C18 Computer Vision

David Murray

david.murray@eng.ox.ac.uk
www.robots.ox.ac.uk/~dwm/Courses/4CV

Michaelmas 2015

Course Content

DWM: Intro, basic image features, basic multiview geometry ...

1. **Introduction; imaging geometry; camera calibration**
2. **Salient feature detection – edges, line and corners**
3. **Recovering 3D from two images I: epipolar geometry.**
4. **Recovering 3D from two images II: stereo correspondence algorithms; triangulation.**

Slides, videos, etc at www.robots.ox.ac.uk/~dwm/Courses/4CV

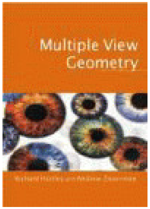
AV: Motion, tracking, recognition ...

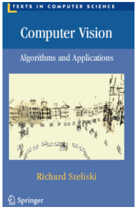
5. **Structure from Motion I: estimating the F matrix**
6. **Structure from Motion II**
7. **Visual Motion and Tracking**
8. **Object Detection and Recognition**

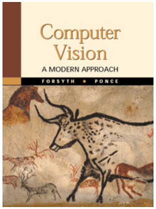
Useful texts

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

Computer Vision: This time ...

1. **Introduction; imaging geometry; camera calibration**
2. Salient feature detection – edges, line and corners
3. Recovering 3D from two images I: epipolar geometry.
4. Recovering 3D from two images II: stereo correspondence algorithms; triangulation.

Lecture 1

1.1 Introduction

1.2 The perspective camera as a geometric device

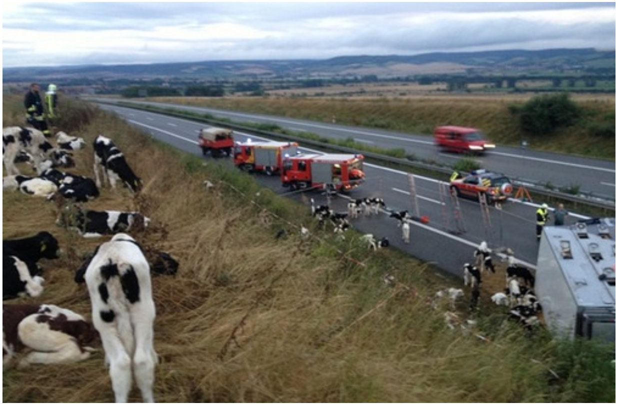
1.3 Perspective using homogeneous coordinates

1.4 Calibrating the elements of the projection model

1.1 Introduction

Aim in computational vision is to take a number of 2D images, and from them obtain an understanding of the 3D environment; what is in it; and how it evolves over time.

What have we here ... ?



So it is easy ...

Wrong! It's the mother of big data problems

Some thoughts why?

From the hardware engineering perspective?

- The raw throughput is upsettingly large:
- Colour stereo pair at 30Hz is $\sim$ 100 MBs^{-1}
- Now multiply by non-trivial processing cost per Byte ...
- Image collections are huge

Certainly challenging, but no longer frightening

From applied maths perspective?

- Scene info is made highly implict or lost by projection
- Inverse mappings 2D$\rightarrow$3D ill-posed, ill-conditioned.
- Can only be solved by introducing **constraints**:
  (i) about the way the world *actually* works; or
  (ii) about the way we *expect* it to work.

We now know about these. No longer frightening

Some thoughts why ...

From the Information-Engineering/AI perspective ...

- Images have uneven information content both absolutely *and* contextually.
- Computational visual semantics
  — what does visual stuff *mean* exactly?
- If we are under time pressure, what is the important visual stuff right now?

Still a massive challenge — if we want genuine autonomy

Natural vision a hard problem

But we see effortlessly! Yep, spot on if one neglects:

- the 10^{11} neurons involved
- aeons of evolution generating hardwired priors $P(I)$
- that we sleep with our eyes shut, and avert gaze when thinking.

Hard work for our brains — does machine vision have a hope?

- Perhaps building a "general" vision system is a flawed concept.
- Evidence that the human visual system is a *bag of tricks* — specialized processing for specialized tasks.
  Perhaps we should expect no more of computer vision?

However, the h.v.s does give us a convincing magic show

- Each trick flows seamlessly into the next. Do we have an inkling how that will be achieved?

So why bother? What are the advantages?

From a sensor standpoint ...

- Vision is a passive sensor (unlike sonar, radar, lasers)
- Wide range of depths, overlaps and complements other sensors
- Wide diversity of methods of recover 3D information: so vision has **process redundancy**.
- Images provide **data redundancy**.

From a natural communications standpoint ...

- The world is awash with information-rich photons
- Because we have eyes, vision provides a natural language of communication. If we want robots/man-machine-interfaces to act and interact in our environment, they have to speak that language.

Organizing the tricks ...

Although human and computer vision might be bags of tricks, it is useful to place the tricks within larger processing paradigms.

For example:

a. Data-driven, bottom-up processing
b. Model-driven, top-down, generative processing
c. Dynamic Vision (mixes bottom-up with top-down feedback)
d. Active Vision (task oriented)
e. Data-driven discriminative approach (machine learning)

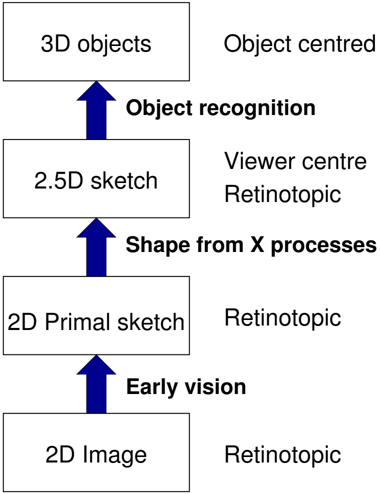
These are neither all-embracing nor exclusive

a. Data-driven, bottom-up processing

Image processing produces map of salient **2D features**.

Features input into a range of **Shape from X** processes whose output was the **$2\frac{1}{2}$D** sketch. This was a thumbnail sketch of the 3D world containing sparse depths, surface orientations and so on, so called because although it contained 3D information, the information is retinotopically-mapped.

Only in the last stage was a fully 3D **object-centred** description achieved.

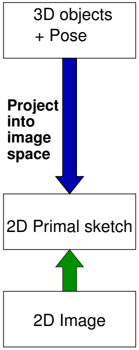


b. Model-driven, and c. Dynamic vision

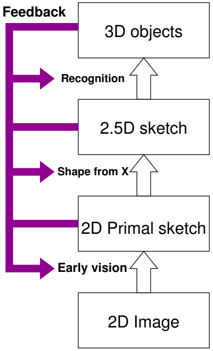
b. Model-driven, top-down, generative processing — a model of the scene is assumed known.

Supply a pose for the object relative to the camera, and use projection to predict where its salient features should be found in the image space.

Search for the features, and refine the pose by minimizing the observed deviation.



Top-Down



Dynamic

c. Dynamic Vision

mixes bottom-up/top-down by introducing feedback

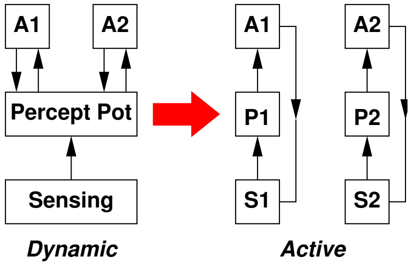
d. Active Vision

A criticism of earlier paradigms was that vision was divorced from the actions that its "owner" might wish to take.

Actions A1 and A2 had to play "lucky dip" in the perceptual bran tub.

By introducing task-oriented sensing-perception-action loops:

1. Visual data need only be "good enough" to drive the particular action.

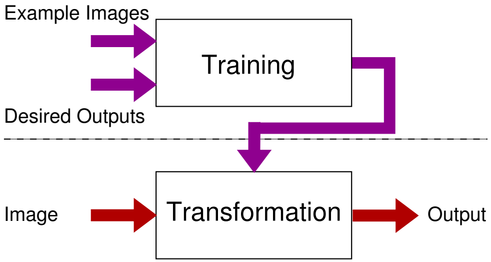


2. No need to build and maintain an overarching representation of the surroundings.

3. Computational resources focussed where they are needed.

e. Data-driven discriminative approach

The aim is to learn a description of the transformation between input and output using exemplars



Geometry is not forgotten, but implicit learned representations are favoured.

The importance of Geometry

A consistent thread through these paradigms is

The geometry of the scene, the lighting and and the properties of the imaging device are key to image formation, and so are also key to understanding reconstruction of the scene from imagery.

That geometry is paramount seems self-evident ...

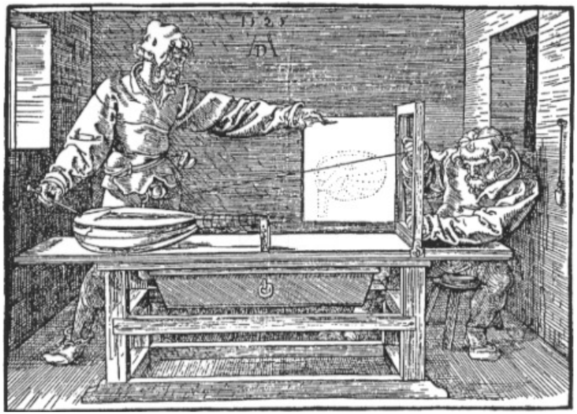
Clear in the 1960's (eg Louis Roberts' 1965 system)

Message nearly lost in the 1970's — expert systems in simple domains led to the *"if above and blue then sky"* rule-based approach. Disaster!

Geometrical modelling regained dominance in 1980's (calibrated) and 1990's (uncalibrated)

Emphasis in 2000's – avoiding explicit modelling of geometry by learning.

1.2 The perspective camera as a geometric device



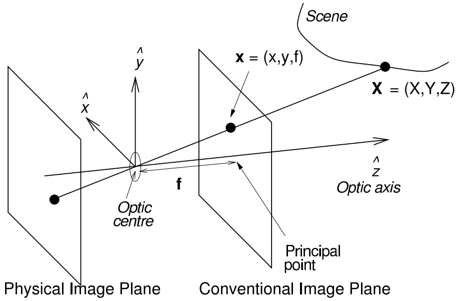
The perspective camera as a geometric device

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

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

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

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



Homogeneous coordinates

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

However, it turns out that the projection can be made linear by adopting **homogeneous coordinates**, which involves representing the image and scene in a higher dimensional space.

They handle limiting cases betters — e.g. points at infinity and vanishing points.

Using homogeneous coordinates also allows transformations to be concatenated more easily —
and here we will start with that 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 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 $\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$

then the transformation can be represented by a 4×4 matrix ...

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

Obviously the matrix has block form:

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

Combining successive transformations

Transformations can now be concatenated by matrix multiplication.

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

then

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

Checking ...

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

as earlier.

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 $\quad X = X_1/X_4 \quad Y = X_2/X_4 \quad Z = X_3/X_4$

- So the following homogeneous vectors represent the same point for any non-zero λ.

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

- 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.

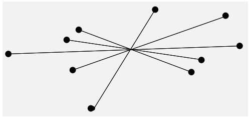
$$(1, 2, 3)^\top \equiv (3, 6, 9)^\top \equiv (-2, -4, -6)^\top$$

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

Projective transformations

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

[**] More 3D-3D & 2D-2D Transformations

Projectivity (15 dof)

$$\begin{bmatrix} X_1' \\ X_2' \\ X_3' \\ X_4' \end{bmatrix} \overset{P}{=} [\boldsymbol{P}_{4\times4}] \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}$$

Affine (12 dof)

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{3\times3} & \mathbf{t}_3 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

Similarity (7 dof)

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} SR_{3\times3} & S\mathbf{t}_3 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

Euclidean (6 dof)

$$\begin{bmatrix} \mathbf{X}' \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3\times3} & \mathbf{t}_3 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

Projectivity (8 dof)

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

Affine (6 dof)

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} A_{2\times2} & \mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

Similarity (4 dof)

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} SR_{2\times2} & S\mathbf{t}_2 \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

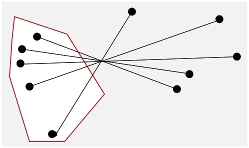
Euclidean (3 dof)

$$\begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix} = \begin{bmatrix} R_{2\times2} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

Perspective 3D-2D transformation

Now constrain the transformed points lie on the plane $z = f$

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



Then, because $z = f$ is fixed, it must be that

$$\lambda \begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ fp_{41} & fp_{42} & fp_{43} & fp_{44} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

The 3rd row is redundant! Remove it and re-label elements:

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

A **perspective transformation**. The matrix is the **projection matrix**.

1.3 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
$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$	$=$	$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$

To check ...

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

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 Matrix**	Projection matrix (vanilla)	**Camera's Extrinsic Matrix**	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

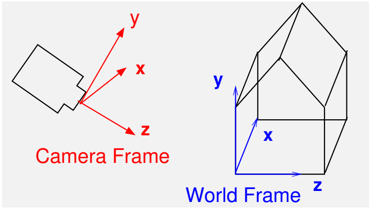
	Intrinsic Matrix	Projection	Extrinsic Matrix	
$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} =$	$\begin{bmatrix} f & sf & u_0 \\ 0 & \gamma f & v_0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} r_{11} & r_{12} & r_{13}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

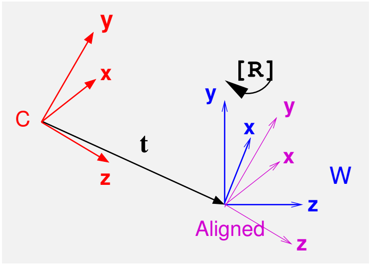
The Extrinsic matrix arameters

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

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

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





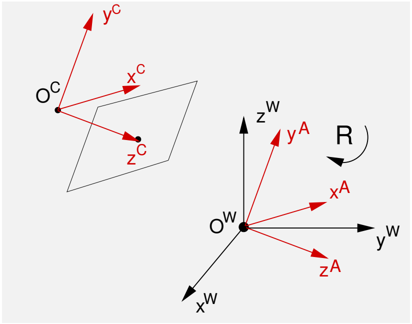
Building the Euclidean transformation in steps

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 ...

$$\boldsymbol{R}^{CW} = \boldsymbol{R}^{AW}$$



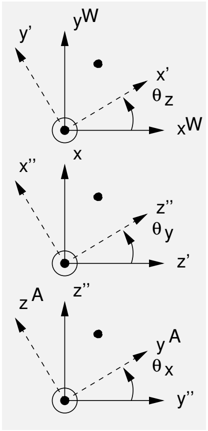
Building a Euclidean transformation in steps /ctd

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 ...

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

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

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



(You think about $\pm$, $\mp$ in the last!)

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

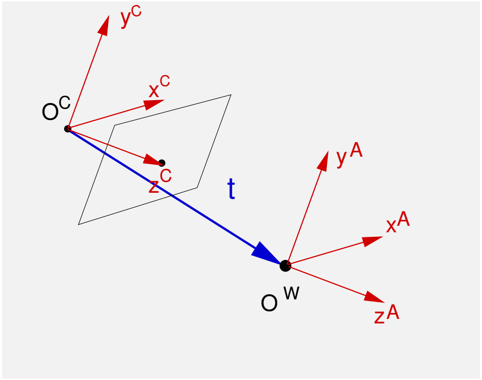
NB: rotation does not commute $\Rightarrow$order important

Build transformations in steps ...

Then deal with translation

$$
\begin{aligned}
\mathbf{X}^C &= \mathbf{X}^A + \mathbf{t}^C_A \\
&= \mathbf{X}^A + \mathbf{t}^C_W \\
&= R^{CW}\mathbf{X}^W + \mathbf{t}^C_W
\end{aligned}
$$

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



[**] Inverse of the 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

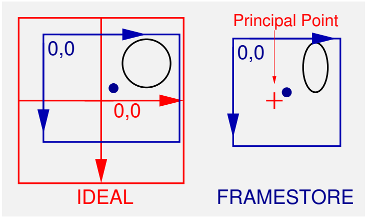
$$\boldsymbol{R}^{WC} = \left[\boldsymbol{R}^{CW}\right]^{-1} = \left[\boldsymbol{R}^{CW}\right]^\top$$

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

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

The Intrinsic Matrix

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.

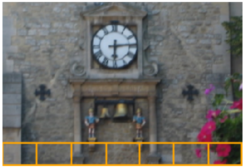


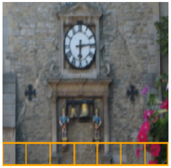
1. Use parameter $s \approx 0$ to account for axis skew.
2. Use an origin offset. (u_0, v_0) is the **principal point**, where where optic axis strikes the image plane. $(u_0/f, v_0/\gamma f$ is the offset before scaling)
3. Allow different scaling in x and y directions. γ is the aspect ratio.

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

How can the aspect ratio γ be non-unity?

There is usally a 1-to-1 mapping between nominal camera pixels 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.

$$\Rightarrow f_x < f_y \quad \Rightarrow f < \gamma f \quad \Rightarrow \gamma > 1 \quad \boxed{\Rightarrow \gamma = w/h}$$

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

Summary of steps from Scene to Image

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

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

Project into an ideal camera ($f = 1$, $\gamma = 1$, $s = 0$, $u_o = v_o = 0$) via a vanilla perspective transformation:

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

Map the ideal image into the real image using the intrinsic matrix:

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

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

1.4 Camera Calibration

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

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

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

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

Here we consider a method of **pre-calibration** using a specially made target.

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 $\boldsymbol{P}_{3\times 4}$.
It has only 11 dof (Why?)

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

Instead, we'll adopt a null-space method, solving (at first, anyway) a system $\boldsymbol{A}\mathbf{p} = \mathbf{0}$, where $\boldsymbol{A}$ is known, and $\mathbf{p}$ is the unknown.

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

Camera Calibration: Preamble

Suppose you know the image positions of 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 to give for point *i*

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

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

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 and perfect image data, $\mathbf{p}$ is the null-space of $\boldsymbol{A}$. That is, $\mathbf{p} = \text{null}(\boldsymbol{A})$.

If the 6 points are noisy, or if you have more points, a least-squares solution is found using SVD:

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

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

See later for an explanation of least squares & SVD.

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

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 $\boldsymbol{P}_{\text{SLEFT}}^{-1}$.

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

Explanation:

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

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

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

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

Step 3: QR decomposition

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

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

Explanation

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

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

There is no need to know how the QR-decomposition works

One of the tute sheet questions gets you to play with it in Matlab

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

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

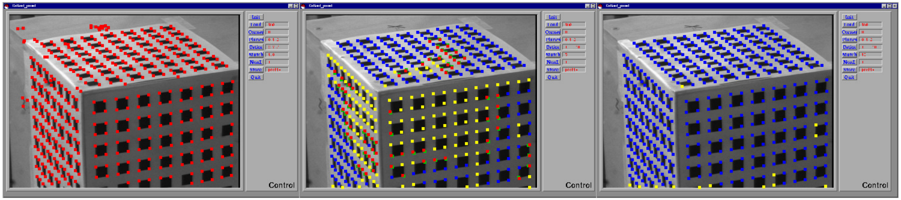
(Note that you could save some time in Steps 5 and 6 by *not* rescaling $\boldsymbol{P}$ and then computing $\mathbf{t} = \mathcal{R}[p_{14} \quad p_{24} \quad p_{34}]^\top$.)

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 Example



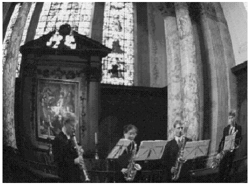
$$\boldsymbol{K} = \begin{bmatrix} 4622 & 0.1 & 267 \\ 0.0 & 4651 & 558 \\ 0.0 & 0.0 & 1 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} -0.03 & 0.34 & -0.94 \\ -0.92 & 0.36 & 0.16 \\ 0.39 & 0.87 & 0.30 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} 0.7 & 7.5 & 115.8 \end{bmatrix}^\top$$

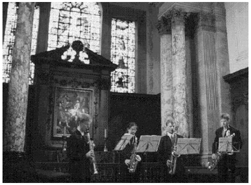
Radial distortion

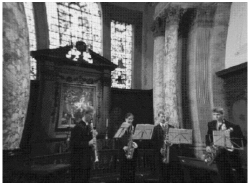
So far, we have figured out the transformations that turn our camera into a notional camera with the world and camera coordinates aligned and an "ideal" image plane of the sort given in the first diagram.

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

Corrections for this distortion is applied before carrying out calibration.


Barrel Distortion


Undistorted


Pincushion Distortion

Summary of Lecture 1

In the lecture we have

1. Introduced the aims of computer vision, and some paradigms (look on the web for videos)
2. Explored linear transformations and introduced homogeneous coordinates
3. Defined perspective projection from a scene, and saw that it could be made linear using homogeneous coordinates
4. Discussed how to pre-calibrate a camera using image of six or more known scene points.

Some further notes

There follow some further notes on

1. Least squares using Eigendecomposition or SVD. **(Essential reading before lecture 3.)**
2. Notes on how homogeneous coords handle points at infinity and vanishing points. **(Essential reading before lecture 3.)**
3. That there are other projection models. (For info only.)

A1: Least Squares, Eigendecomposition and SVD

Suppose you want to find the straight line $p_1 x + p_2 y + p_3 = 0$ from two known points $(x, y)_1$ and $(x, y)_2$ on it.
$p_{1,2,3}$ are the unknowns. You can write

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{or} \quad \boldsymbol{Ap} = \mathbf{0}$$

and solve for $\mathbf{p}$ up to scale as the vector spanning the $\boldsymbol{A}$'s null-space.
Example: points are $(2, 1)$ and $(1, 2)$. Find row-echelon-form

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & 3/2 & 1/2 \end{bmatrix} \quad \text{rank, nullity of} \boldsymbol{A} \text{ are } 2, 1$$

p_1 and p_2 are free variables and p_3 non-free.

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 3/2 & 1/2 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \ldots \text{ grind} \ldots \quad \Rightarrow \mathbf{p} = \alpha \begin{bmatrix} 1 \\ 1 \\ -3 \end{bmatrix}$$

and the line is $x + y - 3 = 0$ — obviously correct!

A1: Least Squares, Eigendecomposition and SVD

If you pile in more points exactly on the line the rank of matrix $\boldsymbol{A}$ does not increase.

```
A =   1      2      1
      2      1      1
      0      3      3
>> rank(A)
ans = 2

A =   1      2      1
      2      1      1
      0      3      1
    -99    102      1
>> rank(A)
ans = 2

>> p=null(A)
p =
    0.3015
    0.3015
   -0.9045
```

Matlab returns **p** with unit modulus.

A1: Least Squares, Eigendecomposition and SVD

Suppose you are trying to solve $\boldsymbol{A}\mathbf{p} = \mathbf{0}$ using more noisy data than there are are degrees of freedom in the system. Eg, three or more points lying approximately on a straight line.

A non-trivial approximate solution can be found using least-squares.

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

The sum of their squares is then $\mathbf{r}^\top\mathbf{r} = \mathbf{p}^\top[\boldsymbol{A}^\top\boldsymbol{A}]\mathbf{p}$.

The least squares solution, the minimum of $\mathbf{p}^\top[\boldsymbol{A}^\top\boldsymbol{A}]\mathbf{p}$, is found when

> $\mathbf{p}$ is the eigenvector of $[\boldsymbol{A}^\top\boldsymbol{A}]$ corresponding to the smallest eigenvalue.

A1: Proof

- $[A^\top A]$ is a $n \times n$, positive semi-definite, symmetric matrix, and hence has real and positive eigenvalues λ_i. Why?
  $[A^\top A]\hat{e}_i = \lambda_i \hat{e}_i \qquad \Rightarrow \hat{e}_i^\top A^\top A \hat{e}_i = \lambda_i \hat{e}_i^\top \hat{e}_i = \lambda_i \qquad \Rightarrow \lambda_i = [A\hat{e}]^\top [A\hat{e}] \geqslant 0$
- The **eigendecomposition** of $[A^\top A]$ is by definition

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

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

$$\mathbf{r}^\top \mathbf{r} = \mathbf{p}^\top [A^\top A]\mathbf{p} = \lambda_1 (\mathbf{p}^\top \hat{e}_1)^2 + \lambda_2 (\mathbf{p}^\top \hat{e}_2)^2 + \ldots + \lambda_n (\mathbf{p}^\top \hat{e}_n)^2$$

- We conclude that

$$\boxed{\mathbf{r}^\top \mathbf{r} = \mathbf{p}^\top [A^\top A]\mathbf{p} \text{ is minimized (and equal to } \lambda_n\text{) when } \mathbf{p} = \hat{e}_n.}$$

A1: 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

$$\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times m} \left[\operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)\right]_{m\times n} \boldsymbol{V}^\top_{n\times n}$$

where

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

The lsq soln for **p** occurs as the column of $\boldsymbol{V}$ associated with the smallest singular value in the SVD of $\boldsymbol{A}$. (Column n in our case.)

A1: Proof

- If $\boldsymbol{U\Sigma V}^\top \leftarrow \boldsymbol{A}$, then

$$[\boldsymbol{A}^\top \boldsymbol{A}] = \boldsymbol{V\Sigma}^\top \boldsymbol{U}^\top \boldsymbol{U\Sigma V}^\top = \boldsymbol{V\Sigma}^\top \boldsymbol{\Sigma V}^\top$$

- But $\boldsymbol{V}$ is fully orthogonal and $\boldsymbol{\Sigma}^\top\boldsymbol{\Sigma}$ is diagonal with elements σ_i^2
- Compare with Eigendecomposition ...

$$[\boldsymbol{A}^\top \boldsymbol{A}] = [\hat{\mathbf{e}}_1|\hat{\mathbf{e}}_2|\ldots|\hat{\mathbf{e}}_n] \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \begin{bmatrix} \hat{\mathbf{e}}_1^\top \\ \hat{\mathbf{e}}_2^\top \\ \vdots \\ \hat{\mathbf{e}}_n^\top \end{bmatrix}$$

- It is obvious that

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

SVD is usually preferred over eigendecomposition because it avoids the relatively expensive computation of $\boldsymbol{A}^\top\boldsymbol{A}$

A1: Least Squares, Eigendecomposition and SVD

SVD: 3 collinear pts

```
A=[1.0   2.0   1; ...
   2.0   1.0   1; ...
   1.5   1.5   1];

>> [u,s,v]=svd(A)
u=-0.5774   0.7071  -0.4082
  -0.5774  -0.7071  -0.4082
  -0.5774   0.0000   0.8165

s= 4.0620        0       0
        0   1.0000       0
        0        0  0.0000

v=-0.6396  -0.7071  -0.3015
  -0.6396   0.7071  -0.3015
  -0.4264        0   0.9045
```

SVD: 3 non-collinear

```
A=[1.0, 2.0, 1; ...
   2.0, 1.0, 1; ...
   1.5, 1.4, 1];

>> [u,s,v]=svd(A)
u=-0.5844   0.6810  -0.4412
  -0.5807  -0.7308  -0.3589
  -0.5669   0.0465   0.8225

s= 4.0257   0        0
   0        1.0016   0
   0        0        0.0248

v=  -0.6308  -0.7143  -0.3032
    -0.6458   0.6999  -0.3052
    -0.4302  -0.0033   0.9027
```

EigD: 3 non-collinear

```
A=[1.0, 2.0, 1; ...
   2.0, 1.0, 1; ...
   1.5, 1.4, 1];

>> [evec,eval]=eig(A'*A)
evec =
    0.3052     0.6999     0.6458
    0.3032    -0.7143     0.6308
   -0.9027    -0.0033     0.4302

eval =
    0.0006          0          0
         0     1.0033          0
         0          0    16.2061
% NB the eigenvectors & values
% are sorted the other way
```

A2: More advantages of homogeneous notation

There are several advantages of using homogeneous notation to represent perspective projection:

- Non-linear projections equations are turned into linear equations.
- The tools of linear algebra can be used.
- Awkward limiting procedures are avoided, especially when dealing with points at infinity.

Let's look at **3D points at infinity** and their projections into the image as **2D vanishing points**

A2: Vanishing points using homogeneous notation

A line in 3D through the point **A** with direction **D** is $\mathbf{X}(\mu) = \mathbf{A} + \mu\mathbf{D}$

Writing this in homogeneous notation

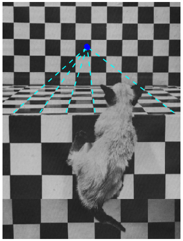
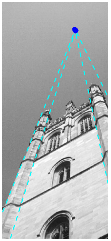
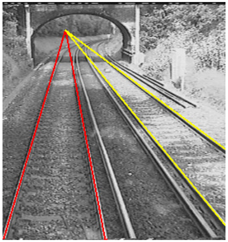
$$\begin{bmatrix} X_1(\mu) \\ X_2(\mu) \\ X_3(\mu) \\ X_4(\mu) \end{bmatrix} \overset{P}{=} \begin{bmatrix} \mathbf{A} \\ 1 \end{bmatrix} + \mu \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix} \overset{P}{=} \frac{1}{\mu} \begin{bmatrix} \mathbf{A} \\ 1 \end{bmatrix} + \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix}$$

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

So, homogeneous vectors with zero last entry represent points "at infinity".

Points at infinity are equivalent to directions

A2: Parallel 3D lines meet at a single 2D point

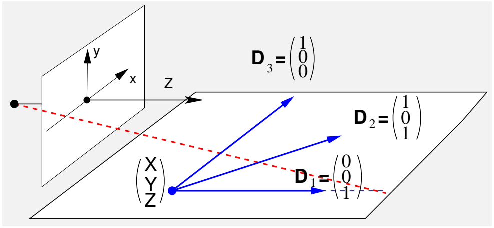
The special image point is call the **vanishing point**

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

$$\mathbf{v} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{D} \\ 0 \end{bmatrix} = \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix}$$

A2: Vanishing points using homogeneous notation

Exercise: Compute the vanishing points of lines on an *XZ* plane: (1) parallel to the *Z* axis; (2) at 45° to the *Z* axix; (3) parallel to the *X* axis.


y
x
Z
$\mathbf{D}_3 = \begin{pmatrix}1\\0\\0\end{pmatrix}$
$\mathbf{D}_2 = \begin{pmatrix}1\\0\\1\end{pmatrix}$
$\mathbf{D}_1 = \begin{pmatrix}0\\0\\1\end{pmatrix}$
$\begin{pmatrix}X\\Y\\Z\end{pmatrix}$

$$1: \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \rightarrow \begin{bmatrix}0\\0\\1\end{bmatrix}$$

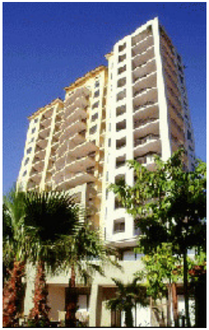
$$2: \begin{bmatrix}1\\0\\1\\0\end{bmatrix} \rightarrow \begin{bmatrix}1\\0\\1\end{bmatrix}$$

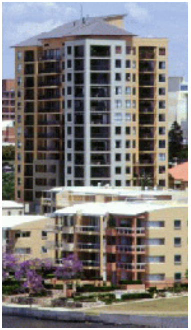
$$3: \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \rightarrow \begin{bmatrix}1\\0\\0\end{bmatrix}$$

A3: There are other projection models

Using a perspective projection model is only sensible when the depth variation across a viewed object is comparable or large compared with the distance to the object – ie when the image displays perspective distortion.

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.





A3: 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**.