.

Geometric Computing

in

Image Analysis and Visualization

Lecture Notes March 2007

Stefan Carlsson
Numerical Analysis and Computing Science
KTH, Stockholm, Sweden
stefanc@bion.kth.se

Contents

Preface

These lecture notes are intended for the course 2D1428: Geometric Computing and Visualisation (Datorgeometri och visualisering) They completely define define the reading part of the of the course . As the title says they are "notes" which means they are somewhat less than a textbook but something more than slides. The intention of the course is to familiarize the students with various computational problems of geometric nature. These occur frequently in image analysis and computer vision where one of the main problems is to understand the relation between the 3 dimensional scene and its projection to 2 dimensional images. A proper understanding of this will permit us to reconstruct 3D models of scenes and objects from images captured in single and multiple views. This field of reconstruction from multiple views has been a very active area of research the last decade. It can be considered as doing inverse computer graphics in the sense that we start with images and construct 3D models. The ultimate purpose however is exactly the same as that of 3D computer graphics, to generate new images of scenes and objects in space. The future of computer graphics and 3D visualization will undoubtly be influenced by these new methods of image based reconstruction and rendering.

We do not assume any background of image analysis or computer graphics. Since the course will deal with problems close to these fields however, it can very well be studied in parallel with the courses of computer vision and computer graphics.

The main intention of the course is to show how basic computational methods from numerical analysis and discrete mathematics can be applied to real problems of geometric nature. The mathematical background assumed are the basic courses in linear algebra and numerical analysis. We will on some occasions need to extend the mathematics beyond these basic courses but that will be dealt with in these lecture notes. Some of the chapters therefore have the character of "mathematical background"

Stockholm 2007-03-12

Stefan Carlsson

1 Cameras and projections

The image of an object depends on the viewpoint of the observer. This observer can be a human looking at the object or a camera capturing an image of it. The image formed on the human retina or in the the camera is the result of an optical process relating the 3 dimensional object to the 2 dimensional image. This process can be described very effectively in pure geometrical terms and is known as a *projection*

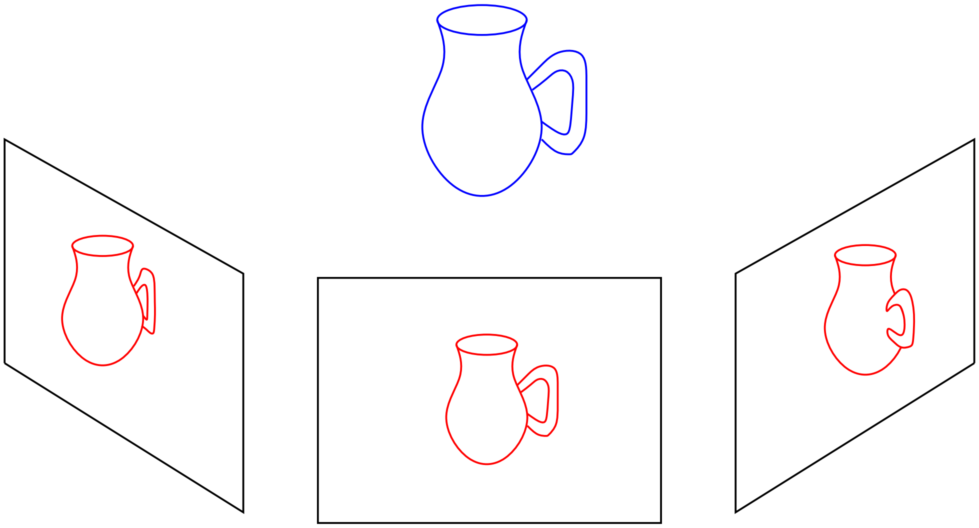

Figure 1: *The projection of a 3 dimensional object to multiple views*

1.1 Parallel projection

If we introduce a 2D coordinate system in the image, each image point can be assigned coordinates (x, y) in this system. The projection relation describes how a point in 3D with coordinates (X, Y, Z) project to these image coordinates. Consider first the case where the x and y axis of the image are parallel to the X and Y axis of the 3D coordinate system and the $Z-$axis of the 3D system goes through the origin of the image system (fig. (2). Any point in 3D projects to the image by a projection line through the point that is *parallel* to the z-axis. We therefore get:

$$\begin{array}{rcl} x & = & X \\ y & = & Y \end{array} \tag{1}$$

This relation is called *parallel projection*. We will later see how it can be extended to more general relative positions of the coordinate systems. The important defining property of parallel projection is the fact that all projection lines are parallel to each other.

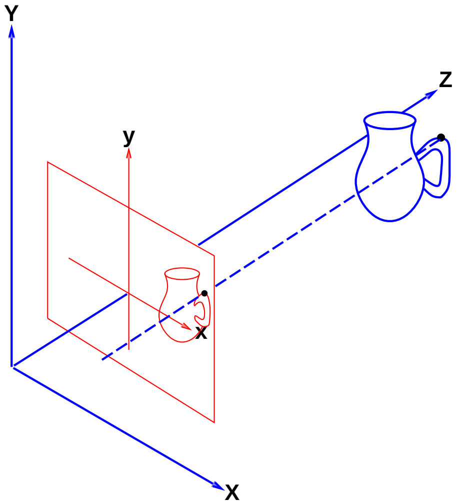

Figure 2: *Parallel projection of 3D coordinates* (X, Y, Z) *to image coordinates* (x, y)

1.2 Perspective projection

An important property of parallel lines in Euclidean geometry is the fact that they never intersect. They remain parallel all the way to infinity. By making all projection lines intersect in a common point however we obtain an alternative relation known as *perspective projection* Let all projection lines intersect in the origin $(0, 0, 0)$ of the 3D coordinate system, and assume that the image plane intersects the $Z-$axis at the point $(0, 0, 1)$ (fig. (3)). Simple relations between triangles then gives us the relation between 3D and image coordinates

$$\begin{aligned} x &= X/Z \\ y &= Y/Z \end{aligned} \tag{2}$$

Just as with the parallel projection case we will later extend this perspective projection relation to more general positions of the 3D coordinate system. In contrast to parallel projection, these relations are *non-linear* in the 3D coordinates. This means the analysis of perspective projection is in general far more complex than that of parallel projection. There are however ways to simplify the analysis of perspective cameras. We can note that the projection relation above is completely equivalent to the vector relation:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3}$$

Where an arbitrary unknown scale factor λ has been introduced. By dividing the components of the left side vectors in this equation, the scale factor is cancelled an eq. (2) results. At the price of introducing the extra scale factor, we now have a linear relation connecting image and 3D coordinates in perspective projection This is a simple example of the us of *homogeneous coordinates* which we will come back to in more detail later.

The intersection point of all projection lines is called the *center of projection*. If we move this along the negative Z axis of the 3D coordinate system, the projection lines will become more and more parallel and eventually in the limit when the projection center is at $-\infty$ we will end up with parallel projection. Parallel projection is therefore essentially

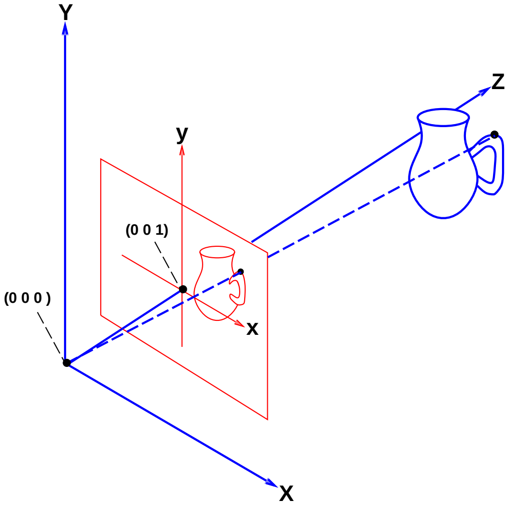

Figure 3: *Perspective projection of 3D coordinates* (X, Y, Z) *to image coordinates* (x, y)

just a special case of perspective projection with the projection center infinitely far away. Perspective projection represents the more general case of projection. It is also in general a better model for how projection takes place in physical imaging systems such as cameras and human eyes. In some cases however we will see that parallel projection is essentially just as good a model as perspective. In those cases we will exploit the relative mathematical simplicity of parallel projection.

1.3 Camera rotations - single axis

The simplicity of the relations (1) and (2) is partly due to the fact that the cameras were aligned with the 3D coordinate system in a particularly simple way. In general we would like to know the projection relations for arbitrary placement of cameras in the 3D system. This will allow us to analyze information from multiple cameras at the same time. Suppose that the perspective camera is rotated around the $Y-$axis. In this new rotated system, the image coordinates will be denoted (x', y') and the 3D coordinates will be (X', Y', Z'). The projection relations are now:

$$\begin{aligned} x' &= X'/Z' \\ y' &= Y'/Z' \end{aligned} \tag{4}$$

If the rotation angle around the $Y-$ axis is θ we get the coordinate transformation:

$$\begin{aligned} X' &= \quad cos(\theta)X \quad + \quad sin(\theta)Z \\ Y' &= \qquad Y \\ Z' &= -sin(\theta)X \quad + \quad cos(\theta)Z \end{aligned} \tag{5}$$

We can write this in a more compact way using a 3×3 matrix for the rotation:

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{6}$$

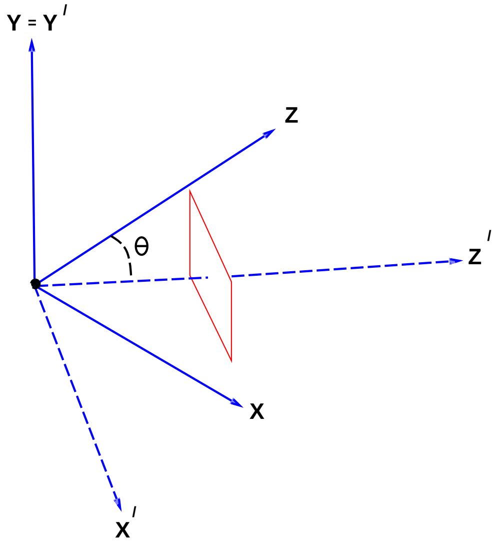

Figure 4: *Rotation of camera angle θ around $Y-$ axis*

If we write the projection relation (4) using the homogeneous coordinate representation of eq. (3):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} \tag{7}$$

we can combine this with eq. (8) and we get:

Using eq. (7) we can write:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{8}$$

This equation tells us how the image coordinates (x, y) of a 3D point (X, Y, Z) varies as the camera with projection center at the origin is rotated the angle θ around the $Y-$ axis.

Using homogeneous image coordinates for both the original and rotated camera positions we get:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' \lambda^{-1} \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{9}$$

This equation relates the image coordinates of a 3D point in two different cameras that are rotated an angle θ relative to each other. Relations like this will be very useful since they involve observable quantities like the image coordinates. A very important property is the fact that this is a *linear* transformation between the two images when the image coordinates are expressed as *homogeneous coordinates* It demonstrates the usefulness of representing coordinates this way. Note however that we achieved this simplification by introducing the unknown scale factors λ, λ'. The advantage of homogeneous coordinates therefore lies in the simplification of deriving relations like these.

1.4 Building simple image mosaics

The knowledge of how two images from a rotated camera are related will permit us to do some simple applications based on putting images together to generate large wide field of view images. Unless special lenses are used, the field of view of a camera is often limited to around 30 - 40 degrees. Since wide field lenses tend to give distorted images it is interesting to investigate alternative ways of building wide field view images using inputs from multiple un distorted ordinary field of view cameras. This technique is known as building image mosaics and we will give a simple example here.

1.4.1 Estimating camera rotation

Fig. 5 shows a top view of two cameras A and B rotated relative to each other around the vertical axis through the projection point. (As was analysed in the previous section). This means that a certain set of 3D points will be visible in camera A only, another set in camera B only and a third set of points will be visible in both cameras. The idea of image mosaics is to map the pixels visible only in camera B to an extended image plane of camera A. In order to do this we have to know how the image coordinates of a point seen in both camera A and B are related. But this is exactly what was derived in the previous section, in the relation of eq. (9) which maps coordinates of points in image A to those of image B. By inverting this relation we get:

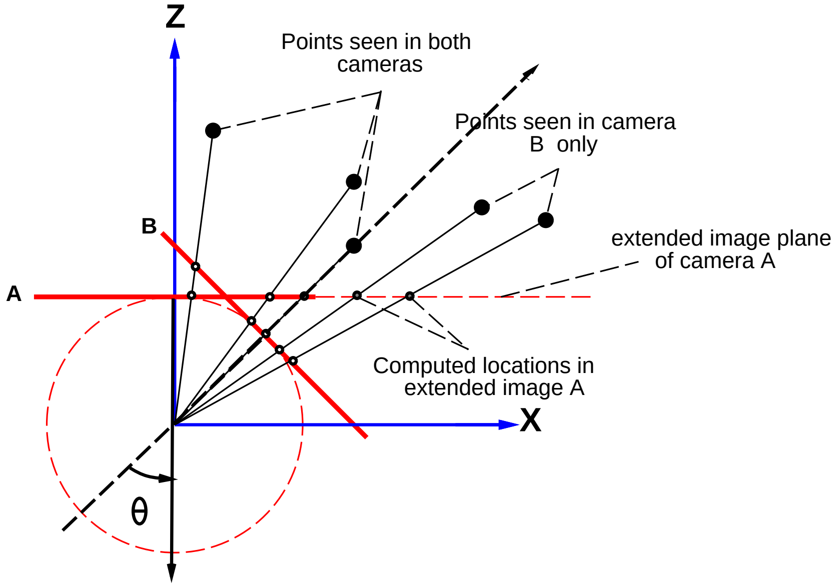
Z
Points seen in both cameras
Points seen in camera B only
B
A
extended image plane of camera A
Computed locations in extended image A
X
θ

Figure 5: *Top view of cameras A and B rotated around vertical axis through common projection point in the origin. If the relative rotation between the cameras is known, we can compute extended image coordinates in camera A for points seen only in camera B*

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \sigma \begin{pmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \qquad (10)$$

Where we have replaced the two scale factors with a common factor σ. In order to use this relation we either have to find the unknown parameters σ and θ or eliminate them from the calculations. We can do a combination of both in by eliminating σ and computing θ using the points that are visible in both camera A and B.

We first eliminate σ by division an we get:

$$\begin{aligned} x^a &= \frac{cos(\theta)\ x^b - sin(\theta)}{sin(\theta)\ x^b + cos(\theta)} \\ y^a &= \frac{y^b}{sin(\theta)\ x^b + cos(\theta)} \end{aligned} \tag{11}$$

These two equations give us two relations for the angle θ

$$\begin{aligned} sin(\theta)\ (x^b\ x^a\ +\ 1) \quad &+ \quad cos(\theta)\ (x^a\ -\ x^b) \quad & &= \quad 0 \\ sin(\theta)\ x^b\ y^a \quad &+ \quad cos(\theta)\ y^a \quad & -y^b &= \quad 0 \end{aligned} \tag{12}$$

1.4.2 Least squares parameter estimation

Every point seen in both cameras gives us two equations for determination of the unknown camera rotation θ. It will always be advantageous from a computational point of view to use more points than necessary and get an over determined system of linear equations for $sin(\theta)$ and $cos(\theta)$. Most computational problems in this field will lead to systems of over determined linear equations like this. We will later look at this problem from a more general point of view and study methods of its solution. For the simple problem of finding $sin(\theta)$ and $cos(\theta)$ above we can note that it is mathematically very similar to fitting a straight line through a set of n points(*see the boxed text*). In order to simplify the computations we make the substitutions:

$$\begin{aligned} &s\ =\ sin(\theta) & &c\ =\ cos(\theta) & & \\ &a_i\ =\ x^b_i\ x^a_i\ +\ 1 & &b_i\ =\ x^a_i\ -\ x^b_i & &e_i\ =\ 0 \\ &c_i\ =\ x^b\ y^a_i & &d_i\ =\ y^a_i & &f_i\ =\ -y^b_i \end{aligned} \tag{13}$$

Eq. 12 can then be written as:

$$\begin{aligned} sa_i + \quad cb_i + \quad e_i \ &=\ 0 \\ sc_i + \quad cd_i + \quad f_i \ &=\ 0 \end{aligned} \tag{14}$$

The problem of finding $sin(\theta)$ and $cos(\theta)$ can be formulated as the problem of finding values of s and c that fulfill the above equations “as well as possible”. Since there are more equations than unknowns and the measurements are noisy we cannot expect the find values of s and c that satisfy the equations exactly. In general we have to seek a best solution in the sense that we minimise the squared sum of the error of the equations:

$$\min_{s,c} \sum_{i=1}^{i=n} (sa_i\ +\ cb_i\ +\ e_i)^2\ +\ (sc_i\ +\ cd_i\ +\ f_i)^2$$

A least squares estimate of s, c can be found by computing the derivatives and setting them to zero:

$$\frac{\partial C(s,c)}{\partial s}\ =\ \sum_{i=1}^{i=n} (sa_i\ +\ cb_i\ +\ e_i)a_i\ +\ (sc_i\ +\ cd_i\ +\ f_i)c_i\ =\ 0$$

$$\frac{\partial C(s,c)}{\partial c} \; = \; \sum_{i=1}^{i=n} (sa_i \; + \; cb_i \; + \; e_i) b_i \; + \; (sc_i \; + \; cd_i \; + \; f_i) d_i \; = \; 0$$

This leads to the *linear* system of equations:

$$\begin{aligned}
s \sum_{i=1}^{i=n} (a_i a_i \; + \; c_i c_i) \; + \; c \sum_{i=1}^{i=n} (b_i a_i \; + \; d_i c_i) \; + \; \sum_{i=1}^{i=n} e_i a_i \; + \; f_i c_i = \; 0 \\
s \sum_{i=1}^{i=n} (a_i b_i \; + \; c_i d_i) \; + \; c \sum_{i=1}^{i=n} (b_i b_i \; + \; d_i d_i) \; + \; \sum_{i=1}^{i=n} e_i b_i \; + \; f_i d_i = \; 0
\end{aligned}$$

Which is easily solved to give:

$$- \begin{pmatrix} s \\ \\ c \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{i=n} b_i b_i + d_i d_i & -\sum_{i=1}^{i=n} b_i a_i + d_i c_i \\ -\sum_{i=1}^{i=n} a_i b_i + c_i d_i & \sum_{i=1}^{i=n} a_i a_i + c_i c_i \end{pmatrix} \begin{pmatrix} \sum_{i=1}^{i=n} e_i a_i + f_i c_i \\ \sum_{i=1}^{i=n} e_i b_i + f_i d_i \end{pmatrix}$$

It should be noted here that by treating s and c as independent variables we have not used all the constraints available in the optimisation. Since $s \; = \; sin(\theta)$ and $c \; = \; cos(\theta)$ and $sin^(\theta) \; + \; cos^2(\theta) \; = \; 1$ we have an extra constraint that should be satisfied by the solution to the least squares minimisation. The formally correct way to do this is to pose the problem as a constrained minimisation problem. In the problems we will encounter we will frequently have extra non-linear constraints like this. Very often we will see that we can ignore and use a linear algorithm based on the quadratic cost function only.

1.4.3 Cylindrical backprojection

The extended planar image represents the image of a camera with arbitrary large field of view. This gives in general a somewhat distorted view when it is looked at directly since the viewers position does not coincide exactly with that of the camera. A more natural large image is obtained if the extended image is backprojected to a cylinder, fig. 6. The x-coordinate transformation to the cylindrical coordinate θ is then simply

$$tan(\theta) \; = \frac{x}{f} \tag{15}$$

where f is the focal length of the camera

For more general panoramas involving general rotations, the cylindrical backprojection is replaced by reprojection to a sphere.

1.5 External and internal camera parameters

The camera models in eq. (1) and (2) are very idealized in the sense that we assume that the camera is centered at the origin of the 3D coordinate system with the image plane at unit distance from the projection point and parallel to the $X - Y$ plane. The measure of distance in the image was assumed to be the same as that in the 3D system. In eq (8) we have extended this simple model somewhat to include rotation around the $Y-$ axis.

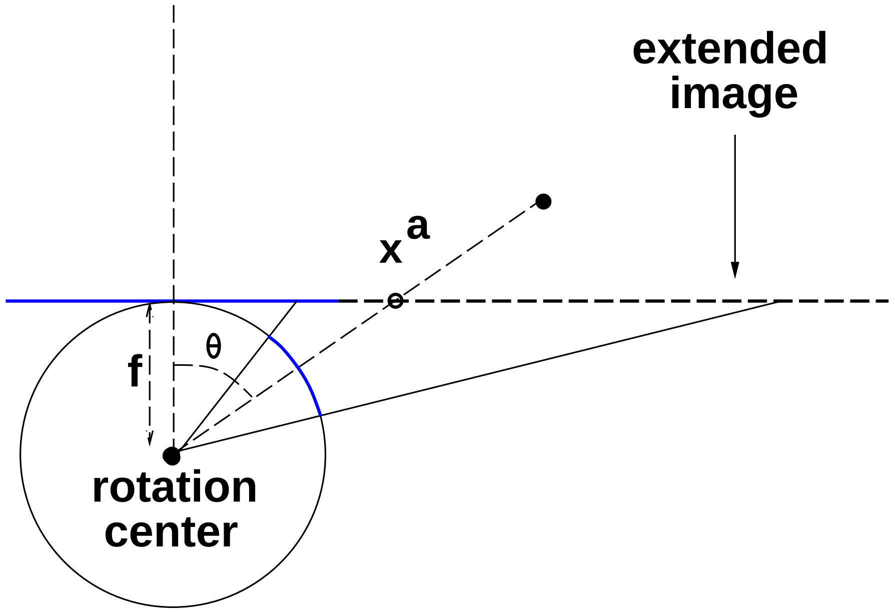

Figure 6: *Cylindrical backprojection of extended image*

In this section we will extend the camera models even further and include a complete description of the position and orientation in 3D , known as *external* parameters, as well as a description of scale factors and other parameters affecting the projection from 3D to the image known as *internal parameters*

1.6 General camera rotation

The external camera parameters are the rotation of the camera and the position or translation of the projection center. We will treat these parameters separately, starting with camera rotation and later coming back to the translation of the projection center.

Just as the camera rotation θ around the $Y-$axis through the projection center $(0,0,0)$ can be represented by a 3×3 matrix eq.(6) so can rotations around the $X-$ and $Z-$ axis. If these rotations are ϕ and ψ respectively we get the three rotation matrices:

$$R_X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{pmatrix} R_Y = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} R_Z = \begin{pmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Any rotation in 3D can be decomposed into simple rotations like these. A general rotation can therefore be expressed as a 3×3 matrix:

$$R \;=\; R_X R_Y R_Z \tag{16}$$

Generalising eq (8) The projection from 3D to image coordinates can then be written as:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \;=\; \lambda' \; R \; \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{17}$$

MATH 1: Least squares straight line estimation

Given a set of points x_i, y_i $i = 1 \ldots n$ we want to estimate parameters a and b for a straight line approximating the points as well as possible. Ideally if all points were collinear we would be able to write:

$$ax_i \; + \; by_i + \; 1 \; = \; 0 \qquad i \; = 1 \ldots \; n \tag{18}$$

If the points are noisy, we cannot in general find parameters$a$ and b that satisfy them all. We therefore seek a least squares solution:

$$\min_{a,b} \sum_{i=1}^{i=n} (ax_i \; + \; by_i + \; 1)^2 \tag{19}$$

The important thing about these problems is the fact that the cost functions are *quadratic* in the parameters a, b. Since they are positive semidefinite (always ≥ 0), we can actually find a global minimum by finding the derivatives of the cost w.r.t. the parameters and setting them to zero.
If we do this we obtain:

$$\begin{aligned}
\frac{\partial C(a,b)}{\partial a} &= \sum_{i=1}^{i=n} (ax_i \; + \; by_i + \; 1)x_i \; = \; 0 \\
\frac{\partial C(a,b)}{\partial b} &= \sum_{i=1}^{i=n} (ax_i \; + \; by_i + \; 1)y_i \; = \; 0
\end{aligned} \tag{20}$$

By factoring out a and b from these expressions we get:

$$\begin{aligned}
a\sum_{i=1}^{i=n} x_i^2 \quad &+ \quad b\sum_{i=1}^{i=n} x_i y_i \quad + \quad \sum_{i=1}^{i=n} x_i \quad = 0 \\
a\sum_{i=1}^{i=n} x_i y_i \quad &+ \quad b\sum_{i=1}^{i=n} y_i^2 \quad + \quad \sum_{i=1}^{i=n} y_i \quad = 0
\end{aligned} \tag{21}$$

This is a simple 2×2 system of equations in the two unknowns a and b. We can write the solution as:

$$\begin{pmatrix} a \\ b \end{pmatrix} = - \begin{pmatrix} \sum_{i=1}^{i=n} x_i^2 & \sum_{i=1}^{i=n} x_i \; y_i \\ \sum_{i=1}^{i=n} x_i \; y_i & \sum_{i=1}^{i=n} y_i^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^{i=n} x_i \\ \sum_{i=1}^{i=n} y_i \end{pmatrix} \tag{22}$$

For higher order systems involving more than two unknowns we will in general not be able to compute explicit solutions in this way but will have to use specific numerical methods for solving linear systems of equations.

MATH 2: Rotation matrices
It is easy to verify that the inverse of any rotation matrix e.g. R_X can be written:

$$R_X^{-1} \;=\; \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} \tag{23}$$

This means that $R_X^{-1} \;=\; R_X^T$, the transpose of R_X. Similarly for R_Y^{-1} and R_Z^{-1}. Using the relations:

$$(AB)^T \;=\; B^T\; A^T \qquad\qquad (AB)^{-1} \;=\; B^{-1}\; A^{-1} \tag{24}$$

which are valid for arbitrary matrices A and B we get:

$$R^T \;=\; (R_X R_Y R_Z)^T \;=\; R_Z^T\; (R_X R_Y)^T \;=\; R_Z^T\; R_Y^T\; R_X^T \;=\; R_Z^{-1}\; R_Y^{-1}\; R_X^{-1} \;=$$

$$=\; R_Z^{-1}\; (R_X R_Y)^{-1} \;=\; (R_X R_Y R_Z)^{-1} \;=\; R^{-1} \tag{25}$$

This means that

$$R\; R^T \;=\; I \tag{26}$$

which is the property defining *orthogonal* matrices. It means that the three rows r_1, r_2, r_3 of R are orthogonal i.e $r_i^T r_j \;=\; 0$ for $i \neq j$ and $r_i^T r_i \;=\; 1$ for all i
We will later see that this property of general rotation matrices will be very useful.

1.7 Internal parameters

The expression eq.(2) relating 3D and image coordinates for perspective projection is based on a simplified model of a camera. Any perspective projection camera will have an *optical axis* which is the line passing through the *projection* center orthogonal to the image plane. The optical axis intersects the image plane in a point called the *principal point.* The distance between the image plane and the projection center is known as the *focal length.* Compared to the simplified model of eq.(2) a general camera can have the following properties:

- Image coordinates will be measured in pixels. In order to relate these to distance of the 3D coordinate system we need scale factors σ_x and σ_y.

- The principal point does not necessarily coincide with the origin of the image coordinate system. We will denote it's image coordinates with x_0, y_0

- The focal length f can have a wide range of values. The zooming of a camera is controlled by varying the focal length.

- Although not very common in practise , the x and y axis of the image coordinate system may not be orthogonal. We will introduce a *skew factor* γ to account for this

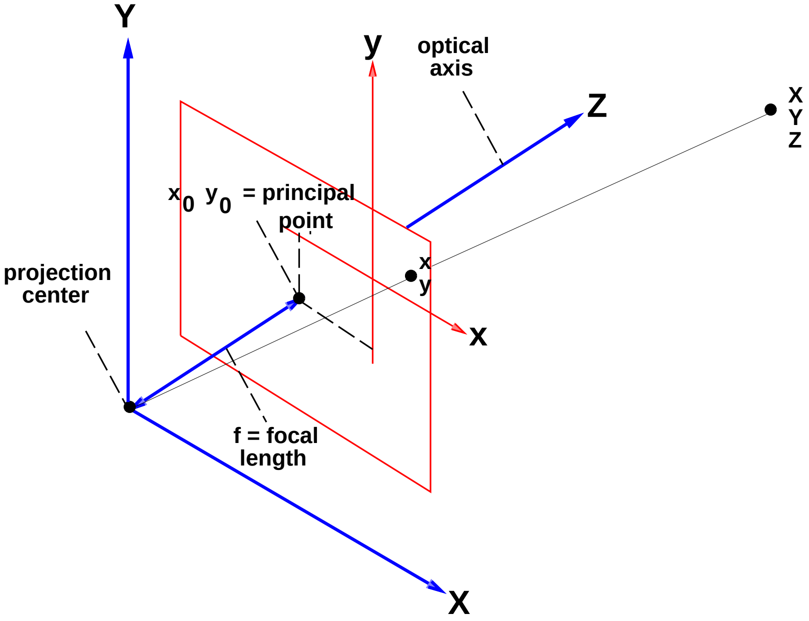

Figure 7: *Perspective projection camera*

Rescaling using the scale factors σ_x, σ_y. for the image $x-$ and $y-$ axis, the perspective projection can be written:

$$\begin{aligned} \frac{x - x_0}{\sigma_x f} &= X/Z \\ \frac{y - y_0}{\sigma_y f} &= Y/Z \end{aligned} \tag{27}$$

Using homogeneous image coordinates we can write this as :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} \sigma_x f & \gamma & x_0 \\ 0 & \sigma_y f & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{28}$$

If the skew factor $\gamma = 0$, the image coordinate $x-$ and $y-$ axis will be parallel to the 3D coordinate $X-$ and $Y-$ axis respectively.

The 3×3 matrix containing the internal camera parameters will be denoted K for short. Using eq. (17) we can write the projection relation for a perspective camera with rotation R relative the 3D reference system:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' \; K \; R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{29}$$

We can now generalise the image coordinate transformation between two cameras, with internal parameters K_a and K_b and rotations R_a and R_b respectively. The geometry of this is illustrated in fig. 8

The projection relations are:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \lambda_a \; K_a \; R_a \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \qquad \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} = \lambda_b K_b \; R_b \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{30}$$

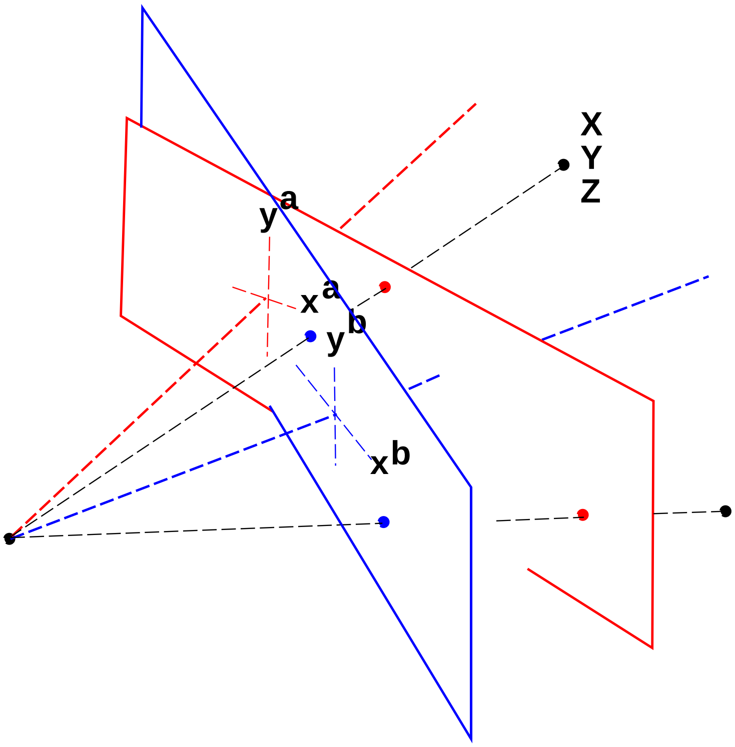

Figure 8: *Two cameras rotated around the projection center*

from which we get after simple manipulation:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \lambda_a {\lambda_b}^{-1} \; K_a \; R_a \; (K_b \; R_b)^{-1} \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \tag{31}$$

Using the relations of eq. (24) we can write this as:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \lambda_a {\lambda_b}^{-1} \; K_a \; R_a \; R_b^{-1} K_b^{-1} \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \tag{32}$$

This is a very central result about rotating cameras. No matter what the internal camera parameters are, the homogeneous image coordinates of corresponding points in two rotated cameras will always be related by a *linear* transformation represented by a 3 × 3 matrix.

$$\boxed{H = K_a \; R_a \; R_b^{-1} K_b^{-1} \; (33)}$$

Since there is an arbitrary scale factor in the relation between the cameras, this matrix can be described by 8 parameters = (number of matrix elements -1). For many applications e.g building image mosaics, it is this matrix that contains the interesting information. Most importantly: even if we don't know the internal parameters, i.e the internal calibration of the cameras, we can estimate this transformation. Each point visible in the two cameras gives two equations for determination of the elements of H. Since H is described by 8 elements, (an arbitrary element can be set to 1), we need at least four corresponding points for this computation. This is an example of a problem that can be solved without actually calibrating the cameras involved. Later we will see more examples of this

1.8 Estimation of linear mappings

We are now in a position to generalize the problem of building image mosaics from multiple rotated uncalibrated cameras. The simple case of rotating a calibrated camera around one axis led to the problem of estimating rotation from eq. (10). The case of general relative rotation of two cameras with unknown internal parameters leads to the problem of estimating the 3×3- matrix H from corresponding points in th two images:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \sim H \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \tag{34}$$

Where we have written $\sim$ instead of $=$ to indicate that we should have an arbitrary scale factor on the right side. From now on we will use this notation $\sim$ to indicate "similarity up to scale"

If we write out the elements of H we have:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \sim \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \tag{35}$$

We can cancel the arbitrary scale factor by division:

$$\begin{aligned} x^a &= \frac{h_{11}x^b + h_{12}y^b + h_{13}}{h_{31}x^b + h_{32}y^b + h_{33}} \\ y^a &= \frac{h_{21}x^b + h_{22}y^b + h_{23}}{h_{31}x^b + h_{32}y^b + h_{33}} \end{aligned} \tag{36}$$

By simple manipulation we can write these equations as:

$$\begin{aligned} h_{11}x^b + h_{12}y^b + h_{13} - h_{31}x^b x^a - h_{32}y^b x^a - h_{33}x^a &= 0 \\ h_{21}x^b + h_{22}y^b + h_{23} - h_{31}x^b y^a - h_{32}y^b y^a - h_{33}y^a &= 0 \end{aligned} \tag{37}$$

We see that each corresponding point gives two *linear* homogeneous equations for determining the nine unknown parameters $h_{11} \ldots h_{33}$ of the linear mapping matrix H. Since this matrix is only defined up to an arbitrary scale factor we actually only have 8 parameters. If we divide each term in the linear equations by the element h_{33} we are left with only eight unknowns since the element h_{33} is cancelled. A homogeneous linear equation can always be normalised this way, reducing the number of unknowns with one. However, there are other ways to handle this normalisation that does not treat any element such as h_{33} as special. If we define the vectors:

$$\begin{aligned} \mathbf{h} &= (h_{11},\ h_{12},\ h_{13},\ h_{21},\ h_{22},\ h_{23},\ h_{31},\ h_{32},\ h_{33})^T \\ \alpha_i &= (x_i^b,\ y_i^b,\ 1,\ 0,\ 0,\ 0,\ -x_i^b x_i^a,\ -y_i^b x_i^a,\ -x_i^a)^T \\ \beta_i &= (0,\ 0,\ 0,\ x_i^b,\ y_i^b,\ 1,\ -x_i^b y_i^a,\ -y_i^b y_i^a,\ -y_i^a)^T \end{aligned} \tag{38}$$

we can write the constraints eq. (37) simply as:

$$\begin{aligned} {\alpha_i}^T \mathbf{h} &= 0 \\ {\beta_i}^T \mathbf{h} &= 0 \end{aligned} \tag{39}$$

Using four corresponding points would give us a total of eight linear constraints for computing **h**. In general we would like to use more points than necessary in order to get a more robust estimate, since the points will in general be affected by measurement noise. Just as in the single axis rotation case we can seek a least squares estimate of **h** by:

$$\min_{\mathbf{h^T h} = 1} \quad \sum_{i=1}^{i=n} ({\alpha_i}^T \mathbf{h})^2 \; + \; ({\beta_i}^T \mathbf{h})^2 \tag{40}$$

where we have normalised **h** by adding the constraint $\mathbf{h^T h} = 1$. Without this constraint the solution for **h** would be the trivial all zero vector.

If we define the matrix Q:

$$Q \;=\; \begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{19} \\ \alpha_{21} & \alpha_{22} & \dots & \alpha_{29} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \dots & \alpha_{n9} \\ \beta_{11} & \beta_{12} & \dots & \beta_{19} \\ \beta_{21} & \beta_{22} & \dots & \beta_{29} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{n1} & \beta_{n2} & \dots & \beta_{n9} \end{pmatrix} \tag{41}$$

the constraint 39 can be written as

$$Q\mathbf{h} \; = 0 \tag{42}$$

and the cost function 40 as:

$$\min_{\mathbf{h^T h} = 1} \quad ||Q\mathbf{h}||^2 \tag{43}$$

which can be expanded as:

$$\min_{\mathbf{h^T h} = 1} \quad \mathbf{h}^T Q^T Q \mathbf{h} \tag{44}$$

This is a minimisation problem with constraints on the variables over which we minimise. We can therefore construct the Lagrange function:

$$\min_{\mathbf{h}} \mathbf{h}^T Q^T Q \mathbf{h} \; - \; \lambda(\mathbf{h^T h} \; - 1) \tag{45}$$

and minimise over this. Since the cost function is quadratic and semidefinite we get the the necessary conditions for a minimum by computing first derivatives w.r.t the components of **h** and setting to zero. This results in:

$$Q^T Q \mathbf{h} \; - \; \lambda \mathbf{h} \; = \; 0 \tag{46}$$

Which is just the eigenvalue problem:

$$Q^T Q \mathbf{h} \; = \; \lambda \mathbf{h} \tag{47}$$

An $n \times n$ matrix has in general n distinct eigenvalues λ_i and eigenvectors. For a specific eigenvector $\mathbf{h}_i$ normalised to unit length, the cost function (50) will be:

$$\mathbf{h}_i Q^T Q \mathbf{h}_i \;=\; \lambda_i \tag{48}$$

Minimal cost is therefore realised by the eigenvector with minimum eigenvalue. In the ideal noise free case, this value will be 0 but in practise the minimum eigenvalue will always be > 0

The procedure of computing the linear mapping (35) relating image coordinates in two rotated cameras a and b with unknown internal parameters can be summarized as:

1. Select a minimum of four corresponding points $(x_i^a, y_i^a) < -- > (x_i^b, y_i^b) \quad i = 1 \ldots n$ in the two images a and b
2. For each corresponding point i, compute the two 9-dimensional vectors α_i, β_i eq. (38)
3. With n corresponding points selected, build the $2n \times 9$ matrix Q (41) from the vectors α_i, β_i
4. Compute the eigenvectors and eigenvalues of the matrix $Q^T Q$
5. Select the eigenvector $\mathbf{h}_0$ with minimum eigenvalue. The nine elements of this vector are the nine elements $h_{11} \ldots h_{33}$ of the linear mapping relating image coordinates in the two rotated cameras a and b

Numerical software for doing this uses what is known as the Singular value decomposition (SVD) of the matrix Q in order to botain this eigenvector. The SVD is described in more detail in the chapter on numerical computations.

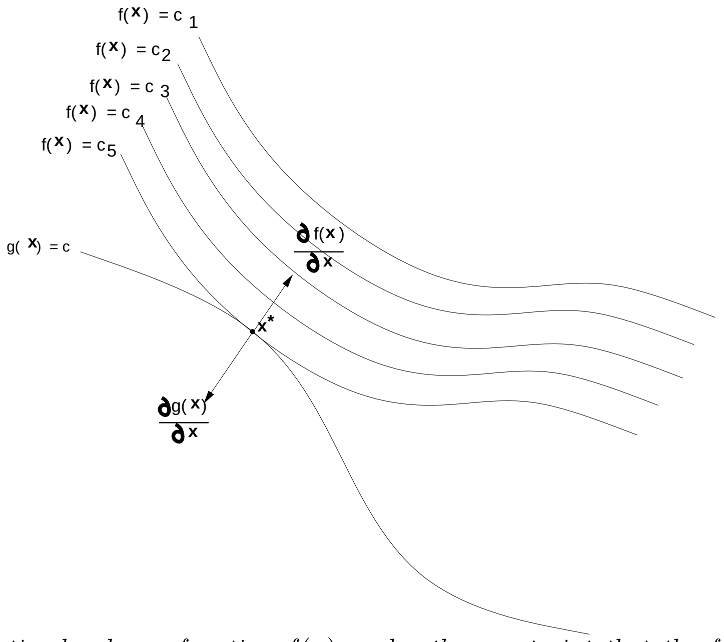

Figure 9: *The optimal value a function $f(\mathbf{x})$ under the constraint that the function $g(\mathbf{x}) = c$ is attained at a point where a level curve of f is tangent to the curve $g(\mathbf{x}) = c$. This means that the normals of these curves are parallell.*

MATH : Constrained optimisation

We will frequently encounter constrained optimisation problems of the kind:

$$\min_{g(\mathbf{x}) = c} \quad f(\mathbf{x}) \tag{49}$$

Consider the point $\mathbf{x}^*$ where the tangent to the curve $g(\mathbf{x}) = c$ is parallel to the tangent of one of the level curves of $f(\mathbf{x})$. As we move away from this point, but stay on the constraint curve $g(\mathbf{x}) = c$, the value of $f(\mathbf{x})$ will increase or decrease no matter which of the two directions we move in. The point $\mathbf{x}^*$ is therefore a local extrema to our optimisation problem. The fact that the tangents at $\mathbf{x}^*$ are parallel also means that the normals of the two curves are parallel, i.e. have the same direction, This can be expressed that one normal is simply a multiple of the other:

$$(\frac{\partial g(\mathbf{x})}{\partial x_1} \dots \frac{\partial g(\mathbf{x})}{\partial x_n}) \; = \; \lambda(\frac{\partial f(\mathbf{x})}{\partial x_1} \dots \frac{\partial f(\mathbf{x})}{\partial x_n}) \tag{50}$$

which can be formulated as that the *Lagrange function* $\lambda f(\mathbf{x}) - g(\mathbf{x})$ should have a local extrema, i.e the first order variation in each component x_i vanishes:

$$\frac{\partial g(\mathbf{x})}{\partial x_i} - \; \lambda \frac{\partial f(\mathbf{x})}{\partial x_i} \; = 0 \qquad i = 1 \dots n \tag{51}$$

1.9 General camera translation

The relation between the coordinates of a point in 3D and its image coordinates for a perspective projection camera with rotation R around the origin and internal parameters K was according to eq. (29)

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \; = \; \lambda \; K \; R \; \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{52}$$

The camera projection center in this relation coincides with the origin of the 3D system. In general we would like to have arbitrary positions X_0, Y_0, Z_0 of the camera center. This means that we simply translate the camera and we get the projection relation:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \; = \; \lambda \; K \; R \; \begin{pmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{pmatrix} \tag{53}$$

This is the most general expression for a perspective camera that we can derive. It contains the *internal parameters* K and *external parameters* R and X_0, Y_0, Z_0. Note that if we have two cameras with *different* camera centers, we will no longer have the simple linear relation (35) relating corresponding image coordinates. There is still a relation between camera parameters and corresponding image coordinates but as we will see later it is more complex.

We will see that just as with the image coordinates introducing homogeneous coordinates in 3D will have advantages in the analysis of multiple cameras. Eq. (53) can be written:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \; = \; \lambda \; K \; R \; \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{54}$$

This 3×4 matrix relating 3D and image coordinates will be denoted the *projection matrix*. This matrix contains all information about the camera, internal and external. Much of the problem of 3D reconstruction from image data is centered around the determination of this matrix. We will sometimes have reason to write the projection relation in a more general way using $\mathbf{M}$ for the projection matrix and the relation $\sim$ to indicate that equality is up to an arbitrary scale factor.

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \sim \mathbf{M} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{55}$$

The matrix M contains 12 elements. Since it is homogeneous, it can always be normalised to 11 parameters. If we look at the form 54 of the projection matrix, it contains the components:

- Internal parameters: K 5 parameters
- Rotation: R 3 parameters
- Projection center: X_0, Y_0, Z_0 3 parameters

which together also make up 11 parameters. We will see that in general we can compute unique values of these components from the elements $m_{i,j}$ of the projection matrix M

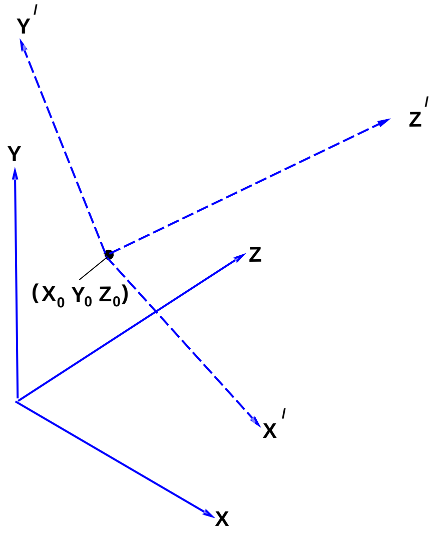

Figure 10: *Translated and rotated coordinate systems*

MATH : Least squares minimisation - Non-homogeneous equations

The problem of finding a solution of an over determined set of linear equations depends on whether the set of equations is homogeneous or not. If a set of equations for a vector $\mathbf{x}$ is *homogeneous*, any vector $\lambda\mathbf{x}$ is a solution where λ is an arbitrary real number.
The least squares fitting of straight lines that was treated in the previous section is an example of a non homogeneous problem. Suppose we are looking for a n-dimensional vector $\mathbf{x}$ which is the solution to the over determined system:

$$A\mathbf{x} \ = \ b \tag{56}$$

where A is an $m \times n$ matrix $m \geq n$ and b is an m-dimensional vector. Due to noise in the matrix A and vector b we will in general not be able to find a solution that satisfies all equations. We therefore look for the least squares solution to:

$$\min_{\mathbf{x}} \ ||A\mathbf{x} \ - \ b||^2 \tag{57}$$

Where $|| \dots ||$ denotes the Euclidean norm of a vector.
We can write this as:

$$\min_{\mathbf{x}} \ (A\mathbf{x} \ - \ b)^T \ (A\mathbf{x} \ - \ b) \tag{58}$$

The cost function is always ≥ 0 so the necessary conditions for a minimum is that all partial derivative w.r.t. the components $x_1 \dots x_n$ of $\mathbf{x}$ vanish, i.e:

$$\frac{\partial (A\mathbf{x} \ - \ b)^T \ (A\mathbf{x} \ - \ b)}{\partial x_i} \ = \ 0 \qquad i = 1 \dots n \tag{59}$$

If we develop the square we get:

$$\frac{\partial (\mathbf{x}^T A^T A\mathbf{x} \ - \mathbf{x}^T A^T b \ - \ b^T A\mathbf{x} \ + \ b^T b)}{\partial x_i} \ = \ 0 \tag{60}$$

Which can be computed as:

$$A^T A\mathbf{x} \ - A^T b \ = \ 0 \tag{61}$$

We end up with a linear system of equation with the quadratic system matrix $A^T A$ instead of the (in general) rectangular matrix A. We can formally get the solution for $\mathbf{x}$ as:

$$\mathbf{x} \ = \ (A^T A)^{-1} A^T b \tag{62}$$

This is the formal solution to the non-homogeneous least squares problem. The matrix $(A^T A)^{-1} A^T$ is called the pseudo inverse of A. Note that A is in general a rectangular matrix so it has no inverse in the ordinary sense.
We will later come back to methods of numerical solutions of these kind of problems.

MATH : Least squares minimisation - homogeneous equations

A *homogeneous* system of equations for the unknown vector $\mathbf{x}$ can be written:

$$A\mathbf{x} \; = \; 0 \tag{63}$$

Where A is a general $m \times n$ matrix. If $\mathbf{x}$ is a solution to this system obviously any vector $\lambda\mathbf{x}$ with arbitrary λ is a solution. In order to treat this problem we therefore have to normalised the solution in some way to get a unique vector. We can always put e.g $x_n = 1$ If we define the $n-1$-dimensional vector $\mathbf{x}' \; = \; (x_1 \ldots x_{n-1})$ and the $m \times n - 1$ matrix A' by deleting the n:th row a_n in A we can write the homogeneous system as:

$$A'\mathbf{x}' \; + \; a_n \; = \; 0 \tag{64}$$

Which in a non-homogeneous system for the vector **x'**. By normalising the homogeneous vector this way We can therefore in principle use the same technique for homogeneous as for non-homogeneous systems. This kind of normalisation could lead to numerical problems however if the last component x_n is very small. We would therefore like to have a normalisation procedure that treats all components of $\mathbf{x}$ equally. This can be done if we require $\mathbf{x}$ to have unit length, i.e:

$$||\mathbf{x}||^2 \; = \; \mathbf{x}^T\mathbf{x} \; = \; 1 \tag{65}$$

We can then formulate the problem of finding $\mathbf{x}$ as the constrained least squares problem:

$$\min_{||\mathbf{x}||^2 = 1} \quad ||A\mathbf{x}||^2 \tag{66}$$

Which can be written as:

$$\min_{\mathbf{x^T x} = 1} \quad \mathbf{x}^T A^T A\mathbf{x} \tag{67}$$

This constrained minimisation problem is equivalent to minimizing the Lagrange function:

$$\mathbf{x}^T A^T A\mathbf{x} - \lambda(\mathbf{x^T x} - 1) \tag{68}$$

Setting partial derivatives to zero:

$$\frac{\partial(\mathbf{x}^T A^T A\mathbf{x} - \lambda(\mathbf{x^T x} - 1))}{\partial x_i} \qquad i \; = \; 1 \ldots n \tag{69}$$

we get:

$$A^T A\mathbf{x} \; = \; \lambda\mathbf{x} \tag{70}$$

i.e. any eigenvector of the matrix $A^t A$ is a minimum. For a specific eigenvector we can compute the cost function as :

$$\mathbf{x}^T A^T A\mathbf{x} \; = \; \lambda\mathbf{x}^T\mathbf{x} \; = \; \lambda \tag{71}$$

since the length of $\mathbf{x}$ is normalised to unity. The least squares solution is therefore the eigenvector of $A^t A$ with the minimum eigenvalue.

2 3D Reconstruction from two views

2.1 Two calibrated cameras

When a 3D scene or object is projected to an image, information about the position 3D is lost. A certain image point can be the projection of multiple points in 3D. All points on the 3D line through the projection center and the image points are possible origins of the projected image point. If a point is projected to two images with different projection centers however, it's position in 3D has to be on two lines. One line for each camera. Provided we know the orientations, positions and internal parameters we can compute the projection matrices of the two cameras. Using those , we can compute the 3D position of every image point.

$$x = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$y = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}} \tag{72}$$

These equations can be simply rearranged to:

$$(x\ m_{31} - m_{11})X + (x\ m_{32} - m_{12})Y + (x\ m_{33} - m_{13})Z + (x\ m_{34} - m_{14}) = 0$$

$$(y\ m_{31} - m_{21})X + (y\ m_{32} - m_{22})Y + (y\ m_{33} - m_{23})Z + (y\ m_{34} - m_{24}) = 0 \tag{73}$$

These are two linear equations for three unknowns, which is the algebraic form of the fact that 3D information is lost when projecting to an image. However, if we consider two cameras a and b we get *four* equations in three unknowns:

$$\begin{pmatrix} (x^a\ m^a_{31} - m^a_{11}) & (x^a\ m^a_{32} - m^a_{12}) & (x^a\ m^a_{33} - m^a_{13}) & (x^a\ m^a_{34} - m^a_{14}) \\ (y^a\ m^a_{31} - m^a_{21}) & (y^a\ m^a_{32} - m^a_{22}) & (y^a\ m^a_{33} - m^a_{23}) & (y^a\ m^a_{34} - m^a_{24}) \\ (x^b\ m^b_{31} - m^b_{11}) & (x^b\ m^b_{32} - m^b_{12}) & (x^b\ m^b_{33} - m^b_{13}) & (x^b\ m^b_{34} - m^b_{14}) \\ (y^b\ m^b_{31} - m^b_{21}) & (y^b\ m^b_{32} - m^b_{22}) & (y^b\ m^b_{33} - m^b_{23}) & (y^b\ m^b_{34} - m^b_{24}) \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = 0$$

which allows us to compute the 3D coordinates X, Y, Z of a point from the image coordinates of it's projections in two images a and b. We are however also required to know the projection matrices of the two cameras. Collecting this knowledge is known as *calibration* and it is of course a central part of 3D reconstruction from multiple views. Knowing internal camera parameters K is called *internal* calibration and knowledge of camera rotation R and translation X_0, Y_0, Z_0 is called *external* calibration. Internal and external calibration are somewhat different and are in general treated separately. Both kinds of calibration can be computed from the image of calibration objects with completely known 3D structure relative to the reference coordinate system. This is however often not practically possible. We will therefore study approaches to calibration that uses no or very little knowledge in the form of pre defined calibration objects. By making some simple assumptions about the cameras involved in a multi view situations, we shall see that it is possible to compute both internal and external calibration by having the cameras look at an unknown 3D object. This is known as *self calibration* which essentially means calibration without any external information. We will also consider intermediate cases where some external calibration information is exploited as well as more general cases with no assumptions at all about the cameras.

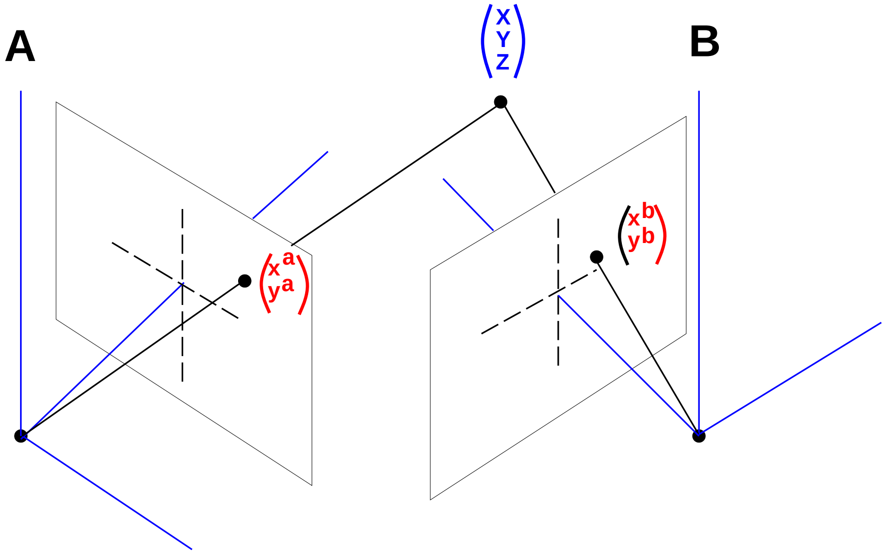

Figure 11: *3D reconstruction of point X, Y, Z from two views*

2.2 Calibration using known 3D object

Suppose we know the 3D coordinates X_i, Y_i, Z_i, of a set of points $i = 1 \dots n$ and their image coordinates x_i, y_i. Using this information, we would like to compute the projection matrix M of the camera. Since this is a 3×4 homogeneous matrix it has 12 elements which by normalisation can be reduced to 11. Starting with the projection equation (72)we get two equations for the unknown elements $m_{11} \dots m_{34}$ for each point. By rearranging terms, these equations can be written as:

$$\begin{aligned} m_{11}X + m_{12}Y + m_{13}Z + m_{14} - m_{31}xX - m_{32}xY - m_{33}xZ - m_{34}x &= 0 \\ m_{21}X + m_{22}Y + m_{23}Z + m_{24} - m_{31}yX - m_{32}yY - m_{33}yZ - m_{34}y &= 0 \qquad (74) \end{aligned}$$

These equations are exactly the same as eq. (73) with terms rearranged in a different way. Given that we have 11 unknowns and two equations for each point we see that we can compute the elements of the projection matrix if we have at least 6 points giving us a total of 12 equations. For n points we get the system of equations:

$$\begin{pmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{pmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{pmatrix} = 0$$

Just as reconstruction, calibration is achieved by solving an (in general) over determined system of linear equations. As we saw in the previous chapter, this applies to the determination of the linear mapping between two rotated cameras. In general we would like the solution to these problem to be as robust as possible so we will try to make them over determined.

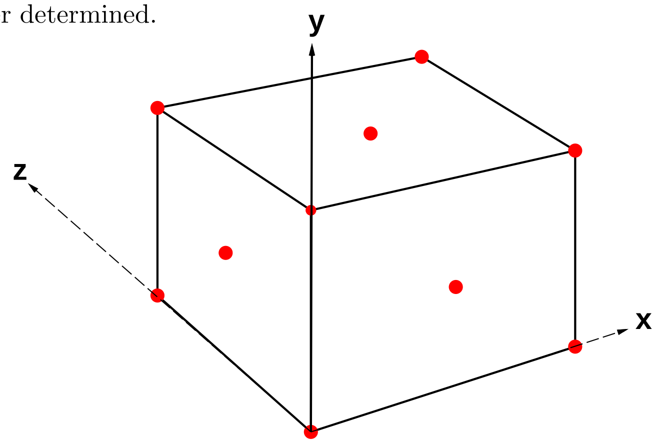

Figure 12: *By using points with known 3D locations in the coordinate system of the calibration object, we can find the projection matrix of the camera*

2.3 Computing camera parameters from the projection matrix

From equations (53) and (55) we have:

$$\lambda\ K\ R \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \tag{75}$$

We can factor the right hand side of this equation and get:

$$\lambda\ K\ R \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & m'_{14} \\ 0 & 1 & 0 & m'_{24} \\ 0 & 0 & 1 & m'_{34} \end{pmatrix} \tag{76}$$

with

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} m'_{14} \\ m'_{24} \\ m'_{34} \end{pmatrix} = \begin{pmatrix} m_{14} \\ m_{24} \\ m_{34} \end{pmatrix} \tag{77}$$

we can make the identifications:

$$\lambda\ K\ R = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} = -\begin{pmatrix} m'_{14} \\ m'_{24} \\ m'_{34} \end{pmatrix} = -\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}^{-1} \begin{pmatrix} m_{14} \\ m_{24} \\ m_{34} \end{pmatrix} \tag{78}$$

This gives us directly the camera projection center X_0, Y_0, Z_0 expressed in terms of the camera projection matrix elements $m_{11} \dots m_{33}$ We also get a relation between the internal parameters K, the rotation matrix R and the projection matrix elements. It is not obvious that we can find unique matrices K and R from this relation. We can always include the scale factor λ in the unknown matrix K. Taking the transpose of both sides we get:

$$\begin{pmatrix} m_{11} & m_{21} & m_{31} \\ m_{12} & m_{22} & m_{32} \\ m_{13} & m_{23} & m_{33} \end{pmatrix} \; = \; (KR)^T \; = \; R^T K^T \tag{79}$$

The problem is then to find an *upper triangular* matrix K and an *orthogonal* matrix R that satisfies this relation. We will write the columns of the left side matrix as $\mathbf{m_1}, \mathbf{m_2}, \mathbf{m_3}$ Writing out the columns vectors for the left side M-matrix and the orthogonal matrix R^T and writing out the elements of the lower triangular matrix K^T (being the transpose of an upper triangular matrix) we get:

$$(\mathbf{m_1}, \mathbf{m_2}, \mathbf{m_3}) \; = \; (\mathbf{r_1}, \mathbf{r_2}, \mathbf{r_3}) \begin{pmatrix} a & 0 & 0 \\ b & d & 0 \\ c & e & f \end{pmatrix} \tag{80}$$

We can write this out as:

$$\begin{array}{llll} \mathbf{m_1} & = a\mathbf{r_1} & + b\mathbf{r_2} & + c\mathbf{r_3} \\ \mathbf{m_2} & = d\mathbf{r_2} & + e\mathbf{r_3} & \\ \mathbf{m_3} & = f\mathbf{r_3} & & \end{array} \tag{81}$$

Starting with the third equation and remembering that the vectors $\mathbf{r_i}$ are orthogonal, i.e $\mathbf{r_i}^T\mathbf{r_j} \; = \; 0$ for $i \neq j$ and $|\mathbf{r_i}| \; = \; 1$ for all i, we get:

$$f \; = \; |\mathbf{m_3}| \quad \mathbf{r_3} \; = \; f^{-1}\mathbf{m_3} \; = \; \frac{\mathbf{m_3}}{|\mathbf{m_3}|} \tag{82}$$

Taking the inner product with $\mathbf{r_3}$ on both sides of the second equation and using $\mathbf{r_3}^T\mathbf{r_2} \; = \; 0$ and $\mathbf{r_3}^T\mathbf{r_3} \; = \; 1$ we have:

$$\mathbf{m_2}^T\mathbf{r_3} \; = \; e \tag{83}$$

Using this in the second equation we have:

$$d^{-1}(\mathbf{m_2} - e\mathbf{r_3}) = \mathbf{r_2} \tag{84}$$

Using $|\mathbf{r_2}| \; = \; 1$ we can compute the value of d:

$$d \; = \; |(\mathbf{m_2} - e\mathbf{r_3})| \tag{85}$$

So far we have determined $f, e, d, \mathbf{r_3}$ and $\mathbf{r_2}$. Taking inner products with $\mathbf{r_3}$ and $\mathbf{r_2}$ in the first equation and using orthogonality properties we get:

$$\mathbf{m_1}^T\mathbf{r_2} = b$$

$$\mathbf{m_1}^T\mathbf{r_3} = c \tag{86}$$

which determines b and c. Writing the first equation as:

$$a^{-1}(\mathbf{m_1} - \; b\mathbf{r_2} - \; c\mathbf{r_3}) \; = \; \mathbf{r_1} \tag{87}$$

we get from $|\mathbf{r_1}| \; = \; 1$:

$$a \; = \; |(\mathbf{m_1} - \; b\mathbf{r_2} - \; c\mathbf{r_3})| \tag{88}$$

This concludes the description of the procedure of computing matrices R and K such that the first 3×3 part of the projection matrix factorises as KR. Decomposing a matrix into orthogonal and upper/lower triangular matrix, known as QR-factorisation, is a general procedure that can be applied to matrices of any dimension. As can be seen it is closely related to the idea of Gram-Schmidt orthogonalization.

Note that in our case the matrix M is defined up to an arbitrary scale factor only. Since the rotation matrix R is normalized and cannot be scaled, this means that the matrix K will have an arbitrary scale factor associated with it. We can therefore conclude:

> Given the elements $m_{11} \ldots m_{33}$ of the projection matrix up to an arbitrary scale factor, we can find unique values of camera position X_0, Y_0, Z_0 and rotation R. The internal parameter matrix K can be determined uniquely up to an arbitrary scale factor.

This gives a characterization of all cameras that have the same projection matrix up to an arbitrary scale factor, or equivalently all cameras that have exactly the same projected image coordinates for a set of 3D points.

From a practical point of view, calibration using a known 3D calibration object is in general used in order to find internal parameters only. External calibration in general has to be done from case to case as we will see in the next section.

2.4 Internally calibrated cameras - The essential matrix

By using a 3D calibration object we can determine the internal parameters of a camera. We can also determine the external parameters, position and orientation, relative to the calibration object. In most cases of interest we will not have a known calibration object visible in the scene that we want to reconstruct. The external calibration is therefore in general of no use. Finding the internal calibration can be very useful however, provided it does not change between the calibration moment and the moment of capturing the images we want to use for reconstruction. We will see however that cameras can actually be externally calibrated without the use of a known 3D calibration object. This is probably the most useful fact to be learnt in multi view 3D reconstruction.

Let us assume that we have two cameras with known internal calibration. We can always let the projection point of the first camera be at the origin with the image plane at unit distance along the Z-axis. The projection point of the second camera is at X_0, Y_0, Z_0 and it has rotation R. Since the internal parameters are known, we can assume that the K-matrices are unit matrices. We therefore have the projection matrices: Eq. (53) can be written:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \lambda_a \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} = \lambda_b \; R \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{89}$$

Consider the three vectors:

$$R^T \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \quad \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \quad \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \tag{90}$$

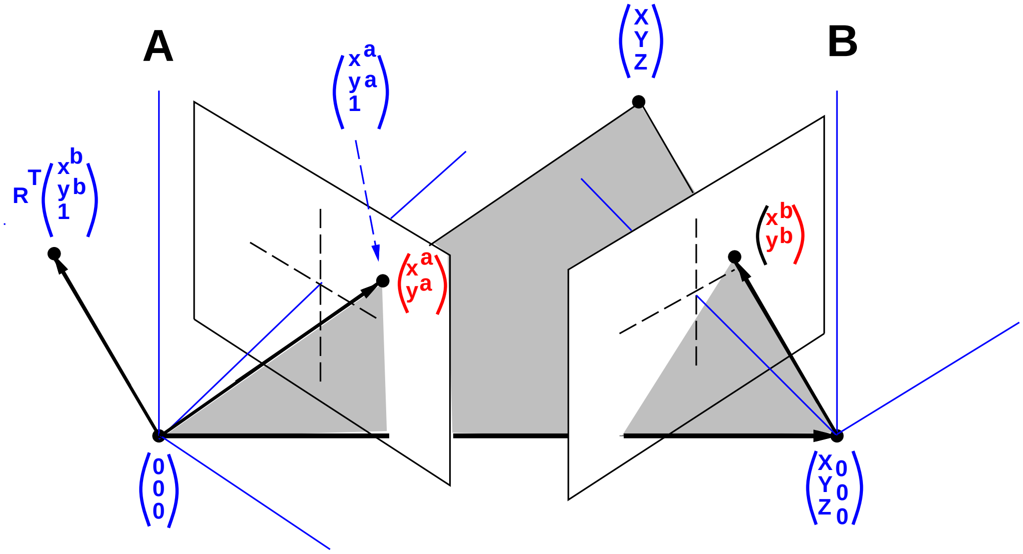

Figure 13: *The epipolar plane formed by the projection points of cameras A and B and the 3D point P. The epipolar constraint is generated by the coplanarity of the 3 boldface indicated vectors*

From fig. 13 we see that they all lie in the same plane. The normal vector of a plane containing vectors a, b and c can be expressed as the cross product $b \times c$. The vector a in the plane is orthogonal to this normal vector, i.e:

$$a^T(b \times c) \;=\; 0 \tag{91}$$

Identifying the vectors in eq. (90)we get:

$$(\, x^b \; y^b \; 1 \,) \;\; R \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \times \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \;=\; 0 \tag{92}$$

The cross product can expressed as:

$$\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \times \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \begin{pmatrix} Y_0 & - & Z_0 y^a \\ Z_0 x^a & - & X_0 \\ X_0 y^a & - & Y_0 x^a \end{pmatrix} = \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix} \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \tag{93}$$

If we plug this into eq. (92) we get:

$$\boxed{(\, x^b \; y^b \; 1 \,) \;\; R \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix} \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \;=\; 0} \tag{94}$$

This is a bilinear form in the image coordinates of image a and b. called the *epipolar constraint.* It is completely defined by the matrix:

$$\boxed{E \;=\; R \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix}} \tag{95}$$

known as the *essential matrix*

The essential matrix plays a very important role in calibration and reconstruction from two camera views. It's application is actually twofold depending on whether we know the point to point correspondence in the two images or not:

1. If we do not know which points in image a corresponds to points in image b, but we know the essential matrix, then the epipolar constraint can be used as a correspondence constraint, i.e we can use it to verify that two points are in correspondence or not
2. If we know sufficiently many ≥ 5 point to point correspondences between image a and b, we can use the coordinates of these points to compute the essential matrix.

Note that these two statements are complimentary .The essential matrix gives us point correspondence constraints and the point correspondences give us the essential matrix. This means that correspondence and calibration is often computed in an iterative way. In practice, point correspondence is computed using the information in the surrounding grey value image. This information degrades however as the two cameras have very different positions and orientations. The purely geometric constraint given by eq. (94) is valid for arbitrary positions and orientations of the cameras.

The epipolar constraint 94 can be expressed using the essential matrix with elements $e_{i,j}$:

$$(\, x^b \;\, y^b \;\, 1\,) \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \;=\; 0$$

We can write this out as:

$$x^b x^a e_{11} + x^b y^a e_{12} + y^b x^a e_{21} + y^b y^a e_{22} + x^b e_{13} + y^b e_{23} + x^a e_{31} + y^a e_{32} + e_{33} = 0$$

This is a linear equation in the nine unknown elements $e_{11} \ldots e_{33}$. Every corresponding point pair in image a and b contribute one such equation. Since the equations are homogeneous, we actually only have 8 unknowns since we can e.g normalize by dividing all equations with the element e_{33}. This means that the elements can only be determined up to an unknown scale factor. If we have at least 8 point correspondences, we can solve this linear system of equations and obtain the E-matrix up to scale. We will later look at the specific problems associated with numerical computation of the essential matrix.

2.5 Computing the projection matrix from E

In order to reconstruct the 3D coordinates of points from their projections in multiple views, we saw from section 2 that we need to know the camera matrices for all cameras involved. For internally calibrated cameras, this means that we have to know the rotation R and translation of camera center $(X_0, Y_0, Z_0)^T$ relative the first reference camera. These quantities are encoded in the E-matrix defined in eq. 95 and we will show that they can actually (with some minor restrictions) be computed from it.

2.5.1 Computing the translation of the camera center

To get a more convenient formalism we will denote the camera center and its associated antisymmetric matrix as:

$$\mathbf{t} = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \qquad [\mathbf{t}]_\times = \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix} \tag{96}$$

Eq. 95 can then be written:

$$E = R[\mathbf{t}]_\times \tag{97}$$

The matrix $[t]_\times$ has the property that when multiplied by an arbitrary 3-vector, it returns the cross product with the vector t.

$$[\mathbf{t}]_\times \mathbf{p} = \mathbf{t} \times \mathbf{p} \tag{98}$$

this means that

$$E\mathbf{t} = R[\mathbf{t}]_\times \mathbf{t} = R\mathbf{t} \times \mathbf{t} = 0 \tag{99}$$

The vector $\mathbf{t}$ is therefore the null-vector to the E-matrix, i.e the eigenvector with eigenvalue 0. It can be found using standard procedures for eigenvector computations once the E-matrix has been computed. Note that being an eigenvector i.e a homogeneous quantity, $\mathbf{t}$ can only be computed up to an arbitrary scale factor. This is the best we can do when computing the camera center relative to the reference camera. It is a simple example of the fact that we cannot compute unique solutions for certain parameters when trying to calibrate. This will of course have the consequence that when we want to do 3D reconstructions we will end up with a whole equivalence class of solutions. In this case the equivalence class is particularly simple however. It just means that all 3D structures will have an unknown scale just as the camera center.

Using this approach we are still left with computing the camera rotation R however. This requires some more elaborate computations involving the E matrix

2.5.2 Computing the camera translation and rotation from E

We will describe a method of computing both the camera translation and rotation from the E-matrix. This involves computing the singular value decomposition (SVD) of the E-matrix as described in chapter 5. For that purpose we will define three matrices that we will make use of:

$$Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad W = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{100}$$

The main feature of the matrix $[\mathbf{t}]_\times$ is that it is antisymmetric or equivalently, skewsymmetric, i.e:

$$[\mathbf{t}]_\times^T = \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix}^T = \begin{pmatrix} 0 & Z_0 & -Y_0 \\ -Z_0 & 0 & X_0 \\ Y_0 & -X_0 & 0 \end{pmatrix} = -[\mathbf{t}]_\times \tag{101}$$

Let V e an arbitrary *orthonormal* 3×3 matrix. i.e $VV^T = V^TV = I$ Using this and the matrix Z as defined above we construct the matrix

$$VZV^T \tag{102}$$

Taking the transpose we get:

$$(VZV^T)^T \;=\; V(VZ)^T \;= VZ^TV^T \;= -VZV^T \tag{103}$$

where the last equality follows from the fact that Z is antisymmetric. The matrix VZV^T obviously equals its own negative on transposition. It is therefore also an antisymmetric matrix. It turns out that for any antisymmetric matrix, e.g $[\mathbf{t}]_\times$ we can always find an orthonormal matrix V such that:

$$[\mathbf{t}]_\times \;= \phi\; VZV^T \tag{104}$$

Where ϕ is a scalar. This is slightly intricate to prove but intuitively obvious since V has three degrees of freedom just as $[\mathbf{t}]_\times^T$

From eq. 100 we get:

$$Z \;=\; WD \tag{105}$$

which means that we can write:

$$[\mathbf{t}]_\times \;= \phi\; VZV^T \;= \phi\; VWDV^T \tag{106}$$

The E-matrix can therefore be expressed as:

$$E \;=\; R[\mathbf{t}]_\times \;= \phi RVWDV^T \tag{107}$$

Since the matrices R, V and W are all orthonormal, their product RVW is also orthonormal. Using:

$$U \;=\; RVW \tag{108}$$

we can express E as:

$$E \;=\; \phi UDV^T \tag{109}$$

Since both U and V are orthonormal matrices, this represents the singular value decomposition (SVD) of E, with the singular values given by the diagonal elements of the matrix ϕD i.e. ϕ, ϕ and 0. Since E is homogeneous, all values of ϕ are equally valid. By normalizing the norm $||E||$ we can fix the value of ϕ.

Given an E matrix we find its singular value decomposition, $E \;=\; \phi UDV^T$, i.e. the orthonormal matrices U and V. If we normalise E so that $||E|| = ||UDV^T|| = 1$, we get the scale factor values $\phi = \pm 1$. The translation vectors associated with this normalised E-matrix are then:

$$[\mathbf{t_1}]_\times \quad = VZV^T \quad [\mathbf{t_2}]_\times \quad = -VZV^T \tag{110}$$

The remaining problem is then to find all possible rotation matrices R compatible with the normalisation $||E|| = ||UDV^T||$, i.e all matrices R such that

$$R[\mathbf{t}]_\times \;=\; E \;=\; \pm UDV^T \tag{111}$$

with $\mathbf{t} = \mathbf{t_1}$ or $\mathbf{t} = \mathbf{t_2}$ as defined above

Given a rotation matrix R and orthogonal matrices U and V we can always find another rotation matrix Q such that:

$$R \; = \; UQV^T \tag{112}$$

If this is used in eq 111 above, we get the condition on Q

$$R[\mathbf{t}]_\times \; = \; UQV^T[\mathbf{t}]_\times \; = \; \pm UQV^TVZV^T \; = \; \pm UQZV^T = \; \pm UDV^T \tag{113}$$

From which we deduce that:

$$QZ = \; \pm D \tag{114}$$

which gives the possible solutions

$$Q \; = \; W \quad\; Q = W^T \tag{115}$$

and rotation matrices R:

$$R_1 \; = \; UW^TV^T \qquad\quad R_2 \; = \; UWV^T \tag{116}$$

We can compute the associated E-matrices by taking the four combinations:

$$\begin{array}{lllllllll}
R_1[\mathbf{t_1}]_\times & = & UW^TV^TVZV^T & = & UW^TZV^T & = & UDV^T & = & E \\
R_1[\mathbf{t_2}]_\times & =- & UW^TV^TVZV^T & =- & UW^TZV^T & =- & UDV^T & =- & E \\
R_2[\mathbf{t_1}]_\times & = & UW\;\;V^TVZV^T & = & UW\;\;ZV^T & = & UDV^T & =- & E \\
R_2[\mathbf{t_2}]_\times & =- & UW\;\;V^TVZV^T & =- & UW\;\;ZV^T & =- & UDV^T & = & E
\end{array} \tag{117}$$

All these combinations represents the same matrix E with the same norm $||E||$, they are therefore all equally valid decompositions of E into translation t and rotation R. It turns out that any decomposition of E into t and R has to have this form. These four solutions therefore represent all valid solutions for a given E. Note that the two translation vectors have the same length with opposite signs. We will denote them as $+\mathbf{t}$ and $-\mathbf{t}$. The four solutions represent the four projection matrices:

$$M_1 \; = \; R_1(I| \;\; \mathbf{t}) \quad M_2 \; = \; R_1(I| - \mathbf{t}) \quad M_3 \; = \; R_2(I| \;\; \mathbf{t}) \quad M_1 \; = \; R_2(I| - \mathbf{t}) \tag{118}$$

Given a 3D point set, all these cameras together with the reference camera $M \; = \; (I| \;\; 0)$ represent camera pairs in which all image coordinates in both images are the same, although each camera pair has a different relative orientation. The situation is illustrated by the figure in the lab instructions on page 10

We can then summarize the steps of the algorithm for computing projection matrices from the essential matrix E:

1. Compute the singular value decomposition (SVD) of the essential matrix $E = UDV^T$

2. Use the matrix V in the SVD to compute translation vectors $\mathbf{t} = \pm VZV^T$ with the matrix Z as defined in eq: 100

3. Use the matrices U and V from the SVD together with the matrix W from defined in eq. 100 to compute the possible rotation matrices $R_1 \; = \; UW^TV^T$ and $R_2 \; = \; UWV^T$

4. All possible projection matrices are then given by the list in eq. 118

2.6 Projection matrices and 3D reconstruction

We see that we have managed to express the columns of the rotation matrix R^T in terms of the columns of the essential matrix E^T and the vector $\mathbf{t}$ representing the unit vector in the direction of the camera center translation. Starting from two internally calibrated cameras we have shown that it is possible to compute a external parameters of the camera system, rotation and direction of translation of the second camera relative to the first, using projected image data. Let $\mathbf{p}$ and $\mathbf{P}$ be the vectors for the image and 3D coordinates respectively, I the 3×3 identity matrix and $\mathbf{t}$ as above the unit direction of translation of the camera center. All coordinates refer to the 3D coordinate system of the first camera which has it's projection point located in the origin and is rotated with optical axis along the $Z-$axis. Let the center of projection of the second camera be $\sigma\mathbf{t}$, where σ is an unknown parameter. The two cameras can then be represented as:

$$\begin{aligned} \mathbf{p^a} &= \lambda_a(\ I \mid 0 \ )\mathbf{P} \\ \mathbf{p^b} &= \lambda_b R(\ I \mid \sigma\mathbf{t} \ )\mathbf{P} \end{aligned} \tag{119}$$

For each choice of the parameter σ we get an over determined linear system of equations and can compute the 3D coordinates P of every point projected to in the two cameras. By varying σ we get a whole class of solutions for the 3D points. The question is then how are these solutions related ?. Let $\mathbf{P} = (X, Y, Z, 1)^T$ be the solution for the 3D coordinate vector for the choice of $\sigma = 1$:

$$\begin{aligned} \mathbf{p^a} &= \lambda_a(\ I \mid 0 \ )\mathbf{P} \\ \mathbf{p^b} &= \lambda_b R(\ I \mid \mathbf{t} \ )\mathbf{P} \end{aligned} \tag{120}$$

Now consider the vector $\mathbf{P'} = (X, Y, Z, \sigma^{-1})^T$ and the system 119. We obviously have:

$$\begin{aligned} \mathbf{p^a} &= \lambda_a(\ I \mid 0 \ )\mathbf{P'} &= \lambda_a(\ I \mid 0 \ )\mathbf{P} \\ \mathbf{p^b} &= \lambda_b R(\ I \mid \sigma\mathbf{t} \ )\mathbf{P'} &= \lambda_b R(\ I \mid \mathbf{t} \ )\mathbf{P} \end{aligned} \tag{121}$$

The vector $\mathbf{P'} = (X, Y, Z, \sigma^{-1})^T$ is therefore s solution for any arbitrary value of σ. Since it is a homogeneous coordinate' vector, we can scale it arbitrarily and we see that it is equivalent to the vector $(\sigma X, \sigma Y, \sigma Z, 1)^T$, i.e. it represents the Cartesian 3D coordinates $\sigma(X, Y, Z)$. If (X, Y, Z) and $\mathbf{t}$ are solutions to the two camera reconstruction problem so are any arbitrarily scalings: $\sigma(X, Y, Z)$ and $\sigma\mathbf{t}$. We say that the solution is known up to an arbitrary scale factor only. In homogeneous coordinates, the solutions are related by the transformation:

$$\begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \begin{pmatrix} \sigma & 0 & 0 & 0 \\ 0 & \sigma & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{122}$$

We will frequently meet this phenomena of “ambiguous” or multiple solutions in 3D reconstruction that are related by a linear transformation. In this case of internally calibrated cameras the ambiguity is not particularly severe. It simply means that we cannot determine the absolute length scale of the object or scene we are reconstructing. I.e. the external calibration is not perfect. In order to do that we have to supply extra information about some known length in the scene. In the next section however, we will consider the case of internally uncalibrated cameras, i.e we do not know the internal parameters. We will particularly investigate to what extent we still can achieve 3D reconstruction.

2.7 Summary: 3D reconstruction internally calibrated cameras

The steps from internal calibration of the cameras to 3D reconstruction can now be summarized:

Find internal camera parameters using known 3D calibration object

⇓

Compute essential matrix E from image projections p^a, p^b in two cameras using epipolar constraint ${p^a}^T E p^b = 0$

⇓

Compute camera rotation R and translation $\mathbf{t}$ of camera b by finding the singular value decomposition of the essential matrix: $E = R[\mathbf{t}]_\times$

⇓

Use projection relations $p^a = \lambda_a (I \mid 0)\mathbf{P} \quad \mathbf{p^b} = \lambda_\mathbf{b} \mathbf{R}(\mathbf{I} \mid \sigma \mathbf{t})\mathbf{P}$ to compute 3D coordinates $\mathbf{P}$ from image projections p^a, p^b in two cameras. Note that $\mathbf{P}$ can only be computed up to an arbitrary scale factor.

A main disadvantage here is the fact that we have to calibrate the cameras before doing anything else. This will prevent us from changing any camera parameters during the image acquisition. Calibration parameters of cameras in generally drift with time, so the calibration has to be repeated regularly. In the next section we will investigate the effects of using uncalibrated cameras, i.e we do not know any internal parameters such as scale factors or optical centers.

2.8 Uncalibrated cameras - The fundamental matrix

If we don't know the internal calibration of the cameras, we cannot write the projection relations using normalised image coordinates as in eq. (89), but we have to include the calibration matrices K_a and K_b:

$$\begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} = \lambda_a \; K_a \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} = \lambda_b \; K_b \; R \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{123}$$

This means that the transformed image coordinates:

$$K_a^{-1} \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \qquad K_b^{-1} \begin{pmatrix} x^b \\ y^b \\ 1 \end{pmatrix} \tag{124}$$

obey the same projection equation (89) as the normalised coordinates of the calibrated cameras. For uncalibrated cameras, we can therefore derive exactly the same epipolar constraint as for calibrated cameras by simply substituting the image coordinates according to eq.(124). Doing this we get:

$$\boxed{(\, x^b \;\, y^b \;\, 1\,) \;\; K_b^{-1^T} \;\, R \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix} K_a^{-1} \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \; = \; 0} \tag{125}$$

which is the epipolar constraint for uncalibrated cameras. The essential matrix is now substituted by the matrix:

$$\boxed{F \; = \; K_b^{-1^T} \;\; R \begin{pmatrix} 0 & -Z_0 & Y_0 \\ Z_0 & 0 & -X_0 \\ -Y_0 & X_0 & 0 \end{pmatrix} K_a^{-1}} \tag{126}$$

which we call the *fundamental matrix*

The important message here is that even if we do not know anything about our two cameras, neither internal nor external calibration, we still know that corresponding points are related by:

$$(\, x^b \;\, y^b \;\, 1\,) \;\; F \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \; = \; 0 \tag{127}$$

where F is a 3×3 matrix.

The F matrix can be computed using corresponding image points in exactly the same way as the E matrix. The exact computation of the projection matrices from F is not feasible however since we do not know the internal calibration of the cameras. We will see however, that we can still compute modified forms of projection matrices that will allow us to compute a 3D reconstruction with specific properties. In order to see this we first have to discuss the general problem of multiple ambiguous solutions to projection equations.

2.9 Projective reconstruction

In its most general form the problem of computing a 3D reconstruction from two cameras can be formulated as finding the solutions $\mathbf{P}$ to the equations:

$$\begin{aligned} \mathbf{p^a} &= M_a \mathbf{P} \\ \mathbf{p^b} &= M_b \mathbf{P} \end{aligned} \tag{128}$$

For the case of completely uncalibrated cameras which we will consider here, the projection matrices M_a and M_b are unknown. We only know the image coordinates $\mathbf{p^a}, \mathbf{p^b}$ in the two views. The question then is: Can we compute the 3D coordinates $\mathbf{P}$ and the projection matrices M_a and M_b ? The answer to this is yes, but not in the same sense as for calibrated cameras. Even for calibrated cameras we saw that we could not compute the 3D reconstruction exactly. There would always be an unknown scale factor associated with the solution. Neither could we compute the second camera exactly since it's center would also have this unknown scale factor. We therefore actually computed a whole equivalence class of 3D reconstructions and cameras, one for each choice of the unknown scale factor. This class of 3D reconstructions could be represented by a linear parameterized transformation applied to the homogeneous coordinates of a specific solution eq. 122.

For uncalibrated cameras, we similarly have a whole equivalence class of possible solutions for 3D reconstruction and cameras. As we can expect, this class is much "larger" than that for calibrated cameras. It can actually be determined very simply from the two equations for the cameras provided we know at least one solution in the class. Apply an arbitrary linear transformation represented by the 4×4 matrix T and its inverse T^{-1} to the 3D coordinate in the above equations:

$$\begin{aligned} \mathbf{p^a} &= M_a\mathbf{P} = M_aT^{-1}T\mathbf{P} \\ \mathbf{p^b} &= M_b\mathbf{P} = M_bT^{-1}T\mathbf{P} \end{aligned} \tag{129}$$

We see that if M_a ,M_b and $\mathbf{P}$ are possible cameras and 3D points so are the cameras M_aT^{-1} ,M_bT^{-1} and points $T\mathbf{P}$. I.e given a solution for 3D coordinates, we can apply an *arbitrary* linear transformation to these coordinates and we still have a solution. The possible solutions are therefore known up to an arbitrary linear transformation only. Linear transformations applied to homogeneous coordinates are known as *projective* transformations. We therefore often refer to reconstruction from uncalibrated cameras as *projective reconstruction*. In a similar way of course , we cannot uniquely determine the projection ,matrices of the cameras since multiplication with the inverse T^{-1} still gives a solution. Obviously if we want to compute projective reconstructions we just have to find at least one possible solution. Once we have that, we know that any linear transformation applied to this defines a solution. Strictly speaking we have not proved that these solutions are the only ones. There could be even more solutions. It can be shown however that in general the projective reconstructions , represents all solutions except for special cases of 3d point sets known as critical configurations.

2.10 Computing the projection matrices from F

The fundamental matrix is completely determined by corresponding image coordinates acc. to eq. 127. Alternatively it is completely determined by internal and external camera parameters contained in the projection matrices M_a and M_b. In the previous section we saw that for arbitrary linear transformations T, the camera pair M_aT^{-1}, M_bT^{-1} would give exactly the same image coordinates for points $T\mathbf{P}$ as the camera pair M_a, M_b for points $\mathbf{P}$. These two camera pairs therefore have the same fundamental matrix. The fundamental matrix therefore has the property of being invariant to linear transformations of camera matrices. Let the 3×4 matrix M_a be decomposed into the 3×3 matrix A and 3-vector $\mathbf{a}$ as:

$$M_a = (A \mid \mathbf{a}) \tag{130}$$

We can in general assume that A is invertible, i.e the inverse A^{-1} exists. Choose the linear transformation represented by the 4×4 matrix T^{-1} as :

$$T^{-1} \;=\; \begin{pmatrix} A^{-1} & -A^{-1}\mathbf{a} \\ 0^T & 1 \end{pmatrix} \tag{131}$$

We then get:

$$M_a T^{-1} \;=\; (A \mid \mathbf{a}) \begin{pmatrix} A^{-1} & -A^{-1}\mathbf{a} \\ 0^T & 1 \end{pmatrix} \;=\; (\,AA^{-1}| - AA^{-1}\mathbf{a} + \mathbf{a}\,) \;=\; (I \mid 0) \tag{132}$$

The projection matrix of the second camera can always be decomposed as:

$$M_b T^{-1} \;=\; (H \mid \mathbf{h}) \tag{133}$$

If we want to find the projective reconstruction for the camera system M_a, M_b we can therefore just as well work with the cameras: $(I \mid 0), (H \mid \mathbf{h})$ since these camera pairs are related to each other by a linear transformation. This simplifies the analysis substantially since the problem of recovering both camera matrices M_a and M_b has been reduced to determining the 3×3 matrix H and the vector $\mathbf{t}$.

The fact that all projectively equivalent cameras have the same fundamental matrix is a simple consequence of the fact that they can always be made to produce exactly the same set of image coordinates by a proper choice of 3D points. An important but more difficult result is the fact that there is only one set of projectively equivalent cameras associated with a certain fundamental matrix. We will not prove this here but just note that this means that if we want to compute projection matrices and 3D reconstruction from image data, we can proceed in very much the same way as for calibrated cameras by first computing the fundamental matrix. Given the fundamental matrix we only have to produce one example of projection matrices associated with it. The projective equivalence class of solutions is obtained by applying an arbitrary linear transformation to this specific solution.

The epipolar constraint can be written

$$\mathbf{p}^{b^T} F \mathbf{p}^{\mathbf{a}} \;=\; 0 \tag{134}$$

If we use the projection equations:

$$\begin{aligned} \mathbf{p}^{\mathbf{a}} \;&=\; (I \mid 0\,)\,\mathbf{P} \\ \mathbf{p}^{\mathbf{b}} \;&=\; (H \mid \mathbf{h}\,)\,\mathbf{P} \end{aligned} \tag{135}$$

the epipolar constraint becomes:

$$((H \mid \mathbf{h}\,)\,\mathbf{P})^T \; F \; (I \mid 0\,)\,\mathbf{P} \;=$$

$$=\; \mathbf{P}^T \begin{pmatrix} H^T \\ - \\ \mathbf{h}^T \end{pmatrix} \; F \; (I \mid 0)\mathbf{P} \;=\; \mathbf{P}^{\mathbf{T}} \begin{pmatrix} H^T F & 0 \\ & \\ \mathbf{h}^T F & 0 \end{pmatrix} \mathbf{P} \;=\; \mathbf{0} \tag{136}$$

The fact that this quadratic form is valid for all vectors $\mathbf{P}$ means that the 4×4 matrix in the form has to have a special structure. Consider a general $n \times n$ matrix Q with the property:

$$\mathbf{x}^T Q \mathbf{x} \;=\; 0 \tag{137}$$

for all vectors $\mathbf{x}$. It can be developed as:

$$\sum_{i=1}^{i=n}\sum_{j=1}^{j=n} x_i q_{i,j} x_j \;=\; \sum_{i=1}^{i=n}\sum_{j=i+1}^{j=n} x_i x_j (q_{i,j} + q_{j,i}) \;=\; 0 \tag{138}$$

Since this is valid for arbitrary vectors $x_1 \ldots x_n$ we can take the vector with $x_i \neq 0, x_j \neq 0$ and $x_k = 0, k = 1 \ldots n, k \neq i, k \neq j$. The expression above then becomes:

$$x_i x_j (q_{i,j} + q_{j,i}) \;=\; 0 \tag{139}$$

Since $x_i \neq 0, x_j \neq 0$ are arbitrary the only possibility for this expression to be generally valid is that

$$q_{i,j} + q_{j,i} \;=\; 0 \tag{140}$$

This is the general property defining an *anti symmetric* matrix. Note that we encountered matrices of this type earlier when we defined the equivalent matrix $[\mathbf{t}]_\times$ for the cross product with a vector $\mathbf{t}$ Looking at eq. 136 we see that for this to be an antisymmetric matrix, we must have:

$$H^T F \text{ antisymmetric}$$

$$\mathbf{h}^T F \;= 0$$

The second of these constraints means that the vector $\mathbf{h}$ is the left null vector of F. Alternatively writing: $\mathbf{h}^T F \;= (F^T \mathbf{h})^T \;= 0$ it can be seen as the right null vector of the matrix F^T, i.e. the eigenvector with eigenvalue 0 In order to make the matrix $H^T F$ antisymmetric , we choose H as: $H \;=\; SF$ where S is an arbitrary 3×3 antisymmetric matrix. We then get:

$$H^T F \;=\; (SF)^T F \;=\; F^T S^T F \tag{141}$$

If S is antisymmetric it is easy to see that S^T is. Consider the quadratic form:

$$\mathbf{x}^T H^T F \mathbf{x} \;=\; \mathbf{x}^T F^T S^T F \mathbf{x} \;=\; (F\mathbf{x})^T S^T F \mathbf{x} \tag{142}$$

but since S^T is an antisymmetric matrix this must be zero for arbitrary $\mathbf{x}$ which proves that $H^T F$ is antisymmetric for any choice of $H \;=\; SF$ where S is antisymmetric. We have therefore found two camera matrices that correspond to a specific fundamental matrix F, namely the pair:

$$(I \mid 0\,)\,\mathbf{P}$$

$$(SF \mid \mathbf{h}\,)\,\mathbf{P} \tag{143}$$

where S is an arbitrary 3×3 antisymmetric matrix and $\mathbf{h}$ is the null vector: $F^T \mathbf{h} \;=\; 0$

It is interesting to contrast these computations with those for the case of calibrated cameras since they are substantially less complex. The main differences are that 1) No calibration procedure to obtain internal parameters of the camera is needed and 2) The somewhat complex procedure of factorising the essential matrix is replaced by the multiplication of F by an arbitrary antisymmetric matrix.

The price for this increased simplicity is of course that we are not able to compute a metric 3D reconstruction. All 3D points $\mathbf{P}$ can only be computed up to an arbitrary unknown linear transformation $\mathbf{TP}$. The effect of a linear transformation on a 3D object will be a distortion of the object. Fig 14 illustrates a the effect of a linear transformation on a cube. Note that right angles e.g are not preserved in general. Planar surfaces and straight lines remain planar and straight however. This is a consequence of the linearity of the transformation.

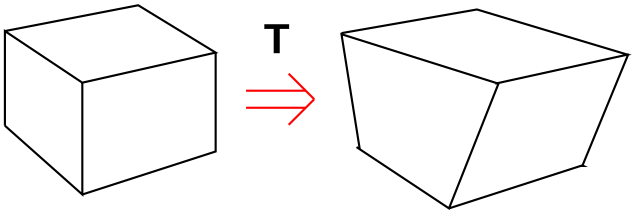

Figure 14: *Example of a linear transformation* $\mathbf{T}$ *on a cube.*

2.11 Rectification using metric information

For a given projective reconstruction $\mathbf{P_1} \dots \mathbf{P_n}$ like this we can formulate the calibration problem afterwards as finding the transformation $\mathbf{T}$ that relates the metrically correct reconstruction with that of the projective reconstruction, i.e. $\mathbf{P'} = \mathbf{TP}$. This can be done if we have some knowledge of the metric properties of the scene like lengths or angles. A direct way would be if we have a priori knowledge of a set of 3D points $\mathbf{P'_1} \dots \mathbf{P'_n}$ in the scene as when we use a calibration object. Given the projective reconstruction of these points $\mathbf{P_1} \dots \mathbf{P_n}$ we can formulate the problem of finding the rectifying transformation $\mathbf{T}$ in exactly the same way as we found the linear mapping H in the previous chapter. Since each point in in 3D, we now get 3 equations for determining the 16 elements $t_{11} \dots t_{44}$ of the matrix $\mathbf{T}$ from each transformation relation:

$$\mathbf{P`_i} = \mathbf{TP_i} \qquad \mathbf{i = 1 \dots n} \tag{144}$$

Since $\mathbf{T}$ is homogeneous, i.e defined up to an arbitrary scale facto, we actually have only 15 parameters. This means that 5 points would give a sufficient number of $3 \times 5 = 15$ equations.

In principle we can use any kind of metric information such as distances between points or knowledge that specific lines in 3D are orthogonal This is slightly more complicated however since it does not give the simple direct linear problem of determining $\mathbf{T}$ as the explicit knowledge of 3D coordinates. Alternatively we will demonstrate that if we have more than two cameras and they all have the same (but unknown) internal calibration, we can actually find this correcting transformation. This is known as *self calibration.*

2.12 Summary: 3D reconstruction uncalibrated cameras

The steps from external calibration to 3D reconstruction for uncalibrated cameras can now be summarized:

Compute fundamental matrix F from image projections p^a, p^b in two cameras using epipolar constraint ${p^a}^T F p^b = 0$

$\Downarrow$

Find the null vector of fundamental matrix $F\mathbf{h} = 0$

$\Downarrow$

Choose arbitrary anti symmetric matrix S Use projection relations $p^a = \lambda_a(I \mid 0\ )\ \mathbf{P} \quad \mathbf{p^b} = \lambda_{\mathbf{b}}(\mathbf{SF} \mid \mathbf{h}\ )\ \mathbf{P}$ to compute 3D coordinates $\mathbf{P}$ from image projections p^a, p^b in two cameras. Note that $\mathbf{P}$ can only be computed up to an arbitrary linear transformation

⇓

Use a priori metric information $\mathbf{P}'$ to find rectification transformation $\mathbf{T}$ in: $\mathbf{P}' \ = \mathbf{TP}$

This scheme is obviously simpler than the corresponding scheme for calibrated cameras. The main disadvantage is that we get reconstruction up to an arbitrary projective transformation. The crucial step then lies in finding metric information to be used for the rectification to metric structure. This information can frequently be found in the scene itself, especially when structured man made objects are considered. For general cases there are more complex options exploiting camera properties but they in general require more than two cameras. The methods of 3D reconstruction using calibrated vs. un calibrated cameras is essentially just a question of interchanging the processes of calibration and reconstruction. The flexibility gained by not having to calibrate the camera beforehand is a main factor deciding in favor of using the uncalibrated camera approach.

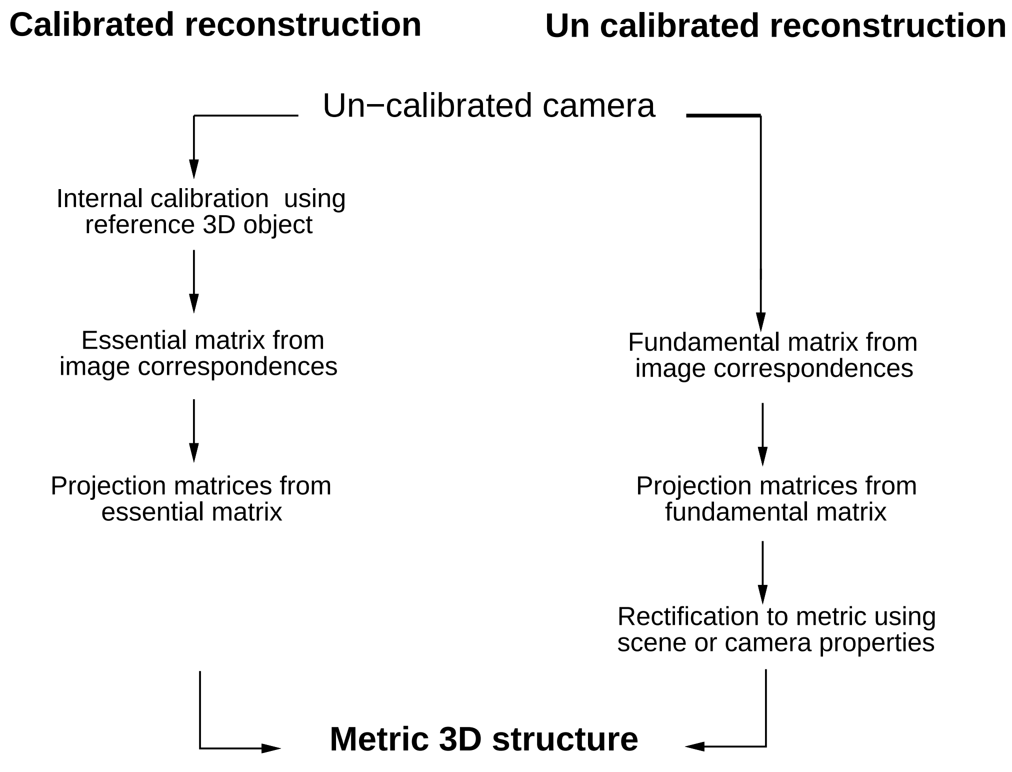
Calibrated reconstruction
Un calibrated reconstruction
Un−calibrated camera
Internal calibration using reference 3D object
Essential matrix from image correspondences
Projection matrices from essential matrix
Fundamental matrix from image correspondences
Projection matrices from fundamental matrix
Rectification to metric using scene or camera properties
Metric 3D structure

Figure 15: *Calibrated vs. uncalibrated 3D reconstruction*

3 Surface reconstruction and rendering

3D reconstructions of selective point sets only gives a sparse set of points in 3D while objects in general consists of continuous surfaces. These surfaces then have to be interpolated from the sparse point sets. There are several ways of computing interpolating surfaces from point sets with varying degree of complexity. We will just consider the simplest case corresponding to piecewise linear interpolation with planar surfaces. For simple one dimensional $x - y$ graphs, piecewise linear interpolation is a trivial matter of connecting neighboring points with straight lines. Alternatively we can say that we are interpolating a curve in 2D. The simplicity comes from the fact that we know which pairs of points to connect since they are ordered by there $x-$ values. For point sets in 3D, considered as functions $z = f(x, y)$, piecewise linear interpolation means connecting points with planar surfaces. This can be done if we select 3 points since they define a unique plane. Our problem then becomes how to select combinations of 3 points in the $x - y$ plane so that every point that we want to interpolate lies within the triangle formed by the points. Contrary to the case of one dimensional graphs, there is no trivial ordering of the points however. There is no unique way to select triplets of points so that every point in the plane becomes a member of one triangle. This is known as the problem of triangulating a point set. It turns out that some triangulations are more "natural" than others and we will look at ways of defining these. Once we have defined a triangulation for a set of points x_i, y_i in an image, we have defined a linear mapping from the image to 3D defined by the plane formed by the points z_i, x_i, y_i in 3D. Since each point in the image is a member of a unique triangle, we get a unique corresponding point in 3D. This unique correspondence can also be used to map the pixel grey value of the image to the surface in 3D. Every point on the piecewise planar surface in 3D therefore gets a unique grey value. This is known as "image based rendering" and it will allow us to reproject the 3D reconstruction to an arbitrary viewpoint and thereby create new images of the object.

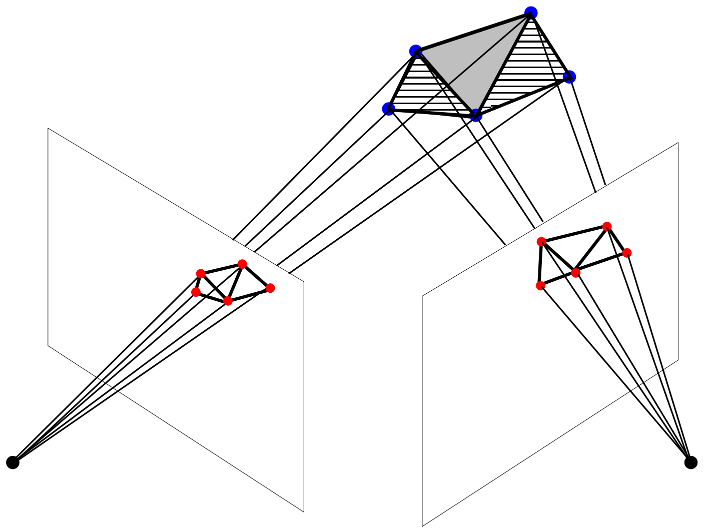

Figure 16: *Triangulations in image and 3D*

It may happen that we know that the surface in 3D actually is more or less exactly planar. This important special case occurs frequently for man made 3D objects such as buildings. We can then exploit more efficient mapping techniques defined by the geometry of the viewing situation.

3.1 Triangulation of point sets

Suppose that the image coordinates of our point set in an image is $x_1, y_1 \ldots x_n, y_n$. The corresponding reconstructed 3D coordinates will be denoted $X_1, Y_1, Z_1 \ldots X_n, Y_n, Z_n$. fig. 3.1. Consider a point x, y inside the triangle formed by points 1,2 and 3. We can use these points to define a local coordinate system in which we can express the coordinates of the point x, y by:

$$\begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \alpha \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \beta \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} \tag{145}$$

The values α, β are known as the *affine coordinates* in this system. They can be computed by solving the linear system of equations that is implied by their definition:

$$\begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{146}$$

to give:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \frac{1}{D} \begin{pmatrix} y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{pmatrix} \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix} \tag{147}$$

where D is the determinant:

$$D = \begin{vmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{vmatrix} \tag{148}$$

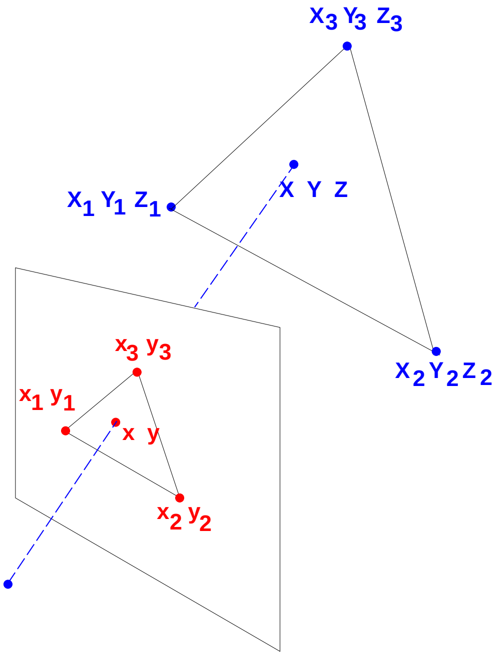

Figure 17: *Corresponding triangles in image and 3D for defining affine coordinates*

The 3D coordinates X, Y, Z corresponding to the point x, y using the interpolating plane defined by points 1,2 and 3 can now be computed as:

$$\begin{pmatrix} X - X_1 \\ Y - Y_1 \\ Z - Z_1 \end{pmatrix} = \alpha \begin{pmatrix} X_2 - X_1 \\ Y_2 - Y_1 \\ Z_2 - Z_1 \end{pmatrix} + \beta \begin{pmatrix} X_3 - X_1 \\ Y_3 - Y_1 \\ Z_3 - Z_1 \end{pmatrix} \tag{149}$$

It is easily verified that this maps points $1, 2$ and 3 with affine coordinates $(0, 0), (1, 0), (0, 1)$ respectively, to points $(X_1, Y_1, Z_1), (X_2, Y_2, Z_2)$ and (X_3, Y_3, Z_3). It must therefore be the unique linear mapping from the image point x, y to the 3D point X, Y, Z that maps these three image points to their respective 3D points. Provided each point to be mapped to 3D belongs to a unique triangle defined by three points in the point set, this mapping is unique. We therefore have to consider ways of computing triangulations of point sets.

3.1.1 Voronoi cells and Delaunay triangulations

The fact that there are several possible ways to triangulate a point set means of course that we cannot compute the 3D surface with complete certainty. A specific choice of triangulation implies a specific choice of values of the 3D surface at points not present in the original point set. Since the 3D surface is assumed to have linear variation within the triangle, a good heuristics for choice of triangulation is to make the triangles as compact as possible, i.e avoiding long extended triangles. In this way we try to make the linear approximation as "local" as possible. A specific way of triangulation that leads to compact triangles is called *Delaunay triangulation*. This is a way of forming triangles by grouping intuitively close points. The Delaunay triangulation of a finite set of points in the plane starts by dividing the the plane into cells by assigning each point in the plane to the closest point in the point set (fig. 3.1.1). These cells are called *Voronoi cells* The boundaries between two neighboring Voronoi cells are then defined by the points that have the same distance to two points. These boundaries are therefore always straight lines. This can be seen easily if we compute the squared distance from a point (x, y) to two points (x_1, y_1) and (x_2, y_2):

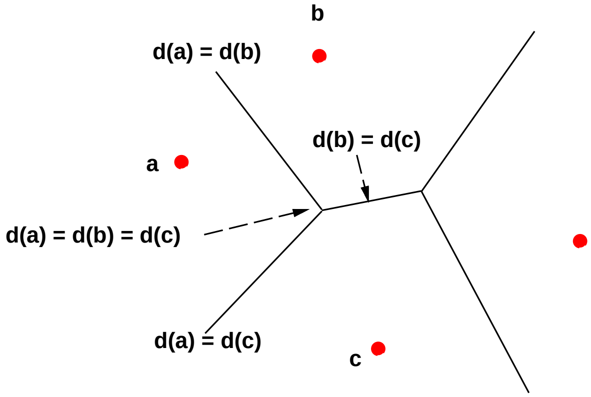

Figure 18: *Voronoi cells from a point set are generated by assigning each point in the plane to the nearest point in the set. Boundaries between the cells are then straight lines with the same distance* **d** *to two points in the set*

$$(x - x_1)^2 + (y - y_1)^2 \;=\; (x - x_2)^2 + (y - y_2)^2 \tag{150}$$

evaluating the squares we get:

$$x^2 - 2xx_1 + x_1^2 + y^2 - 2yy_1 + y_1^2 = x^2 - 2xx_2 + x_2^2 + y^2 - 2yy_2 + y_2^2 \tag{151}$$

which can be shortened to:

$$2x(x_2 - x_1) + 2y(y_2 - y_1) + x_1^2 - x_2^2 + y_1^2 - y_2^2 \;=\; 0 \tag{152}$$

which defines a straight line in the plane. Consider the line defined by the same distance to points a and b, $d(a) = d(b)$ and the line defined by the same distance to points a and c, $d(a) = d(c)$ These lines will meet at a point where: $d(a) = d(b) = d(c)$, i.e their point of intersection has the property that $d(b) = d(c)$ which is the definition of the line of points with same distance to points b and c. We can therefore conclude that boundary lines in general intersect in triplets, i.e 3 lines meet at a common junction. This then gives us an opportunity to define a triangulation with nice properties. Each junction of three lines is uniquely associated with three points and these points are grouped together to a triangle. This is called *Delaunay triangulation* and in general gives rise to compact triangles in the sense that the smallest angle in the triangle is as large as possible, i.e it tries to form triangles as close to equilaterals as possible. There are various algorithms to compute Voronoi cells and Delaunay triangulations and most software libraries contain such an algorithm.

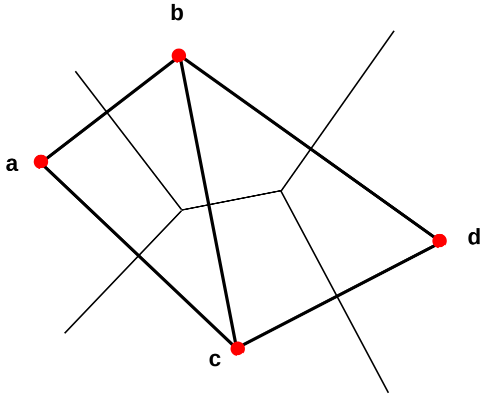

Figure 19: *Delaunay triangulation generated by grouping together triplets of points with common triplet line junctions*

3.2 Texture mappings - image based rendering

Once we have defined a triangulation of our point set, we can treat each triangle separately and compute the 3D plane associated with it from the known 3D coordinates of it's corner points. We now also want to paint or render a texture on this plane using the values $I(x_i, y_j)$ of the image intensity at the discrete image point (x_i, y_j) Every point on the planar surface triangle in 3D therefore has to be given unique integer coordinates and the intensity value associated with those coordinates should be computed from the corresponding image coordinates. This correspondence was computed in the previous section using the affine coordinates α, β of a point (x, y) in the image. Discrete values α_i, β_j can be used as integer coordinates for points on the planar surface. The corresponding image point(x, y) can then be computed from the defining eq. 145 for α, β.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \alpha_i \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} + \beta_j \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \end{pmatrix} \tag{153}$$

Note that this point will in general not coincide with a discrete image point, but will fall in between four points $(i, j), (i + 1, j), (i, j + 1), (i + 1, j + 1)$, fig. 24. The value I' to be used at the surface point α_i, β_j therefore has to be interpolated from these four points. If we denote the pixel values at these points as:

$$\begin{aligned} A &= f(x_i, y_j) & B &= f(x_i, y_{j+1}) \\ C &= f(x_{i+1}, y_j) & D &= f(x_{i+1}, y_{j+1}) \end{aligned} \tag{154}$$

and the fractional coordinates as

$$d_x = \frac{x - x_i}{x_{i+1} - x_i} \qquad d_y = \frac{y - y_j}{y_{j+1} - y_j} \tag{155}$$

Note that:

$$0 \leq d_x \leq 1 \qquad 0 \leq d_y \leq 1 \tag{156}$$

We can use the weighted combination:

$$I' = (1 - d_x)(1 - d_y)A + d_x(1 - d_y)B + (1 - d_x)d_yC + d_xd_yD \tag{157}$$

which can be seen to coincide with the values of A, B, C and D respectively as the point x, y is close to the points corresponding to these values.

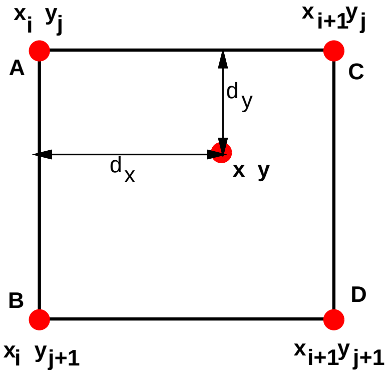

Figure 20: *Interpolation of non-integer coordinates for texture mapping*

3.3 Planar surfaces, linear mappings

Frequently we know that the 3D surface we are computing actually is a planar surface. This is common e.g when reconstructing buildings and other man made objects. In this case we can render the whole plane more efficiently, avoiding the triangulation. This is due to the fact that the projection of planes in an image can be expressed as a linear mapping in homogeneous coordinates similar to the one we encountered in the case of rotating cameras in the first chapter. Consider a perspective camera mapping:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{158}$$

Suppose the point $(X, Y, Z)^T$ is lies in a plane containing points: $(X_1, Y_1, Z_1), (X_2, Y_2, Z_2)$ and (X_3, Y_3, Z_3). As in eq. 149 we can then express it as the linear combination:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} + \alpha \begin{pmatrix} X_2 - X_1 \\ Y_2 - Y_1 \\ Z_2 - Z_1 \end{pmatrix} + \beta \begin{pmatrix} X_3 - X_1 \\ Y_3 - Y_1 \\ Z_3 - Z_1 \end{pmatrix} \tag{159}$$

which can be written as:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_2 - X_1 & X_3 - X_1 & X_1 \\ Y_2 - Y_1 & Y_3 - Y_1 & Y_1 \\ Z_2 - Z_1 & Z_3 - Z_1 & Z_1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} \tag{160}$$

by adding a trivial identity we can write this as:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} X_2 - X_1 & X_3 - X_1 & X_1 \\ Y_2 - Y_1 & Y_3 - Y_1 & Y_1 \\ Z_2 - Z_1 & Z_3 - Z_1 & Z_1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} \tag{161}$$

combining this with eq. 158 we get

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X_2 - X_1 & X_3 - X_1 & X_1 \\ Y_2 - Y_1 & Y_3 - Y_1 & Y_1 \\ Z_2 - Z_1 & Z_3 - Z_1 & Z_1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} \tag{162}$$

But this is a multiplication of a 3 × 4 matrix with a 4 × 3 matrix which is a 3 × 3 matrix. We can therefore write:

$$\boxed{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix}}$$

which states that the homogeneous image coordinates of a point on a planar surfaces and the homogeneous affine coordinates defined for the point on that surface are related by a *linear* transformation. If we can identify this linear mapping,it can be used directly to render the planar surface in 3D thus avoiding the triangulation. As we know from chapter one, the computation of a linear mapping like this requires at least four corresponding points.

As a simple consequence of this we get that the image coordinates of points on a planar surface in two different views are always related by a linear mapping. let $\mathbf{p_a} = (x^a, y^a, 1)^T$ $\mathbf{p_b} = (x^b, y^b, 1)^T$ be the image coordinates in camera a and b respectively of a point on a planar surface. We can then write:

$$\mathbf{p_a} = H_a \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} \qquad \mathbf{p_b} = H_b \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} \tag{163}$$

combining these we get:

$$\mathbf{p_a} = H_a \begin{pmatrix} \alpha \\ \beta \\ 1 \end{pmatrix} = H_a {H_b}^{-1} \mathbf{p_b} \tag{164}$$

Denoting $H_{ab} = H_a {H_b}^{-1}$ we get:

$$\boxed{\mathbf{p_a} = H_{ab} \mathbf{p_b}} \tag{165}$$

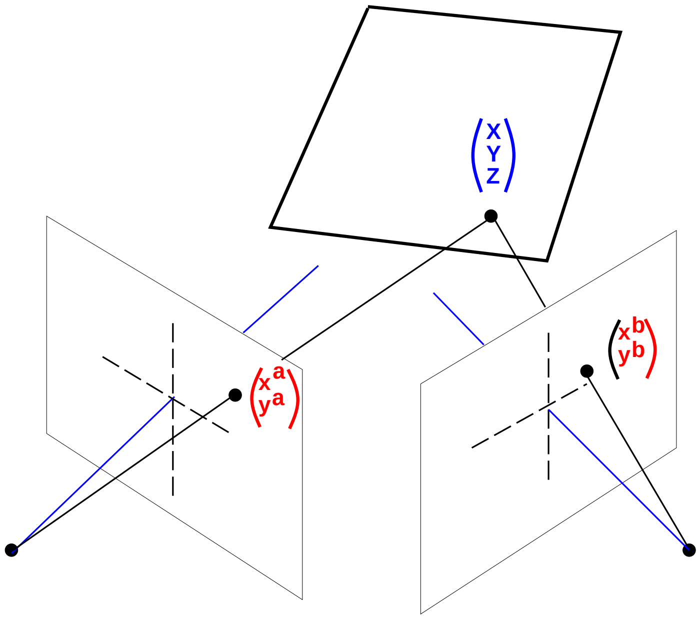

Figure 21: *Corresponding image points in two views projected from points on a planar surface are always related by a linear transformation* $\mathbf{p_a} = H\mathbf{p_b}$

This fact is of central importance in many applications involving planar surfaces. Note that it applies only when the points are on a planar surface. For more general point configurations we cannot find simple transformations like these. In that case the only relation connecting image coordinates is the epipolar constraint.

4 Multiple cameras

4.1 Perspective cameras - bundle adjustment

The procedure for calculating the camera matrix using the epipolar constraint and then the 3D reconstruction of any point by triangulation is based on *pairs* of cameras. What if we have access to more than two cameras ? If they are already calibrated there is no problem. We just stack them together to get an overdetermined system of linear equations for computing 3D coordinates of points from their image projections in each image. If they are not calibrated there are different options depending on the geometry of the viewing and wether all points are visible in all cameras or if only a part of the points are visible in each camera. If all points are visible in all cameras, we can always choose one camera as a reference and compute the camera matrices of each camera using the epipolar constraint relative to the reference camera. This procedure is essentially the best we can come up with for general perspective projection cameras.

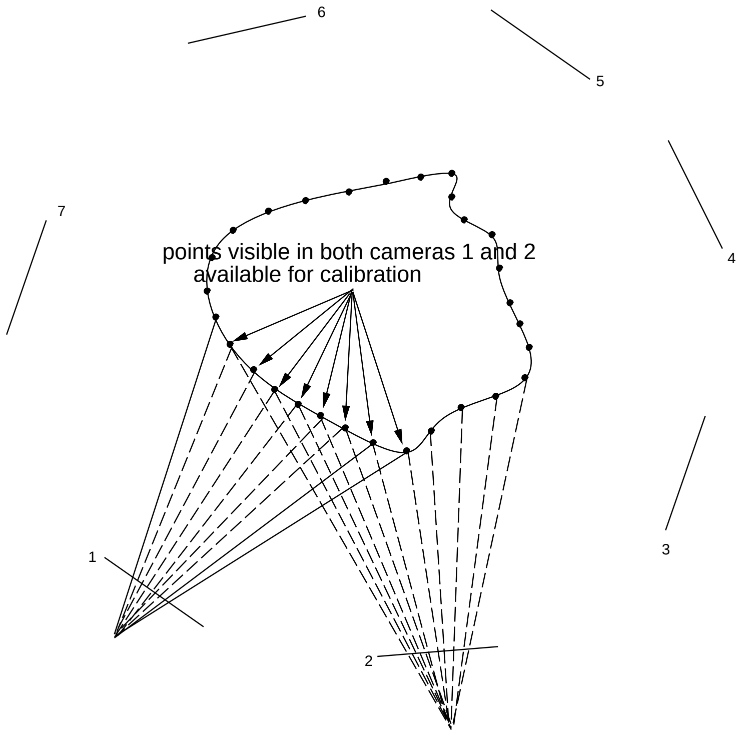
6
5
7
points visible in both cameras 1 and 2
available for calibration
4
3
1
2

Figure 22: *Multiple perspective projection cameras viewing the same object*

If on the other hand we are trying to reconstruct an object from all angles this will in general not be the case unless the object is transparant. However if we can just find a sufficient number of points visible in two nearby cameras, we can calibrate these relative to each other. We will then get F or E matrices between cameras $1-2, 2-3, \ldots k-1, k$ for k cameras. These can be used to compute orientation of the cameras relative to some reference, e.g camera 1 by just accumulating relative orientations between successive cameras. The problem with this approach is that errors in orientations are accumulated in this process and may lead to large errors in the last camera's position relative to the reference camera 1. This procedure is therefore always followed by a final adjustment process known as *bundle adjustment* wherebye all camera matrices and 3D locations of points are adjusted iteratively in order to reduce the total reprojection error in all cameras. Bundle adjustment considers the reprojection error of 3D point $(X_i, Y_i.Z_i)$ in camera j:

$$x_i^{(j)} - \frac{m_{11}^{(j)} X_i + m_{12}^{(j)} Y_i + m_{13}^{(j)} Z_i + m_{14}^{(j)}}{m_{31}^{(j)} X_i + m_{32}^{(j)} Y_i + m_{33}^{(j)} Z_i + m_{34}^{(j)}}$$

$$y_i^{(j)} - \frac{m_{21}^{(j)} X_i + m_{22}^{(j)} Y_i + m_{23}^{(j)} Z_i + m_{24}^{(j)}}{m_{31}^{(j)} X_i + m_{32}^{(j)} Y_i + m_{33}^{(j)} Z_i + m_{34}^{(j)}} \tag{166}$$

and the process tries to minimise the sum of saquared errors in all cameras over all the parameters $m_{11}^{(1)} \ldots m_{34}^{(1)} \ldots m_{11}^{(k)} \ldots m_{34}^{(k)}$, in all k cameras and coordinates $X_1, Y_1, Z_1 \ldots X_n.Y_n.Z_n$ of n 3D points:

$$\begin{aligned} \min \quad & \sum_{i=1}^{i=n} \sum_{j=1}^{j=k} (x_i^{(j)} - \frac{m_{11}^{(j)} X_i + m_{12}^{(j)} Y_i + m_{13}^{(j)} Z_i + m_{14}^{(j)}}{m_{31}^{(j)} X_i + m_{32}^{(j)} Y_i + m_{33}^{(j)} Z_i + m_{34}^{(j)}})^2 + \\ + \quad & \sum_{i=1}^{i=n} \sum_{j=1}^{j=k} (y_i^{(j)} - \frac{m_{21}^{(j)} X_i + m_{22}^{(j)} Y_i + m_{23}^{(j)} Z_i + m_{24}^{(j)}}{m_{31}^{(j)} X_i + m_{32}^{(j)} Y_i + m_{33}^{(j)} Z_i + m_{34}^{(j)}})^2 \end{aligned} \tag{167}$$

Note that summation is of course only over points i that are visible in camera j This is a very complex non-linear minimisation procedure and care must be taken to avoid local minima. The best way to do this is of course to start off the iterations with a good solution as possible from the accumulated relative orientations.

4.2 Parallel projection cameras

It turns out however that for *parallel projection* cameras there is a more efficient direct procedure for computing camera matrices and 3D coordinates of points using all image coordinates from all cameras in one single computational step. This is based of direct factorization of camera matrices and 3D coordinates using the singular value decomposition.

Parallel projection can be considered as a special case of perspective projection when the camera center is moved infinitely far away from the 3D points being imaged. In practise it can be considered as a valid approximation to the projection whenever the spread of points is small compared to the distance between the point center and the camera center. This situation is very common when a certain object with a finite extent is imaged. The general relation between image and 3D coordinates of points for parallel projection is just a special case of the perspective projection equation 55 where the last row of the projection matrix is replaced by $(0, 0, 0, 1)$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{168}$$

The third linear relation in this equation is just a trivial identity and can be deleted.

We therefore get the parallel projection equation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{169}$$

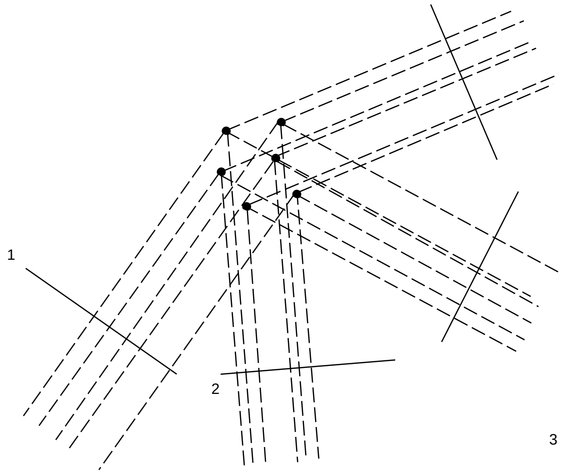

Figure 23: *Multiple parallel projection cameras viewing the same object*

Note that contrary to perspective projection, there is no longer any arbitrary scale factor associated with this equation. The elements $c_{i,j}$ of the parallel projection matrix can be determined accurately and not just up to an arbitrary scale factor.

4.3 Factorization of cameras and 3D coordinates

If we have a set of n points (X_i, Y_i, Z_i) with $i = 1 \dots n$ with image coordinates $(x_i, y_i,)$ we can compute the centroids in 3D and image by taking averages:

$$\begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} = \frac{1}{n} \sum_{i=1}^{i=n} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \qquad \begin{pmatrix} \bar{X} \\ \bar{Y} \\ \bar{Z} \end{pmatrix} = \frac{1}{n} \sum_{i=1}^{i=n} \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} \tag{170}$$

It is easy to see that the image centroid is the projection of the 3D centroid:

$$\begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{pmatrix} \begin{pmatrix} \bar{X} \\ \bar{Y} \\ \bar{Z} \\ 1 \end{pmatrix} \tag{171}$$

Taking the difference between equations 169 and 171 we get:

$$\begin{pmatrix} x - \bar{x} \\ y - \bar{y} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{pmatrix} \begin{pmatrix} X - \bar{X} \\ Y - \bar{Y} \\ Z - \bar{Z} \\ 1 - 1 \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} \begin{pmatrix} X - \bar{X} \\ Y - \bar{Y} \\ Z - \bar{Z} \end{pmatrix} \tag{172}$$

If we define:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x - \bar{x} \\ y - \bar{y} \end{pmatrix} \qquad \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{pmatrix} X - \bar{X} \\ Y - \bar{Y} \\ Z - \bar{Z} \end{pmatrix} \tag{173}$$

we can write:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} \tag{174}$$

Now consider n points in 3D and their 2D image projections. If we stack coordinates together into rectangular $3 \times n$ and $2 \times n$ matrices respectively we can write:

$$\begin{pmatrix} x'_1, & x'_2 & \dots & x'_n \\ y'_1, & y'_2 & \dots & y'_n \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} \begin{pmatrix} X'_1, & X'_2 & \dots & X'_n \\ Y'_1, & Y'_2 & \dots & Y'_n \\ Z'_1, & Z'_2 & \dots & Z'_n \end{pmatrix} \tag{175}$$

This relation is valid for all points viewed in a specific camera. We can also consider multiple cameras. Every camera then gives rise to a relation like the one above. If we have m different cameras, we get m different relations of this kind. Since each relation has 2 equations we get in total $2m$ equations. We can stack all image coordinates into one large $2m \times n$ matrix and all the camera matrices into one large $2m \times 3$ matrix. We then get the relation between the matrices describing image measurements,cameras and 3D coordinates:

$$\begin{pmatrix} x_1'^{(1)}, & x_2'^{(1)} & \dots & x_n'^{(1)} \\ y_1'^{(1)}, & y_2'^{(1)} & \dots & y_n'^{(1)} \\ & & & \\ x_1'^{(2)}, & x_2'^{(2)} & \dots & x_n'^{(2)} \\ y_1'^{(2)}, & y_2'^{(2)} & \dots & y_n'^{(2)} \\ & & & \\ & & \vdots & \\ & & & \\ x_1'^{(m)}, & x_2'^{(m)} & \dots & x_n'^{(m)} \\ y_1'^{(m)}, & y_2'^{(m)} & \dots & y_n'^{(m)} \end{pmatrix} = \begin{pmatrix} c_{11}^{(1)} & c_{12}^{(1)} & c_{13}^{(1)} \\ c_{21}^{(1)} & c_{22}^{(1)} & c_{23}^{(1)} \\ & & \\ c_{11}^{(2)} & c_{12}^{(2)} & c_{13}^{(2)} \\ c_{21}^{(2)} & c_{22}^{(2)} & c_{23}^{(2)} \\ & & \\ & \vdots & \\ & & \\ c_{11}^{(m)} & c_{12}^{(m)} & c_{13}^{(m)} \\ c_{21}^{(m)} & c_{22}^{(m)} & c_{23}^{(m)} \end{pmatrix} \begin{pmatrix} X'_1, & X'_2 & \dots & X'_n \\ Y'_1, & Y'_2 & \dots & Y'_n \\ Z'_1, & Z'_2 & \dots & Z'_n \end{pmatrix} \tag{176}$$

If we call these matrices W, C and P^T respectively we have:

$$W = CP^T$$

We see that the problem of computing camera matrices C and 3D coordinates P is a problem of *factorization* of the data matrix W Just as in the projective case we will see that given a data matrix W built from image coordinates of n points in m images, there is no unique way to factorise it into matrices C and P. This follows from the trivial observation that given certain $2m \times 3$ matrix C and $3 \times n$ matrix P we can choose an arbitrary invertible 3×3 matrix Q such that:

$$W = CP^T = CQQ^{-1}P^T$$

which means that the matrices CQ and $Q^{-1}P^T$ are just as valid solutions as C and P^T. Since the matrix Q has in general 9 degrees of freedom, we can always find a Q such that the matrices CQ and $Q^{-1}P^T$ have *orthogonal* columns and rows respectively. If we choose such a matrix Q and denote by C^* and P^{*T} the matrices obtained by normalising to unit length the columns and rows of the matrices CQ and $Q^{-1}P^T$ respectively we can write:

$$W \; = \; CP^T \; = \; CQQ^{-1}P^T \; = \; C^*DP^{*T}$$

where

$$C^{*T}C^* \; = \; I \qquad P^{*T}P^* \; = \; I$$

and D is a diagonal matrix inserted in order to compensate for the length normalization of C^* and P^{*T} Compare this to the expression for the singular value decomposition for an arbitrary $2m \times n$ matrix W into an $2m \times n$ matrix U with orthogonal columns and and orthogonal $n \times n$ matrices Σ and V where Σ is diagonal:

$$W \; = \; U\Sigma V^T$$

where

$$U^TU \; = \; I \qquad V^TV \; = \; I$$

These expressions are identical provided the diagonal matrix Σ has zero entries from the 4:th row and column and onwards. I.e it has to have the structure:

$$\Sigma \; = \; \begin{pmatrix} \sigma_1, & 0 & 0 & 0 & \ldots & 0 \\ 0, & \sigma_2 & 0 & 0 & \ldots & 0 \\ 0, & 0 & \sigma_3 & 0 & \ldots & 0 \\ 0, & 0 & 0 & 0 & \ldots & 0 \\ & & & & & \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ & & & & & \\ 0, & 0 & 0 & & \ldots & 0 \end{pmatrix}$$

By computing the SVD of the data matrix W we therefore expect it to have rank 3 as seen above. In practise, due to noise in the measured image coordinates this will not happen exactly. The remaining singular values from 4 and onwards will not be exactly zero. By taking the 3 first singular values of the matrix Σ together with the associated 3 column vectors of the matrix U and the associated 3 row vectors of the matrix V^T we can identify these with matrices C^*, D and P^{*T} respectively from the relation

$$W \; = \; U\Sigma V^T \; = \; C^*DP^{*T}$$

4.4 Affine structure

We therefore have a procedure for calculating camera matrices C^* and 3D coordinates P^* from the image coordinate data matrix W built from image coordinates of n points in m views. Given these example camera matrices, and 3D coordinates we can obtain all the possible solutions for cameras and points by multiplication with an arbitrary non-singular 3×3 matrix A

$$C \; = \; C^*A \qquad P^T \; = \; A^{-1}P^{*T}$$

Each individual camera matrix is then obtained as:

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} \; = \; \begin{pmatrix} c^*_{11} & c^*_{12} & c^*_{13} \\ c^*_{21} & c^*_{22} & c^*_{23} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \qquad (177)$$

Or equivalently using the complete camera matrix:

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} c^*_{11} & c^*_{12} & c^*_{13} & c^*_{14} \\ c^*_{21} & c^*_{22} & c^*_{23} & c^*_{24} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{178}$$

where b_1, b_2, b_3 is an arbitrary vector. This is an example of an *affine* transformation which is a special case of general linear transformations. Any affine transformation matrix applied to the computed camera matrices will give an equally valid solution for the cameras. This is the equivalent of the case for the perspective camera where solutions could only be obtained up to a general linear transformation.

The solutions for the 3D coordinates of the points will have the same ambiguity as the solutions from the cameras. We get for the 3D points:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} X^* \\ Y^* \\ Z^* \\ 1 \end{pmatrix} \tag{179}$$

Affine transformations have the property however of being a subgroup of linear transformations. This means that the inverse of any affine transformation is itself an affine transformation. The ambiguity for the solution of 3D coordinates can therefore be expressed as

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & b'_1 \\ a'_{21} & a'_{22} & a'_{23} & b'_2 \\ a'_{31} & a'_{32} & a'_{33} & b'_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X^* \\ Y^* \\ Z^* \\ 1 \end{pmatrix} \tag{180}$$

where the affine transformation defined by the elements $a'_{i,j}$ and b'_i are arbitrary. The solution for the 3D points can therefore be computed up to an arbitrary affine transformation only. We say that we compute the *affine structure* of the point set.

5 Numerical computations

5.1 The singular value decomposition

In the sections describing linear least squares estimation, we started with the non-homogeneous system of equations:

$$A\mathbf{x} \;=\; b \tag{181}$$

formulating the solution of $\mathbf{x}$ as a least squares problem led to the equations:

$$(A^T A)\mathbf{x} \;=\; A^T b \tag{182}$$

which in principle can be solved by inverting the symmetric matrix $(A^T A)$. Note that the matrix A is in general not a square matrix and does not have an inverse. The actual computation of matrix inverses is most often done in standard software libraries containing numerical routines, such as MATLAB where the user supplies the input data A and b and the numerical routines returns the solution $\mathbf{x}$. There are various alternative ways of computing matrix inverses, some of which are better from a numerical standpoint than others. Frequently the most efficient routines make us of the so called *singular value decomposition (SVD)* of a matrix. This is a way to decompose a matrix into products of other matrices which have a particularly simple structure. Using this structure, manipulations of matrices such as taking the inverse and splitting up into factors, becomes particularly simple. We will start by considering the SVD for symmetric matrices of the type $(A^T A)$ and then see how it can be generalised to arbitrary, even rectangular matrices A.

5.1.1 Symmetric matrices

Consider the symmetric $n \times n$ matrix $(A^T A)$. Suppose it has the set of eigenvectors $\mathbf{v_i}$ and with eigenvalues λ_i

$$A^T A\mathbf{v_i} \;=\; \lambda_i \mathbf{v_i} \qquad i = 1 \ldots n \tag{183}$$

If we collect all eigenvectors into one matrix we can write all these relations as a single matrix multiplication:

$$A^T A(\mathbf{v_1} \ldots \mathbf{v_n}) \;=\; (\lambda_1 \mathbf{v_1} \ldots \lambda_\mathbf{n} \mathbf{v_n}) \;=\; (\mathbf{v_1} \ldots \mathbf{v_n}) \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \tag{184}$$

Note that we have written the right hand side of the equation as a product of the eigenvector matrix and a diagonal matrix. This is a simple example of *matrix factorization* which we will encounter frequently in practical applications.

The eigenvectors $\mathbf{v_i}$ of the symmetric matrix $(A^T A)$. have the property of being mutually orthogonal. Consider any symmetric matrix Q with eigenvectors $\mathbf{v_i}$ and with eigenvalues λ_i

$$\begin{aligned} Q\mathbf{v_i} \;&=\; \lambda_i \mathbf{v_i} \\ Q\mathbf{v_j} \;&=\; \lambda_j \mathbf{v_j} \end{aligned} \tag{185}$$

Multiplying these relations by the vectors $\mathbf{v_j}$ and $\mathbf{v_i}$ respectively, we get:

$$\mathbf{v_j}^T Q\mathbf{v_i} \;=\; \lambda_i \mathbf{v_j}^T \mathbf{v_i}$$

$$\mathbf{v_i}^T Q \mathbf{v_j} \;=\; \lambda_j \mathbf{v_i}^T \mathbf{v_j} \tag{186}$$

Since Q is symmetric, the left hand sides of these equations are actually equal. This can be seen by rewriting:

$$\mathbf{v_j}^T Q \mathbf{v_i} \;=\; (Q\mathbf{v_i})^T \mathbf{v_j} \;=\; \mathbf{v_i}^T Q^T \mathbf{v_j} \;=\; \mathbf{v_i}^T Q \mathbf{v_j} \tag{187}$$

where we have made use of the general relation $(AB)^T \;=\; A^T B^T$ This means that the right hand sides of the equations must be equal too:

$$\lambda_i \mathbf{v_j}^T \mathbf{v_i} \;=\; \lambda_j \mathbf{v_i}^T \mathbf{v_j} \tag{188}$$

since inner products of vectors is independent of the order of the vectors, we have $\mathbf{v_j}^T \mathbf{v_i} \;= \mathbf{v_i}^T \mathbf{v_j}$ and:

$$(\lambda_i -\; \lambda_j) \mathbf{v_i}^T \mathbf{v_j} \;=\; 0 \tag{189}$$

From this we conclude that for eigenvectors with *unequal* eigenvalues $\lambda_i \neq \lambda_j$ we must have:

$$\mathbf{v_i}^T \mathbf{v_j} \;=\; 0 \tag{190}$$

All the columns of the matrix $V = (\mathbf{v_1} \ldots \mathbf{v_n})$ will therefore be mutually orthogonal, and the matrix itself is therefore called orthogonal. Since eigenvectors are always only defined up to arbitrary length, we can always normalize them to unity:

$$\mathbf{v_i}^T \mathbf{v_i} \;=\; 1 \tag{191}$$

This means that the matrix is orthogonal:

$$V^T V \;=\; (\mathbf{v_1} \ldots \mathbf{v_n})^T (\mathbf{v_1} \ldots \mathbf{v_n}) \;=\; I \tag{192}$$

from which follows:

$$V^T \;=\; V^{-1} \tag{193}$$

Consider the relation 184. This can be written more compactly as:

$$A^T A V \;=\; V \Lambda \tag{194}$$

where we have defined the diagonal matrix:

$$\Lambda \;=\; \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \tag{195}$$

Multiplying both sides with $V^{-1} = V^T$ we get:

$$\boxed{A^T A \;=\; V \Lambda V^T} \tag{196}$$

This is a general decomposition of a symmetric matrix into orthogonal and a diagonal matrix. If we know the eigenvectors and eigenvalues of a symmetric matrix we can use this decomposition for various problems. The inverse e.g is easily computed as:

$$(A^T A)^{-1} \;=\; (V \Lambda V^T)^{-1} \;=\; (\Lambda V^T)^{-1} V^{-1} \;=\; {V^T}^{-1} \Lambda^{-1} V^{-1} \;=\; V \Lambda^{-1} V^T \tag{197}$$

where we have made frequent use of the general relations $(AB)^T \;=\; A^T B^T$ and $(AB)^{-1} \;=\; A^{-1} B^{-1}$ for two matrices A and B

Since the matrix Λ^{-1} is simply the diagonal matrix with elements λ^{-1} this means that the inverse of $A^T A$ can easily be expressed once it's decomposition 196 is known.

5.1.2 General matrices

Consider a general rectangular matrix A with m rows and n columns. We will assume that $m \geq n$. From this we can form the $n \times n$ symmetric matrix $A^T A$ and the $m \times m$ symmetric matrix AA^T.

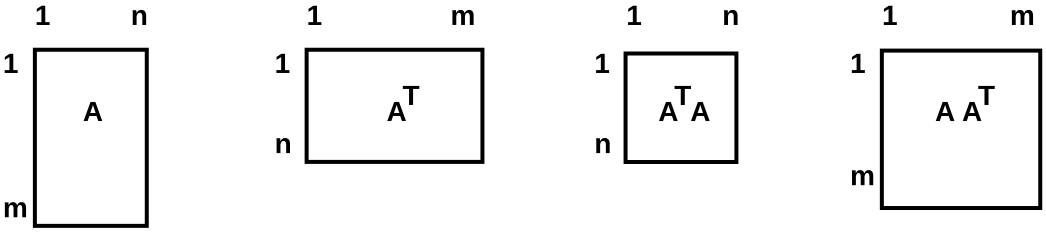

Figure 24: *Sizes of various matrices*

These matrices have eigenvectors and eigenvalues as:

$$\begin{aligned} A^T A \mathbf{v_i} &= \lambda_i \mathbf{v_i} \qquad i = 1 \ldots n \\ AA^T \mathbf{u_i} &= \mu_i \mathbf{u_i} \qquad i = 1 \ldots m \end{aligned} \tag{198}$$

The eigenvalues of the two matrices are of course related. The n first eigenvalues of the $m \times m$ matrix AA^T are actually exactly those of the $n \times n$ matrix $A^T A$. Multiply the second equation above with A^T and we get:

$$A^T AA^T \mathbf{u_i} = \mu_i A^T \mathbf{u_i} \tag{199}$$

Obviously the vector $A^T \mathbf{u_i}$ is an eigenvector of the matrix $A^T A$ with eigenvalue μ_i. We therefore have: $\mu_i = \lambda_i$ for $i = 1 \ldots n$. Note that we had $m \geq n$ so what about $\mu_{n+1} \ldots \mu_m$? Consider the matrix A^T. It consists of n row vectors $a_1 \ldots a_n$ of dimension m, i.e $a_i \in R^m$. n vectors cannot span the whole of m dimensional space, i.e all vectors of R^m cannot be expressed in a linear combination of the vectors $a_1 \ldots a_n$. We can always complete this set of vectors with vectors $a'_{n+1} \ldots a'_m$ orthogonal to all vectors a_i, i.e:

$$a_i^T a'_j = 0 \qquad i = 1 \ldots n \quad j = n+1 \ldots m \tag{200}$$

This means:

$$A^T a'_j = 0 \qquad j = n+1 \ldots m \tag{201}$$

and also:

$$AA^T a'_j = 0 \qquad j = n+1 \ldots m \tag{202}$$

The matrix AA^T therefore have a set of $m - n$ eigenvectors a'_j with eigenvalues 0. The situation can be summarized as:

$$\begin{aligned} A^T A \mathbf{v_i} &= \lambda_i \mathbf{v_i} \qquad i = 1 \ldots n \\ AA^T \mathbf{u_i} &= \lambda_i \mathbf{u_i} \qquad i = 1 \ldots n \\ AA^T \mathbf{u_i} &= 0 \qquad i = n+1 \ldots m \end{aligned} \tag{203}$$

The eigenvectors u_i and v_i are of course also related. In order to see this we assume that they are normalised to unit length:

$$\mathbf{u_i}^T \mathbf{u_i} \;=\; 1 \qquad \mathbf{v_i}^T \mathbf{v_i} \;=\; 1 \tag{204}$$

If we multiply the two eigenvector equations with A and A^T respectively:

$$\begin{aligned} AA^T A\mathbf{v_i} &= \lambda_i A\mathbf{v_i} \qquad i = 1 \ldots n \\ A^T AA^T \mathbf{u_i} &= \lambda_i A^T \mathbf{u_i} \qquad i = 1 \ldots n \end{aligned} \tag{205}$$

we find again that we must have:

$$\begin{aligned} A\mathbf{v_i} &= \alpha_i \mathbf{u_i} \qquad i = 1 \ldots n \\ A^T \mathbf{u_i} &= \beta_i \mathbf{v_i} \qquad i = 1 \ldots n \end{aligned} \tag{206}$$

for some α_i and β_i. Taking Euclidean norms and remembering that the eigenvectors u_i and v_i are normalized to unit length, we get:

$$\begin{aligned} |A\mathbf{v_i}|^2 &= \mathbf{v_i}^T A^T A\mathbf{v_i} = \lambda_i \mathbf{v_i}^T \mathbf{v_i} = \lambda_i = |\alpha_i \mathbf{u_i}|^2 = \alpha_i^2 \mathbf{u_i}^T \mathbf{u_i} = \alpha_i^2 \\ |A^T \mathbf{u_i}|^2 &= \mathbf{u_i}^T AA^T \mathbf{u_i} = \lambda_i \mathbf{u_i}^T \mathbf{u_i} = \lambda_i = |\beta_i \mathbf{v_i}|^2 = \beta_i^2 \mathbf{v_i}^T \mathbf{v_i} = \beta_i^2 \end{aligned} \tag{207}$$

We can therefore write:

$$\alpha_i \;=\; \beta_i \;=\; \sigma_i \qquad \sigma_i^2 \;=\; \lambda_i \tag{208}$$

and

$$\begin{aligned} A\mathbf{v_i} &= \sigma_i \mathbf{u_i} \qquad i = 1 \ldots n \\ A^T \mathbf{u_i} &= \sigma_i \mathbf{v_i} \qquad i = 1 \ldots n \end{aligned} \tag{209}$$

The values σ are called the *singular values* associated with the matrix A and the vectors $\mathbf{u_i}$ and $\mathbf{v_i}$ are the left and right singular vectors respectively. If we define the matrices:

$$U = (\mathbf{u_1} \ldots \mathbf{u_n}) \qquad V = (\mathbf{v_1} \ldots \mathbf{v_n}) \qquad \Sigma \;=\; \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix} \tag{210}$$

we can write the first set of equations 209 as:

$$AV \;=\; U\Sigma \tag{211}$$

Using the orthogonality relations $VV^T \;= I$: we have:

$$\boxed{A \;=\; U\Sigma V^T} \tag{212}$$

Which is the *singular value decomposition (SVD)* of a general rectangular $m \times n$ matrix A ($m > n$) into orthogonal $m \times n$ matrix U, orthogonal $n \times n$ matrix V and diagonal $n \times n$ matrix Σ containing the *singular values* σ_i. The singular values are related to the eigenvalues λ of the symmetric matrix $A^T A$ as $\sigma_i^2 \;=\; \lambda_i$.

5.2 Solving linear systems of equations

5.2.1 Non-homogeneous equations

Going back to the system of linear equations:

$$A\mathbf{x} \;=\; b \tag{213}$$

where A in $m\times$ with $m > n$, we can apply the SVD eq. 212 of A;

$$U\Sigma V^T\mathbf{x} \;=\; b \tag{214}$$

If we multiply both sides with the matrix $V\Sigma^{-1}U^T$

$$V\Sigma^{-1}U^TU\Sigma V^T\mathbf{x} \;=\; V\Sigma^{-1}U^Tb \tag{215}$$

and use the orthogonality properties $U^TU = I, VV^T \;=\; I$ we get:

$$\mathbf{x} \;=\; V\Sigma^{-1}U^Tb \tag{216}$$

i.e. by computing the singular values σ_i and vectors $\mathbf{u}_i, \mathbf{v}_i$ we can directly express the solution $\mathbf{x}$ of the system $A\mathbf{x} \;=\; b$. The rectangular matrix $V\Sigma^{-1}U^T$ obviously plays the role of an inverse to the rectangular matrix A. This kind of pseudo inverse was derived previously in eq. 62

$$\mathbf{x} \;\;=\; (A^TA)^{-1}A^Tb \tag{217}$$

as the solution of $\mathbf{x}$ that minimises the residual $||A\mathbf{x} - b||^2$. Interestingly, these two solutions are actually the same. This can be seen by writing:

$$A^TA \;=\; (U\Sigma V^T)^TU\Sigma V^T \;=\; V\Sigma U^TU\Sigma V^T \;=\; V\Sigma^2V^T \tag{218}$$

Taking the inverse, we get using the orthogonality property $V^{-1} \;=\; V^T$

$$(A^TA)^{-1} \;=\; (V\Sigma^2V^T)^{-1} \;=\; (V^T)^{-1}\Sigma^{-2}V^{-1} \;=\; V\Sigma^{-2}V^T \tag{219}$$

and the pseudo inverse can be computed as:

$$(A^TA)^{-1}A^T \;=\; V\Sigma^{-2}V^TV\Sigma U^T \;=\; V\Sigma^{-1}U^T \tag{220}$$

which is exactly the "inverse" derived using the SVD. This nice property is very useful in numerical calculations.

The singular values of a matrix A is important also in understanding the accuracy of the solution of a linear system. Problems of numerical instability can most often be attributed to the singular value structure of A. Consider the exact system:

$$A\mathbf{x} \;=\; b \tag{221}$$

and then the solution obtained if noise is added the vector b:

$$A(\mathbf{x} + \delta\mathbf{x}) \;=\; b + \delta b \tag{222}$$

if we use the pseudoinverse solutions:

$$\mathbf{x} \;=\; V\Sigma^{-1}U^Tb \qquad \mathbf{x} + \delta\mathbf{x} \;=\; V\Sigma^{-1}U^T(b + \delta b) \tag{223}$$

we get by subtraction:

$$\delta \mathbf{x} \;=\; V \Sigma^{-1} U^T \delta b \tag{224}$$

Denoting $U^T \delta b \;=\; b'$ we have:

$$\delta \mathbf{x} \;=\; V \Sigma^{-1} b' \;=\; \sum_1^n \frac{\delta b_i}{\sigma_i} \mathbf{v}_i \tag{225}$$

This is the error in the solution of x expressed as a linear combination of the orthogonal vectors $\mathbf{v}_i$ Each component in this sum is proportional to the noise component δb_i but also to the inverse of the singular value $1/\sigma_i$ This means that singular values close to 0 will lead to large errors in the vector x. A singular value of 0 actually gives infinite amplification of noise. The matrix is in that case rank deficient or singular and the solution to the linear system is no longer unique.

5.2.2 Homogeneous equations

The set of linear homogeneous equations:

$$A\mathbf{x} \;= 0 \tag{226}$$

will be solved by vectors v_i having singular values $\sigma_i = 0$ This follows from the definition of the singular values in eq. 209. Note that we get non-unique solutions whenever more than one singular value is 0. If the singular values $\sigma_1 = 0 \ldots \sigma_k = 0$ we can take any linear combination of associated singular vectors as a solution:

$$A \sum_1^k c_i \mathbf{v}_i \;=\; \sum_1^k c_i A \mathbf{v}_i = 0 \tag{227}$$

The number of singular values = 0 therefore characterizes the uncertainty or degeneracy of the solution. An $m \times n$ matrix with singular values $\sigma_1 = 0 \ldots \sigma_k = 0$ is said to have $rank(n-k)$

A similar characterization of the uncertainty of the solution is possible in case the data is noisy. The matrix A will then be distorted so that the least singular value will be > 0. In that case we formulated the least squares problem as finding the solution to:

$$\min_{||\mathbf{x}||^2 = 1} \quad ||A\mathbf{x}||^2 \tag{228}$$

where the solution was the eigenvector:

$$A^T A \mathbf{x} \;=\; \lambda \mathbf{x} \tag{229}$$

Choosing the eigenvector u_1 with the least eigenvalue λ_1 will then give us the global minimum of the constrained cost function. This is equivalent to choosing the singular vector of A with the least singular value $\sigma_1 > 0$ since $\sigma_i^2 \;=\; \lambda_i$.

Just as in the case of non-homogeneous equations, the numerical accuracy of this solution depend on the singular value structure of the matrix A. Any vector x can be expressed as a linear combination

$$\mathbf{x} \;=\; \sum_1^n c_i \mathbf{v}_i \tag{230}$$

We have:

$$A\mathbf{x} \;=\; \sum_{1}^{n} c_i A\mathbf{v}_i \;=\; \sum_{1}^{n} c_i \sigma_i \mathbf{v}_i \tag{231}$$

The cost function can then be expressed as:

$$||A\mathbf{x}||^2 \;=\; ||\sum_{1}^{n} c_i \sigma_i \mathbf{v}_i||^2 \;=\; \sum_{1}^{n} c_i^2 \sigma_i^2 \tag{232}$$

where the last equality follows from the fact that the vectors v_i form a orthonormal system, i.e $v_i^T v_j = 0$ for $i \neq j$ and $v_i^T v_i = 1$ for all i And the normalisation criterion becomes:

$$||\mathbf{x}||^2 \;=\; ||\sum_{1}^{n} c_i \mathbf{v}_i||^2 \;=\; \sum_{1}^{n} c_i^2 \;=\; 1 \tag{233}$$

The effect of the singular values on the numerical accuracy of the solution to the least squares problem can be seen from the formulation of the cost function eq. 232 in the coefficients c_i. A small value of σ_i will allow a high value of the corresponding coefficient c_i without increasing the cost function. Several small values of σ_i therefore leads to a large set of vectors x having a small value of the cost function, i.e the best solution is ill-defined. In the extreme case with several singular values equal to 0 we get the complete solution set of eq. 227

5.3 Estimation of E and F matrices

The epipolar constraint 94 can be expressed using the essential matrix with elements $e_{i,j}$ or fundamental matrix with elements $f_{i,j}$:

$$(\, x^b \;\; y^b \;\; 1 \,) \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} x^a \\ y^a \\ 1 \end{pmatrix} \;=\; 0$$

Which can be written as the linear constraint equation:

$$x^b y^a f_{11} + y^b x^a f_{12} + y^b y^a f_{21} + x^b y^b f_{22} + x^a f_{13} + y^a f_{23} + x^b f_{31} + y^b f_{32} + f_{33} = 0$$

Given n corresponding points in two images we get the homogeneous linear system:

$$\begin{pmatrix} x_1^b y_1^a & y_1^b x_1^a & y_1^b y_1^a & x_1^b y_1^b & x_1^a & y_1^a & x_1^b & y_1^b & 1 \\ x_2^b y_2^a & y_2^b x_2^a & y_2^b y_2^a & x_2^b y_2^b & x_2^a & y_2^a & x_2^b & y_2^b & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^b y_n^a & y_n^b x_n^a & y_n^b y_n^a & x_n^b y_n^b & x_n^a & y_n^a & x_n^b & y_n^b & 1 \end{pmatrix} \begin{pmatrix} f_{11} \\ f_{12} \\ f_{21} \\ f_{22} \\ f_{13} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} \;=\; 0$$

Calling the left side matrix A and the vector with F-matrix elements $\mathbf{f}$ we have:

$$A\mathbf{f} \;=\; 0 \tag{234}$$

Computing the F-matrix is therefore achieved by finding the singular vector associated with the smallest singular value of the matrix A. Since the vector $\mathbf{f}$ has nine homogeneous components we need at least 8 points in order to compute a unique solution up to an arbitrary scale factor.

5.3.1 Normalisation of coordinates

It has been found in practise that for image points with large values of coordinates, the numerical accuracy of estimating the vector $\mathbf{f}$ is very poor. Note that in this case the elements of the matrix A will be either very large (the bilinear elements such as $x^a y^b$) or small (the linear elements x^a etc.). This is the source of the numerical difficulties encountered. Ideally the coordinates would spread around the origin of the image coordinate systems in the two images. We can always ensure this by computing $\mathbf{f}$ using transformed image coordinates and then compensate for the transformation. The epipolar constraint for F-matrix can be written compactly as:

$$\mathbf{p_a}^T F \mathbf{p_b} \;=\; 0 \tag{235}$$

where $\mathbf{p}$ denotes the homogeneous image coordinate vector $(x, y, 1)^T$ If we apply linear transformations T_a and T_b to the image coordinates

$$\mathbf{q_a} \;=\; T_a \mathbf{p_a} \qquad \mathbf{q_b} \;=\; T_b \mathbf{p_b} \tag{236}$$

we can write the epipolar constraint using the transformed coordinates as:

$$\mathbf{p_a}^T F \mathbf{p_b} \;=\; \mathbf{q_a}^T (T_a^{-1})^T F T_b^{-1} \mathbf{q_b} \;=\; \mathbf{q_a}^T (T_a^T)^{-1} F T_b^{-1} \mathbf{q_b} \;=\; \mathbf{q_a}^T F' \mathbf{q_b} \;=\; 0 \tag{237}$$

In the last equality we made use of the fact that $(T_a^{-1})^T \;=\; (T_a^T)^{-1}$ If we now solve for the F-matrix using the transformed image coordinates $\mathbf{q_a}$ and $\mathbf{q_b}$ we will compute the matrix

$$F' \;=\; (T_a^T)^{-1} F T_b^{-1} \tag{238}$$

The correct F-matrix is then easily computed by applying the transformation:

$$F \;=\; T_a^T F' T_b \tag{239}$$

The main idea now is to choose the transformations T_a and T_b such that the transformed coordinates $\mathbf{q_a}$ and $\mathbf{q_b}$ become "well behaved" in the sense that the computation of the F.-matrix is numerically stable. Ideally the values of the coordinates should spread around the origin which can be achieved simply by letting the transformations correspond to the process of moving the center of gravity of all points $p_{a1} \ldots p_{an} \quad p_{b1} \ldots p_{bn}$ to the origin. For that purpose we define the c.g:s of the two point sets as:

$$p_a^{cg} \;=\; \frac{\sum_{i=1}^{n} p_{ai}}{n} \qquad p_b^{cg} \;=\; \frac{\sum_{i=1}^{n} p_{bi}}{n} \tag{240}$$

and define the coordinates $\mathbf{q_a}$ and $\mathbf{q_b}$ as:

$$q_{ai} \;=\; p_{ai} \;-\; p_a^{cg} \qquad q_{bi} \;=\; p_{bi} \;-\; p_b^{cg} \tag{241}$$

which corresponds to the choice of transformation matrices as:

$$T_a \;=\; \begin{pmatrix} 1 & 0 & -x_a^{cg} \\ 0 & 1 & -y_a^{cg} \\ 0 & 0 & 1 \end{pmatrix} \qquad T_b \;=\; \begin{pmatrix} 1 & 0 & -x_b^{cg} \\ 0 & 1 & -x_b^{cg} \\ 0 & 0 & 1 \end{pmatrix} \tag{242}$$

This normalisation procedure can be applied to other cases where the system matrix is built up from expressions of image coordinates, This is the case e.g for the computation of the linear mapping H previously considered.

5.3.2 Imposing singularity constraints

If $\mathbf{t} = (X_0, Y_0, Z_0)^T$ is the translation of the camera center of camera b from the origin, we demonstrated in eq. 99 that the essential matrix has the property

$$E\mathbf{t} = 0 \tag{243}$$

Defining the vector $\mathbf{t}' = K_a\mathbf{t}$ we see from eq. 126 that we have:

$$F\mathbf{t}' = 0 \tag{244}$$

These two relations demonstrate that the E- and F-matrices have non-trivial nullvectors,i.e they have at least one zero singular value. The matrices E and F are therefore singular matrices. This has the consequence that their determinants vanishes, i.e:

$$det(E) = det(F) = 0 \tag{245}$$

The determinant of F can be computed as:

$$det \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} = \tag{246}$$

$$= f_{11}(f_{22}f_{33} - f_{23}f_{32}) - f_{21}(f_{12}f_{33} - f_{13}f_{32}) + f_{31}(f_{12}f_{23} - f_{13}f_{22}) = 0 \tag{247}$$

This in a *non-linear* constraint that is always satisfied by E- and F-matrices. In case these matrices are computed by a least squares method with noisy image coordinates, there is no guarantee that this constraint is fulfilled. In general it is not. In principle this constraint should be added just as the normalisation constraint $||\mathbf{f}|| = 1$ in the minimisation procedure. This will give a quite complex minimisation problem where we would have to solve a set of non-linear equations however. A simpler way is to impose it after the F-matrix has been computed using the SVD. The F-matrix itself can always be decomposed using the SVD as:

$$F = U\Sigma V^T \tag{248}$$

The singular values are:

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix} \tag{249}$$

With perfect noise free data, F would be singular, i.e $\sigma_1 = 0$ In practise this is not the case. An algorithm for imposing the singularity constraint on the F-matrix is then:

1. Compute the F matrix as the nullvector of the system eq. 234
2. Find the SVD of F (eq. 248)
3. Put $\sigma_1 = 0$
4. Recompute F using eq. 248

This gives a very efficient algorithm for computing an F-matrix with singularity constraint imposed.