Computer Vision

Single and two-view geometry

David Murray

david.murray@eng.ox.ac.uk
www.robots.ox.ac.uk/~dwm/Courses/Aims

2016-17

3.1 Introduction: Photometry & Geometry

If one starts counting from Louis Roberts' blocks-world recognition system built at MIT in the mid 1960s, computer vision is some 50 years old.

There have been many successful paradigms that have focussed research effort ...

Blocks world	Shape-from-X	Model-based vision
Dynamic Vision	Active Vision	Ego-centric vision
Appearance-based vision	Learning-based vision	... etc ...

... but a thread running through all is one that implicitly states ...

L G Roberts *Machine perception of 3D solids* In (Tippett, Berkowitz, Clapp, Koester, and Vanderburgh eds.) Optical and Electrooptical Information Processing, pp159-197. MIT Press, 1965

Intro: Photometry & Geometry

Because

- the geometry of the scene,
- its lighting and reflectance (BRDF), and
- the geometric and photometric properties of the imaging device

are key to 2D image *formation* ...

then

- these same properties are key to image understanding.

Seems a bit obvious ...

... but the plot was (almost) lost in the 1970's with the rise of expert systems. These adopted a

- shallow rule based-approach: *"if above and blue then sky"*

Not a success! Unlike some areas, computer vision was not, and is not, an area where humans can claim that sort of rule-based expertise.

Intro: Photometry & Geometry

Photometry considers the way image formation depends on:

- the nature of scene surface (reflecting, absorbing ...)
- the relative orientations surface, light source and camera
- the power and spectral properties of source
- the spectral properties of the imaging system.

The important overall outcome (eg, Forsyth & Ponce, p62) is that

image irradiance is proportional to the scene radiance

Relief: this means that *what we see* tells us about *what we are looking at*!

Intro: Photometry & Geometry

Eg, by detailed modelling of how air light scatters on water droplets, Shree Nayar's group at Columbia U de-fogged the streets of San Francisco

Original

De-fogged

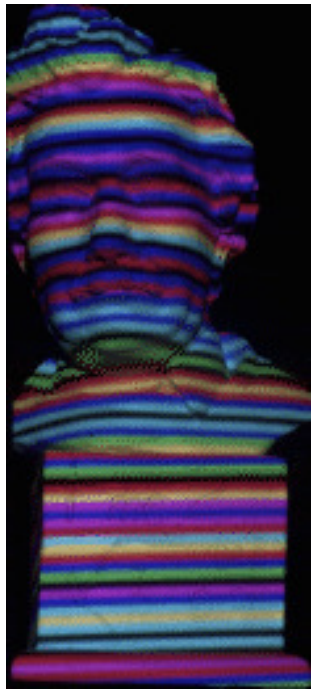
But how does most of computer vision avoid going into such detail?

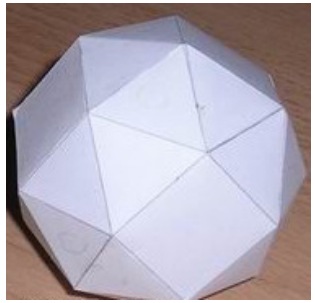
By considering **characteristic changes** in image irradiance that are **glued to aspects of scene geometry.**

Simplest are step changes.

Intro: Photometry & Geometry

1. Changes in Scene Radiance.
Natural (eg lights/shadows) or deliberately introduced via artificial illumination: light stripes are generated either by a mask or from a laser.





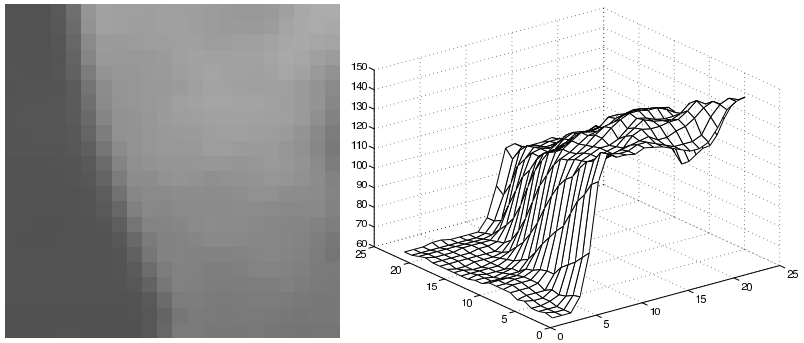
2. Changes in the scene reflectance at sudden changes in surface orientation.
These arise at the intersection of two surfaces, and thus represent geometrical entities fixed on the object in the scene.

3. Changes in reflectance properties due to changes in surface albedo.
The reflectance properties are scaled by a changing albedo arising from surface markings. Again these are fixed to the object.

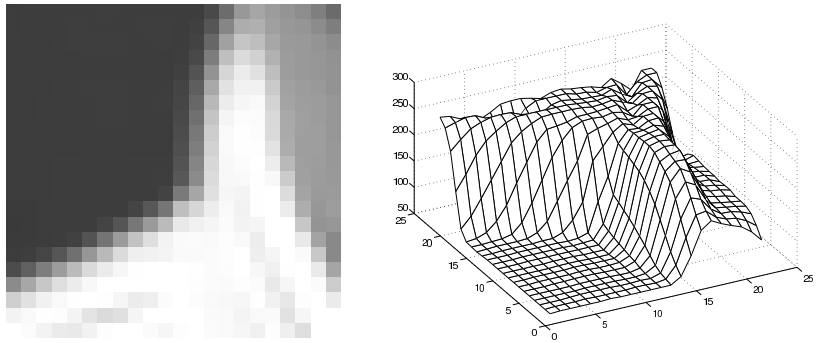
Intro: Photometry & Geometry

Then, at base level, we can consider a classification of changes in image irradiance $I(x, y)$ into

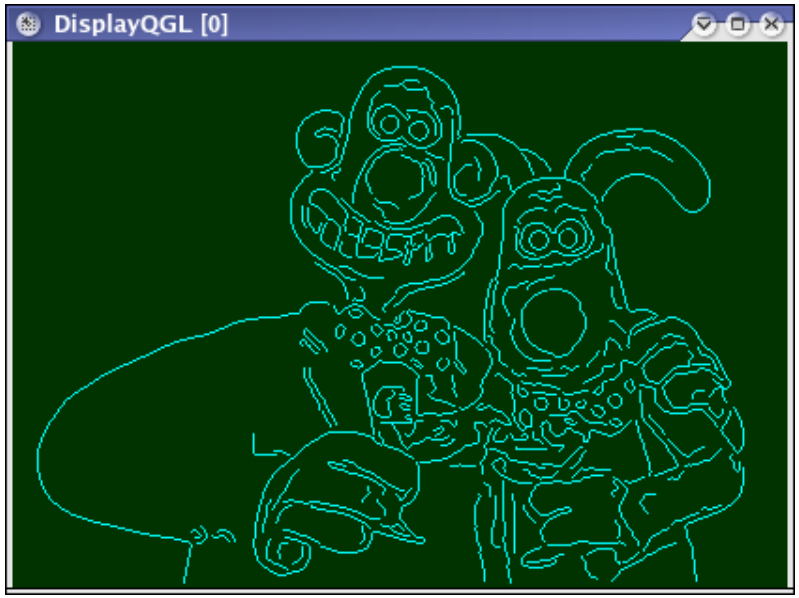
Local 1D structure



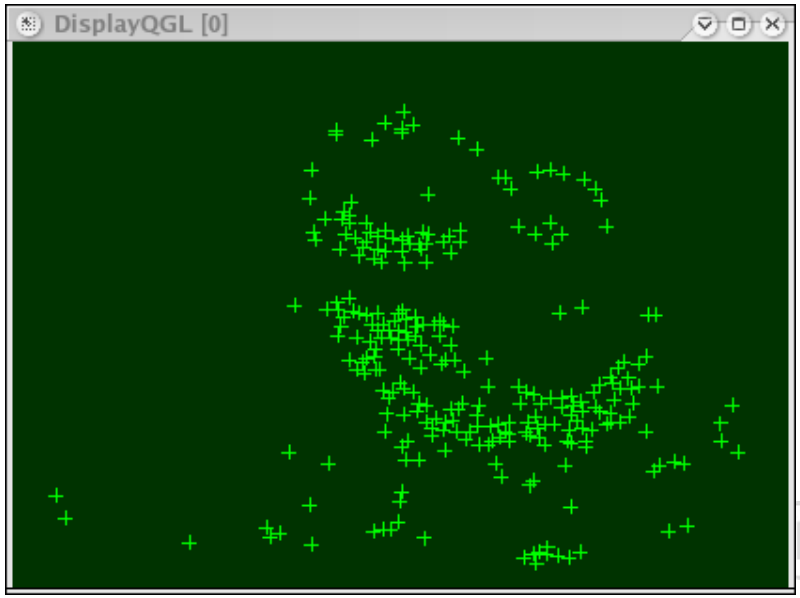
Local 2D structure



$\Rightarrow$ Edge Detectors



$\Rightarrow$ Corner Detectors

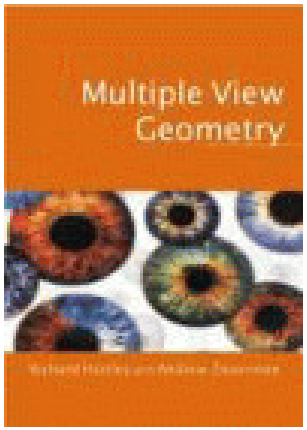


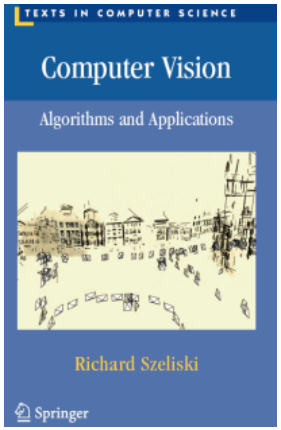
Contents

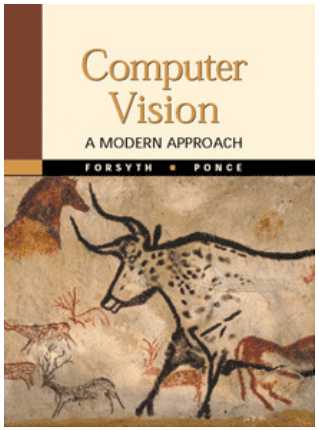
Useful texts

Multiple View Geometry in Computer Vision
Richard Hartley, Andrew Zisserman Cambridge U. Press. ISBN:0521623049

Computer Vision: Algorithms and Applications
Richard Szeliski Springer: online at szeliski.org/book



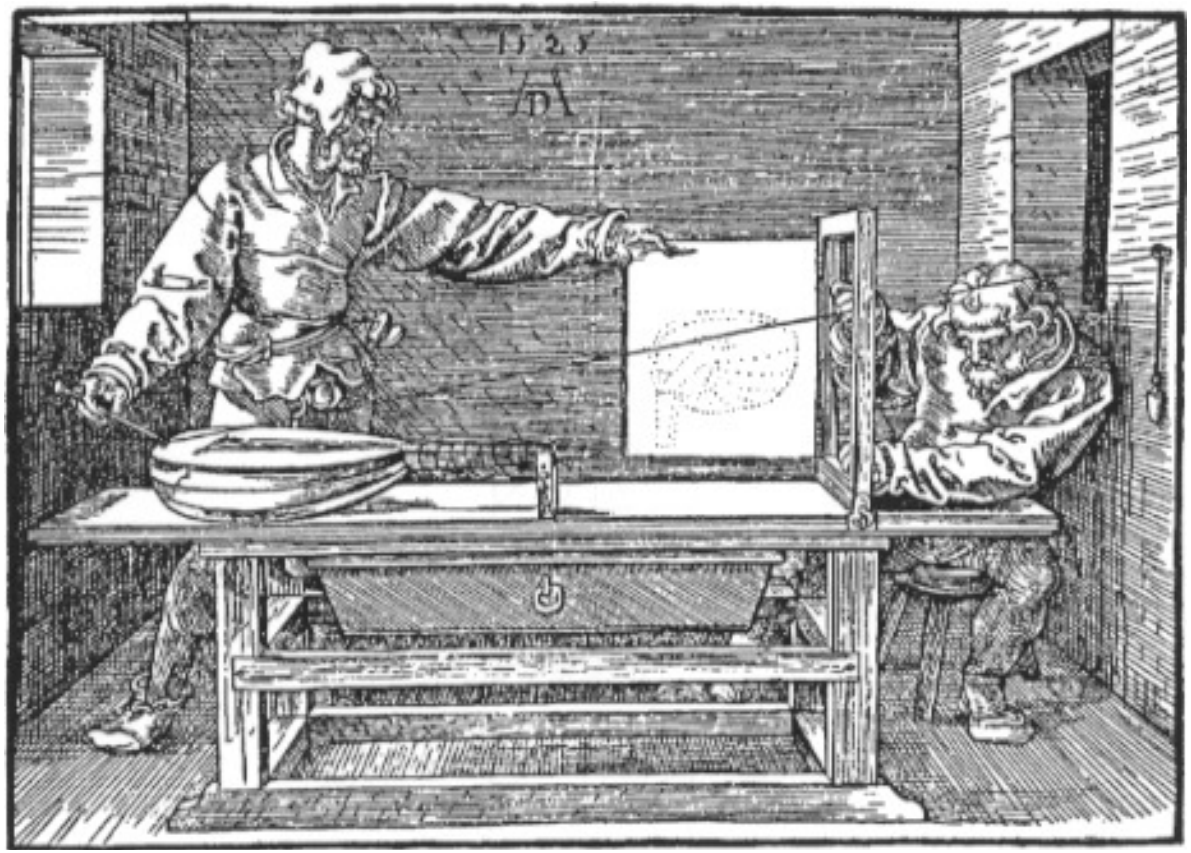




Computer Vision, A Modern Approach
David Forsyth, Jean Ponce Prentice Hall; ISBN:0130851981

Contents

3.2 The perspective camera as a geometric device



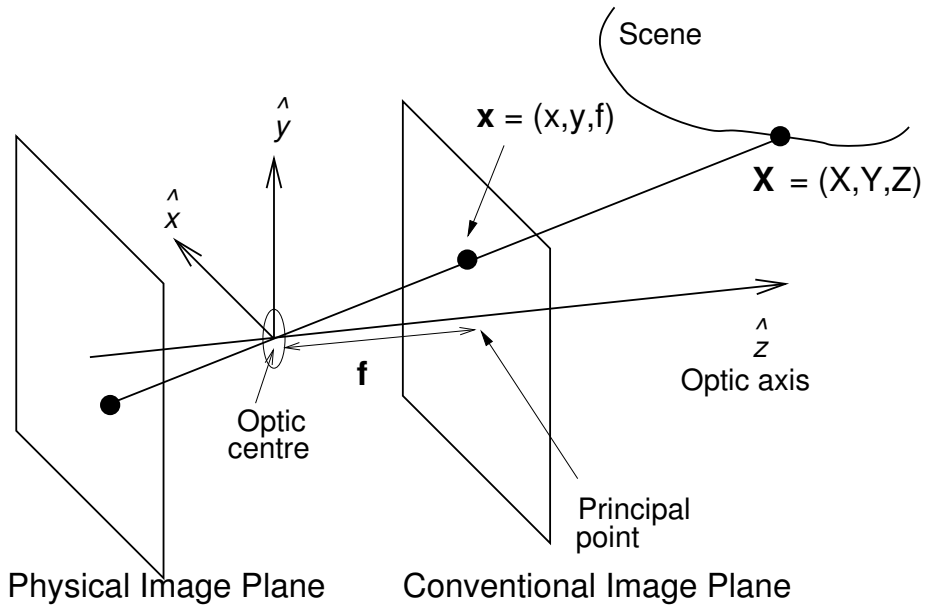
The perspective camera as a geometric device

Standard cameras project images as though light rays enter via pinhole at the **optical centre**.

The z-axis or **optical axis** is perpendicular to the image plane and passes through the optical centre.

Trivial to show that scene point at $\mathbf{X} = (X, Y, Z)^\top$ is imaged at $\mathbf{x} = (x, y)^\top$ where

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix} .$$



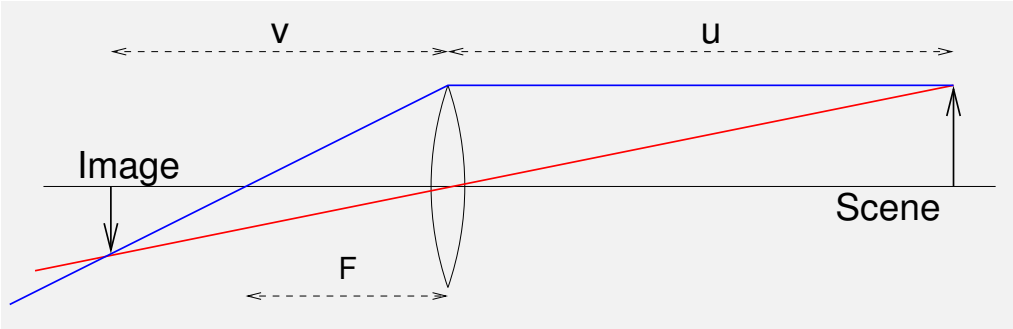
NB! Pinhole f is not lens focal length

Quantity f is usually called the **focal length** but is really
the distance from the image plane to the optical centre

It is **NOT** the focal length F that occurs in the optics lens formula ...

$$\frac{1}{F} = \frac{1}{u} + \frac{1}{v} = \frac{1}{Z} + \frac{1}{v}$$

where $u = Z$ is the object distance, and v is the image distance.



The construction shows that in this case the scaling between **X** and **x** is not F/Z, but v/Z!

$\Rightarrow$Our pinhole f is equivalent to lens formula v.

Homogeneous coordinates

Unfortunately ... The projection equation $\mathbf{x} = f\mathbf{X}/Z$ is non-linear.

Better news ... The projection can be made linear by adopting **homogeneous coordinates**, which involves representing the image and scene in a higher dimensional space.

Limiting cases — *eg* points at infinity and vanishing points — are handled better.

Using homogeneous coordinates also allows transformations to be concatenated more easily ...

... we will start with the latter as motivation.

3D Euclidean transforms: inhomogeneous coords

A **Euclidean transformation** in 3D involves a rotation matrix and translation. Using inhomogeneous coordinates:

$$\mathbf{X}'_{3\times 1} = \boldsymbol{R}_{3\times 3}\mathbf{X}_{3\times 1} + \mathbf{t}_{3\times 1}$$

where $\boldsymbol{R}$ is the orthogonal rotation matrix $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}$ and $\mathbf{X}'$ etc are column vectors.

Concatenation of successive transformations is a mess!

$$\begin{aligned}
\mathbf{X}_1 &= \boldsymbol{R}_1\mathbf{X} + \mathbf{t}_1 \\
\mathbf{X}_2 &= \boldsymbol{R}_2\mathbf{X}_1 + \mathbf{t}_2 \\
&= \boldsymbol{R}_2(\boldsymbol{R}_1\mathbf{X} + \mathbf{t}_1) + \mathbf{t}_2 \quad = \quad (\boldsymbol{R}_2\boldsymbol{R}_1)\mathbf{X} + (\boldsymbol{R}_2\mathbf{t}_1 + \mathbf{t}_2)
\end{aligned}$$

Euclidean transformations using homogeneous coords

If instead 3D points $\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$ are represented by a four-vector $\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ the transformation can be represented by a 4×4 matrix ...

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \boldsymbol{E} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

Transformations can now be concatenated by matrix multiplication. For example suppose

$$\begin{bmatrix} \mathbf{X}_1 \\ 1 \end{bmatrix} = \boldsymbol{E}_{10} \begin{bmatrix} \mathbf{X}_0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} \mathbf{X}_2 \\ 1 \end{bmatrix} = \boldsymbol{E}_{21} \begin{bmatrix} \mathbf{X}_1 \\ 1 \end{bmatrix}$$

then

$$\begin{bmatrix} \mathbf{X}_2 \\ 1 \end{bmatrix} = \boldsymbol{E}_{21}\boldsymbol{E}_{10} \begin{bmatrix} \mathbf{X}_0 \\ 1 \end{bmatrix}$$

Easy to check ...

Though you have to think about how matrices within matrices work ...

$$\begin{bmatrix} \mathbf{X}_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1 & \mathbf{t}_1 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1\mathbf{X}_0 + \mathbf{t}_1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{X}_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_2 & \mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_2 & \mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_1 & \mathbf{t}_1 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{R}_2\boldsymbol{R}_1 & \boldsymbol{R}_2\mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_0 \\ 1 \end{bmatrix} = \begin{bmatrix} (\boldsymbol{R}_2\boldsymbol{R}_1)\mathbf{X}_0 + (\boldsymbol{R}_2\mathbf{t}_1 + \mathbf{t}_2) \\ 1 \end{bmatrix}$$

Reminder: the matrix has block form:

$$\begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} = \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Homogeneous notation — definition in R^3

- $\mathbf{X} = (X, Y, Z)^\top$ is represented in homogeneous coordinates by **ANY** 4-vector

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

  such that $X = X_1/X_4 \qquad Y = X_2/X_4 \qquad Z = X_3/X_4$

- So the following homogeneous vectors represent the same point for any $\lambda \neq 0$.

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \quad \text{and} \quad \lambda \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

- E.g. $(2, 3, 5, 1)^\top$, $(4, 6, 10, 2)^\top$ & $(-3, -4.5, -7.5, -1.5)^\top$ all represent the **same** inhomogeneous point $(2, 3, 5)^\top$

Homogeneous notation – rules for use

1. Convert the inhomogeneous point to an homogeneous vector:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2. Apply a 4×4 transformation.

3. Dehomogenize the resulting vector:

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \rightarrow \begin{bmatrix} X_1/X_4 \\ X_2/X_4 \\ X_3/X_4 \end{bmatrix}$$

NB general transformations only to be defined up to scale.
E.g.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ -2 & 3 & -4 & 5 \\ 6 & 2 & 1 & 9 \\ 4 & -1 & 0 & 1 \end{bmatrix}$$

and

$$\begin{bmatrix} 2 & 4 & 6 & 8 \\ -4 & 6 & -8 & 10 \\ 12 & 4 & 2 & 18 \\ 8 & -2 & 0 & 2 \end{bmatrix}$$

represent the SAME transformation

The Euclidean transformation is a special case, as $\boldsymbol{R}\boldsymbol{R}^\top = \boldsymbol{I}$

Homogeneous notation — definition in R^2

A 2D inhomogeneous vector $\mathbf{x} = (x, y)^\top$ is represented in homogeneous coordinates by any **3-vector**

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

such that

$$x = x_1/x_3 \qquad y = x_2/x_3$$

E.g.

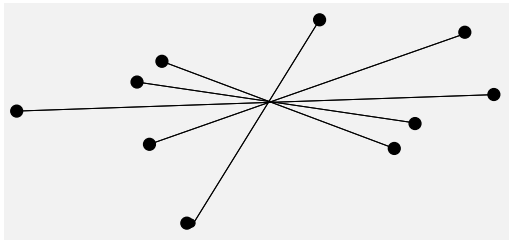
$$(1, 2, 3)^\top \equiv (3, 6, 9)^\top \equiv (-2, -4, -6)^\top$$

all represent the **same** inhomogeneous 2D point $(0.33, 0.66)^\top$

Transformations using hc's: 3D Projective

A projective transformation is a linear transformation on homogeneous 4-vectors represented by a non-singular 4×4 matrix.

$$\begin{bmatrix} X_1' \\ X_2' \\ X_3' \\ X_4' \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$



The effect on the homogenous points is that the original and transformed points are linked through a projection centre

The 4×4 matrix is defined up to scale, and so has 15 dof

HC: More 3D-3D & 2D-2D Transformations

Projectivity (15 dof)

$$\begin{bmatrix} X'_1 \\ X'_2 \\ X'_3 \\ X'_4 \end{bmatrix} \stackrel{P}{=} [\boldsymbol{P}_{4\times 4}] \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

Projectivity aka Homography (8 dof)

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \stackrel{P}{=} \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Affine (12 dof)

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{3\times 3} & \mathbf{t}_3 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

Affine (6 dof)

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{2\times 2} & \mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

Similarity (7 dof)

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} s\boldsymbol{R}_{3\times 3} & s\mathbf{t}_3 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

Similarity (4 dof)

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} s\boldsymbol{R}_{2\times 2} & s\mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

Euclidean (6 dof)

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{3\times 3} & \mathbf{t}_3 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

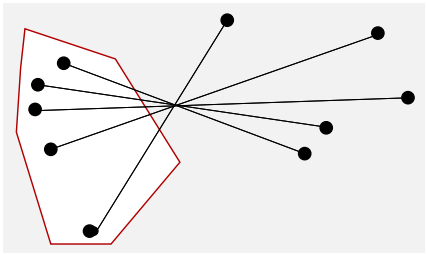
Euclidean (3 dof)

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{2\times 2} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

Perspective 3D-2D transformations

Think about a 3D-3D projectivity, then constrain the transformed points to lie on plane $z = f$

$$z = f \Rightarrow \quad \mathbf{X}_{\text{image}} = \begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix}$$



Because f is fixed, it must be that

$$\lambda \begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ fp_{31} & fp_{32} & fp_{33} & fp_{34} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

But the 3rd row is completely redundant, so that

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \boldsymbol{P}_{3\times 4} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

This is a **perspective transformation**.
$\boldsymbol{P}_{3\times 4}$ is the **projection matrix**.

Perspective using homogeneous coordinates

We now write down the form of the projection matrix for a simple pin-hole camera with focal length f, where the camera frame coincides with the world frame.

Image Point	Projection Matrix	World Point
$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$	$= \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$

Then

$$\lambda x = fX \qquad \lambda y = fY \qquad \lambda = Z$$

and

$$\Rightarrow \quad x = fX/Z \qquad y = fY/Z$$

— exactly what we wrote down using similar triangles

Perspective using homogeneous coordinates

It is useful to split up the overall projection matrix into (i) a part that depends on the internals of the camera, (ii) a vanilla projection matrix, and (iii) a Euclidean transformation between the world and camera frames.

First assume that the scene or world coords are aligned with the camera coords, so that the **Extrinsic Calibration Matrix** is an identity:

Image Point	**Camera's Intrinsic Calibration**	Projection matrix (vanilla)	**Camera's Extrinsic Calibration**	World Point

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{X}$$

Perspective using homogeneous coordinates /ctd

Now make the projection matrix more general ...

1. Insert a Rotation $\boldsymbol{R}$ and translation $\mathbf{t}$ between world and camera coordinates
2. Insert extra terms in the intrinsic calibration matrix

Intrinsic Calibration	Vanilla Projection	**Extrinsic Calibration**

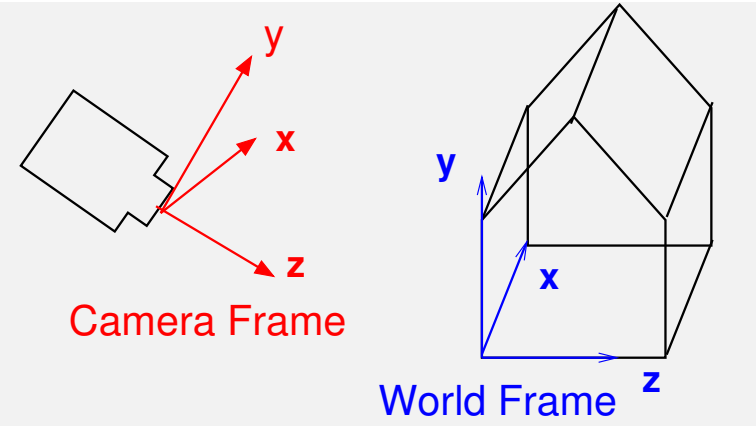
$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & sf & u_0 \\ 0 & \gamma f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{X}$$

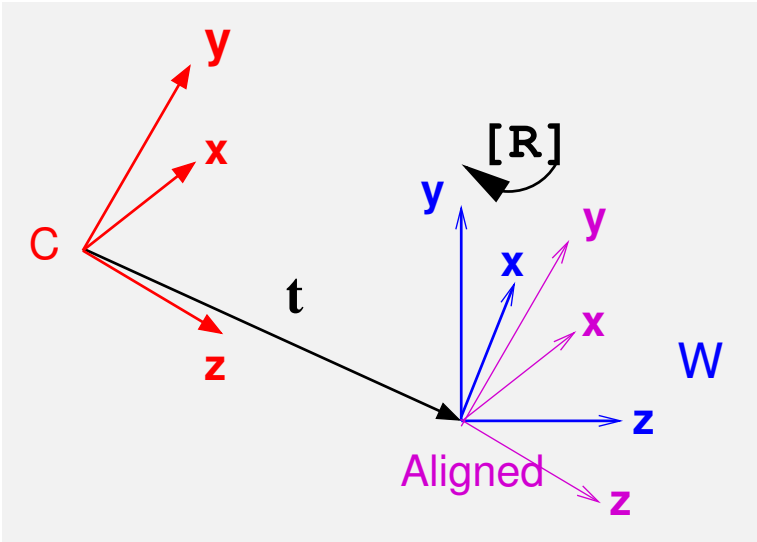
The extrinsic calibration parameters

The camera's extrinsic calibration is just the **rotation** $\boldsymbol{R}$ and **translation** $\mathbf{t}$ that take points in the world or scene frame into the camera frame

$$\begin{bmatrix} \mathbf{X}^C \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}^W \\ 1 \end{bmatrix}$$

The vector $\mathbf{t}$ is the origin of the world frame expressed in the camera frame.



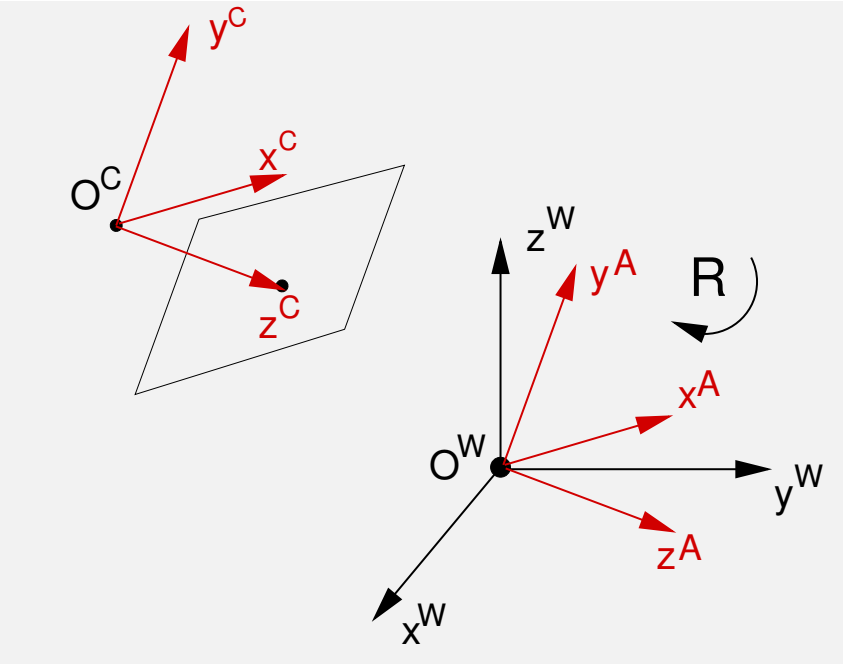


Building the Euclidean transformation in steps

It is convenient to build the transformation in stages.

Let A be a frame aligned with the camera.

Think about the rotation between the world and the aligned frame A ...



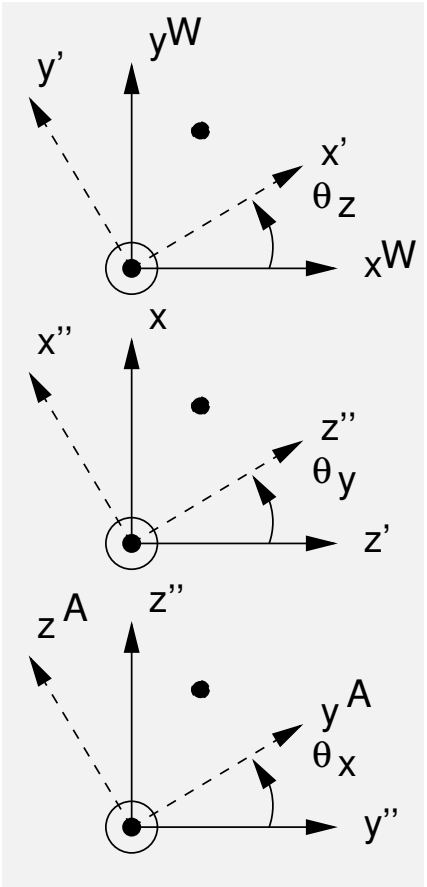
Building a Euclidean transformation in steps /ctd

Break the rotation into simple ones through θ_z about z, then θ_y about y, and θ_x about x. Let frames ' and '' be intermediate frames ...

$$\mathbf{X}' = \boldsymbol{R}_z \mathbf{X}^W = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{X}^W$$

$$\mathbf{X}'' = \boldsymbol{R}_y \mathbf{X}' = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \mathbf{X}'$$

$$\mathbf{X}^A = \boldsymbol{R}_x \mathbf{X}'' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \pm\sin\theta_x \\ 0 & \mp\sin\theta_x & \cos\theta_x \end{bmatrix} \mathbf{X}''$$



NB order is important!

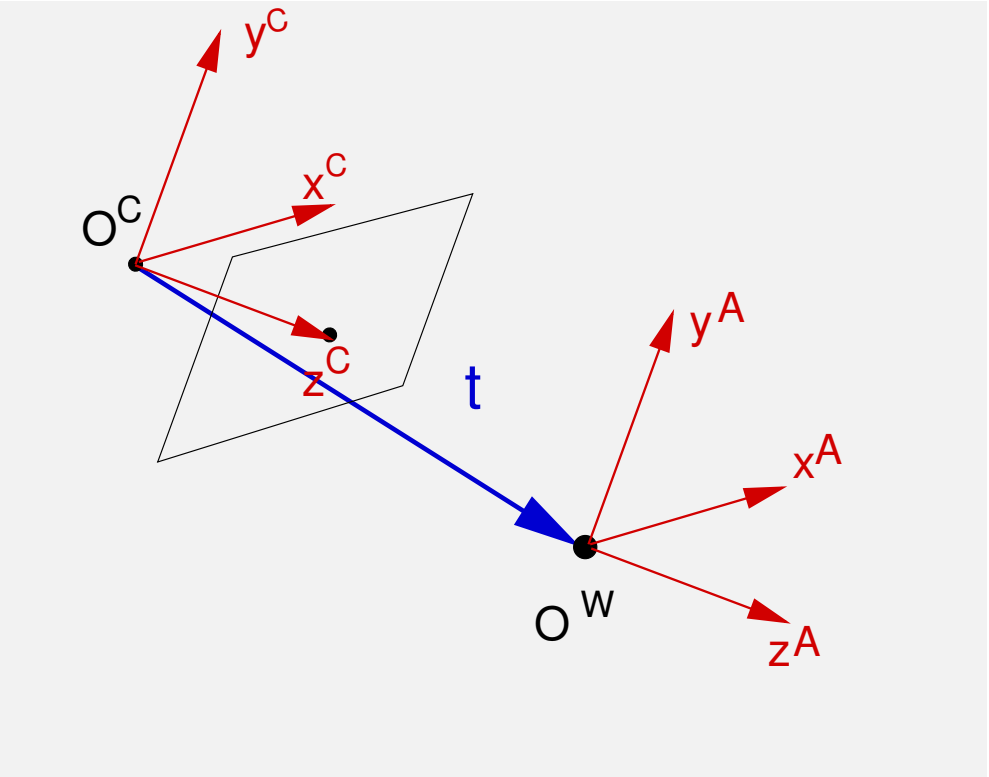
$$\boldsymbol{R}^{CW} = \boldsymbol{R}_x \boldsymbol{R}_y \boldsymbol{R}_z$$

Build transformations in steps ...

Then deal with translation

$$\begin{aligned} \mathbf{X}^C &= \mathbf{X}^A + \mathbf{t}_A^C \\ &= \mathbf{X}^A + \mathbf{t}_W^C \\ &= \boldsymbol{R}^{CW}\mathbf{X}^W + \mathbf{t}_W^C \end{aligned}$$

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{R}^{CW} & \mathbf{t}_W^C \\ \mathbf{0}^\top & 1 \end{bmatrix}$$



The inverse of a Euclidean transformation?

It must be the case that

$$\begin{bmatrix} \boldsymbol{R}^{CW} & \mathbf{t}_W^C \\ \mathbf{0}^\top & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{R}^{WC} & \mathbf{t}_C^W \\ \mathbf{0}^\top & 1 \end{bmatrix}$$

Now, we know that

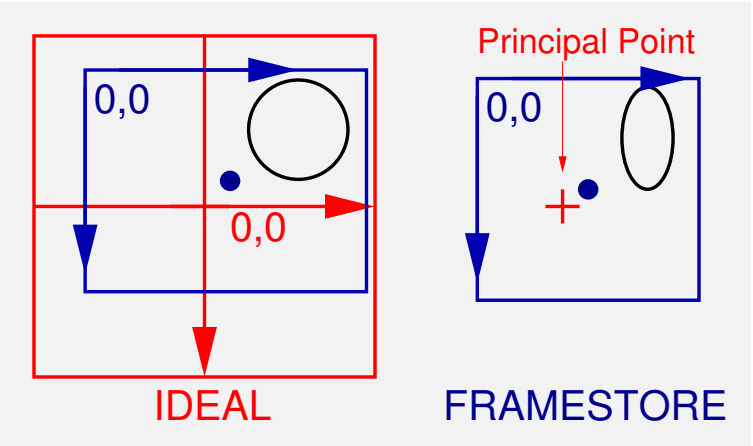
$$\boldsymbol{R}^{WC} = \left[\boldsymbol{R}^{CW}\right]^{-1} = \left[\boldsymbol{R}^{CW}\right]^\top$$

but what is $\mathbf{t}_C^W$? — tempting to say $-\mathbf{t}_W^C$, but NO!

$$\begin{aligned}
\mathbf{X}^C &= \boldsymbol{R}^{CW}\mathbf{X}^W + \mathbf{t}_W^C && (\mathbf{t}_W^C \textit{ is Origin of } W \textit{ refd to } C) \\
\Rightarrow \mathbf{X}^W &= \boldsymbol{R}^{WC}(\mathbf{X}^C - \mathbf{t}_W^C) \\
\Rightarrow \mathbf{X}^W &= \boldsymbol{R}^{WC}\mathbf{X}^C - \boldsymbol{R}^{WC}\mathbf{t}_W^C \\
\textit{But by def'n}: \mathbf{X}^W &= \boldsymbol{R}^{WC}\mathbf{X}^C + \mathbf{t}_C^W && (\mathbf{t}_C^W \textit{ is Origin of } C \textit{ refd to } W) \\
\Rightarrow \mathbf{t}_C^W &= -\boldsymbol{R}^{WC}\mathbf{t}_W^C
\end{aligned}$$

Perspective Projection: Intrinsic Parameters

1. In a real camera the image plane might be skewed.
2. The central axis of the lens might not line up with the optical axis.
3. The light gathering elements might be rectangular, not square.



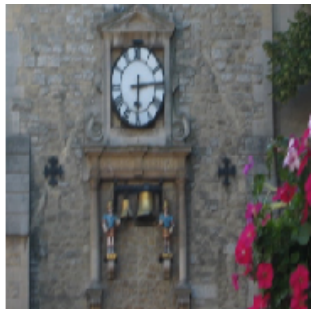
1. Introduce a parameter $s \approx 0$ to account for axis skew.
2. Introduce an origin offset. (u_0, v_0) is the principal point, where where optic axis strikes the image plane. (So u_0/f, $v_0/\gamma f$ is the offset before scaling!)
3. Allow different scaling in x and y directions. γ is the aspect ratio.

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & \gamma f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & u_0/f \\ 0 & 1 & v_0/\gamma f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f & sf & u_0 \\ 0 & \gamma f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Why can the aspect ratio γ be non-unity?

Assume a 1-to-1 mapping between camera and framestore pixels. However, the camera pixels (that is, the elements gathering light) might not be square.

Camera has
Rectangular pixels $w > h$



Framestore Image

Look at the framestore image. To take such an image the camera lens would be wide-angle in the $x-$direction, but narrow-angle in $y-$dirn.

$$\Rightarrow f_x < f_y \qquad \Rightarrow f < \gamma f \qquad \Rightarrow \gamma > 1 \qquad \Rightarrow \gamma = w_{\text{pix}}/h_{\text{pix}}$$

Good news! Modern digital cameras have $\gamma \approx 1$.

Summary of projection

Move scene point is $(\mathbf{X}^W, 1)^\top$ into camera coord frame by a 4×4 extrinsic transform

$$\begin{bmatrix} \mathbf{X}^C \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}^W \\ 1 \end{bmatrix}$$

Project into an "ideal camera" with $f = 1$, $\gamma = 1$, $s = 0$ by

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} \overset{P}{=} [\boldsymbol{I}|\mathbf{0}] \begin{bmatrix} \mathbf{X}^C \\ 1 \end{bmatrix}$$

Map into the real image using the intrinsics

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix}$$

NB: an equals sign: the bottom right element of $\boldsymbol{K}$ is $K_{33} = 1$.

3.3 Camera Calibration

To make Euclidean 3D measurements the camera's intrinsic and extrinsic calibration must be known.

An uncalibrated camera can only recover 3D information up to a unknown projective transformation of the scene. Then the $\mathbf{X}$ recovered relates to the Euclidean $\mathbf{X}_E$ by an unknown 4×4 projectivity

$$\mathbf{X}_E \stackrel{P}{=} \boldsymbol{H}_{4\times4}\mathbf{X} \ .$$

There are a variety of methods for **self-calibration** or **auto-calibration** of cameras. These can recover $\boldsymbol{H}_{4\times4}$.

Here we consider a method of **pre-calibration** using a specially made "known" visual scene

Camera Calibration: Preamble

Multiply out the intrinsic, projection and extrinsic matrices

$$\lambda \mathbf{x} \quad = \quad \boldsymbol{K}\ [\boldsymbol{I}|\mathbf{0}] \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \quad = \quad \boldsymbol{K}\ [\boldsymbol{R}|\mathbf{t}]\mathbf{X} \quad = \quad \boldsymbol{P}_{3\times 4}\mathbf{X}$$

First we need to recover the overall projection matrix $\boldsymbol{P}_{3\times 4}$.
Recall, it has 11 dof.

It is possible to fix one of the elements (say $p_{34} = 1$), but this is risky.

Instead, adopt a null-space method, solving (at first, anyway) a system $\boldsymbol{A}\mathbf{p} = \mathbf{0}$, where $\boldsymbol{A}$ is known, and $\mathbf{p}$ contains the unknown elements of $\boldsymbol{P}$

$$\mathbf{p} = [p_{11}, p_{12}, \ldots, p_{34}]^\top$$

Of course, $\mathbf{p}$ can only be recovered up to scale.

Camera Calibration: Preamble

You've measured image positions $\mathbf{x}$ for a number of known scene points.
For each point i:

$$\lambda_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \boldsymbol{P}\mathbf{X}_i = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \mathbf{X}_i$$

Hence

$$\begin{aligned} \lambda_i &= p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34} \\ (p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34})x_i &= (p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}) \\ (p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34})y_i &= (p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}) \end{aligned}$$

Re-arrange these to give

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -X_i x_i & -Y_i x_i & -Z_i x_i & -x_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -X_i y_i & -Y_i y_i & -Z_i y_i & -y_i \end{bmatrix} \mathbf{p} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

where vector $\mathbf{p} = (p_{11}, p_{12}, \ldots, p_{33}, p_{34})^\top$ contains the unknowns.

So, use 6 points $i = 1 \ldots 6$, to build up a known 12×12 matrix $\boldsymbol{A}$ ...

Calibration: Step 1

Step 0: Create target with known scene points This is the hard bit!

Step 1: Determine the projection matrix

Measure the image positions $\mathbf{x}$ of $n \geqslant 6$ KNOWN scene points $\mathbf{X}$ and build up the system $\boldsymbol{A}\mathbf{p} = \mathbf{0}$

With exactly 6 points, $\mathbf{p}$ is the null-space of $\boldsymbol{A}$. That is, $\mathbf{p} = \ker(\boldsymbol{A})$.

With more points, a least-squares solution is found using SVD:

Find the Singular Value Decompoisiton of $\boldsymbol{A}$: $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top \leftarrow \boldsymbol{A}$

$\mathbf{p}$ is the column of $\boldsymbol{V}$ corresponding to smallest singular value.

(SVD is explained along with Eigen-Decomposition at the end. The smallest singular value would $= 0$ if the data were exact.)

The projection matrix $\boldsymbol{P}$ is now known (up to scale).

Step 2: Invert left block of projection matrix

Step 2:

Construct $\boldsymbol{P}_{\text{LEFT}}$ from the leftmost 3×3 block of $\boldsymbol{P}$.
Invert it to obtain $\boldsymbol{P}_{\text{LEFT}}^{-1}$.

$\boldsymbol{P}_{\text{LEFT}}^{-1}$ has form [Rotation Matrix] $\times$ [Upper triangular matrix].

Explanation:

Remember that $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\mathbf{t}]$, so $\boldsymbol{P}_{\text{LEFT}} = \boldsymbol{K}\boldsymbol{R}$

Hence $\boldsymbol{P}_{\text{LEFT}}^{-1} = \boldsymbol{R}^{-1}\boldsymbol{K}^{-1}$.

You know that the inverse of a rotation matrix is a rotation matrix

The inverse of an upper triangular matrix is also upper triangular.

Step 3: QR decomposition

Step 3.
Apply QR-decomposition technique to $\boldsymbol{P}_{\text{LEFT}}^{-1}$.

Outputs $\mathcal{Q}$ and $\mathcal{R}$ are $\boldsymbol{R}^{-1}$ and $\boldsymbol{K}^{-1}$ respectively.

Explanation

Standard technique decomposes a matrix into a rotation matrix $\mathcal{Q}$ and an upper triangular matrix $\mathcal{R}$.

Watch out!!
In QR decomp, $\mathcal{Q}$ is the rotation matrix — NOT $\mathcal{R}$

Steps 4–7 Mop up

Reminder: Outputs $\mathcal{Q}$ and $\mathcal{R}$ correspond to $\boldsymbol{R}^{-1}$ and $\boldsymbol{K}^{-1}$

Step 4. To find our rotation, set

$\boldsymbol{R} \leftarrow \mathcal{Q}^{-1}$.

Step 5. You'd think that $\boldsymbol{K} = \mathcal{R}^{-1}$, but because the scale of $\boldsymbol{P}$ was unknown, we must re-scale every element of $\boldsymbol{K}$ so that, by convention, $K_{33} = 1$.

Set $\boldsymbol{Temp} \leftarrow \mathcal{R}^{-1}$.

Each $K_{ij} \leftarrow \boldsymbol{Temp}_{ij} / \boldsymbol{Temp}_{33}$.

Each $P_{ij\ new} \leftarrow P_{ij\ old} / \boldsymbol{Temp}_{33}$.

Step 6. Recall $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\mathbf{t}]$, hence $\mathbf{t} = \boldsymbol{K}^{-1}[p_{14} \quad p_{24} \quad p_{34}]^\top$.

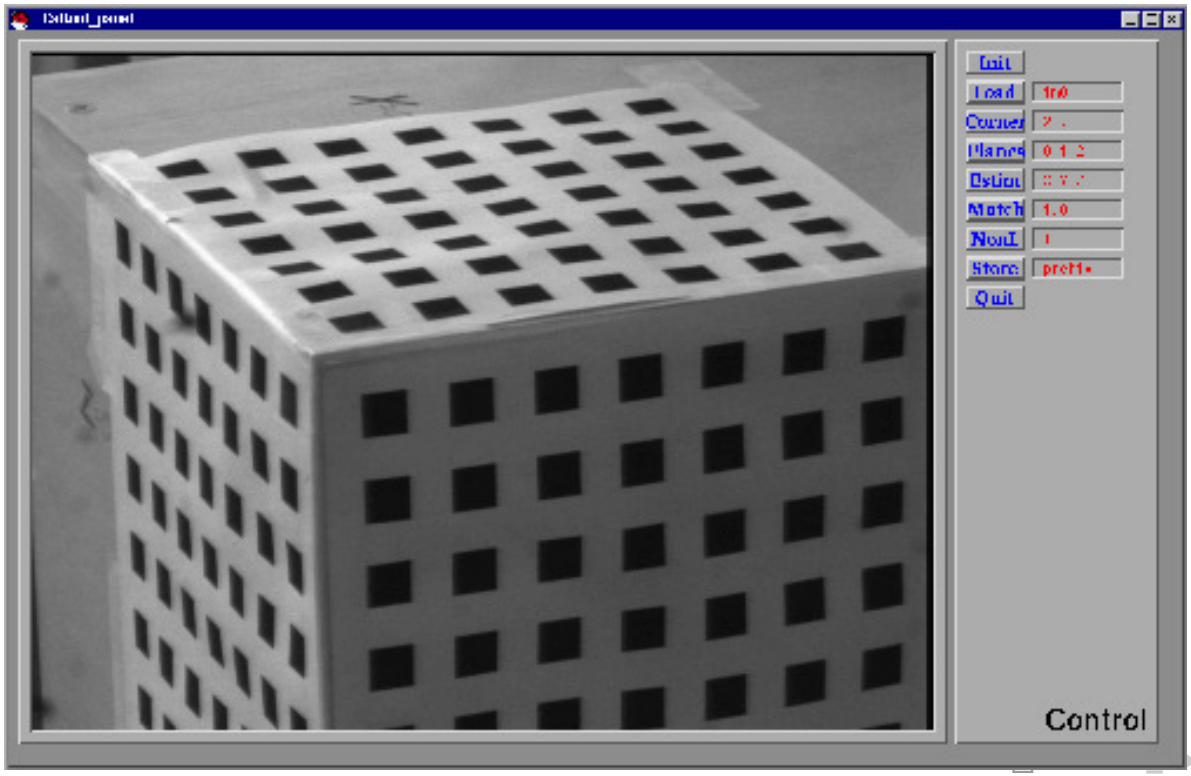
$\mathbf{t} \leftarrow \boldsymbol{K}^{-1}[p_{14} \quad p_{24} \quad p_{34}]^\top$

(Note that you could save some time in Steps 5 and 6 by *not* rescaling $\boldsymbol{P}$ and then computing $\mathbf{t} = \mathcal{R}[p_{14} \quad p_{24} \quad p_{34}]^\top$.)

Step 7. Sort out remaining sign ambiguities. (See Qsheet).

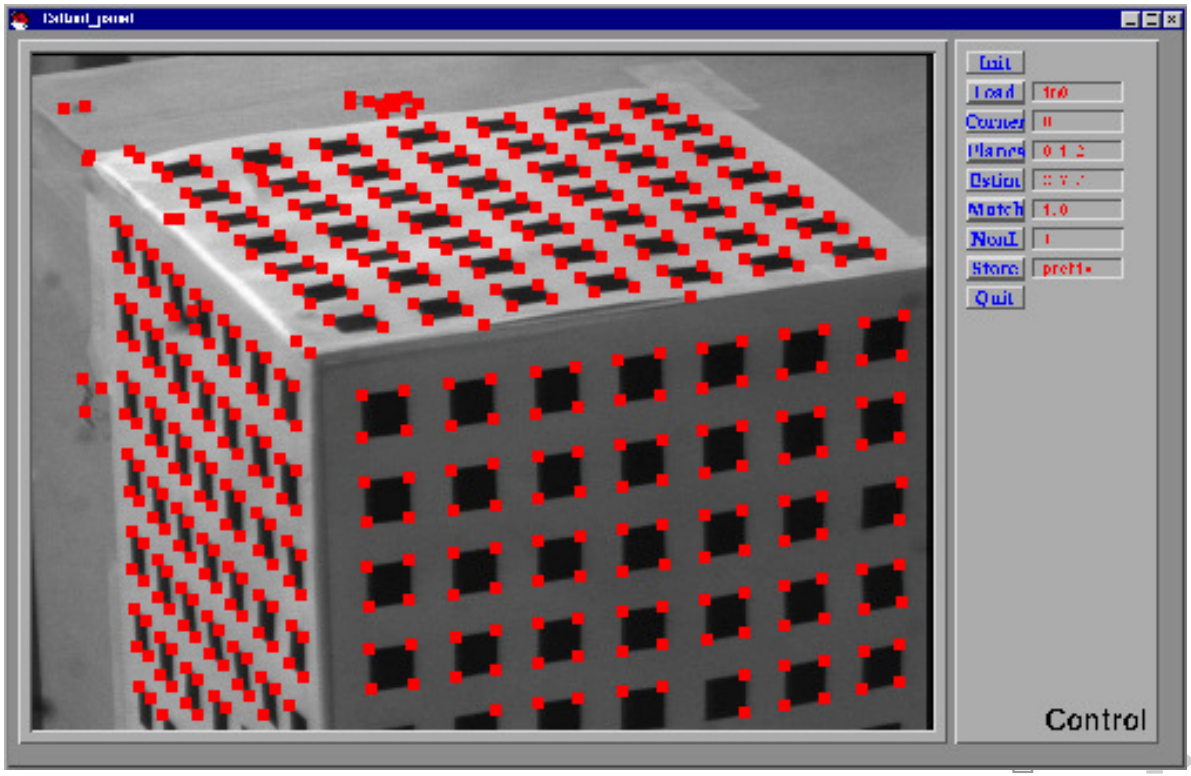
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



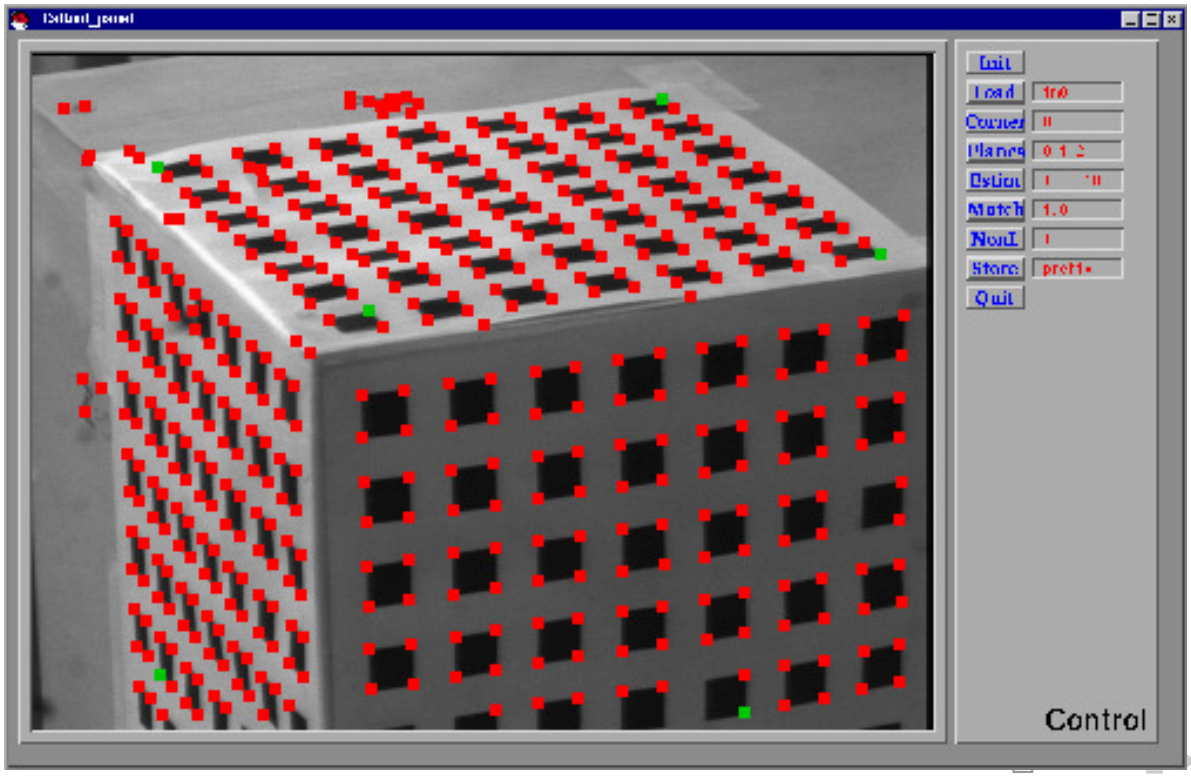
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



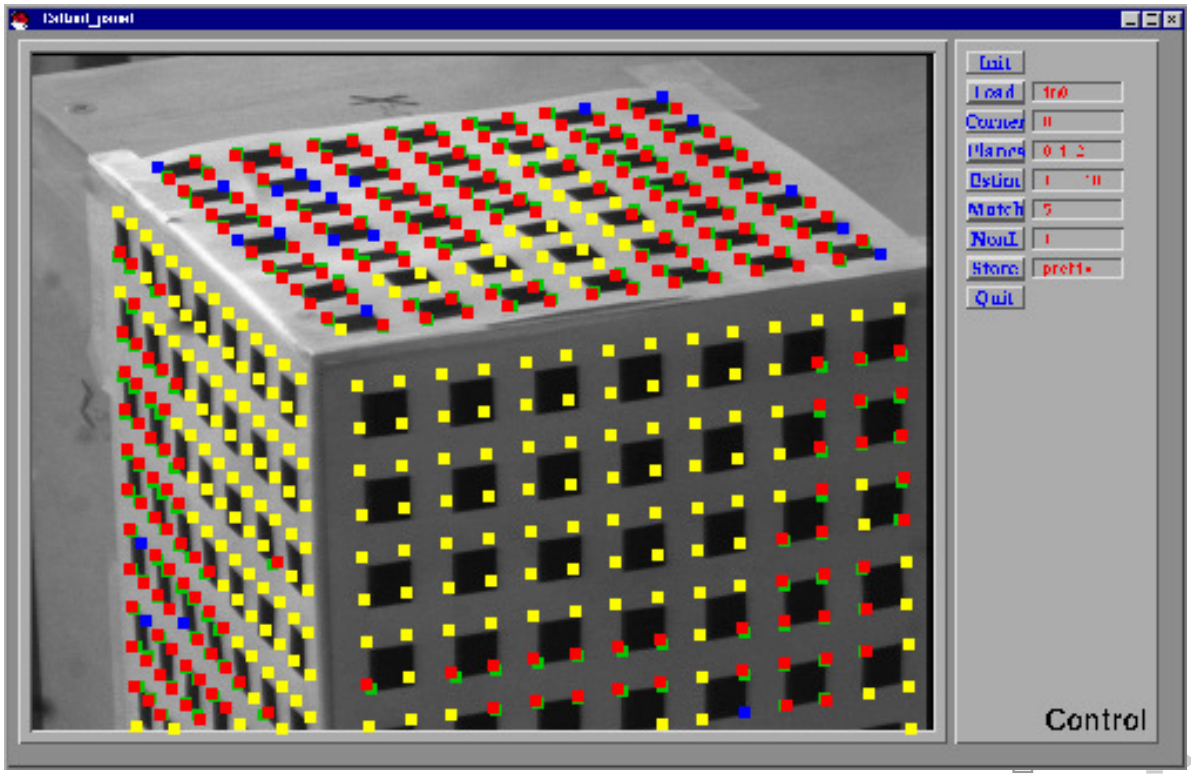
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



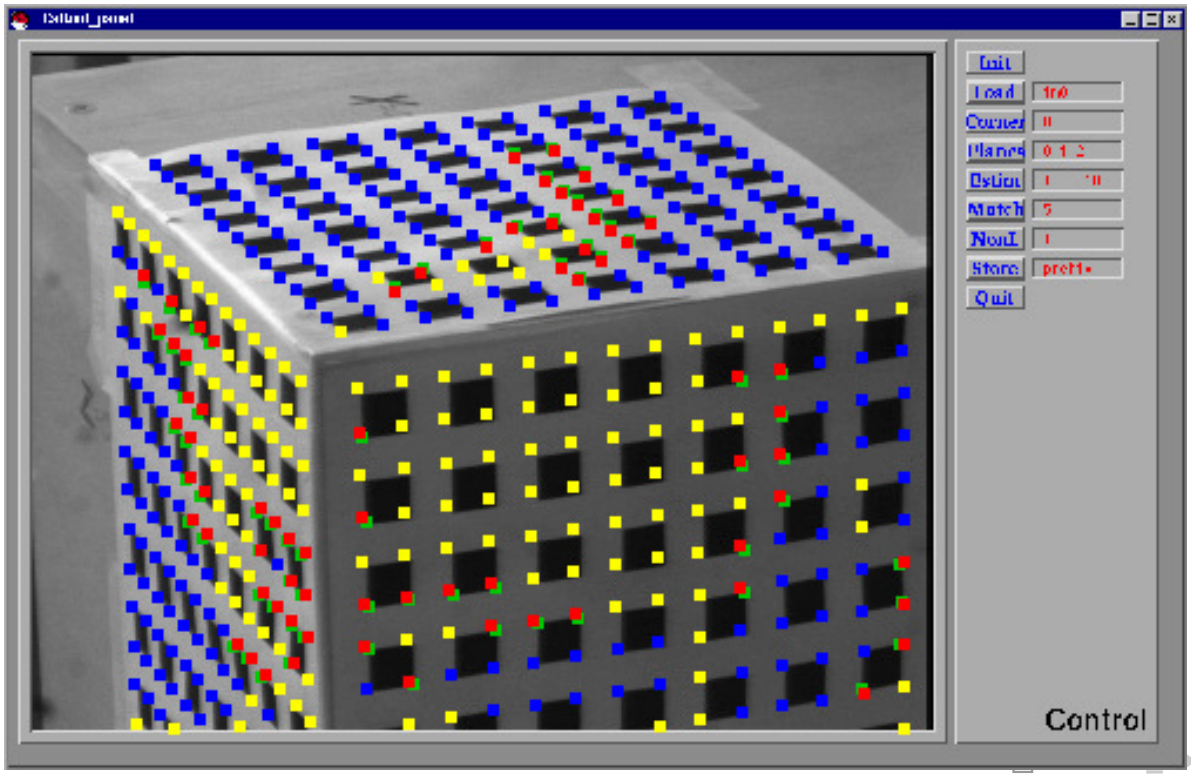
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



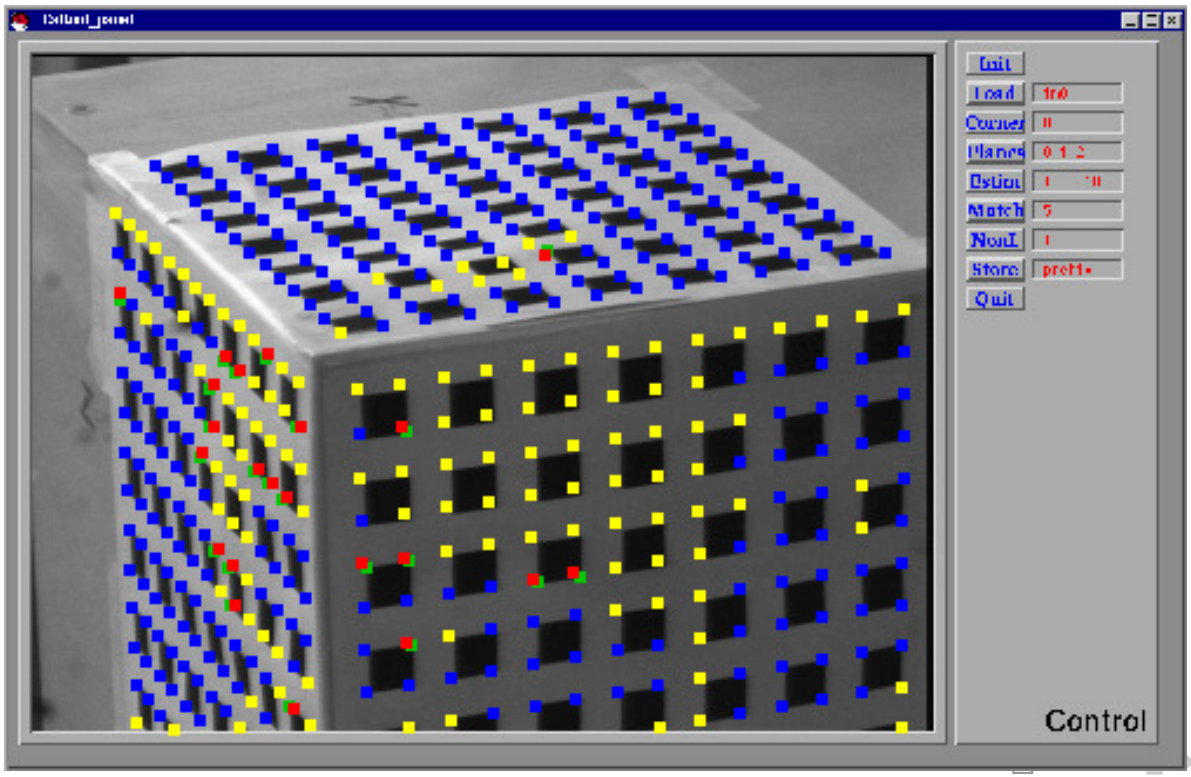
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



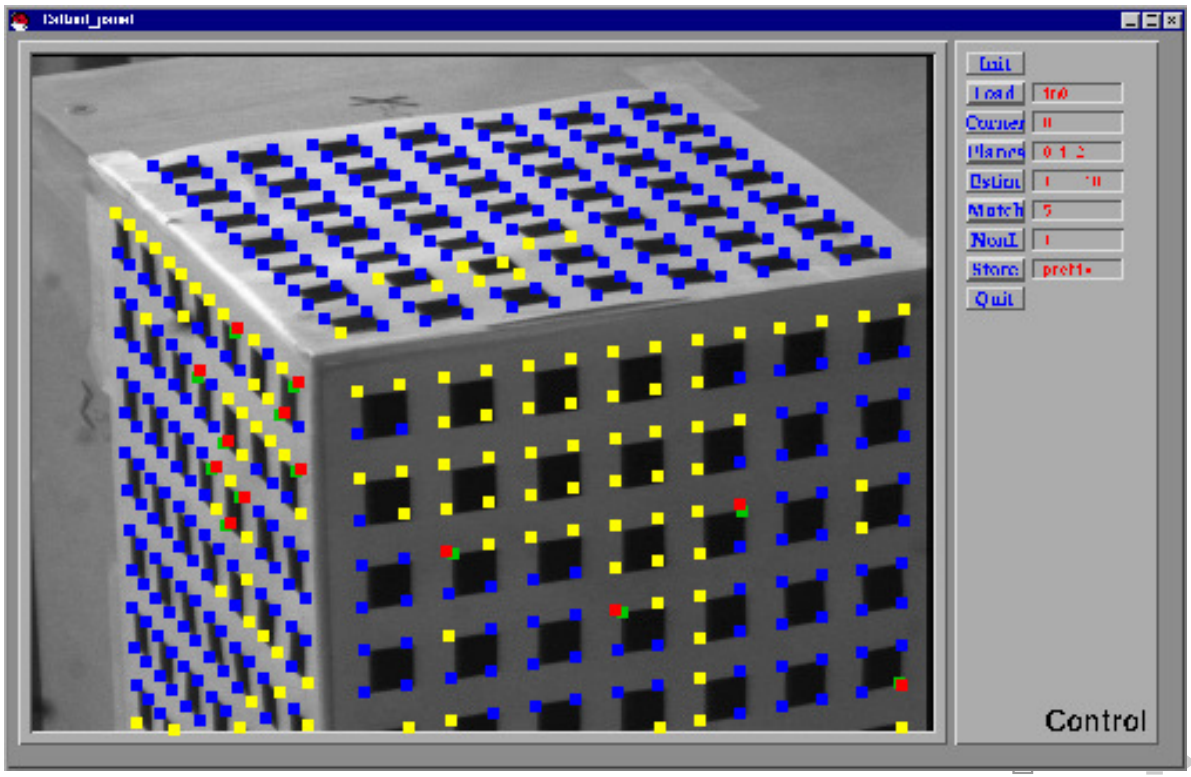
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



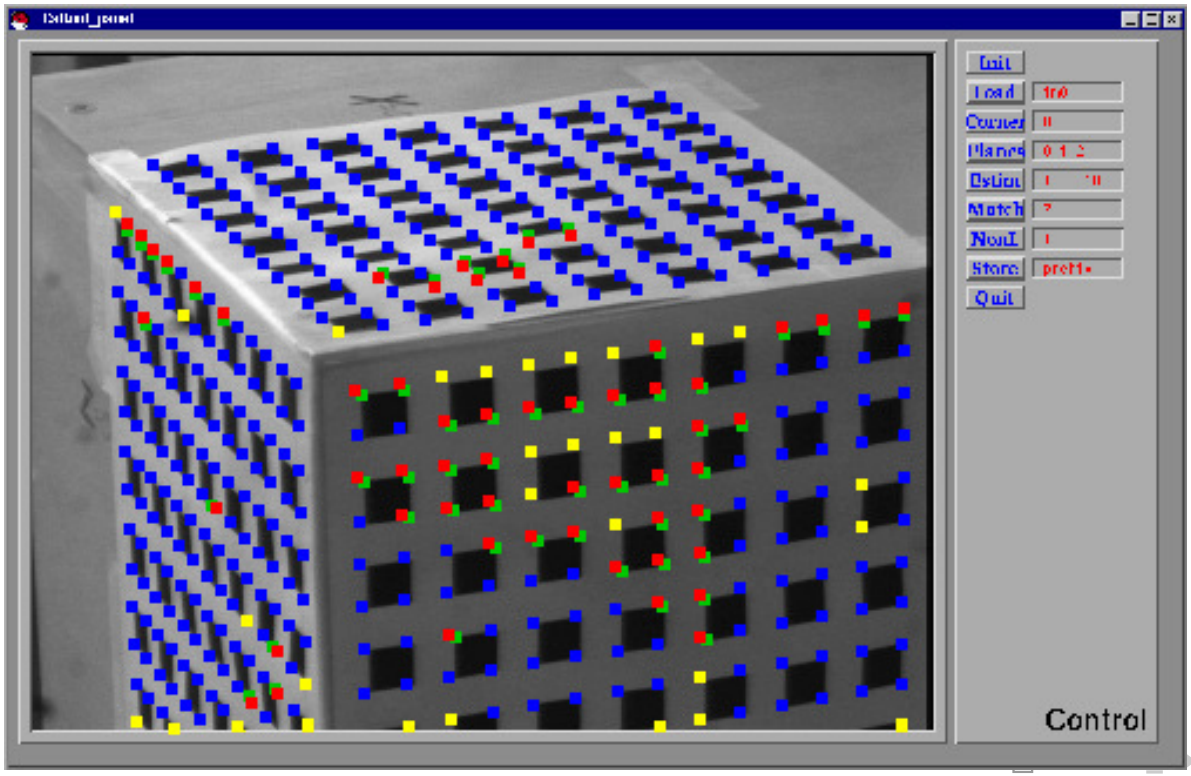
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



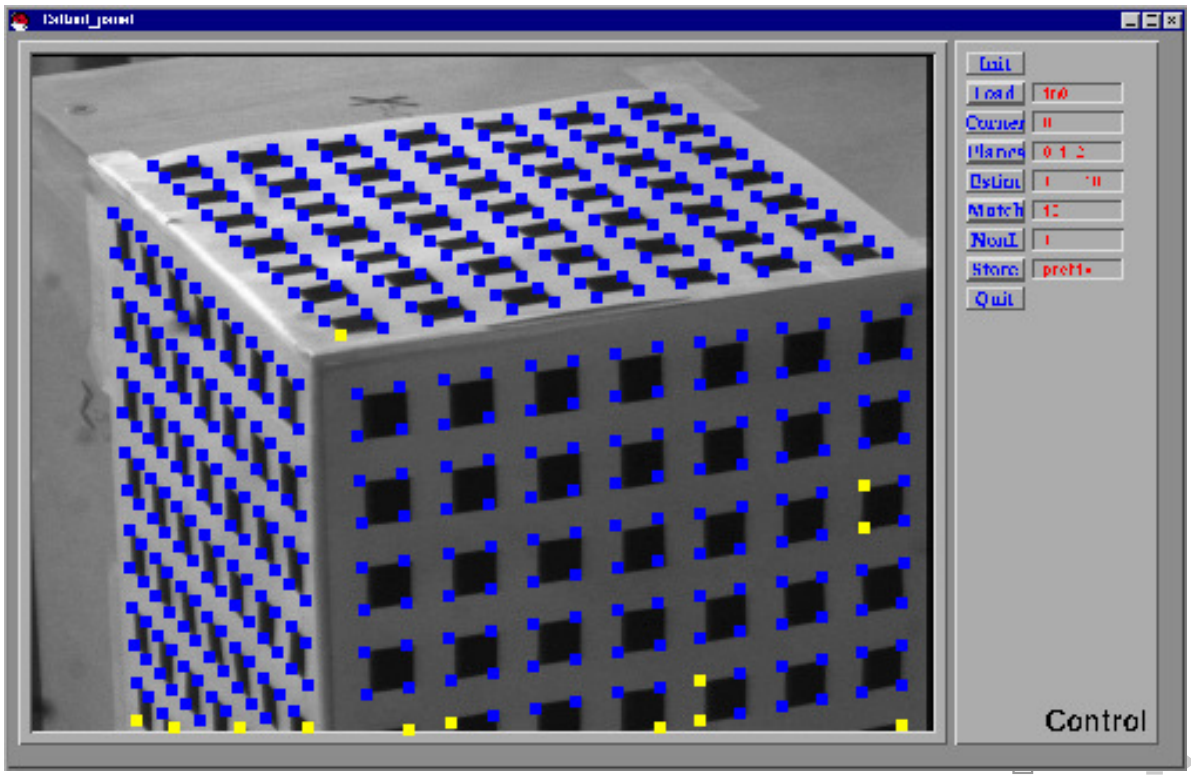
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



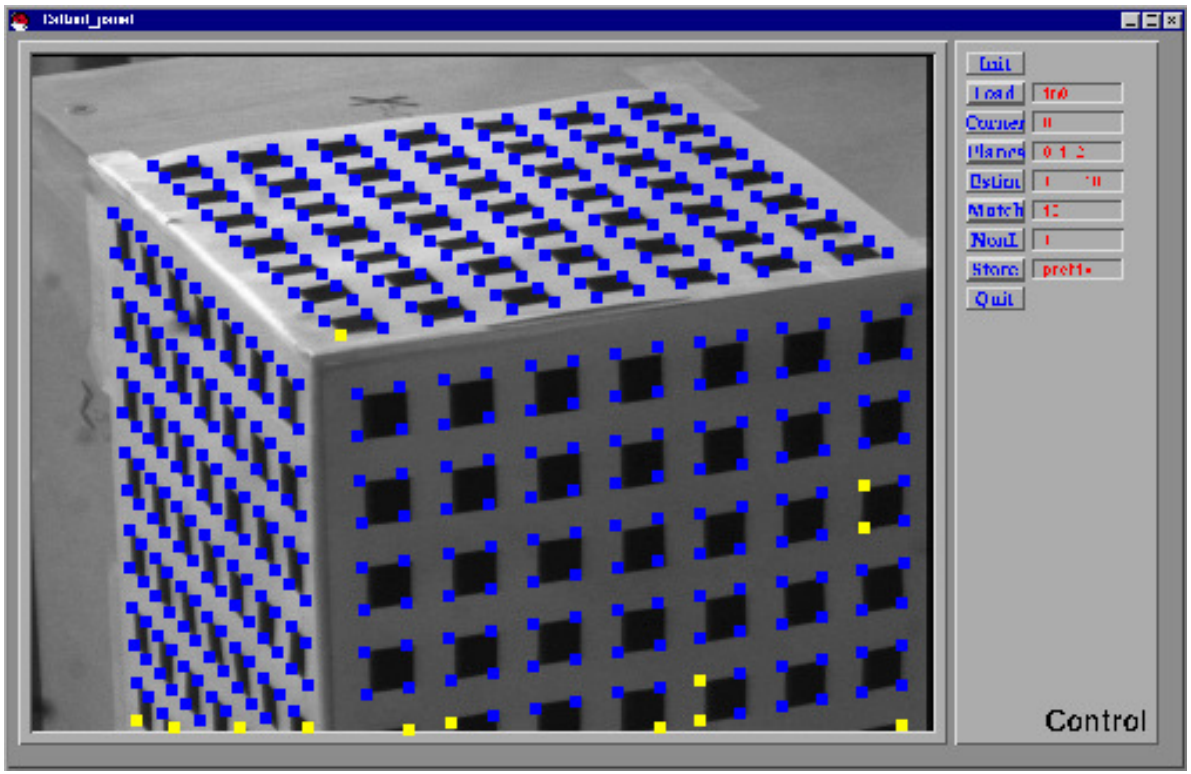
Calibration Scheme

1. Compute corner features
2. Hand match 6 (or more) image points to world points
3. Compute Calibration K, R, t
4. While unmatched points exist
   - 4.1 Project all known world points into image using current calibration
   - 4.2 Match to measurements using threshold on image distance
   - 4.3 Recompute calibration
   - 4.4 Relax threshold if match confusion unlikely



Calibration Example

$$\boldsymbol{K} = \begin{bmatrix} 4622 & 0.1 & 267 \\ 0.0 & 4651 & 558 \\ 0.0 & 0.0 & 1 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} -0.03 & 0.34 & -0.94 \\ -0.92 & 0.36 & 0.16 \\ 0.39 & 0.87 & 0.30 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} 0.7 & 7.5 & 115.8 \end{bmatrix}^\top$$



Radial distortion

So far, we have figured out the transformations that turn our camera into a notional camera with the world and camera coordinates aligned and an "ideal" image plane of the sort given in the first diagram.

One often has to correct for other optical distortions and aberrations. Radial distortion is the most common.

Corrections for this distortion is applied before carrying out calibration.

Barrel Distortion

Undistorted

Pincushion Distortion

Modest radial distortion can be modelled by a single parameter κ. The ideal r is moved to

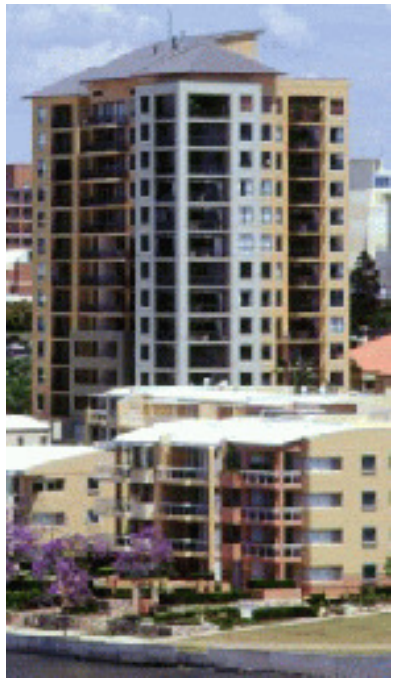
$$r_d = r/\sqrt{1-2\kappa r^2}$$

[**] There are other projection models

Assuming perspective projection is only sensible when the depth variation across a viewed object is comparable with or bigger than the distance to the object

If the depth variation is small (say 10%) of range, trying to recover full perspective is not feasible. It is better to model the projection as **weak-perspective**, akin to choosing an average depth for the scene.

Orthographic projection ignores depth variation entirely.



[**] Eg: Weak perspective is an affine projection

Weak-perspective is a special case of a general linear or *affine* projection from scene to image. Using inhomogenous coordinates

$$\begin{bmatrix} x \\ y \end{bmatrix} = \boldsymbol{M}\mathbf{X} + \mathbf{t} \quad .$$

Using homogeneous coordinates, one sees that the projection matrix gains a specialized form with fewer degrees of freedom

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \overset{P}{=} \boldsymbol{P}_{\text{affine}} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} \qquad \boldsymbol{P}_{\text{affine}} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{12} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & p_{34} \end{bmatrix}$$

$\boldsymbol{M}$ is a 2×3 matrix $m_{ij} = p_{ij}/p_{34}$ and $\mathbf{t} = (p_{14}/p_{34}, p_{24}/p_{34})^\top$

Affine projection matrices correspond to **parallel projections**, or equivalently, projections for which the optic centre is at **infinity**.

[**] Least Squares, Eigendecomposition and SVD

Suppose you are trying to solve $\boldsymbol{A}\mathbf{f} = \mathbf{0}$ using more data than there are degrees of freedom in the system.

That is, three or more points in a straight line, etc.

With noise in the data, a non-trivial, but non-exact, solution can be found using least-squares.

The vector of residuals $\mathbf{r}$ is such that $\boldsymbol{A}\mathbf{f} = \mathbf{r}$.

The sum of their squares is then $||\mathbf{r}||^2 = \mathbf{r}^\top \mathbf{r} = \mathbf{f}^\top \boldsymbol{A}^\top \boldsymbol{A}\mathbf{f} = \mathbf{f}^\top \boldsymbol{M}\mathbf{f}$.

The aim in least squares is therefore is to minimize $\mathbf{f}^\top \boldsymbol{M}\mathbf{f}$.

[**] Task: minimize $\mathbf{f}^\top \boldsymbol{M}\mathbf{f}$

$\boldsymbol{M} = \boldsymbol{A}^\top \boldsymbol{A}$ is a $n \times n$, positive semi-definite, symmetric matrix.
It has real and positive eigenvalues λ_i. Why?
$\boldsymbol{A}^\top \boldsymbol{A}\hat{\mathbf{e}}_i = \lambda_i \hat{\mathbf{e}}_i \quad \Rightarrow \hat{\mathbf{e}}_i^\top \boldsymbol{A}^\top \boldsymbol{A}\hat{\mathbf{e}}_i = \lambda_i \hat{\mathbf{e}}_i^\top \hat{\mathbf{e}}_i = \lambda_i \quad \Rightarrow \lambda_i = [\boldsymbol{A}\hat{\mathbf{e}}]^\top [\boldsymbol{A}\hat{\mathbf{e}}] \geqslant 0$

The **eigendecomposition** of $\boldsymbol{M}$ is

$$\boldsymbol{M} = [\hat{\mathbf{e}}_1 | \hat{\mathbf{e}}_2 | \ldots | \hat{\mathbf{e}}_n] \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & .. & \\ & & & \lambda_n \end{bmatrix} \begin{bmatrix} \hat{\mathbf{e}}_1^\top \\ \hat{\mathbf{e}}_2^\top \\ \vdots \\ \hat{\mathbf{e}}_n^\top \end{bmatrix} = \sum_i \lambda_i [\hat{\mathbf{e}}_i \hat{\mathbf{e}}_i^\top]$$

where we order the eigenvalues $0 \leqslant \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_n$.
Then, noting that $\mathbf{f}^\top \hat{\mathbf{e}}\hat{\mathbf{e}}^\top \mathbf{f} = (\mathbf{f}^\top \hat{\mathbf{e}})^2$ is just a scalar, we have

$$\mathbf{f}^\top \boldsymbol{M}\mathbf{f} = \lambda_1 (\mathbf{f}^\top \hat{\mathbf{e}}_1)^2 + \lambda_2 (\mathbf{f}^\top \hat{\mathbf{e}}_2)^2 + \ldots + \lambda_n (\mathbf{f}^\top \hat{\mathbf{e}}_n)^2$$

Conclude $\mathbf{f}^\top \boldsymbol{M}\mathbf{f}$ minimized when $\mathbf{f} = \hat{\mathbf{e}}_1$. This is the LS solution (up to scale).

[**] Least Squares, Eigendecomposition and SVD

Any $m \times n$ matrix where $m > n$ can be decomposed as $\boldsymbol{U\Sigma V}^\top \leftarrow \boldsymbol{A}$ where

$$\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times n} \begin{bmatrix} \sigma_1 & & \\ & .. & \\ & & \sigma_n \end{bmatrix}_{n\times n} \boldsymbol{V}^\top_{n\times n}$$

where $\boldsymbol{U}$ is column-orthogonal, $\boldsymbol{V}$ is fully orthogonal, and $\boldsymbol{\Sigma}$ contains the **singular values** ordered so $0 \leqslant \sigma_1 \leqslant \sigma_2 \leqslant \ldots \leqslant \sigma_n$.

Least squares achieved by setting $\mathbf{f}$ equal to the column of $\boldsymbol{V}$ corresponding to the smallest singular value σ_1.

Why? Because $\boldsymbol{M} = \boldsymbol{A}^\top\boldsymbol{A} = \boldsymbol{V\Sigma}^\top\boldsymbol{U}^\top\boldsymbol{U\Sigma V}^\top = \boldsymbol{V\Sigma}^2\boldsymbol{V}^\top$
But $\boldsymbol{V}$ is fully orthogonal and $\boldsymbol{\Sigma}^2$ is diagonal with elements σ_i^2

Now compare with Eigendecompostion ... Obvious that

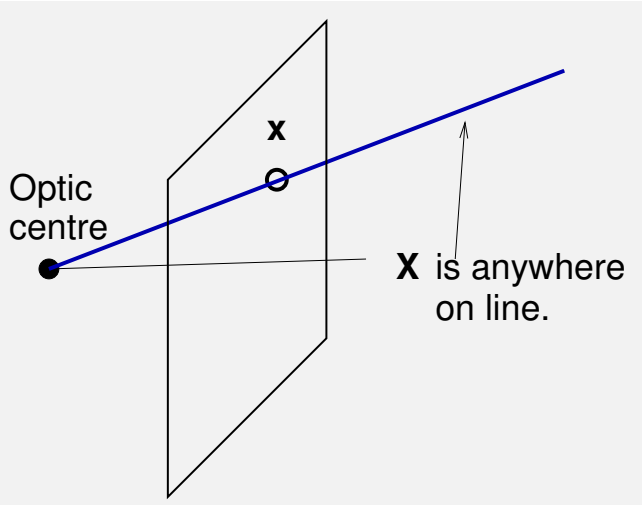
$$\boldsymbol{V} = [\hat{\mathbf{e}}_1|\hat{\mathbf{e}}_2|\ldots|\hat{\mathbf{e}}_n] \quad \text{and} \quad \lambda_i = \sigma_i^2$$

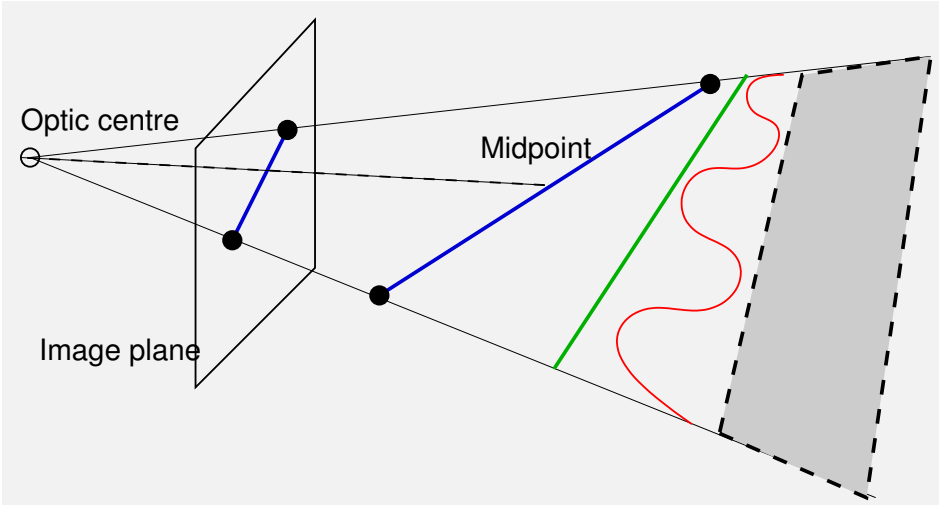
Enough said! SVD is usually preferred because it avoids the expensive computation of $\boldsymbol{A}^\top\boldsymbol{A}$

Epipolar geometry & the Fundamental matrix

Geometrical image entities **back-project** to entities of higher dimensionality in the scene.

So, points in the image back-project to **lines** in the scene, lines to planes ...





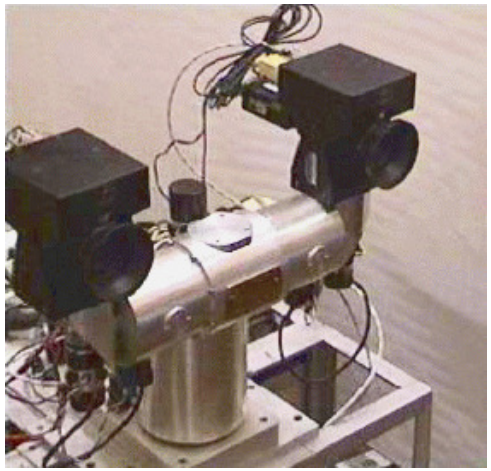
$$\mathbf{X} = \alpha\mathbf{x} \qquad 0 < \alpha < \infty$$

Single views are not sufficient to solve geometric problems in *data-driven* vision.

We need methods of understanding multiple views.

Shape-from-stereo

- different cameras
- different viewpoints
- same time



Structure-from-motion

- same camera
- different viewpoints
- different times

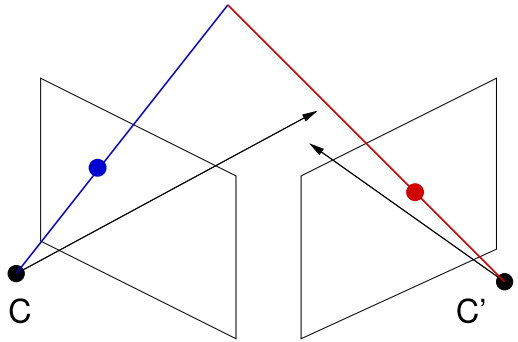
Note that single views may be sufficient to solve geometric problems in *model-driven* vision.

Two views ... straightforward in principle

Given a bit of the scene that is observable in two or more cameras, backproject the two rays to find their intersection

Right (camera $\mathcal{C}'$) View

Left (camera $\mathcal{C}$) View



Backprojection

There are three things to cover:

1. Understanding the geometry — **Epipolar geometry**
2. Find which points in the images are from the same scene location — **the correspondence problem**
3. Find the 3D structure by back-projecting rays — **triangulation**

What clues from identical parallel cameras?

Assume two cameras with the same f, separated by t_x in the x-direction. Trivial to see what happens using inhomogenous coordinates. Same scene Y, Z but different X ...

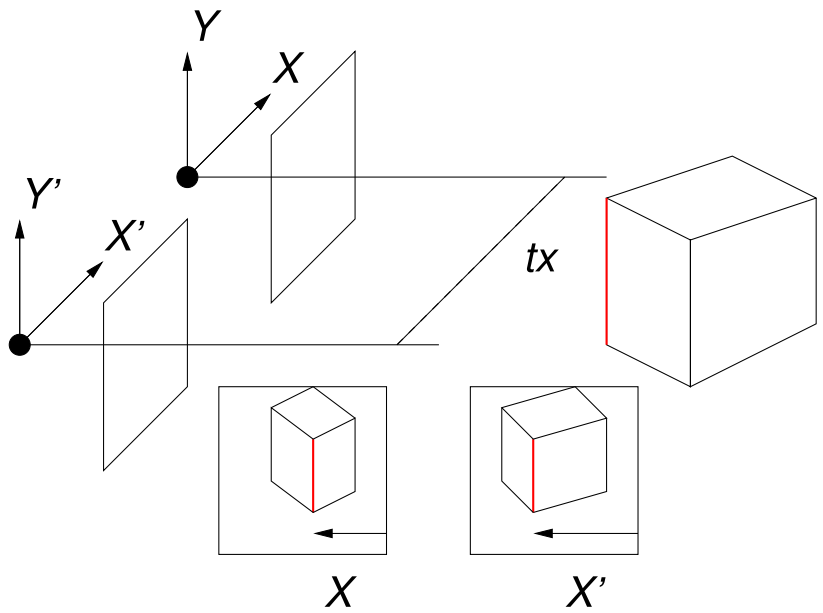
$\Rightarrow Y' = Y$, $Z' = Z$, but

$$X' = X + t_x,$$

As

$$x' = fX'/Z \quad x = fX/Z$$

$$\Rightarrow \frac{1}{Z} = \frac{1}{ft_x}(x' - x)$$



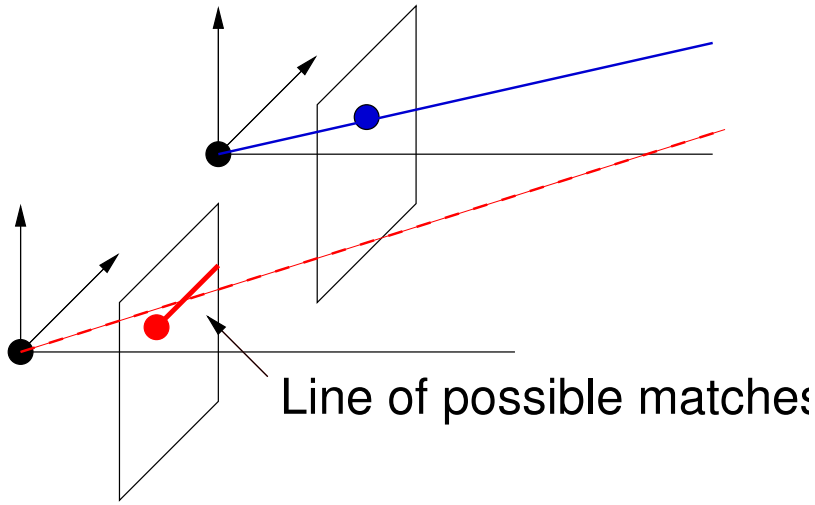
In this case:
the **reciprocal depth** $1/Z$
is proportional to the **horizontal disparity** $(x' - x)$.

What clues from identical parallel cameras? /ctd

The point **x** in the left image can come from any point **X** on the backprojected ray from the left camera $\mathcal{C}$

Could the corresponding $\mathbf{x}'$ be anywhere in $\mathcal{C}'$? No ...

$$\begin{aligned} x' &= (f/Z)(X + t_x) \\ y' &= (f/Z)(Y) \\ \Rightarrow x' &= x + ft_x/Z \\ y' &= y \end{aligned}$$



where as Z ranges from 0 to ∞

$$0 < ft_x/Z < \infty$$

So the locus of possible matches appears to be on a straight line, which would make finding a correspondence a **1D search**.

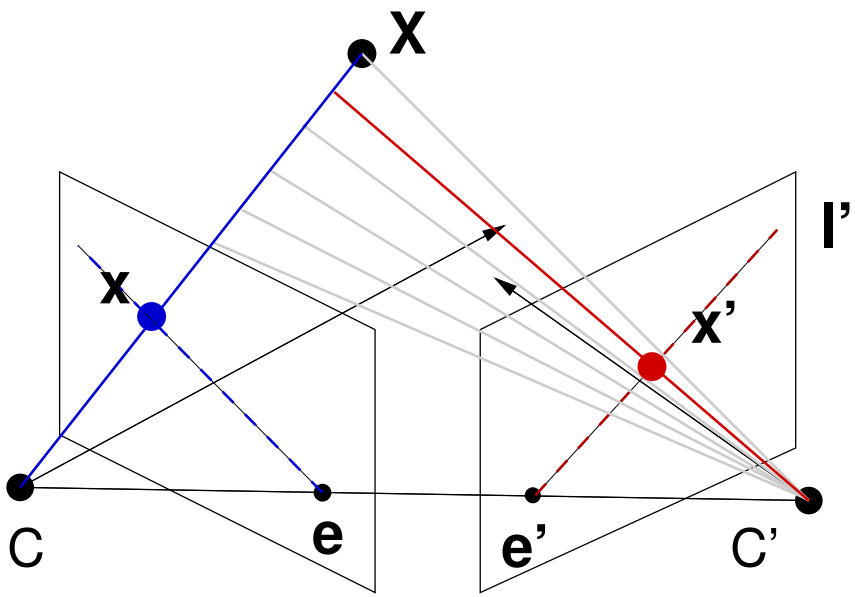
Does this generalize to arbitrary camera geometry?

Epipolar geometry in arbitrary cameras

The locus of matches is the projection into $\mathcal{C}'$ of the backprojected ray in $\mathcal{C}$.

This is **always a straight line**, and is called the **epipolar line**, labelled $\mathbf{l}'$.

As **X** moves along the ray, the other ray sweeps out the **epipolar plane**. The intersection of the epipolar plane with the image plane is the epipolar line.

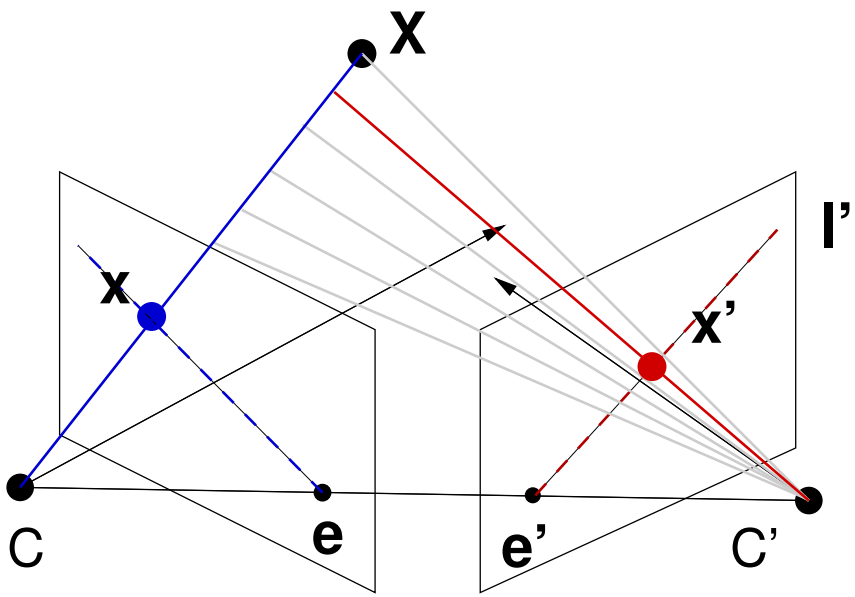


Epipolar geometry in arbitrary cameras

The locus of matches is the projection into $\mathcal{C}'$ of the backprojected ray in $\mathcal{C}$.

This is **always a straight line**, and is called the **epipolar line**, labelled $\mathbf{l}'$.

As **X** moves along the ray, the other ray sweeps out the **epipolar plane**. The intersection of the epipolar plane with the image plane is the epipolar line.

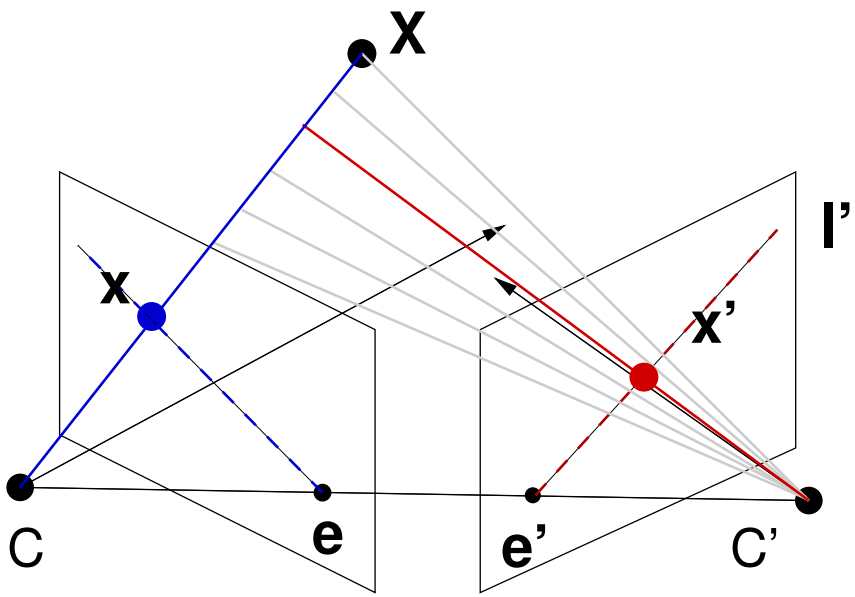


Epipolar geometry in arbitrary cameras

The locus of matches is the projection into $\mathcal{C}'$ of the backprojected ray in $\mathcal{C}$.

This is **always a straight line**, and is called the **epipolar line**, labelled $\mathbf{l}'$.

As **X** moves along the ray, the other ray sweeps out the **epipolar plane**. The intersection of the epipolar plane with the image plane is the epipolar line.

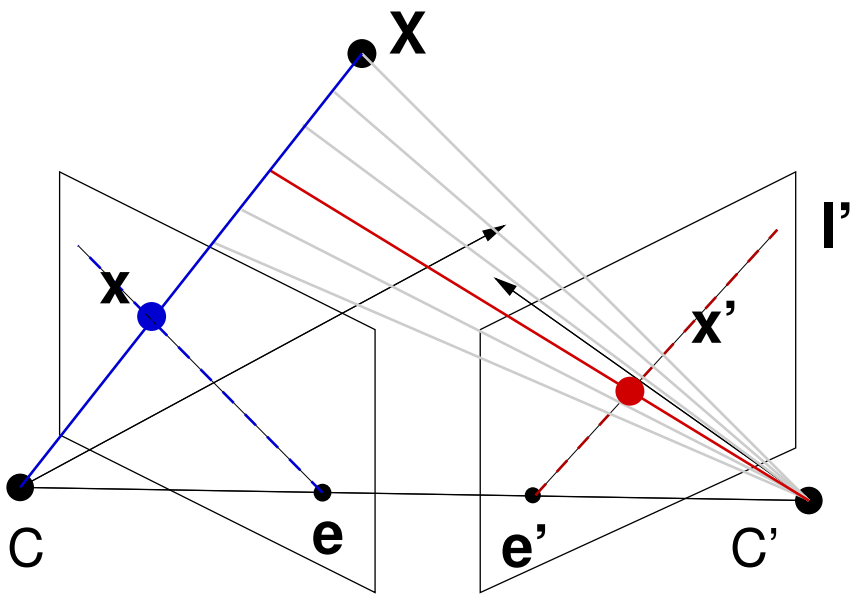


Epipolar geometry in arbitrary cameras

The locus of matches is the projection into $\mathcal{C}'$ of the backprojected ray in $\mathcal{C}$.

This is **always a straight line**, and is called the **epipolar line**, labelled $\mathbf{l}'$.

As **X** moves along the ray, the other ray sweeps out the **epipolar plane**. The intersection of the epipolar plane with the image plane is the epipolar line.

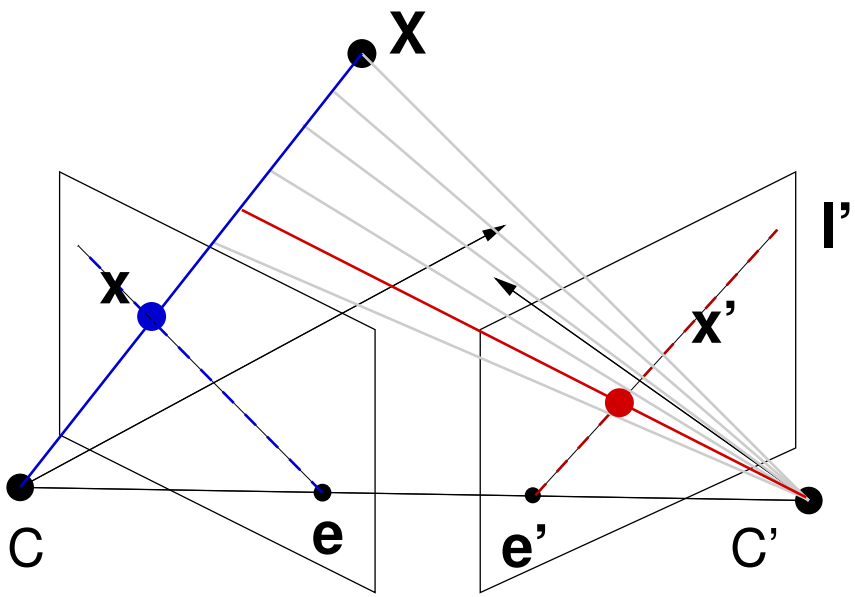


Epipolar geometry in arbitrary cameras

The locus of matches is the projection into $\mathcal{C}'$ of the backprojected ray in $\mathcal{C}$.

This is **always a straight line**, and is called the **epipolar line**, labelled $\mathbf{l}'$.

As **X** moves along the ray, the other ray sweeps out the **epipolar plane**. The intersection of the epipolar plane with the image plane is the epipolar line.

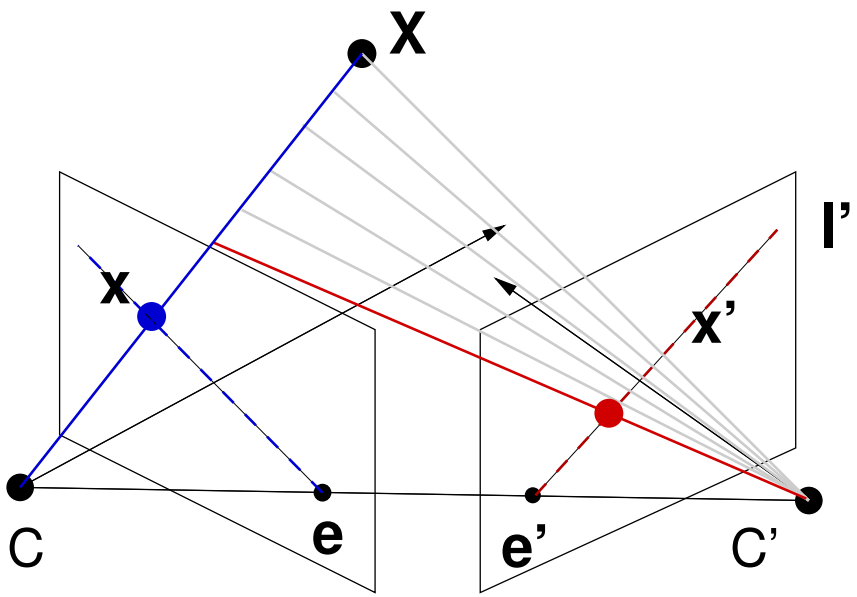


Epipolar geometry in arbitrary cameras

The locus of matches is the projection into $\mathcal{C}'$ of the backprojected ray in $\mathcal{C}$.

This is **always a straight line**, and is called the **epipolar line**, labelled $\mathbf{l}'$.

As **X** moves along the ray, the other ray sweeps out the **epipolar plane**. The intersection of the epipolar plane with the image plane is the epipolar line.



Epipolar geometry in arbitrary cameras

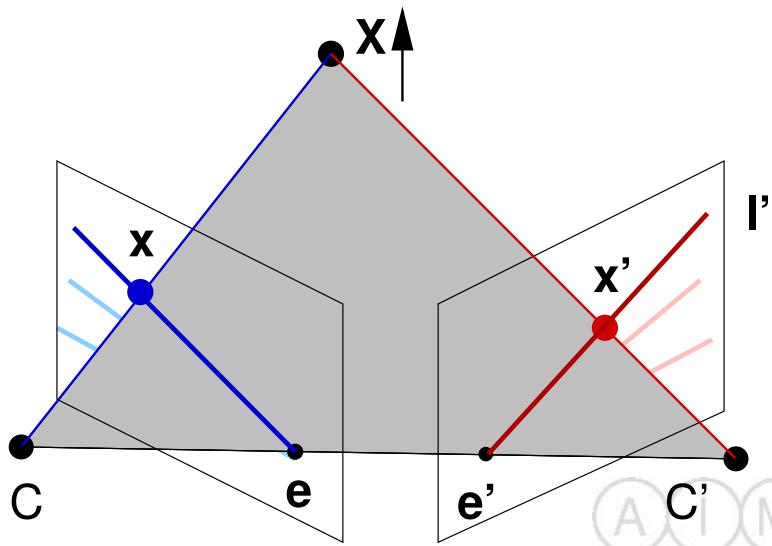
If **x** is moved then the entire plane moves too, generating a new epipolar line.

Epipolar planes hinge about the **camera baseline**, forming a pencil of planes.

This means that all the epipolar lines in camera $\mathcal{C}'$ meet at its **epipole** $\mathbf{e}'$, where the baseline pierces the image plane.

This point is the projection of the optical centre of camera $\mathcal{C}$ into $\mathcal{C}'$.

There are equivalent constructs in camera $\mathcal{C}$. All points on the same epipolar line in $\mathcal{C}$ share the same epipolar line in $\mathcal{C}'$.



Epipolar geometry in arbitrary cameras

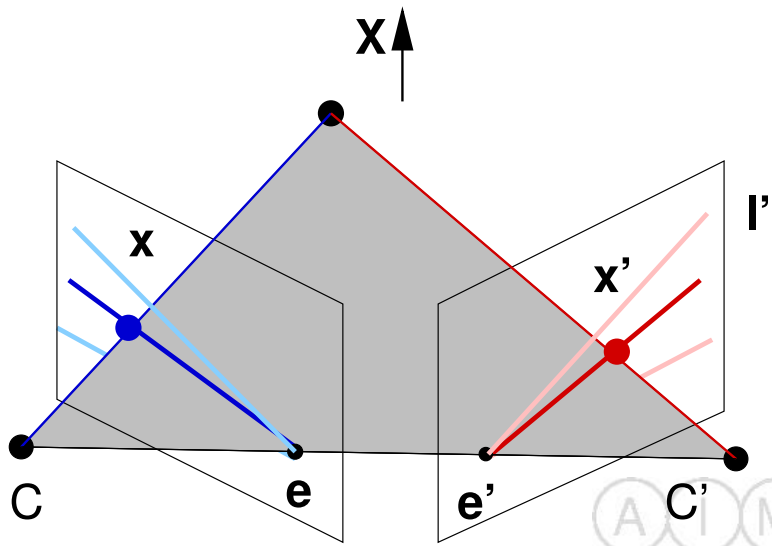
If **x** is moved then the entire plane moves too, generating a new epipolar line.

Epipolar planes hinge about the **camera baseline**, forming a pencil of planes.

This means that all the epipolar lines in camera $\mathcal{C}'$ meet at its **epipole** $\mathbf{e}'$, where the baseline pierces the image plane.

This point is the projection of the optical centre of camera $\mathcal{C}$ into $\mathcal{C}'$.

There are equivalent constructs in camera $\mathcal{C}$. All points on the same epipolar line in $\mathcal{C}$ share the same epipolar line in $\mathcal{C}'$.



Epipolar geometry in arbitrary cameras

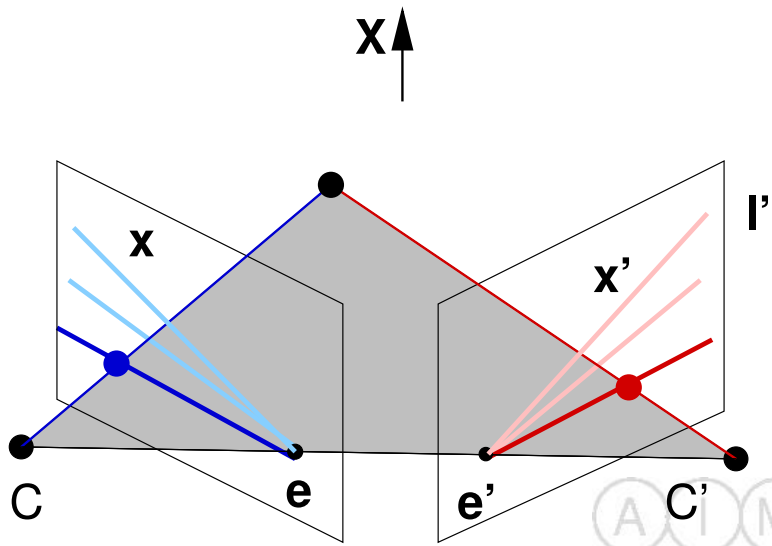
If **x** is moved then the entire plane moves too, generating a new epipolar line.

Epipolar planes hinge about the **camera baseline**, forming a pencil of planes.

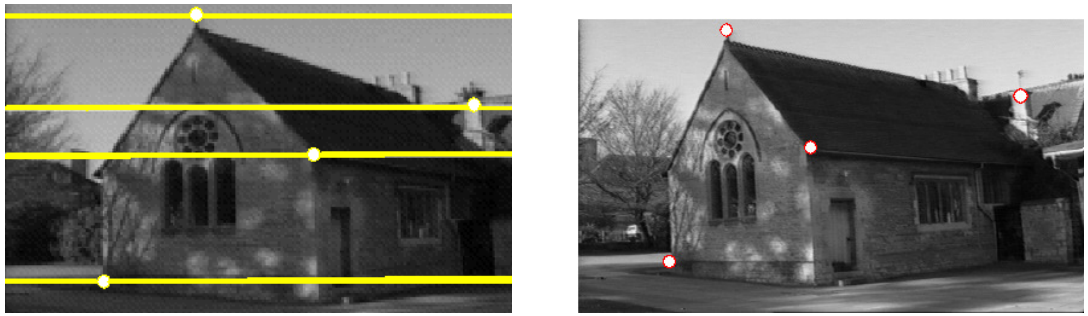
This means that all the epipolar lines in camera $\mathcal{C}'$ meet at its **epipole** $\mathbf{e}'$, where the baseline pierces the image plane.

This point is the projection of the optical centre of camera $\mathcal{C}$ into $\mathcal{C}'$.

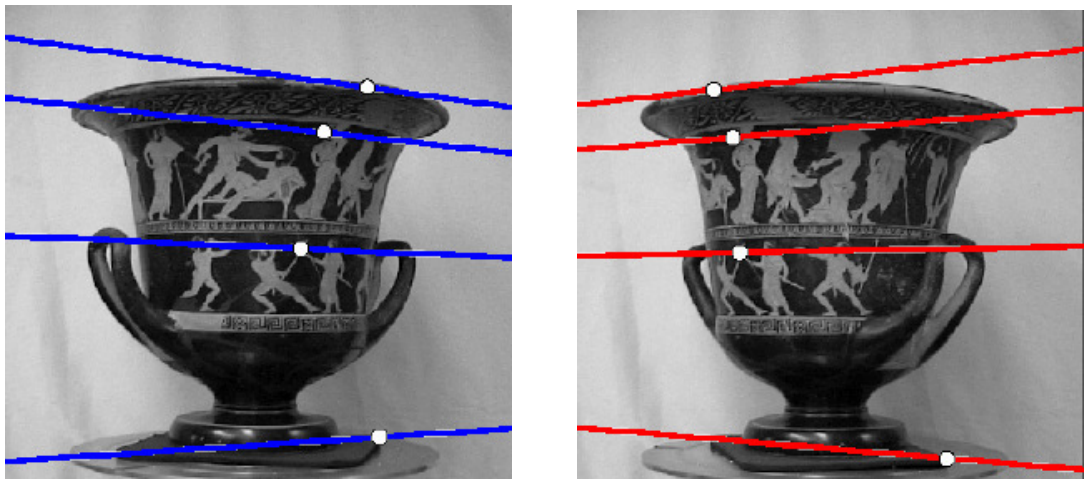
There are equivalent constructs in camera $\mathcal{C}$. All points on the same epipolar line in $\mathcal{C}$ share the same epipolar line in $\mathcal{C}'$.



Epipolar geometry examples



Parallel cameras



Converging cameras

The epipoles most often lie off the physical image planes
What's a quick test *you* could carry out to see whether the epipole was on the image plane?

Algebraic representation and the Fundamental matrix

There are several apparently different ways of deriving detailed epipolar geometry ...

... but all effectively describe how the epipolar plane pierces the image plane to give epipolar lines.

The method here is from Hartley and Zisserman

There are three bits of preamble to get across:

1. The first explains how homogeneous notation handles points at infinity.

2. The second introduces the homogeneous notation for lines.
   You may notice a duality between lines and points ...

3. The last is the matrix representation of the vector product.

Preamble 1: Points at ∞ in homogeneous coordinates

A line of points in 3D through the point **A** with direction **D** is

$$\mathbf{X}(\mu) = \mathbf{A} + \mu\mathbf{D}$$

Writing this in homogeneous notation

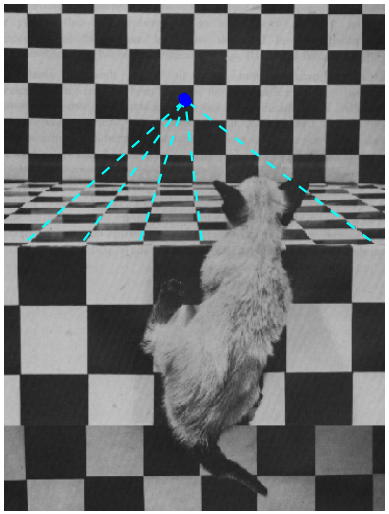
$$\begin{bmatrix} X_1(\mu) \\ X_2(\mu) \\ X_3(\mu) \\ X_4(\mu) \end{bmatrix} \overset{P}{=} \begin{bmatrix} \mathbf{A} + \mu\mathbf{D} \\ 1 \end{bmatrix} \overset{P}{=} \begin{bmatrix} \mathbf{A} \\ 1 \end{bmatrix} + \mu \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix} \overset{P}{=} \frac{1}{\mu} \begin{bmatrix} \mathbf{A} \\ 1 \end{bmatrix} + \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix}$$

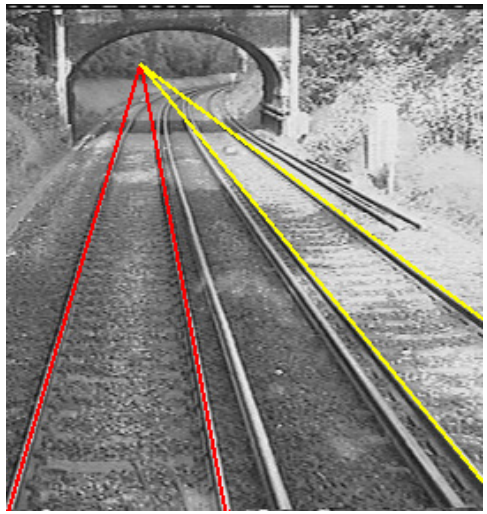
In the limit $\mu \to \infty$ the point on the line is $\begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix}$.

So, homogeneous vectors with $X_4 = 0$ represent points "at infinity".

- **Points at infinity are equivalent to directions**
- **Parallel lines in the scene meet at the same point at infinity**

Points at ∞ and vanishing points





The projection of a point at ∞ into the image is its **vanishing point**

To find it, simply project the point-at-∞ into the image ...

$$\mathbf{v} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} \mid \mathbf{t} \end{bmatrix} \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix}$$

Preamble 2: Homogeneous notation for lines

Recall that a point $(x, y)^\top$ in 2D is represented by the homogeneous 3-vector

$$\mathbf{x} \stackrel{P}{=} (x_1, x_2, x_3)^\top$$

where $x = x_1/x_3$, $y = x_2/x_3$.

Equivalently $\mathbf{x} = \lambda(x, y, 1)^\top$.

The line $l_1 x + l_2 y + l_3 = 0$ in 2D is represented by the homogeneous 3-vector

$$\mathbf{l} \stackrel{P}{=} (l_1, l_2, l_3)^\top$$

Eg: line $y = 1$ is $-y + 1 = 0$, hence $\mathbf{l} \stackrel{P}{=} (0, -1, 1)^\top$

Any point x on the line l has

$$\mathbf{l}^\top \mathbf{x} \;=\; \mathbf{x}^\top \mathbf{l} \;=\; 0$$

Preamble 2: Homogeneous notation for lines

Reminder: A point $\mathbf{x}$ on the line $\mathbf{l}$ has $\mathbf{l}^\top \mathbf{x} = 0$, $\mathbf{x}^\top \mathbf{l} = 0$, or $\mathbf{l} \cdot \mathbf{x} = 0$.

The line through two points $\mathbf{p}$ and $\mathbf{q}$ is given by

$$\mathbf{l} \overset{P}{=} \mathbf{p} \times \mathbf{q}.$$

Proof: Use the properties of the scalar triple product,

$$\mathbf{p} \cdot \mathbf{l} = \mathbf{p} \cdot (\mathbf{p} \times \mathbf{q}) \equiv 0 \quad \mathbf{q} \cdot \mathbf{l} = \mathbf{q} \cdot (\mathbf{p} \times \mathbf{q}) \equiv 0 \ .$$

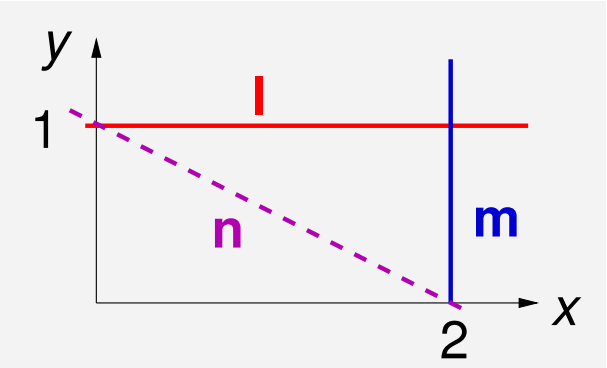
The intersection of two lines is the point

$$\mathbf{x} \overset{P}{=} \mathbf{l}_1 \times \mathbf{l}_2$$

Proof: For you to fill in ...

[**] DIY Examples ...

♣ Find (i) the point of intersection of the lines **l** and **m**; (ii) the point of intersection of the line **l** and the x-axis; and (iii) the equation of the line **n** joining $(0, 1)$ and $(2, 0)$



$\mathbf{l} = (0, -1, 1)$ and $\mathbf{m} = (-1, 0, 2)$.

$$\mathbf{x} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 0 & -1 & 1 \\ -1 & 0 & 2 \end{vmatrix} = \begin{bmatrix} -2 \\ -1 \\ -1 \end{bmatrix}$$ which is the point $(2, 1)$.

The x-axis is $(0, 1, 0)$ and so it and **l** meet at

$$\mathbf{x} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 0 & -1 & 1 \\ 0 & 1 & 0 \end{vmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

which is a 2D point at infinity at $x = \pm\infty$.

$$\mathbf{n} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 0 & 1 & 1 \\ 2 & 0 & 1 \end{vmatrix} = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}$$ which is the line $x + 2y - 2 = 0$ or $y = -(1/2)x + 1$

Preamble 3: Matrix representation of vector products

The vector product $\mathbf{a} \times \mathbf{b} =$

$$\begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [\boldsymbol{a}]_\times \mathbf{b}$$

$[\boldsymbol{a}]_\times$ is a 3×3 skew-symmetric matrix and has rank=2.

$\mathbf{a}$ is the kernel of $[\boldsymbol{a}]_\times$. (Why?)

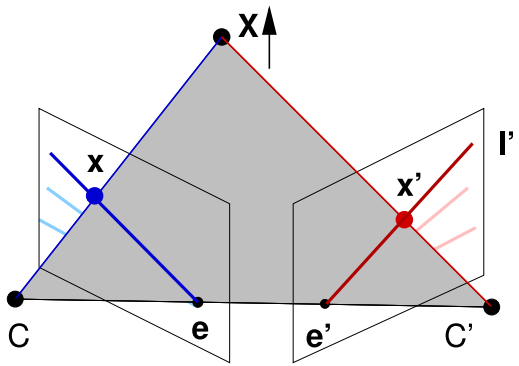
DIY Example: compute the vector product of $\mathbf{l} = (1, 2, 3)$ and $\mathbf{m} = (2, 3, 4)$.
Pseudo-determinant method gives $[., ., .,]^\top$.
Skew-sym method gives

$$\begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}$$

Algebraic representation of Epipolar Geometry

We now know that the epipolar geometry defines the

mapping from point $\mathbf{x}$ to line $\mathbf{l}'$.



The mapping depends only the cameras, not on the structure.

$\Rightarrow$ The mapping depends on the overall projection matrices $\boldsymbol{P}$ and $\boldsymbol{P}'$.

We now show that the mapping is linear, and can be written as

$\mathbf{l}' = \boldsymbol{F}\mathbf{x}$, where $\boldsymbol{F}$ is the **fundamental matrix**

Algebraic representation of Epipolar Geometry /ctd

With no loss of generality we can use the first camera $\mathcal{C}$ to define the world coordinate frame, so that its overall 3×4 projection matrix is

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{I}|\mathbf{0}]$$

We will define the rotation and translation between cameras frames as

$$\mathbf{X}' = \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}$$

So that

$$\boldsymbol{P}' = \boldsymbol{K}'[\boldsymbol{R}|\mathbf{t}]$$

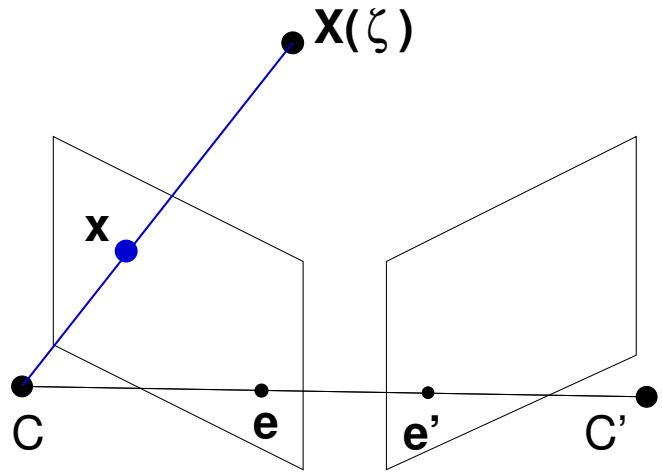
NB, the camera intrinsics $\boldsymbol{K}$ and $\boldsymbol{K}'$ can be different.

Algebraic representation of Epipolar Geometry /ctd

Step 1: back project a ray from $\mathcal{C}$
The back-projected $\mathbf{X}(\zeta)$ satisfies

$$\mathbf{x} = \boldsymbol{K}[\boldsymbol{I}|\mathbf{0}]\mathbf{X}(\zeta)$$

where we use ζ as a parameter.

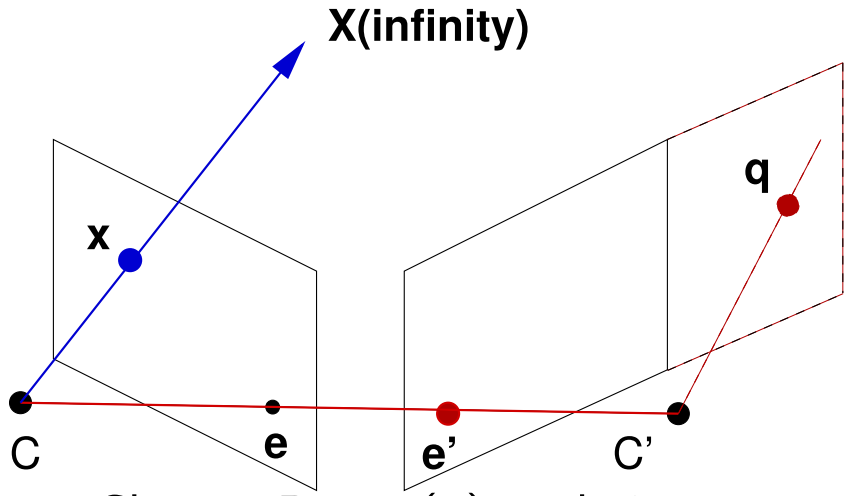


$$\Rightarrow \quad \mathbf{X}(\zeta) = \begin{bmatrix} \zeta[\boldsymbol{K}]^{-1}\mathbf{x} \\ 1 \end{bmatrix} \text{ and } \quad \Rightarrow \mathbf{X}(\infty) = \begin{bmatrix} [\boldsymbol{K}]^{-1}\mathbf{x} \\ 0 \end{bmatrix}$$

Here, $[\boldsymbol{K}]^{-1}$ corrects the direction of the ray. Direction $\begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}$ would be incorrect, because $\mathbf{x}$ was measured in a non-ideal camera.

Algebraic representation of Epipolar Geometry /ctd

Step 2: Choose two points on the ray and project the into the second camera $\mathcal{C}'$



Choose Point (1) with $\zeta = 0$

$\Rightarrow$It is the optical centre $\begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$

Choose Point (2) with $\zeta = \infty$

$\Rightarrow$A point at infinity $\begin{bmatrix} \boldsymbol{K}^{-1}\mathbf{x} \\ 0 \end{bmatrix}$

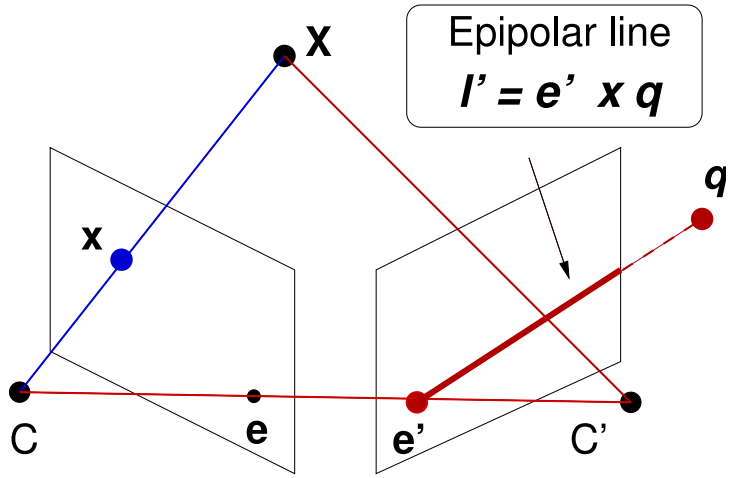
Project these two points into $\mathcal{C}'$

(1) $\mathbf{e}' = \boldsymbol{K}'[\boldsymbol{R}|\mathbf{t}]\begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} = \boldsymbol{K}'\mathbf{t}$ (2) $\mathbf{q} = \boldsymbol{K}'[\boldsymbol{R}|\mathbf{t}]\begin{bmatrix} \boldsymbol{K}^{-1}\mathbf{x} \\ 0 \end{bmatrix} = \boldsymbol{K}'\boldsymbol{R}\boldsymbol{K}^{-1}\mathbf{x}$

Note that the first point is the epipole.

Algebraic representation of Epipolar Geometry /ctd

Step 3: Use vector product to find epipolar line.



$$\mathbf{l}' = (\boldsymbol{K}'\mathbf{t}) \times \boldsymbol{K}'\boldsymbol{R}\boldsymbol{K}^{-1}\mathbf{x}$$

Now tidy up ...

(1) use $(\boldsymbol{M}\mathbf{a}) \times (\boldsymbol{M}\mathbf{b}) = \boldsymbol{M}^{-\top}(\mathbf{a} \times \mathbf{b})$, where $\boldsymbol{M}^{-\top} = [\boldsymbol{M}^{-1}]^{\top}$,
(2) using the skew symmetric matrix $[\boldsymbol{t}]_{\times}$

Hence

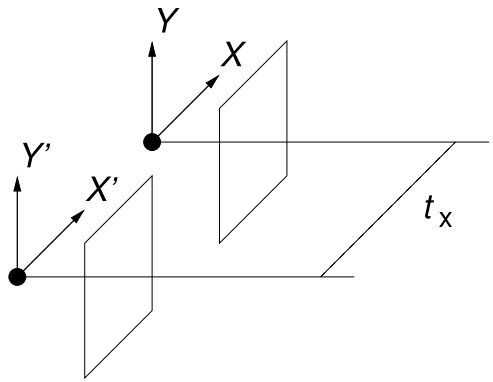
$$\mathbf{l}' = [\boldsymbol{K}']^{-\top}(\mathbf{t} \times \boldsymbol{R}\boldsymbol{K}^{-1}\mathbf{x}) = [\boldsymbol{K}']^{-\top}[\boldsymbol{t}]_{\times}\boldsymbol{R}[\boldsymbol{K}]^{-1}\mathbf{x}$$

So

Epipolar Line is:	$\mathbf{l}' = \boldsymbol{F}\mathbf{x}$
Fundamental matrix is:	$\boldsymbol{F} = [\boldsymbol{K}']^{-\top}[\boldsymbol{t}]_{\times}\boldsymbol{R}\boldsymbol{K}^{-1}$
As $\mathbf{x}'^{\top}\mathbf{l}' = 0$ ...	$\mathbf{x}'^{\top}\boldsymbol{F}\mathbf{x} = 0$

Example: identical parallel cameras

$$\boldsymbol{K} = \boldsymbol{K}' = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{R} = \boldsymbol{I}, \quad \mathbf{t} = (t_x, 0, 0)^\top$$



$$\begin{aligned} \boldsymbol{F} &= [\boldsymbol{K}']^{-\top}[\boldsymbol{t}]_\times[\boldsymbol{R}][\boldsymbol{K}]^{-1} \\ &= \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_x \\ 0 & t_x & 0 \end{bmatrix} [\boldsymbol{I}] \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \end{aligned}$$

$$\Rightarrow \quad \mathbf{x}'^\top \boldsymbol{F}\mathbf{x} = [x' \; y' \; 1] \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

which reduces to .....

Example /ctd

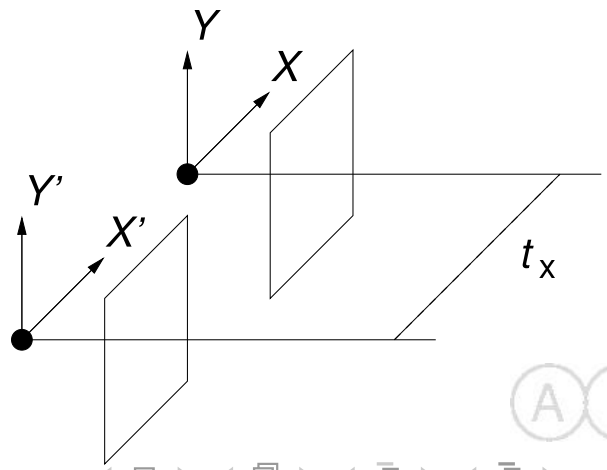
In the exercises you are asked to show that the epipole is the kernel of the fundamental matrix — that is,

$$\boldsymbol{F}\mathbf{e} = \mathbf{0}$$

$\mathbf{e}$ is in the right nullspace of $\boldsymbol{F}$
By inspection, for the parallel identical cameras example:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{0} \qquad \Rightarrow \text{Evidently } \mathbf{e} \overset{P}{=} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

What is the geometric interpretation of this result?



Summary of properties

$\boldsymbol{F}$ is a rank 2 matrix with 7 d.o.f.

If $\mathbf{x} \leftrightarrow \mathbf{x}'$ then $\mathbf{x}'^{\top} \boldsymbol{F} \mathbf{x} = 0$

The epipolar lines in $\mathcal{C}$ and $\mathcal{C}'$ are

$$\mathbf{l} = \boldsymbol{F}^{\top} \mathbf{x}' \qquad \mathbf{l}' = \boldsymbol{F} \mathbf{x}$$

The epipoles in $\mathcal{C}$ and $\mathcal{C}'$ are obtained from

$$\boldsymbol{F} \mathbf{e} = \mathbf{0} \qquad \boldsymbol{F}^{\top} \mathbf{e}' = \mathbf{0}$$

For $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{I}|\mathbf{0}]$ and $\boldsymbol{P}' = \boldsymbol{K}'[\boldsymbol{R}|\mathbf{t}]$ the fundamental matrix is derived as

$$\boldsymbol{F} = [\boldsymbol{K}']^{-\top} [\boldsymbol{t}]_{\times} [\boldsymbol{R}] [\boldsymbol{K}]^{-1}$$

where $^{-\top}$ denotes transpose of the inverse.

Computing F: Algebraic minimizations

The basis for several methods of computing $\boldsymbol{F}$ lies in re-writing the constraint $\mathbf{x}'^{\top}\boldsymbol{F}\mathbf{x} = 0$ for each match $\mathbf{x} \leftrightarrow \mathbf{x}'$ as

$$\begin{bmatrix} x'x & x'y & x' & y'x & y'y & y' & x & y & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ \vdots \\ F_{33} \end{bmatrix} = 0 \ .$$

Inserting a row for each of M matches builds the system

$$\boldsymbol{A}_{M\times 9}\mathbf{f}_{9\times 1} = \mathbf{0}_{M\times 1}$$

The properties of $\boldsymbol{A}$ and $\mathbf{f}$ are then exploited in various ways. Eg

- 8-point method
- Non-linear least squares, using a proper image cost
- Minimal 7-point method, which we review now ...

Minimal linear 7-point solution for F

With 7 entries, matrix $\boldsymbol{A}_{7\times 9}$ has rank 7 and nullity 2.

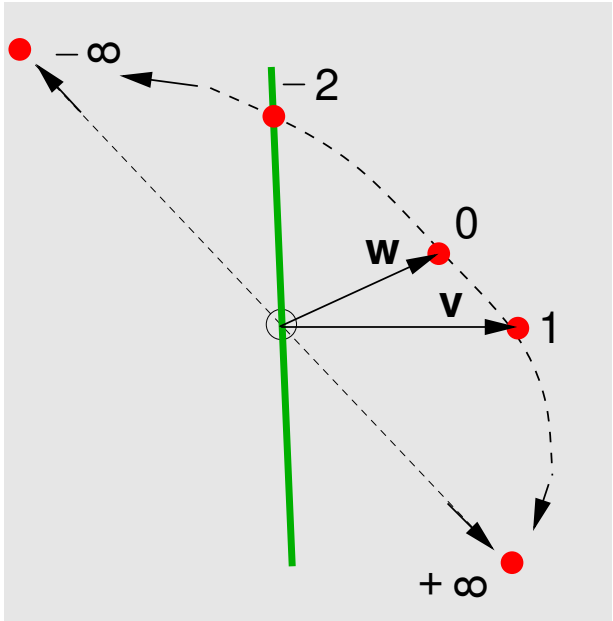
Let vectors $\mathbf{v}$ and $\mathbf{w}$ be two vectors spanning the nullspace of $\boldsymbol{A}$. Every $\mathbf{f} = (\alpha\mathbf{v} + \beta\mathbf{w})$ satisfies $\boldsymbol{A}\mathbf{f} = \mathbf{0}$, but we need to find an $\mathbf{f}$ that give $\det\boldsymbol{F} = 0$.

Surprisingly, we don't need to consider every α and β ...

For any $\mathbf{f}$ which is a solution, any scaling (+ve of -ve) of $\mathbf{f}$ is also a solution. So to explore the entire plane containing $\mathbf{v}$ and $\mathbf{w}$ we need only map out a **path** that traverses a **half plane**.

One such path is $\mathbf{f} = \alpha\mathbf{v} + (1-\alpha)\mathbf{w}$, where α runs from $-\infty$ to $+\infty$.

Eg, $\alpha = -2$ deals with all solutions on the green line.



Notice that $\alpha \to \pm\infty$ generates points $\to \pm\alpha(\mathbf{v}-\mathbf{w})$ which define the half plane division.

7-point solution for F

We want to find an α along that path that makes $\det \boldsymbol{F} = 0$.

However, we have to take care!
$\det \boldsymbol{F} = 0$ is a cubic in α

$$\det \boldsymbol{F} = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 = 0 \ .$$

So there are 3 solutions.

The algorithm goes like

- Generate the matches
- Statistically centre all sets of x and y values.
- Build $\boldsymbol{A}$ from seven of the matches.
- Use SVD to find the two vectors $\mathbf{v}$, $\mathbf{w}$ spanning the nullspace.
- Use $\mathbf{v}$, $\mathbf{w}$ to find the coeffs of the cubic
- Solve the cubic, and test which α is best.

Skeleton Matlab code

```
% xc1 and xc2 are 3 x npoint
% matrices of matched and
% centred image points
%
% Choose 7 of them
% Generate a complete A matrix
  for (i=1:7)
    A(i,1) = xc2(1,i) * xc1(1,i);
    A(i,2) = xc2(1,i) * xc1(2,i);
    A(i,3) = xc2(1,i);
    A(i,4) = xc2(2,i) * xc1(1,i);
    A(i,5) = xc2(2,i) * xc1(2,i);
    A(i,6) = xc2(2,i);
    A(i,7) = xc1(1,i);
    A(i,8) = xc1(2,i);
    A(i,9) = 1;
end
% Perform SVD to obtain
% nullspace of A
  [U,S,V] = svd(A);
  v = V(:,8);
  w = V(:,9);
```

```
% find the cubic coeffs
% from a subroutine
  [a0,a1,a2,a3] = ...
              f_cubic_coeffs(v,w);
  coeffs=[a3,a2,a1,a0];

% Solve the roots of polynomial
  alpha=roots(coeffs);

% for alpha(1), (2), (3)
% generate f 9x1 vector
  f = alpha(1)*v +(1-alpha(1))*w;
% convert into 3x3 matrix
  F1 = fvec_to_Fmat(f);

% ... etc ... for F2 and F3




% then test which is best!
```

Cubic coefficients ...

Why cubic? Writing $\mathbf{d} = \mathbf{v} - \mathbf{w}$, then $\mathbf{f} = \alpha\mathbf{d} + \mathbf{w}$, and

$$\det \boldsymbol{F} = \begin{vmatrix} \alpha d_1 + w_1 & \alpha d_2 + w_2 & \alpha d_3 + w_3 \\ \alpha d_4 + w_4 & \alpha d_5 + w_5 & \alpha d_6 + w_6 \\ \alpha d_7 + w_7 & \alpha d_8 + w_8 & \alpha d_9 + w_9 \end{vmatrix}$$

Minutes of fun later, you'll find

$$a_0 = +w_1 w_5 w_9 + w_2 w_6 w_7 + w_3 w_4 w_8 - w_1 w_6 w_8 - w_2 w_4 w_9 - w_3 w_5 w_7$$

$$\begin{aligned} a_1 = {} & d_1 w_5 w_9 + w_1 d_5 w_9 + w_1 w_5 d_9 + d_2 w_6 w_7 + w_2 d_6 w_7 + w_2 w_6 d_7 + d_3 w_4 w_8 + w_3 d_4 w_8 + w_3 w_4 d_8 \\ & - d_1 w_6 w_8 - w_1 d_6 w_8 - w_1 w_6 d_8 - d_2 w_4 w_9 - w_2 d_4 w_9 - w_2 w_4 d_9 - d_3 w_5 w_7 - w_3 d_5 w_7 - w_3 w_5 d_7 \end{aligned}$$

$$\begin{aligned} a_2 = {} & +d_1 d_5 w_9 + d_1 w_5 d_9 + w_1 d_5 d_9 + d_2 d_6 w_7 + d_2 w_6 d_7 + w_2 d_6 d_7 + d_3 d_4 w_8 + d_3 w_4 d_8 + w_3 d_4 d_8 \\ & - d_1 d_6 w_8 - d_1 w_6 d_8 - w_1 d_6 d_8 - d_2 d_4 w_9 - d_2 w_4 d_9 - w_2 d_4 d_9 - d_3 d_5 w_7 - d_3 w_5 d_7 - w_3 d_5 d_7 \end{aligned}$$

$$a_3 = +d_1 d_5 d_9 + d_2 d_6 d_7 + d_3 d_4 d_8 - d_1 d_6 d_8 - d_2 d_4 d_9 - d_3 d_5 d_7$$

Tedious, but easy enough to write a matlab routine

```
function [a0,a1,a2,a3] = f_cubic_coeffs(v,w)
d= v-w;
a0 =   w(1)*w(5)*w(9) + w(2)*w(6)*w(7) + w(3)*w(4)*w(8) ...
     - w(1)*w(6)*w(8) - w(2)*w(4)*w(9) - w(3)*w(5)*w(7);
a1 = ... blah blah blah ...
```

Centering the data

For numerical stability it is essential that the elements of $\boldsymbol{A}$ are not of markedly different orders of magnitude.

The standard method of "statistical centering" an image quantity x involves shifting and scaling the data so that the centred data have $\mu_{x\boldsymbol{c}} = 0$ and standard deviation of $\sigma_{x\boldsymbol{c}} = \sqrt{2}$.

Treating x and y as independent, a centered point is

$$\mathbf{x}^{\boldsymbol{C}} = \boldsymbol{C}\mathbf{x} = \begin{bmatrix} \sqrt{2}/\sigma_x & 0 & -\sqrt{2}\mu_x/\sigma_x \\ 0 & \sqrt{2}/\sigma_y & -\sqrt{2}\mu_y/\sigma_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

and similarly for points $\mathbf{x}'$ in the other image.

Using centered points in $\boldsymbol{A}$, the required matrix is obtained from the $\boldsymbol{F}^{\boldsymbol{C}}$ recovered via SVD as

$$\boldsymbol{F} = [\boldsymbol{C}']^{\top} \boldsymbol{F}^{\boldsymbol{C}} [\boldsymbol{C}]$$

You are just undoing the linear transformation.

Using more points: 8, then many

We require $\det \boldsymbol{F} = 0$. So, if (for example) you have defined $F_{11} \ldots F_{32}$, then F_{33} is known. In turn the last column of $\boldsymbol{A}$ is not independent. So, for perfect data, the maximum rank of $\boldsymbol{A}$ is 8.

Hence, with 8 perfect matches, $\mathbf{f}$ is the kernel of $\boldsymbol{A}$.

For imperfect data, and more than 8 matches, our aim should be to calculate the $\boldsymbol{F}$ that minimizes the algebraic error

$$\min_{\mathbf{f}} \|\boldsymbol{A}\mathbf{f}\|, \quad \text{subject to} \quad \|\mathbf{f}\| = 1$$

Both of these can be achieved using SVD

$$\boldsymbol{U}[\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_9)]\boldsymbol{V}^\top \leftarrow \boldsymbol{A}$$

$\mathbf{f}$ is the column of $\boldsymbol{V}$ associated with the smallest singular value.

Most algorithms sort singular values high to low, so this will be

$$\mathbf{f} = \mathbf{v}_9$$

Computing the Fundamental Matrix 8 points

However, in neither case will $\boldsymbol{F}$ reconstructed from $\mathbf{f}$ be of rank 2 — that is, $\det \boldsymbol{F}$ will not be zero.

One finds the closest approximation (in the Frobenius sense) of rank 2 using a further SVD.

We want to find $\min_{\boldsymbol{F}_2} \|\boldsymbol{F} - \boldsymbol{F}_2\|\text{Frob}$ subject to $\text{Rank}\boldsymbol{F}_2 = 2$.

The SVD is

$$\boldsymbol{U}[\text{diag}(\sigma_1, \sigma_2, \sigma_3)]\boldsymbol{V}^\top \leftarrow \boldsymbol{F}$$

and the required matrix is

$$\boldsymbol{F}_2 = \boldsymbol{U}[\text{diag}(\sigma_1, \sigma_2, 0)]\boldsymbol{V}^\top$$

You simply switch the smallest singular value to zero.

Optimizing F by minimizing image error

Linear methods minimize an "algebraic error" which is not readily related to the physical error in the image. Ignoring gross mismatches, this is random error in the measured location of features $\mathbf{x}$ and $\mathbf{x}'$.

More correctly, one should allow optimization of a parameter vector containing elements of the fundamental matrix and the feature positions, subject to the contraints

$$min_{\mathbf{p}} \sum_{m=1}^{M} \left\{ d(\mathbf{x}_m, \tilde{\mathbf{x}}_m)^2 + d(\mathbf{x}'_m, \tilde{\mathbf{x}}'_m)^2 \right\}$$

subject to all $[\tilde{\mathbf{x}}'_m]^\top \boldsymbol{F} \tilde{\mathbf{x}}_m = 0$ and $\|\boldsymbol{F}\|_{\text{Frob}} = 1$ and $\det \boldsymbol{F} = 0$. The parameter vector is length $2m + 9$

$$\mathbf{p} = \{F_{11}, \ldots, F_{33}, \tilde{x}_1, \tilde{y}_1, \ldots, \tilde{x}_m, \tilde{y}_m, \tilde{x}'_1, \tilde{y}'_1, \ldots, \tilde{x}'_m, \tilde{y}'_m\}$$

The Optimal F matrix

There is a standard method [**] to find

$$\min_{\mathbf{p}} \frac{1}{2}\mathbf{d}(\mathbf{p})^\top \boldsymbol{W}\mathbf{d}(\mathbf{p}) \quad \text{subject to} \quad \mathbf{c}(\mathbf{p}) = \mathbf{0}$$

and we can use this by setting

$$\mathbf{d}(\mathbf{p}) = \begin{bmatrix} \mathbf{x}_1 - \tilde{\mathbf{x}}_1 \\ \vdots \\ \mathbf{x}_M - \tilde{\mathbf{x}}_M \\ \mathbf{x}'_1 - \tilde{\mathbf{x}}'_1 \\ \vdots \\ \mathbf{x}'_M - \tilde{\mathbf{x}}'_M \end{bmatrix} \quad \text{and} \quad \mathbf{c}(\mathbf{p}) = \begin{bmatrix} [\tilde{\mathbf{x}}'_1]^\top \boldsymbol{F}\tilde{\mathbf{x}}'_1 \\ \vdots \\ [\tilde{\mathbf{x}}'_M]^\top \boldsymbol{F}\tilde{\mathbf{x}}'_M \\ \|\boldsymbol{F}\|_{\text{Frob}} - 1 \\ \det \boldsymbol{F} \end{bmatrix}$$

A note on the Essential Matrix

The fundamental matrix $\boldsymbol{F} = [\boldsymbol{K}']^{-\top}[\boldsymbol{t}]_\times \boldsymbol{R}\boldsymbol{K}^{-1}$ requires $\mathbf{x}'^{\top}\boldsymbol{F}\mathbf{x} = 0$

If however we know the intrinsic calibrations, we can transform the matching points into their respective ideal images.

Points in the ideal images are related by

$$\mathbf{x}'^{\top}\boldsymbol{E}\mathbf{x} = 0$$

where the Essential Matrix

$$\boldsymbol{E} = [\boldsymbol{t}]_\times \boldsymbol{R}$$

Properties of the essential matrix were derived (in a variety of forms) in the early 1980's. The broad sweep of methods to be described for computing $\boldsymbol{F}$ have counterparts for $\boldsymbol{E}$, although opportunities (and challenges) arise from its reduced number of degrees of freedom and special form.

3.5 The correspondence problem

We've seen that because the projection into $\mathcal{C}'$ of the backprojected ray from a point in $\mathcal{C}$ is a straight line, the search for potential matches can be carried out along straight epipolar lines

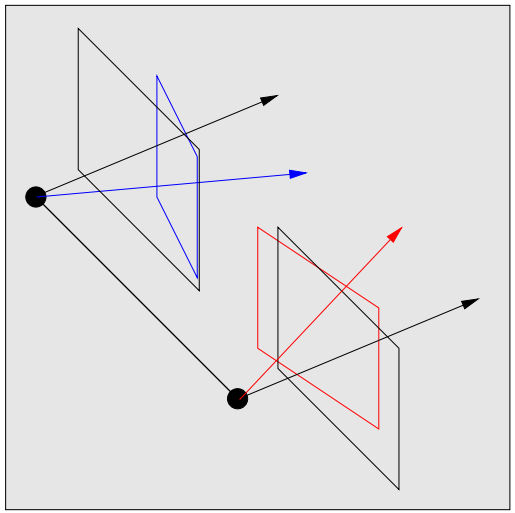
1D search must be less demanding than 2D search throughout the image

In some cases it makes sense first to **rectify** the images from a stereo rig so that parallel camera geometry applies

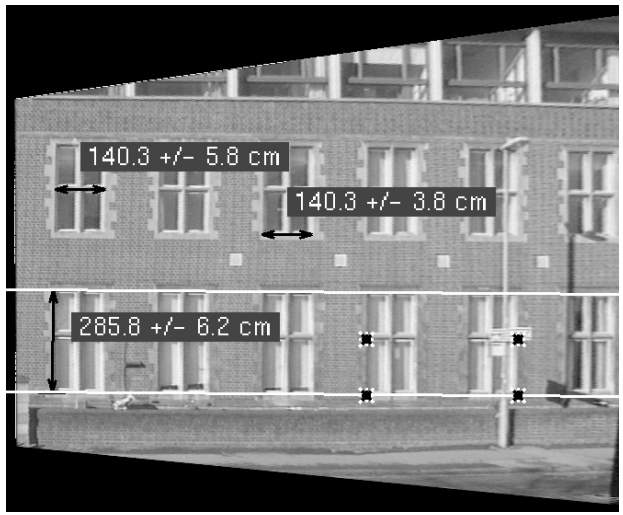
Search is then along images rows (or rasters) which is particularly cheap.

Adjusting epipolar geometry: Rectification

New optical axes are chosen to be coplanar and perpendicular to the base line, and the new image planes set with the same intrinsic matrix $\boldsymbol{K}_{rect}$



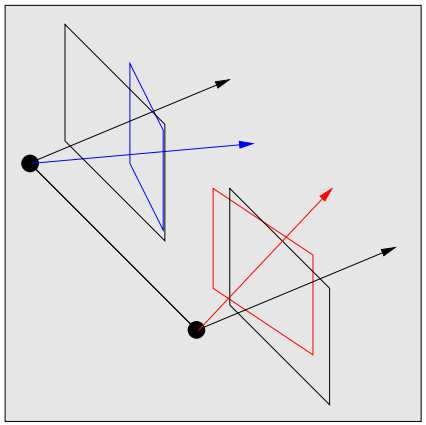
Actual Left image



Rectified

Adjusting epipolar geometry: Rectification

For each camera the image transformation required turns out to be a plane-plane projectivity, or **homography**, $\boldsymbol{H}_{3\times 3}$.



The scene point in the rotated camera frame is related to that in the actual camera by:

$$\mathbf{X}^{rect} = \begin{bmatrix} \boldsymbol{R} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}^{C} \quad \Rightarrow \quad \mathbf{X}^{C} = \begin{bmatrix} \boldsymbol{R}^{-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}^{rect}$$

The actual measured projection is

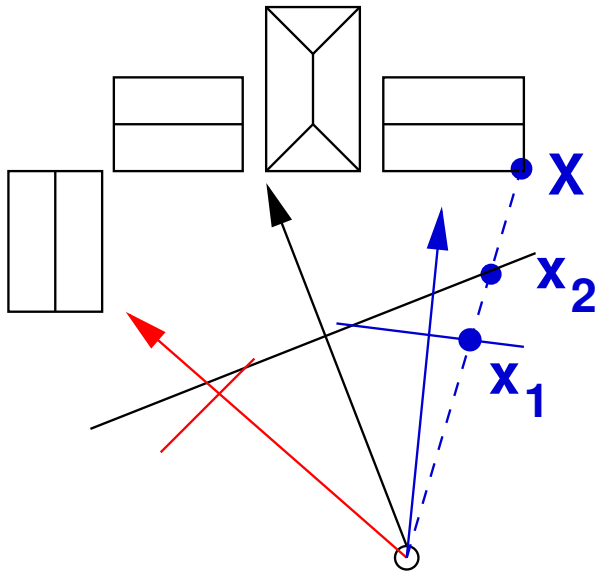
$$\mathbf{x} \overset{P}{=} \boldsymbol{K}\,[\boldsymbol{I}|\mathbf{0}] \begin{bmatrix} \boldsymbol{R}^{-1} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}^{rect} = \boldsymbol{K}\boldsymbol{R}^{-1}\mathbf{X}_3^{rect} \quad \Rightarrow \quad \mathbf{X}_3^{rect} = \boldsymbol{R}\boldsymbol{K}^{-1}\mathbf{x}$$

So, the rectified image point is

$$\mathbf{x}^{rect} \overset{P}{=} \boldsymbol{K}_{rect}\mathbf{X}_3^{rect} = [\boldsymbol{K}_{rect}\boldsymbol{R}\boldsymbol{K}^{-1}]\mathbf{x} = \boldsymbol{H}_{3\times 3}\mathbf{x}$$

An aside: Homographies and Mosaicing from rotated images

An image is the intersection of a plane with the cone of rays between points in 3-space and the optical centre. Any two such "images" (with the same optical centre) are related by a planar projective transformation.



As the camera is rotated the points of intersection of the rays with the image plane are related by a planar projective transformation.

Image points $\mathbf{x}_1$ and $\mathbf{x}_2$ correspond to the same scene point $\mathbf{X}$.

Homographies and Mosaicing

For corresponding points $\mathbf{x}_1$ and $\mathbf{x}_2$ in two views 1 and 2,

$$\mathbf{x}_1 \overset{P}{=} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_1 & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \boldsymbol{R}_1 \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad \mathbf{x}_2 \overset{P}{=} \boldsymbol{R}_2 \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Hence

$$\mathbf{x}_2 = \boldsymbol{R}_2 \boldsymbol{R}_1^{-1} \mathbf{x}_1$$

The cameras could have different focal lengths — so one can do all of this while rotating *and* zooming. Then

$$\mathbf{x}_1 \overset{P}{=} \boldsymbol{K}_1 \boldsymbol{R}_1 \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad \mathbf{x}_2 \overset{P}{=} \boldsymbol{K}_2 \boldsymbol{R}_2 \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad \Rightarrow \mathbf{x}_2 = \boldsymbol{K}_2 \boldsymbol{R}_2 \boldsymbol{R}_1^{-1} \boldsymbol{K}_1^{-1} \mathbf{x}_1$$

where in the simplest case

$$\boldsymbol{K}_1 = \begin{bmatrix} f_1 & 0 & 0 \\ 0 & f_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Computing an homography

The equations,

$$\begin{aligned} x'(h_{31}x + h_{32}y + h_{33}) &= h_{11}x + h_{12}y + h_{13} \\ y'(h_{31}x + h_{32}y + h_{33}) &= h_{21}x + h_{22}y + h_{23} \end{aligned}$$

can be rearranged as

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y & -y' \end{bmatrix} \mathbf{h} = \mathbf{0}$$

where $\mathbf{h} = (h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{33})^\top$ is the matrix $\boldsymbol{H}$ written as a 9-vector.

continued ...

For 4 points,

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2'x_2 & -x_2'y_2 & -x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y_2'x_2 & -y_2'y_2 & -y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3'x_3 & -x_3'y_3 & -x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y_3'x_3 & -y_3'y_3 & -y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4'x_4 & -x_4'y_4 & -x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y_4'x_4 & -y_4'y_4 & -y_4' \end{bmatrix} \mathbf{h} = \mathbf{0}$$

which has the form $\boldsymbol{A}\mathbf{h} = \mathbf{0}$, with $\boldsymbol{A}$ a 8×9 matrix. The solution $\mathbf{h}$ is the (one dimensional) null space of $\boldsymbol{A}$.
If using many points, one can use least squares. Solution best found then using SVD of $\boldsymbol{A}$ — ie $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top \leftarrow \boldsymbol{A}$ Then $\mathbf{h}$ is the column of $\boldsymbol{V}$ corresponding to smallest singular value. (The smallest singular value would be zero of all the data were exact ...)

But really, use RANSAC ...

Some Matlab

```
npoints = 4  (or  5 later -- 5th point is noisy)
x  = [0,1,0,1, 1.01];   y  = [0,0,1,1, 0.99];
xd = [0,1,0,2, 2.01];   yd = [0,0,1,1, 1.01];
A = zeros(2*npoints,9);
for i=1:npoints,
 A(2*i-1,:)= [x(i),y(i),1,0,0,0,  -x(i)*xd(i),-xd(i)*y(i),-xd(i)];
 A(2*i,  :)= [0,0,0,x(i),y(i),1,  -x(i)*yd(i),-yd(i)*y(i),-yd(i)];
end;
if npoints==4
  h = null(A);
else
  [U,S,V] = svd(A);
  h=V(:,9);
end;
H=[h(1),h(2),h(3);h(4),h(5),h(6);h(7),h(8),h(9);]
```

With the 4 exact points ...

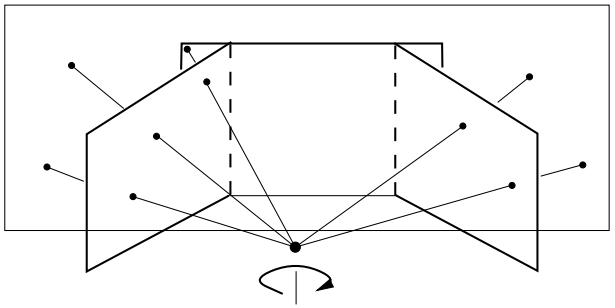
$$\boldsymbol{H} = \begin{bmatrix} 0.6325 & -0.0000 & 0.0000 \\ 0.0000 & 0.3162 & -0.0000 \\ 0.0000 & -0.3162 & 0.6325 \end{bmatrix}$$

Adding a noisy point ...

$$\boldsymbol{H} = \begin{bmatrix} 0.6295 & -0.0000 & -0.0000 \\ -0.0001 & 0.3188 & 0.0001 \\ -0.0050 & -0.3155 & 0.6344 \end{bmatrix}$$

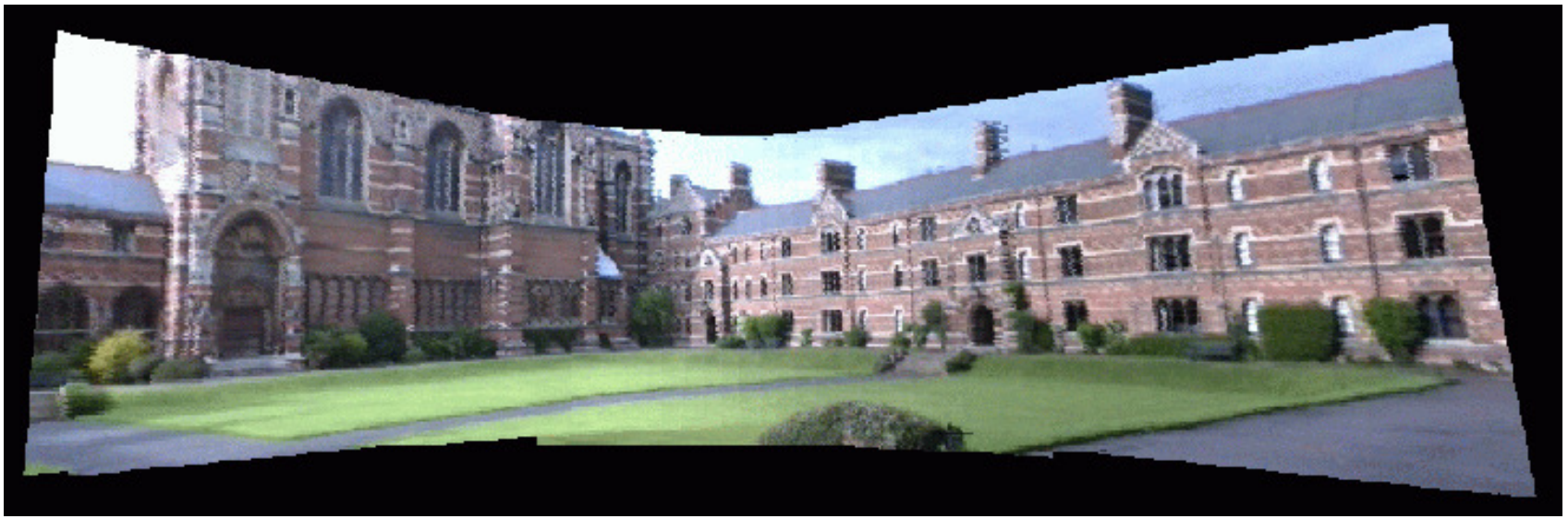
Homographies, Mosaics

- Input sequence: 30 frames, hand held camcorder



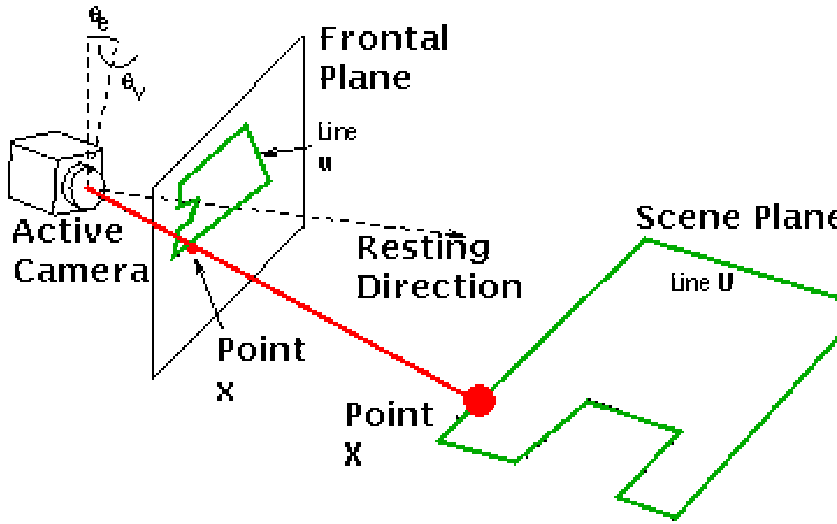
Register all the images to one reference image by projective transformations.

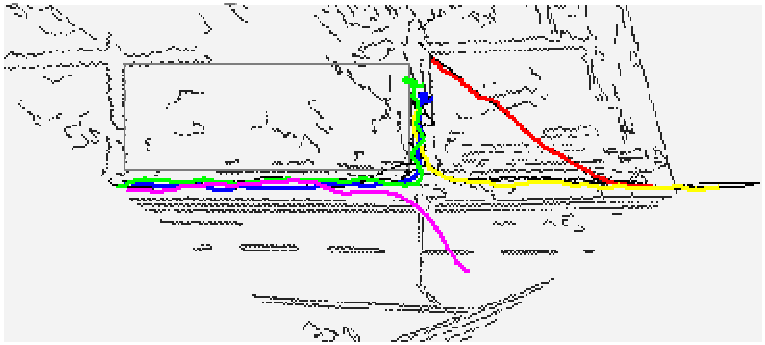
Homographies, Mosaics, Keble



Homographies: a modern Dürer

Here the rotation joint of a pan-tilt camera become the projection centre, and tracking people in the ground plane produces a track on a notional frontal plane.
These frontal plane tracks are then converted in Cartesian tracks viewed from above.





Whether rectified or not ...

The task now is to determine pairs of points in the images from the same scene location

This **the correspondence problem** can be solved

- Sparsely: matches are sought between individual corner or edge features; or
- Densely: matches are sought for all the pixels in an image.

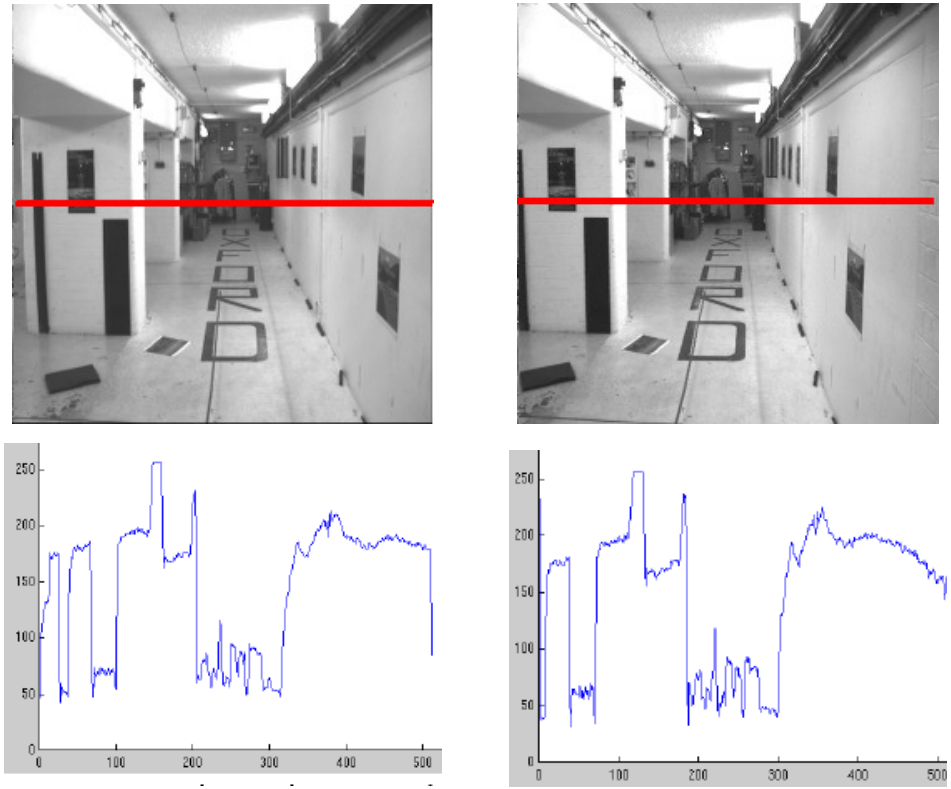
We'll explore a dense correlation-based approach suitable for short inter-camera baselines.

Will also consider a dynamic programming approach to incorporate further constraints on matching

Correlation-based

The basic assumption in "short baseline" correspondence algorithms is that the image bands around the two epipolar lines generated by an epipolar plane are similar.

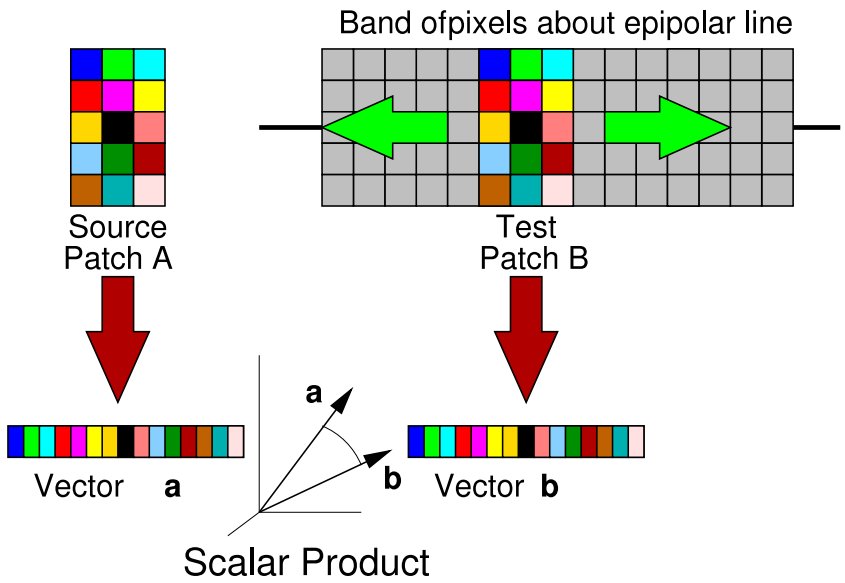
The example, for || cameras, shows that this is broadly the case, though there are regions of noise, ambiguity, and occlusion.



Zero-Normalized cross-correlation

Images are from different cameras — may be global gains and offsets
Model by $I' = \alpha I + \beta$ and use zero-normalized cross-correlation.

Step 1. Set source patch A around $\mathbf{x}$



Step 2. Subtract the patch mean $A_{ij} \leftarrow A_{ij} - \mu_A$

Step 3. For each $\mathbf{x}'$ on the epipolar line

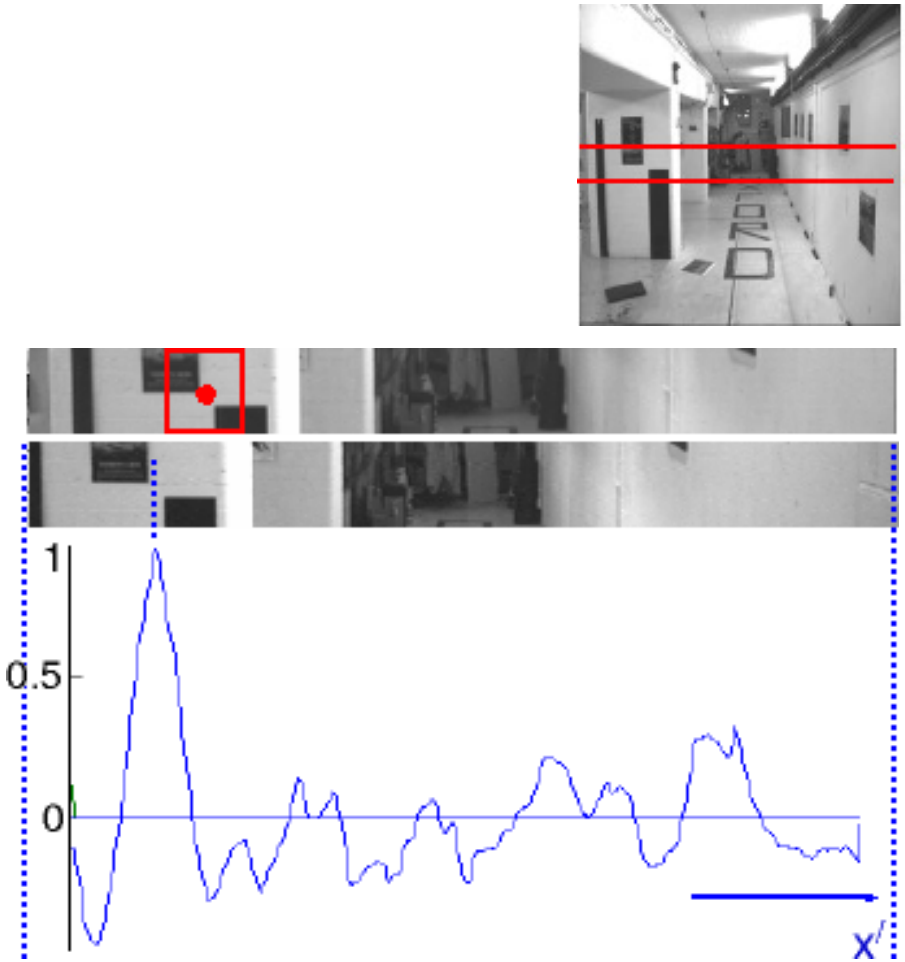
3a. Generate patch B, and $B_{ij} \leftarrow B_{ij} - \mu_B$.

3b. Compute

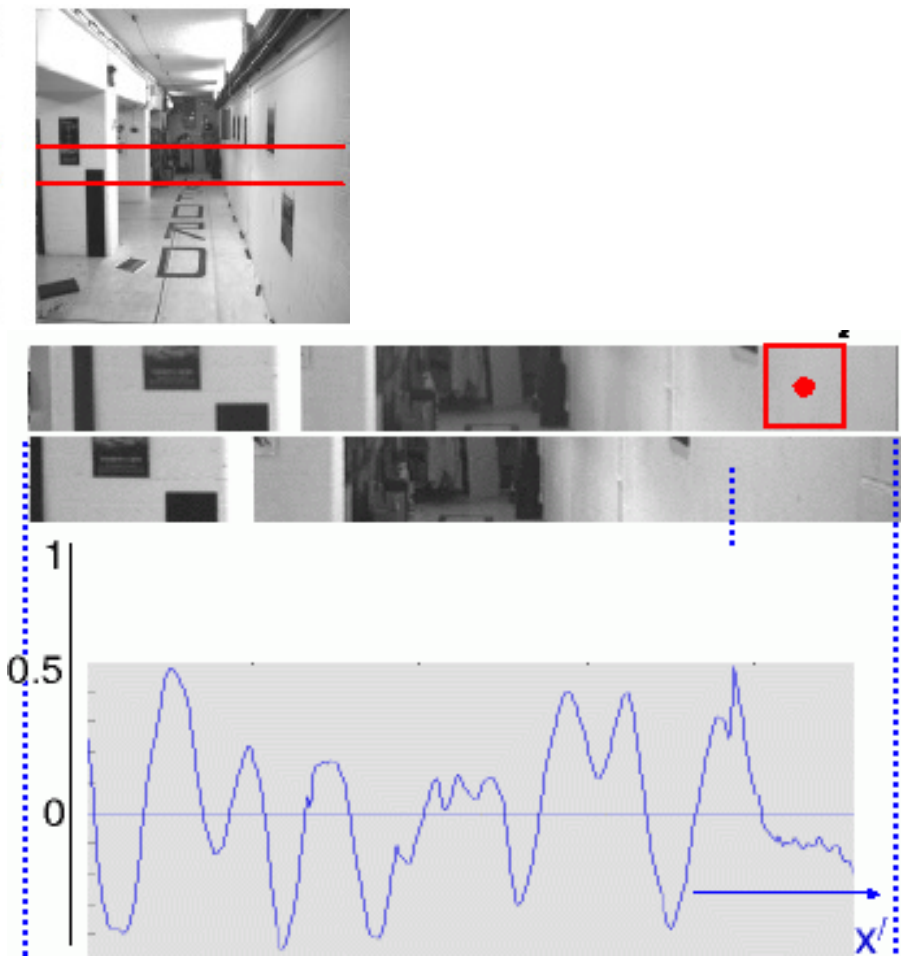
$$NCC(\mathbf{x}, \mathbf{x}') = \frac{\sum_i \sum_j A_{ij} B_{ij}}{\sqrt{\sum_i \sum_j A_{ij}^2} \sqrt{\sum_i \sum_j B_{ij}^2}}$$

Useful analogy: Imagine the regions as vectors $A \rightarrow \mathbf{a}$ $B \rightarrow \mathbf{b}$.
$NCC \equiv \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$, a scalar product of UNIT vectors. So $-1 \leqslant NCC \leqslant 1$

Examples



Example 1: Strong correlation

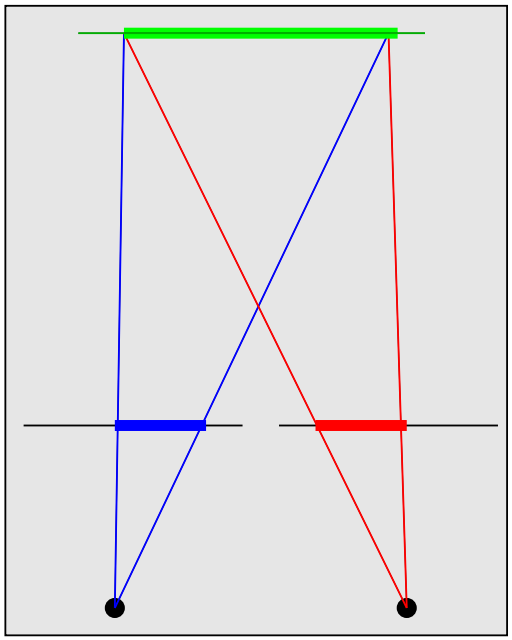


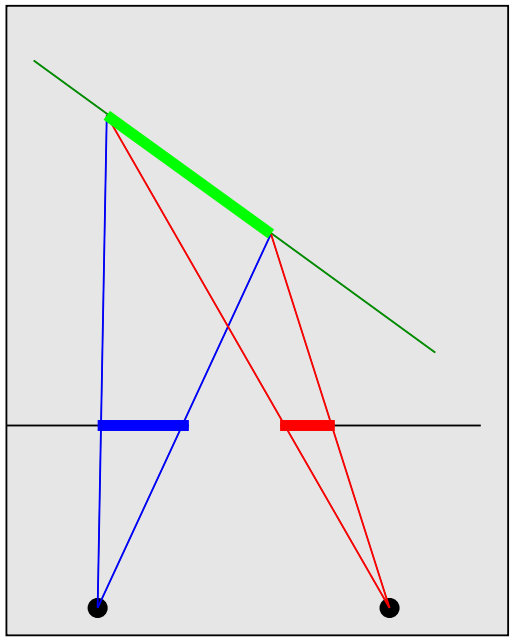
Example 2: Weak correlation

Examples /ctd

Why is cross-correlation a poor measure in Example 2?

1. The neighbourhood region does not have a distinctive spatial intensity. Weak auto-correlation ⇒ambiguous cross-correlation
2. Foreshortening effects — perspective distortion.
   Using Correlation assumes minimal appearance change
   which favours fronto-parallel surfaces.





Outline of a dense correspondence algorithm

Algorithm in a box

1. Rectify images in $\mathcal{C}$, $\mathcal{C}'$
2. For each pixel $\mathbf{x}$ in image $\mathcal{C}$:
   - 2.1 Compute NCC for each pixel $\mathbf{x}'$ along epipolar line $\mathbf{l}'$ in image $\mathcal{C}'$
   - 2.1 Choose the match $\mathbf{x}'$ with the highest NCC

Parameters to adjust

- size of neighbourhood patches
- limit on maximum disparity $(\mathbf{x}' - \mathbf{x})$

Constraints to apply

- uniqueness of match
- match ordering
- smoothness of disparity field (disparity gradient limit)
- figural continuity.

Limitations

- scene must be textured, and
- largely fronto-parallel (related to d.g. limit)

Constraints to apply on correspondence

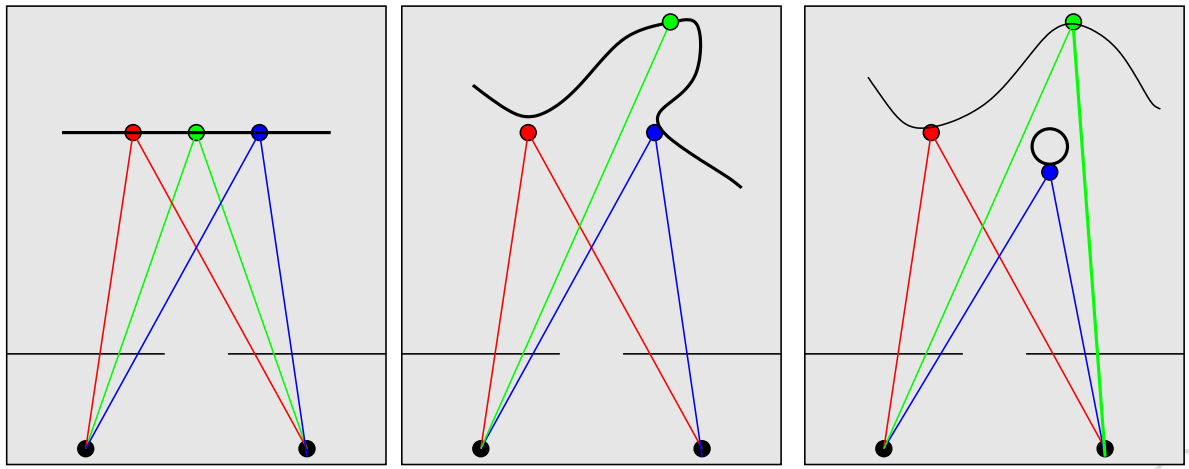
Uniqueness of match: Promote 1-to-1 matches, but there can be 0-to-1 and 1-to-0 because of occlusion.

Ordering: From a continuous opaque surface it is not possible to change the match order.

Disparity smoothness: This is stronger than the previous constraint and favours smooth surfaces with no sudden changes in depth.

Figural continuity: The disparity field along an epipolar line should not be much different from that along a neighbouring epipolar line.

Ordering

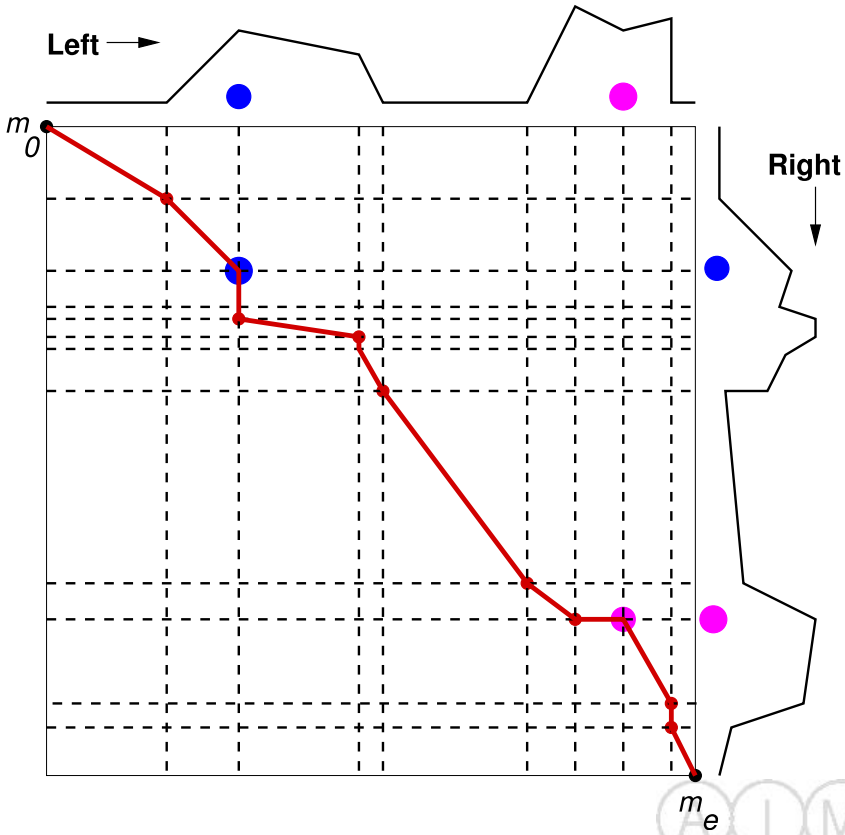


Constraints applied via dynamic programming

These constraints can be imposed using costs and optimization —

eg, using Dynamic Programming (Ohta and Kanade 1985, Polleyfeys *et al* 2000).

1. Imagine the raster intensities from the plotted out on two sides of a grid.
2. Significant features (edges, corners, or pixels) used as key potential matches
3. Crossing dashed lines represent potential matches at nodes.



Correspondence using Dynamic programming /ctd

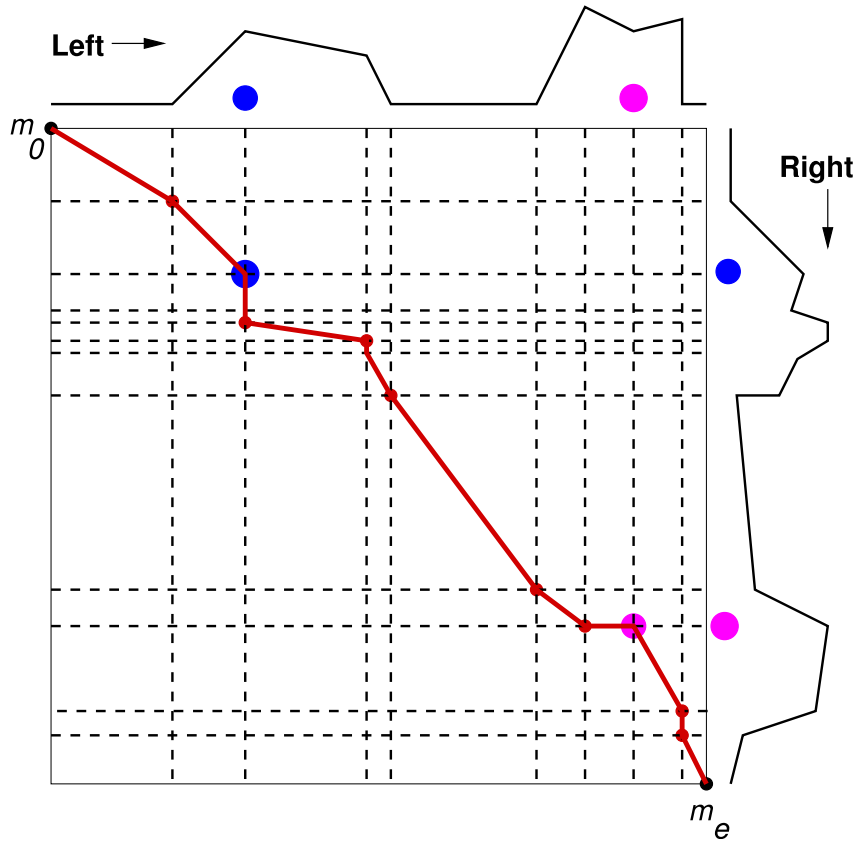
Aim: Construct a minimum cost path from top-left to bottom-right of the grid, which:

Ordering: has negative gradient in South-East quadrant

Uniqueness: rarely goes across or down

Completeness: visits as many potential nodes as possible

Figural Cont: is similar to paths constructed for neighbouring rasters.



Correspondence using Dynamic programming /ctd

Costs: Some costs associated with nodes, others with links.

Nodes: **Eg Similarity**

Features should be similar in images $\mathcal{C}$,$\mathcal{C}'$.

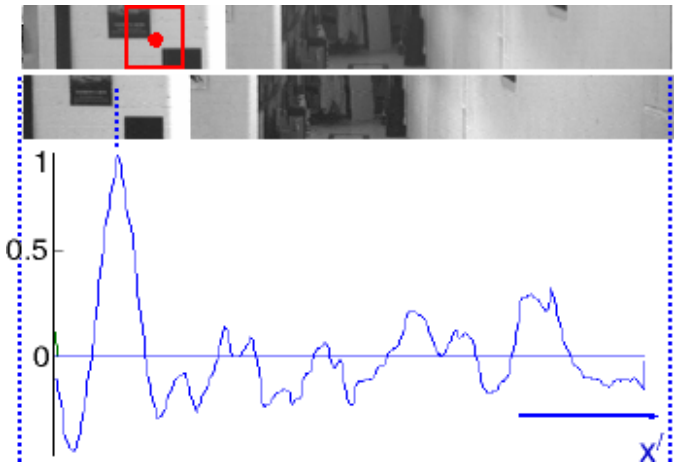
Eg, for NCC between patches

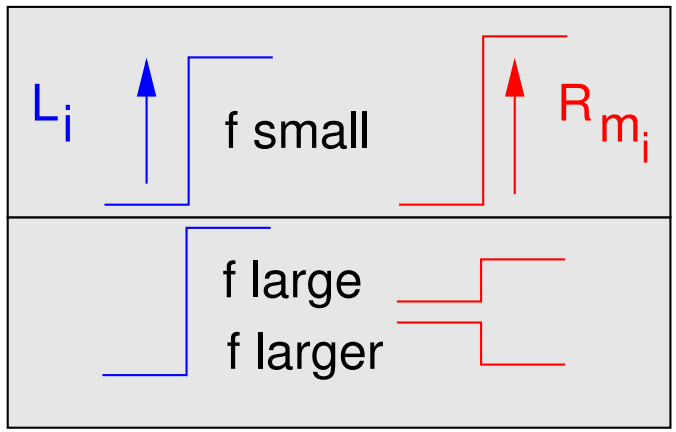
$$f_i(m_i) = \alpha(1 - NCC)^2$$

If you were using edges as features, you could use the constrast similarity

$$f_i(m_i) = \alpha(\mathcal{C}_i - \mathcal{C}'_{m_i})^2$$

α is a factor chosen empirically to trade-off the different costs.

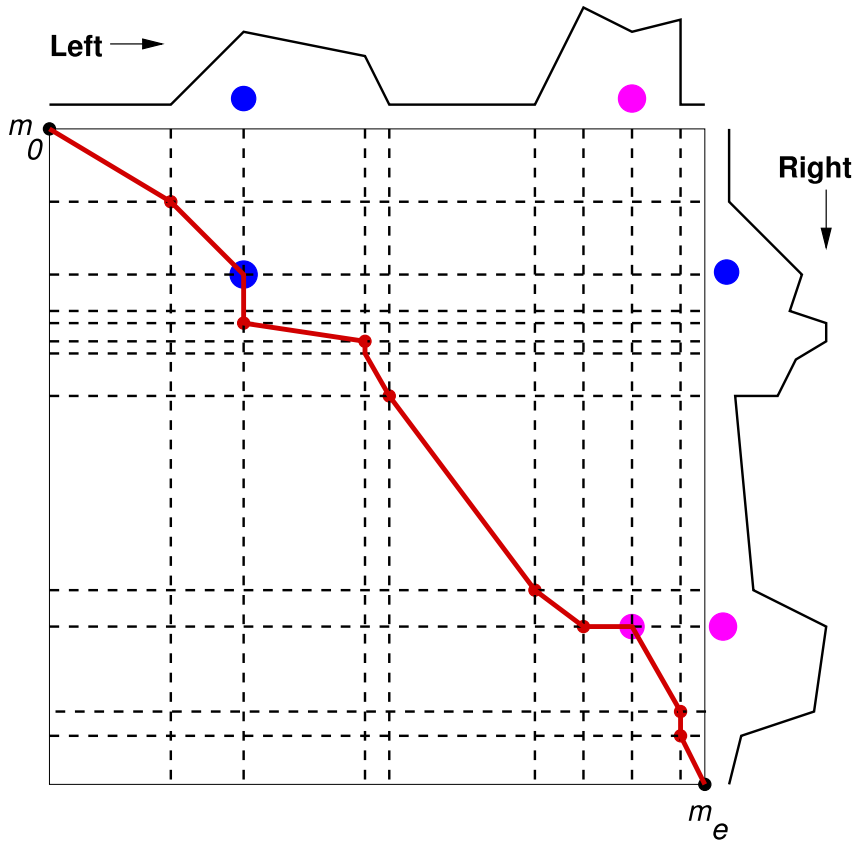




Correspondence using Dynamic programming /ctd

Links: Eg, uniqueness

Devise a link cost $g(m, m')$ between **matches** m and m', where
$(m, m') \equiv (i \to i', j \to j')$



Cost is

$$g(m, m') = \begin{cases} 0 & \text{if } i < i' \ \& \ j < j' \quad \text{OK} \\ \beta & \text{if } i = i' \ \& \ j < j' \quad \text{discourage} \\ \beta & \text{if } i < i' \ \& \ j = j' \quad \text{discourage} \\ \infty & \text{otherwise} \quad \text{verboten} \end{cases}$$

Correspondence using Dynamic programming /ctd

Choose m_0 to m_e to **Minimize sum of node and link costs**

$$C(m_0, m_e) = \sum_{i=0}^{e} f(m_i) + \sum_{i=0}^{e-1} g(m_i, m_{i+1})$$

$MN - 2$ variables, because m_0 and m_e are fixed at the raster ends.

They almost certainly don't match, so why are they in there?
... Because they impose ordering!

The minimum cost $C(m_0, m)$ of a path from m_0 to m is defined recursively

$$C(m_0, m) = \min_{p \in \mathcal{N}(m)} [f(m) + g(p, m) + C(m_0, p)]$$

DP replaces *simultaneous* minimization over $(MN - 2)$ variables by a $(MN - 1)$ *separate* minimizations involving just 1 variable (ie, p).

Correspondence using Dynamic programming /ctd

Algorithm involves exploring a graph or net:

For putative matches at level 1:
$C(m_0, m_1) = g(m_0, m_1)$.

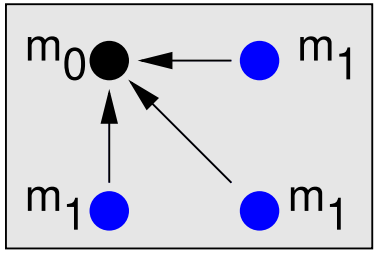
For putative matches at level n:
Find which p in the neighbourhood $\mathcal{N}(m_n)$ of match m_n gives

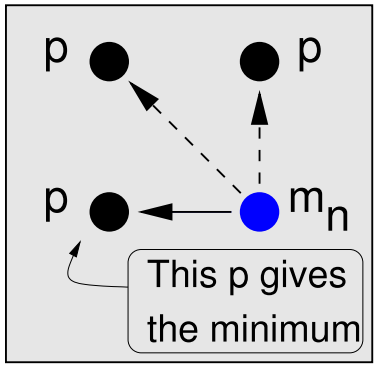
$$C(m_0, m_n) = \min_{p \in \mathcal{N}(m_n)} [f(m_n) + g(p, m_n) + C(m_0, p)]$$

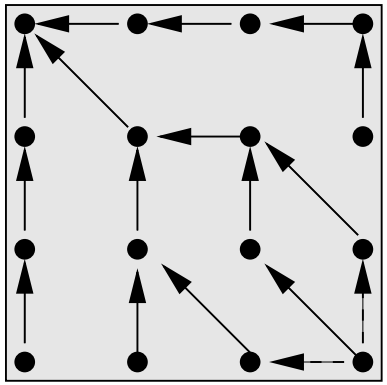
Keep only the minimum route/cost

Finally, **for putative matches at level $MN - 1$:**
Compute $C(m_e)$, then trace back.

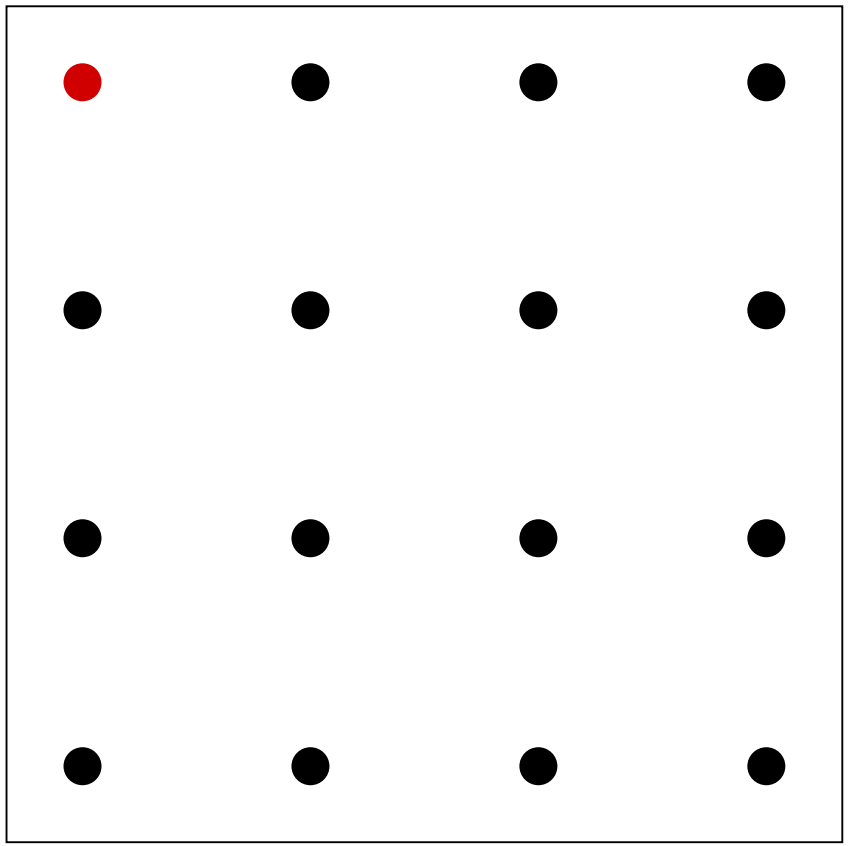
Note that at level n all the $C(m_0, p)$'s required are already derived and in place.



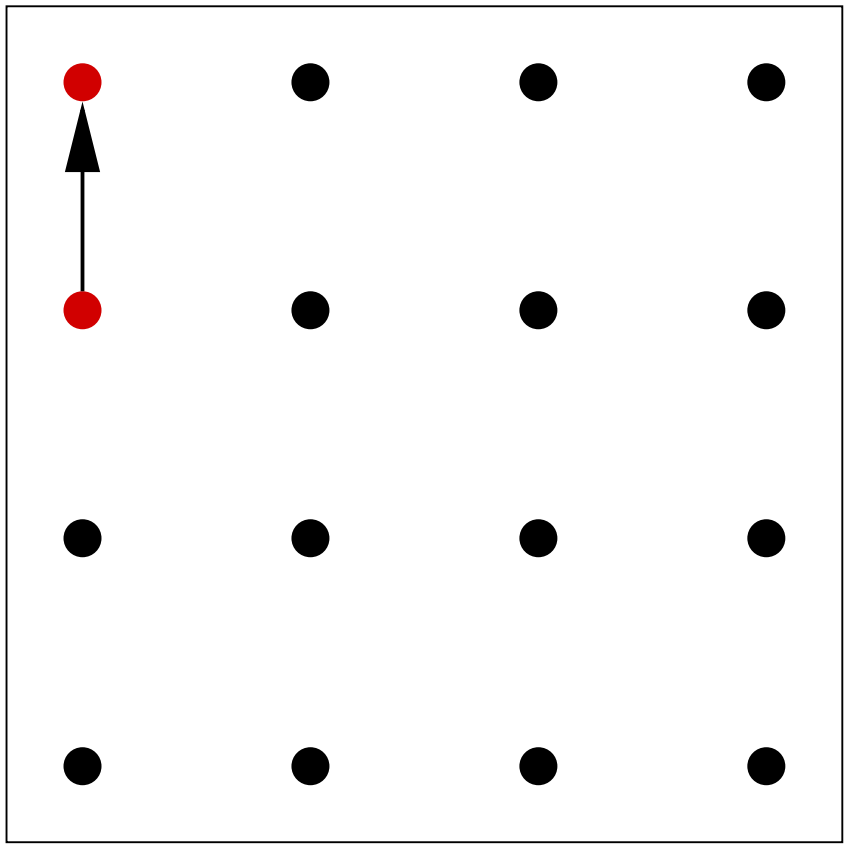




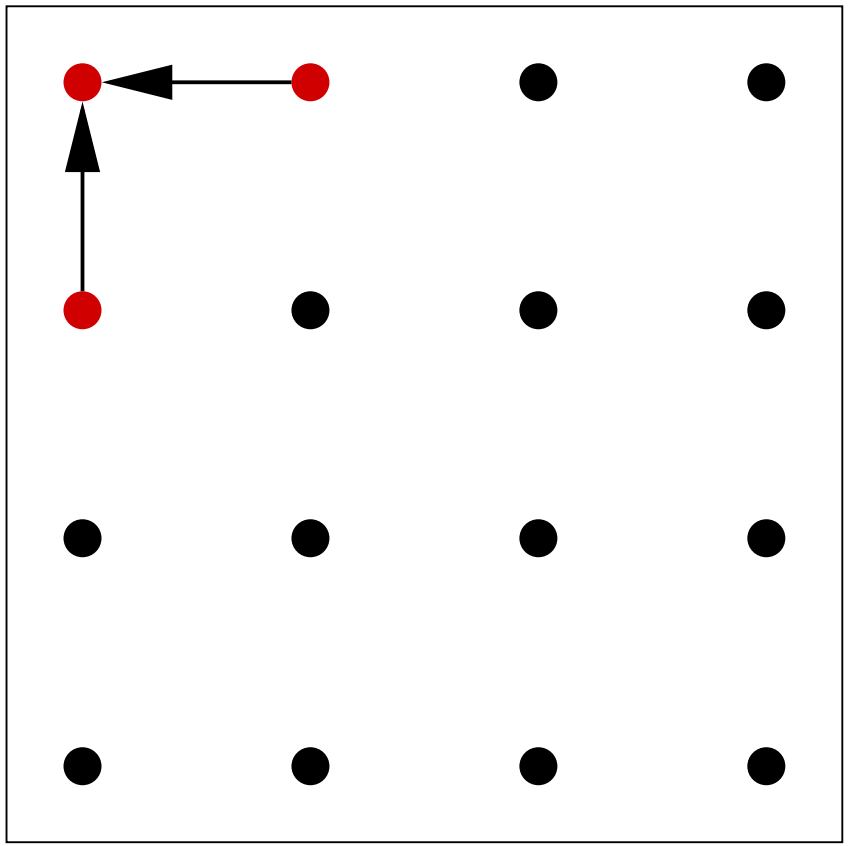
DP



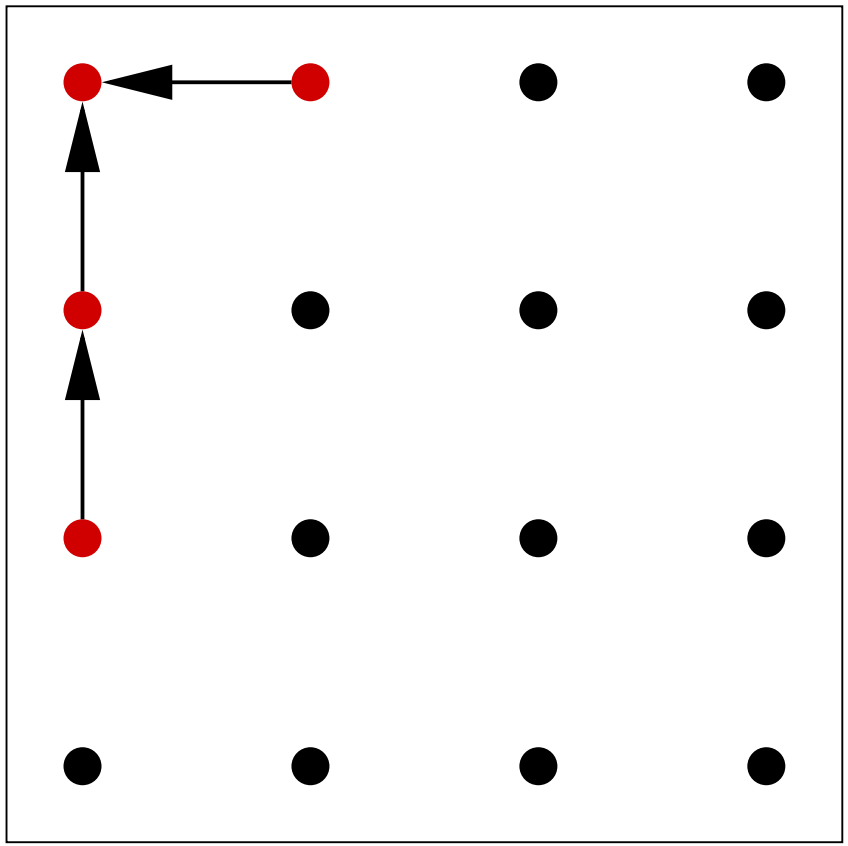
DP



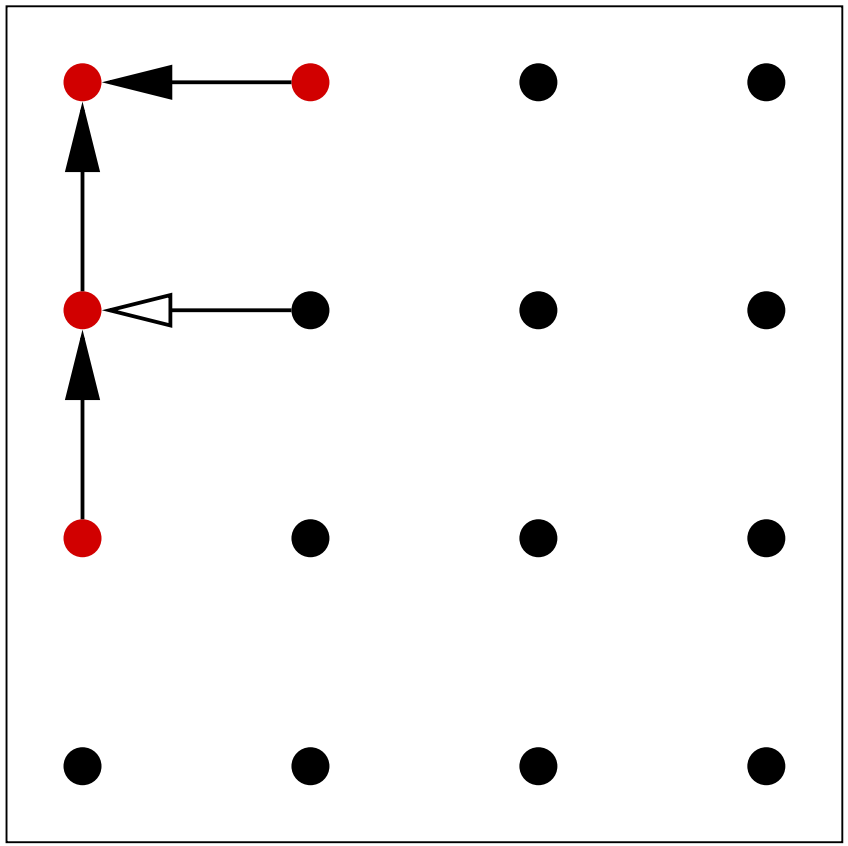
DP



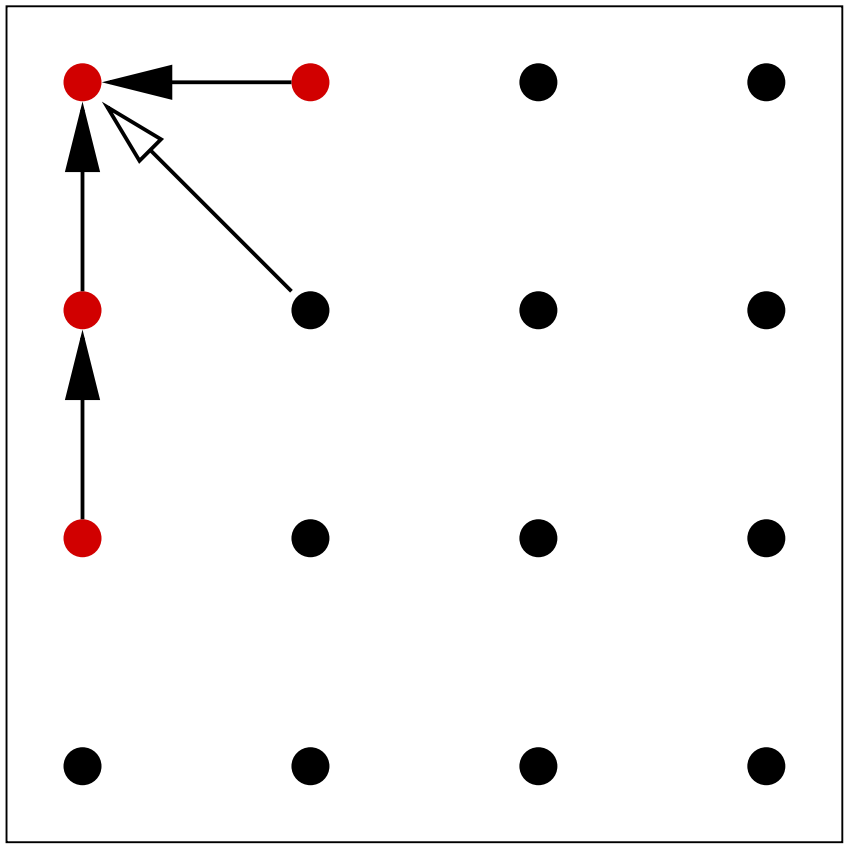
DP



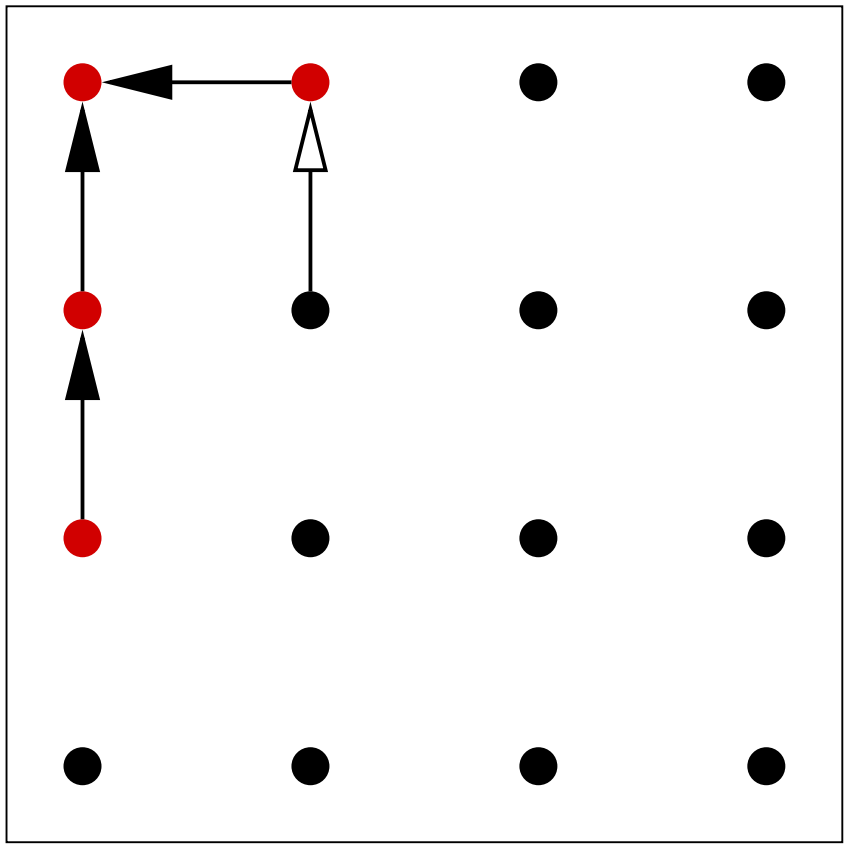
DP



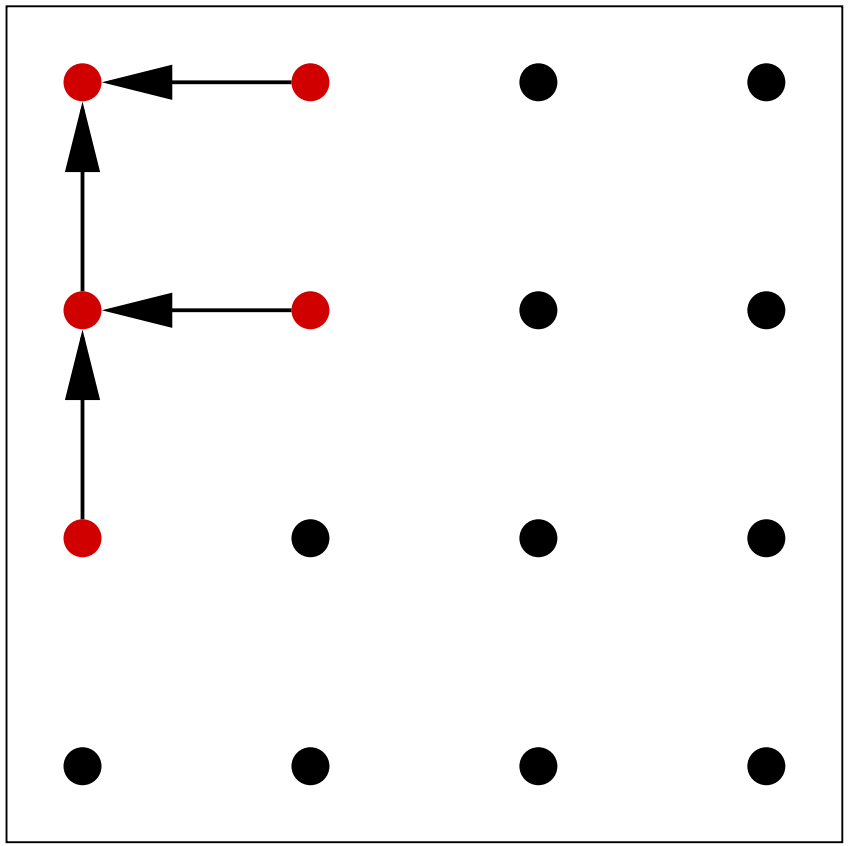
DP



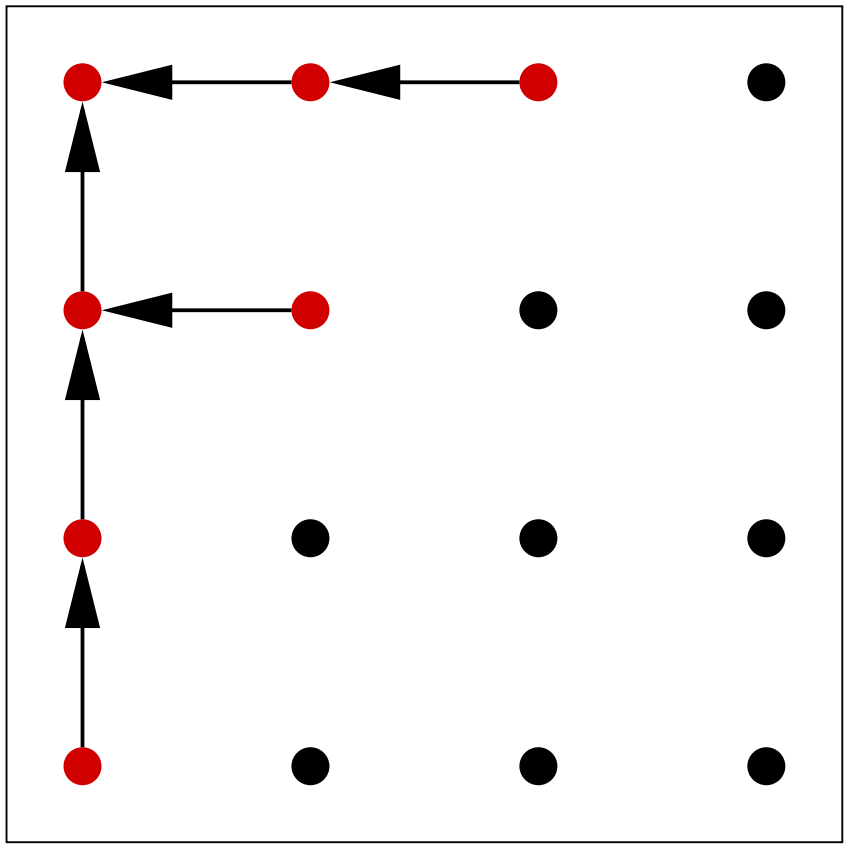
DP



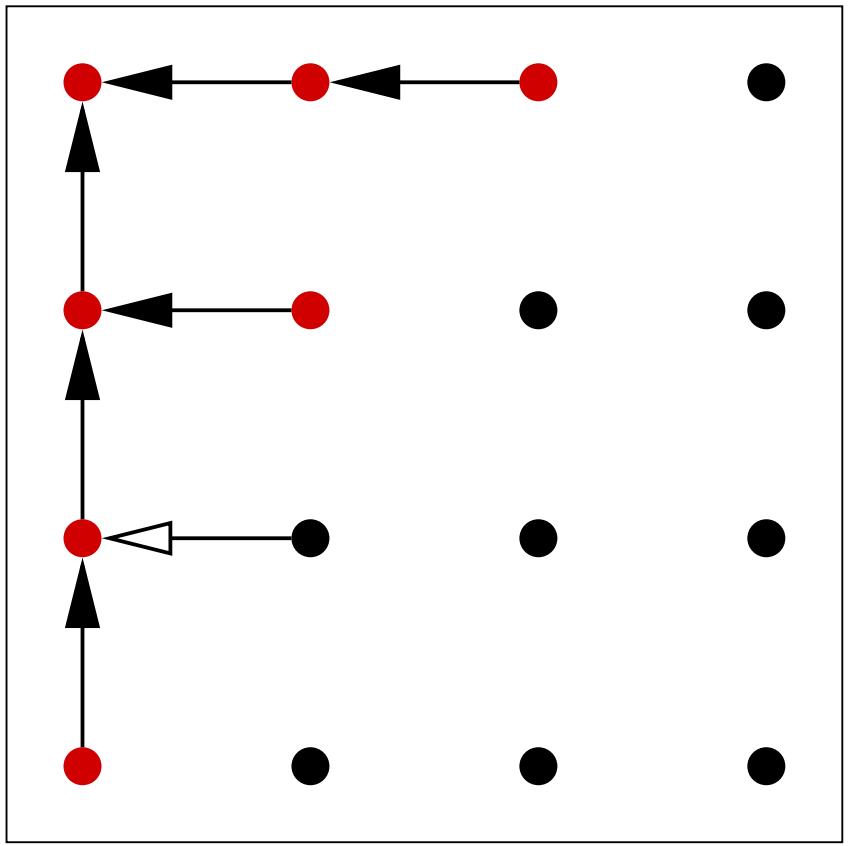
DP



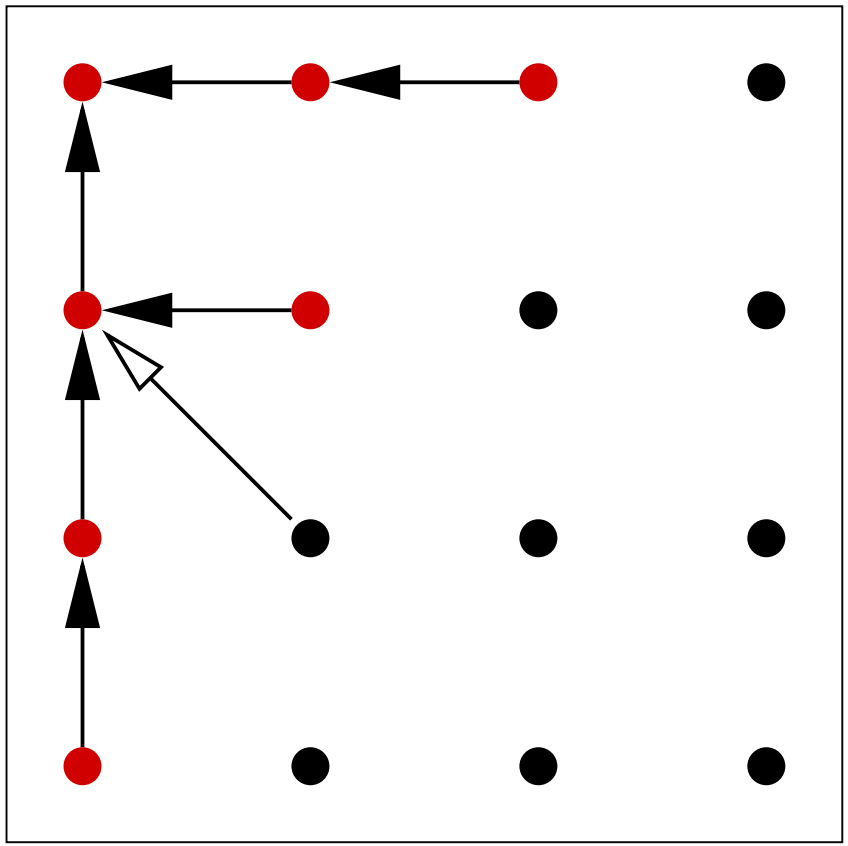
DP



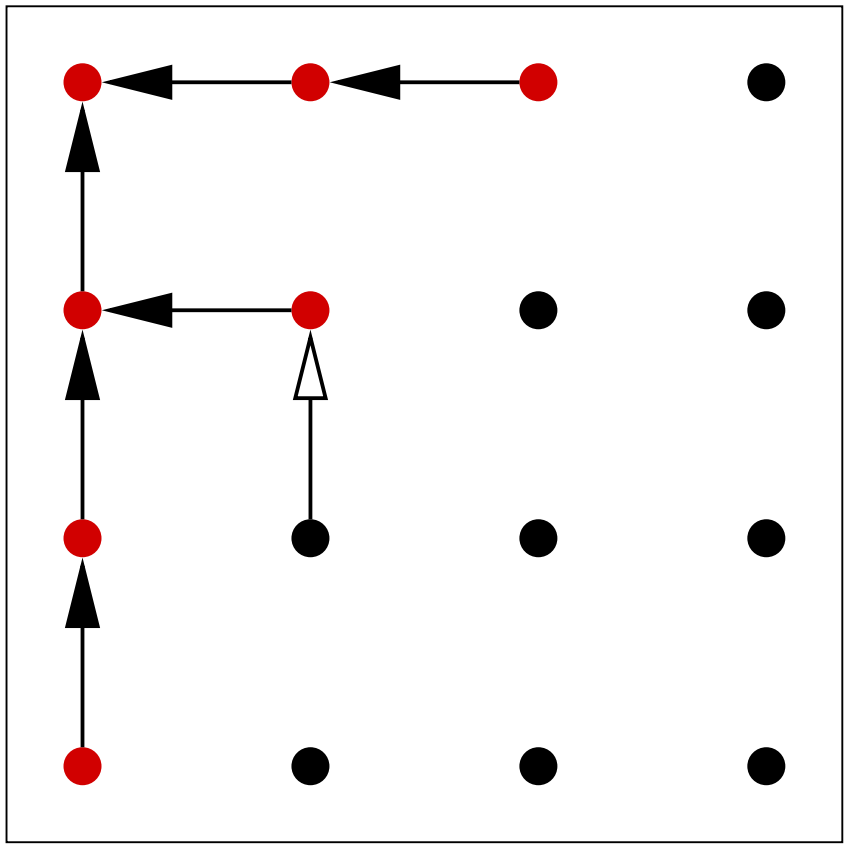
DP



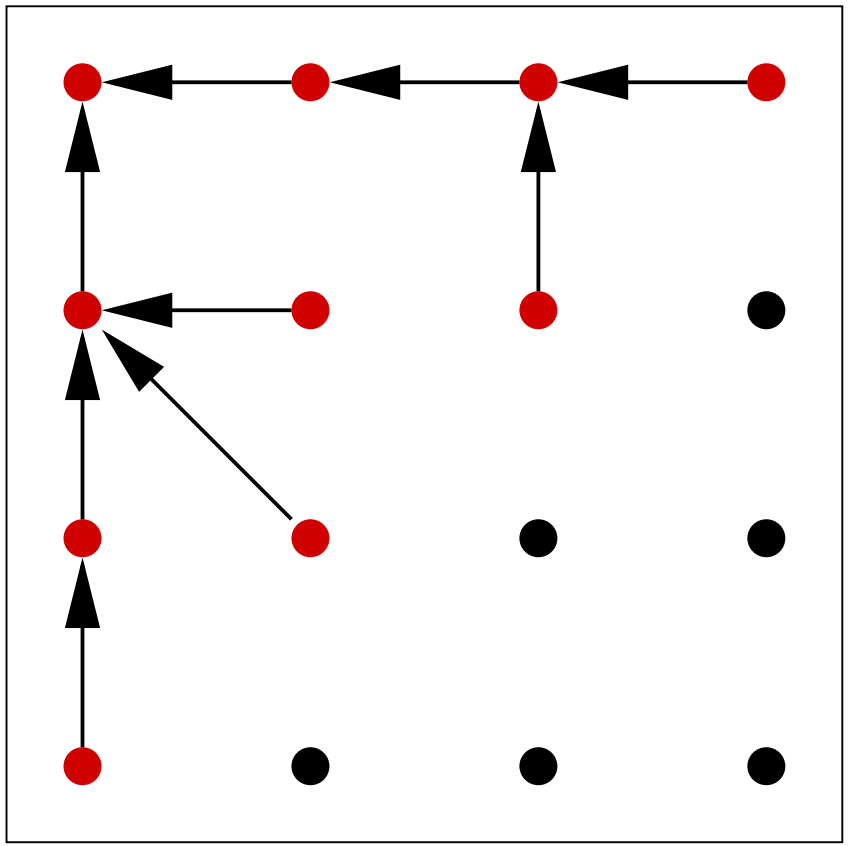
DP



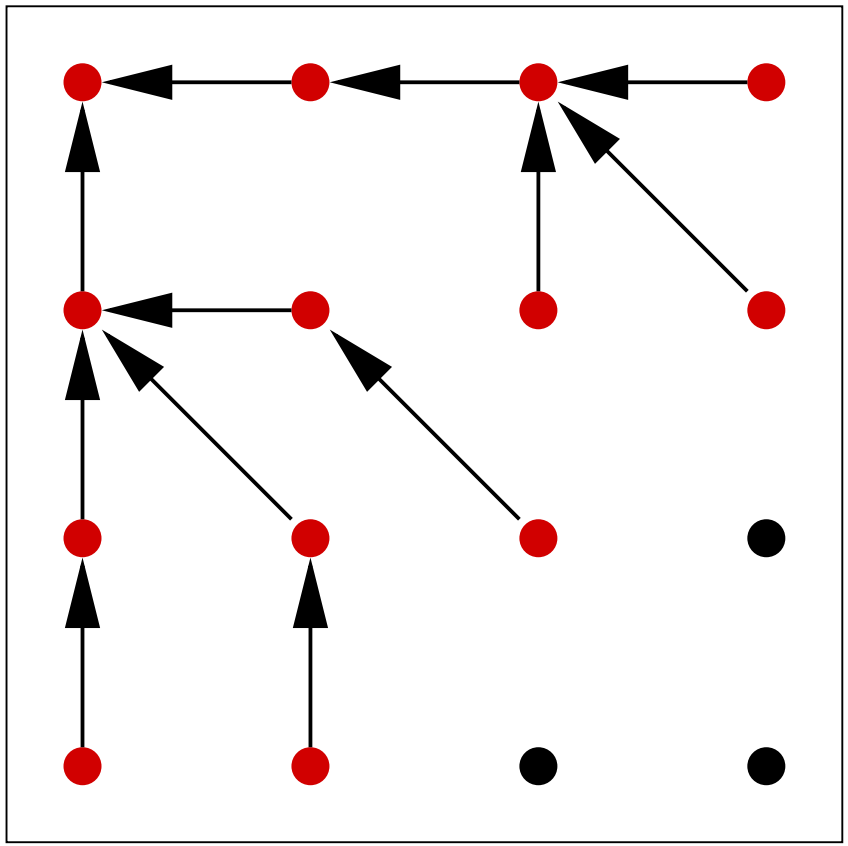
DP



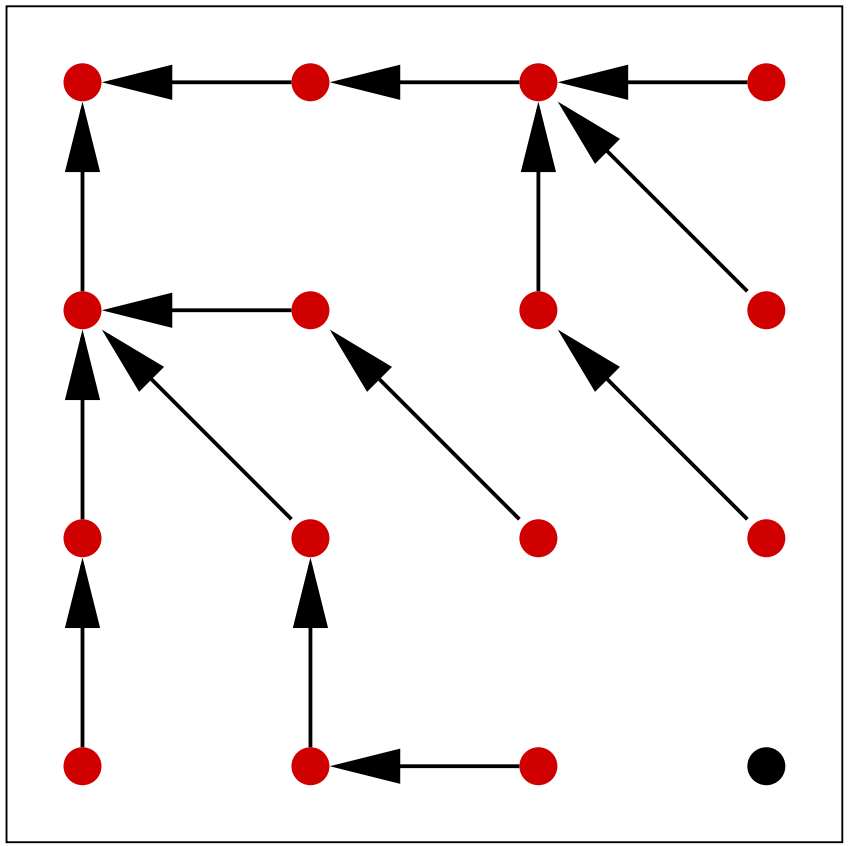
DP



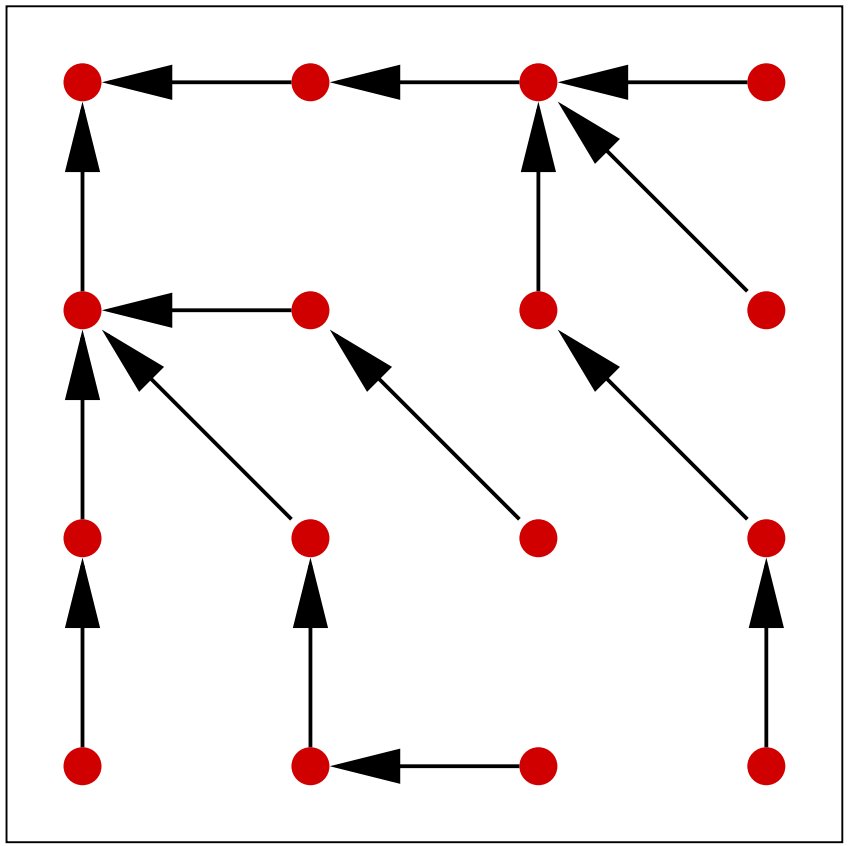
DP



DP



DP

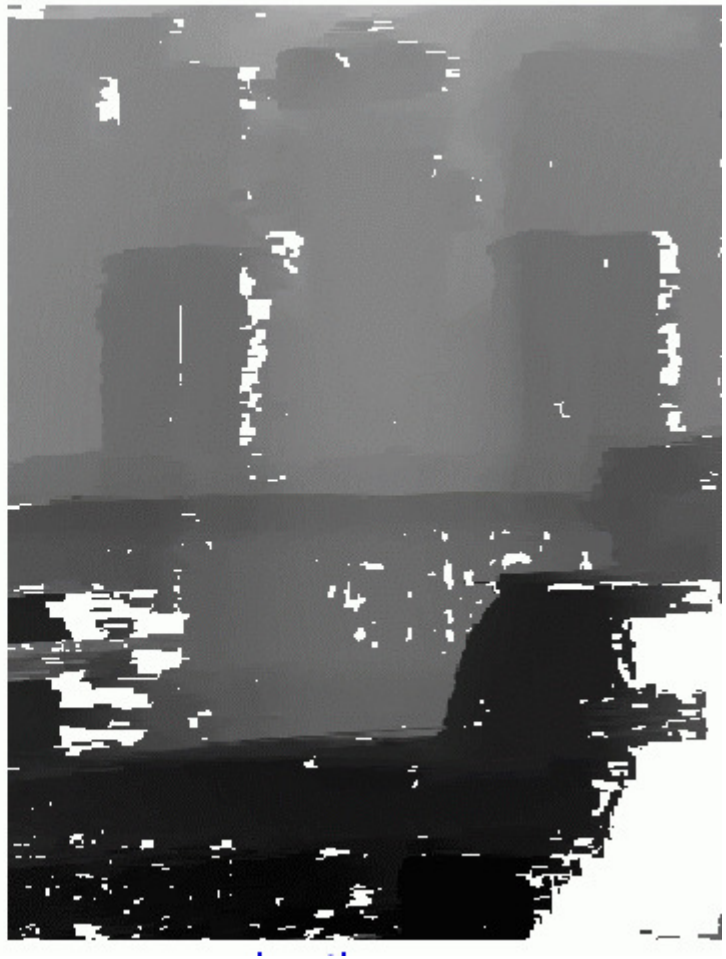


Example: Right and Left Images

Cross-eye fusable

Example: Full sequence

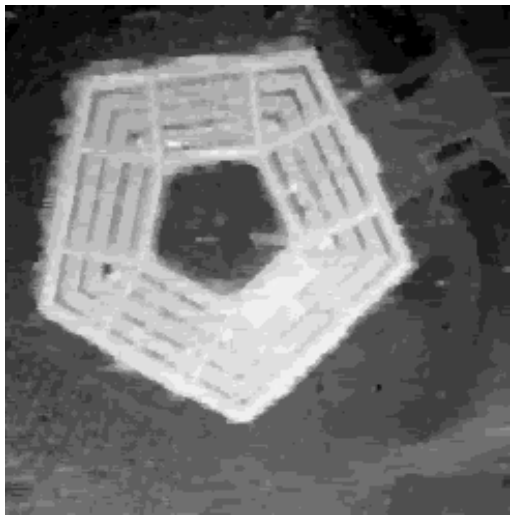
Results: Left Image and 3D Range Map



Results: 3D Reconstruction

Another reconstruction

Fantastic example for cross-eyed fusion
— if you can't do this one, you can't do it



← Range Map

3.6 Reconstruction by triangulation

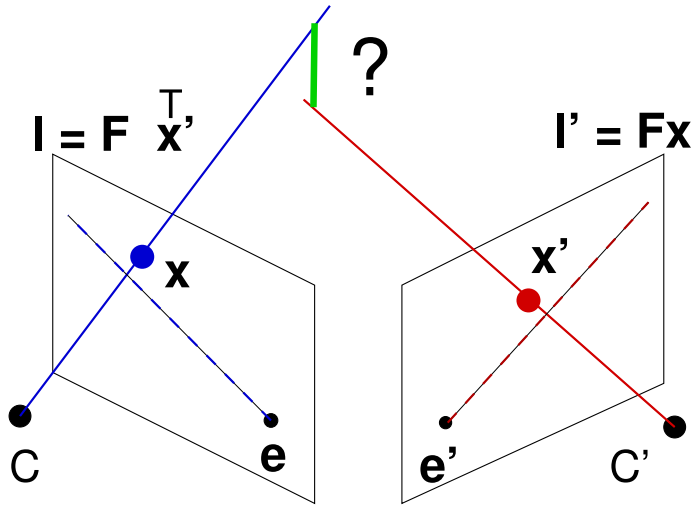
Given

a. exact proj. matrices $\boldsymbol{P}$, $\boldsymbol{P}'$

b. positionally noisy $\mathbf{x} \leftrightarrow \mathbf{x}'$

Compute

the 3D point $\mathbf{X}$.



Difficulties:

a. Back-projected rays will not meet, because

b. Measured points don't lie properly on epipolar lines

There are several cures ...

Approach I: Vector Midpoint

Assume $\mathbf{a}, \mathbf{b}, \mathbf{c}$ defined in $\mathcal{C}$ and use 3 components.

$\Rightarrow \mathbf{X}_{3\times 1} = (\mathbf{0} + \alpha\mathbf{a} + \mathbf{c} + \beta\mathbf{b})/2$ when

$\alpha\mathbf{a} + \gamma(\mathbf{a} \times \mathbf{b}) = \mathbf{c} + \beta\mathbf{b}$

$\alpha(\mathbf{a}\cdot\mathbf{a}) = (\mathbf{a}\cdot\mathbf{c}) + \beta(\mathbf{a}\cdot\mathbf{b})$ and

$\alpha(\mathbf{a}\cdot\mathbf{b}) = (\mathbf{b}\cdot\mathbf{c}) + \beta(\mathbf{b}\cdot\mathbf{b})$

Hence with $D = (\mathbf{a}\cdot\mathbf{a})(\mathbf{b}\cdot\mathbf{b}) - (\mathbf{a}\cdot\mathbf{b})^2$

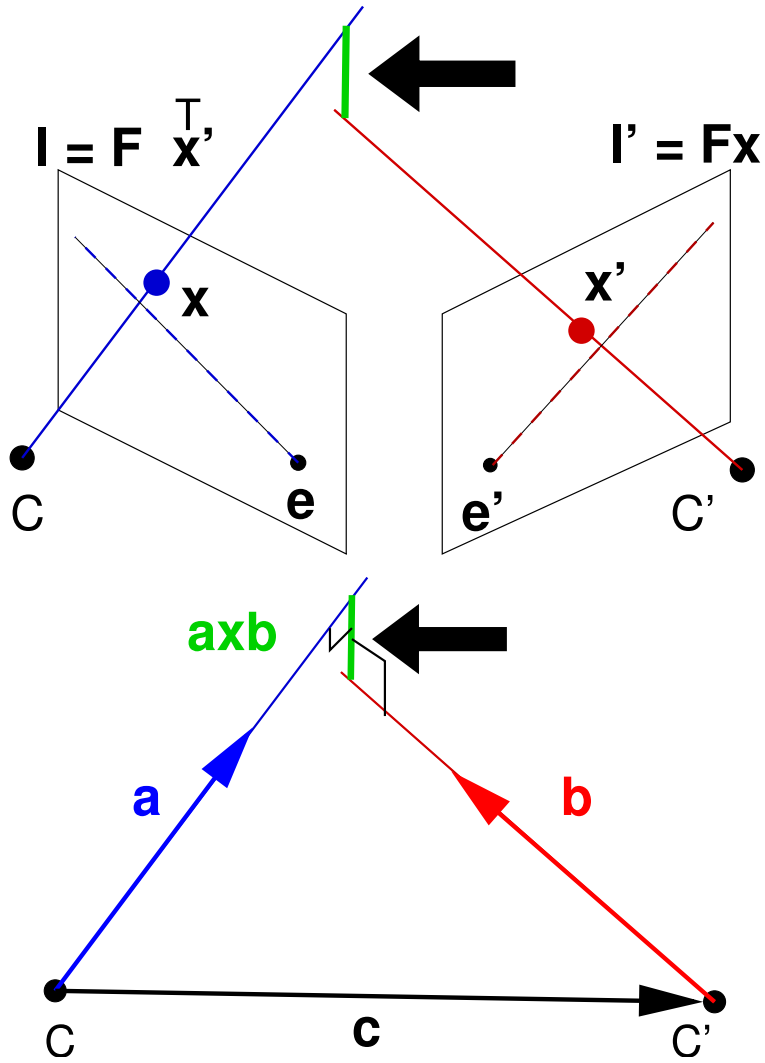
$\alpha = ((\mathbf{b}\cdot\mathbf{b})(\mathbf{a}\cdot\mathbf{c}) - (\mathbf{a}\cdot\mathbf{b})(\mathbf{b}\cdot\mathbf{c}))/D$

$\beta = ((\mathbf{a}\cdot\mathbf{b})(\mathbf{a}\cdot\mathbf{c}) - (\mathbf{a}\cdot\mathbf{a})(\mathbf{b}\cdot\mathbf{c}))/D$

Now specify the vectors in the $\mathcal{C}$ frame.

$$\mathbf{a} = \begin{bmatrix} \boldsymbol{K}^{-1}\mathbf{x} \\ 0 \end{bmatrix}; \mathbf{b} = \begin{bmatrix} \boldsymbol{R}^{-1}\boldsymbol{K}'^{-1}\mathbf{x}' \\ 0 \end{bmatrix};$$

$$\mathbf{c} = \begin{bmatrix} -\boldsymbol{R}^{-1}\mathbf{t} \\ 1 \end{bmatrix}.$$



Assuming all vectors in $\mathcal{C}$.

Approach I: Vector Midpoint /ctd

Why is $\mathbf{c} = \begin{bmatrix} -\boldsymbol{R}^{-1}\mathbf{t} \\ 1 \end{bmatrix}$?

First, recall that $\mathbf{x}' = \boldsymbol{K}'[\boldsymbol{R}|\mathbf{t}]\mathbf{X}$ because

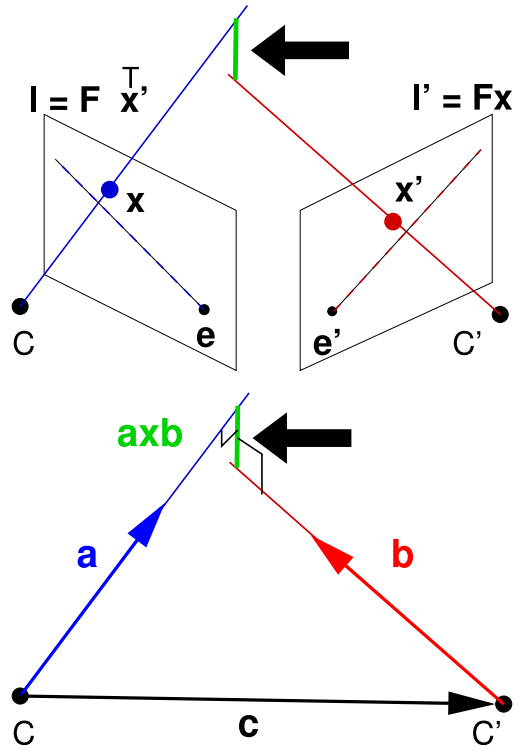
$$\mathbf{X}' = \begin{bmatrix} \boldsymbol{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}$$

$$\Rightarrow \mathbf{X} = \begin{bmatrix} \boldsymbol{R}^{-1} & -\boldsymbol{R}^{-1}\mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}'$$

The camera centre of $\mathcal{C}'$ is at $\begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$ in $\mathcal{C}'$, and so in $\mathcal{C}$

$$\mathbf{c} = \begin{bmatrix} \boldsymbol{R}^{-1} & -\boldsymbol{R}^{-1}\mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} = \begin{bmatrix} -\boldsymbol{R}^{-1}\mathbf{t} \\ 1 \end{bmatrix}$$

All vectors in $\mathcal{C}$.



Approach II: Linear Triangulation

Use the equations $\mathbf{x} = \boldsymbol{P}\mathbf{X}$ and $\mathbf{x}' = \boldsymbol{P}'\mathbf{X}$ to solve for $\mathbf{X}$

Write

$$\boldsymbol{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} \mathbf{p}^{1\top} \\ \mathbf{p}^{2\top} \\ \mathbf{p}^{3\top} \end{bmatrix}$$

Eliminate unknown scale in $\lambda\mathbf{x} = \boldsymbol{P}\mathbf{X}$ by forming a cross product $\mathbf{x} \times (\boldsymbol{P}\mathbf{X}) = \mathbf{0}$. Its three components are

$$\begin{aligned} x(\mathbf{p}^{3\top}\mathbf{X}) - (\mathbf{p}^{1\top}\mathbf{X}) &= 0 \\ y(\mathbf{p}^{3\top}\mathbf{X}) - (\mathbf{p}^{2\top}\mathbf{X}) &= 0 \\ x(\mathbf{p}^{2\top}\mathbf{X}) - y(\mathbf{p}^{1\top}\mathbf{X}) &= 0\ . \end{aligned}$$

Rearrange just the first two as

$$\begin{bmatrix} x\mathbf{p}^{3\top} - \mathbf{p}^{1\top} \\ y\mathbf{p}^{3\top} - \mathbf{p}^{2\top} \end{bmatrix}_{2\times 4} \mathbf{X}_{4\times 1} = \mathbf{0}_{2\times 1}$$

Approach II: Linear Triangulation /ctd

Write the same for the second camera

$$\left[\begin{array}{c} x'\mathbf{p}'^{3\top} - \mathbf{p}'^{1\top} \\ y'\mathbf{p}'^{3\top} - \mathbf{p}'^{2\top} \end{array}\right]_{2\times 4} \mathbf{X}_{4\times 1} = \mathbf{0}_{2\times 1}$$

Put one above the other to obtain $\boldsymbol{A}_{4\times 4}\mathbf{X}_{4\times 1} = \mathbf{0}_{4\times 1}$ where

$$\boldsymbol{A}_{4\times 4}\mathbf{X}_{4\times 1} = \left[\begin{array}{c} x\mathbf{p}^{3\top} - \mathbf{p}^{1\top} \\ y'\mathbf{p}^{3\top} - \mathbf{p}^{2\top} \\ x'\mathbf{p}'^{3\top} - \mathbf{p}'^{1\top} \\ y'\mathbf{p}'^{3\top} - \mathbf{p}'^{2\top} \end{array}\right] \mathbf{X}_{4\times 1} = \mathbf{0}_{4\times 1}$$

This is a null-space problem, from which $\mathbf{X}$ can be recovered up to scale

Advantage: extends to more than two views.

Disadvantage: this is a minimization, but the quantity being minimized is not related to realistic error.

Approach III: Minimizing a geometrical/statistical error

Project estimated scene position

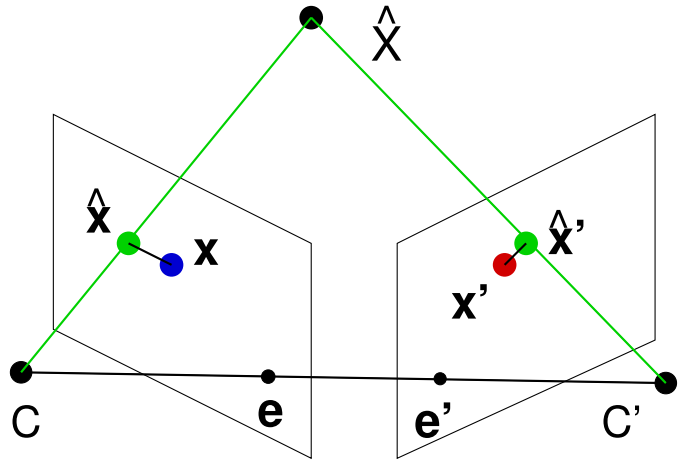
$$\hat{\mathbf{x}} = \boldsymbol{P}\hat{\mathbf{X}} \quad \hat{\mathbf{x}}' = \boldsymbol{P}'\hat{\mathbf{X}}$$

Measure Euclidean displacements

$$d(\mathbf{x}, \hat{\mathbf{x}}) \quad d(\mathbf{x}', \hat{\mathbf{x}}')$$

Compute the cost

$$C(\hat{\mathbf{X}}) = d^2(\mathbf{x}, \hat{\mathbf{x}}) + d^2(\mathbf{x}', \hat{\mathbf{x}}')$$



Adjust $\hat{\mathbf{X}}$ to minimize the cost.
If the measurement noise is a zero-mean Gaussian $\mathcal{N}(0, \sigma^2)$, then this is the Maximum Likelihood Estimator of $\mathbf{X}$.
It looks like a minimization over 3 parameters ...
... but it can be reduced to a single parameter.
Can you see how?

A note on Triangulation depth error

Triangulation involves intersecting rays in the epipolar plane. Depth uncertainty must depend on the uncertainty in point location along the epipolar lines.

Analyze parallel camera formula

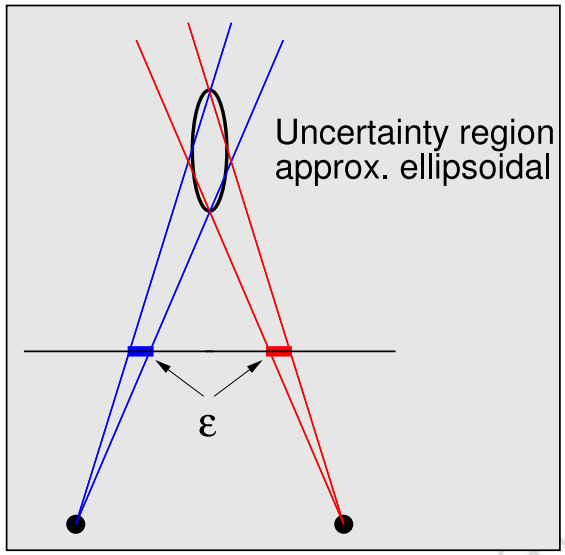
$$\frac{1}{Z} = \frac{1}{ft_x}(x' - x)$$

$$\Rightarrow d\left(\frac{1}{Z}\right) = -\frac{dZ}{Z^2} = \frac{1}{ft_x}(dx' - dx)$$

$$\Rightarrow \left(\frac{\delta Z}{Z^2}\right)^2 \approx \left(\frac{1}{ft_x}\right)^2 \left((\delta x')^2 + (\delta x)^2\right)$$

Conclusion is:
Depth Error: $\delta Z \approx Z^2\sqrt{2}\epsilon/ft_x$
where ϵ is typical image measurement error.

Reminder ...
$f = f(u, v...)$, $\Rightarrow$1st-order Taylor
$\delta f = \frac{\partial f}{\partial u}\delta u + \frac{\partial f}{\partial v}\delta v + \ldots$
Assuming no correlation, we add errors in quadrature:
$\Rightarrow (\delta f)^2 = \left(\frac{\partial f}{\partial u}\delta u\right)^2 + \left(\frac{\partial f}{\partial v}\delta v\right)^2 + \ldots$



Real time reconstruction from Stereo

Miksik, Lidegaard, Golodetz & Torr (2014)