Chapter 1

Notes for Dialogue 2

November 11, 2002

1.1 Multiview Geometry (in Archimedes's lab)

1.1.1 Projective Space

Arc: If you want to start using computer programs and images to really build 3D models of the world, I need to teach you an interesting way of thinking about these problems. Remember what an eye really is. Take all the rays passing through a point and cut them with a surface, say a plane. You get an image. Now it pays to think of the image as the set of rays passing through each image point, because what you find out will be true no matter how you cut the rays.

Mel: So, does this mean that now we must figure out an easy way to reason about these rays?

Arc: Exactly. We have an interesting set: take our space and a point in it. Consider all the lines that pass through this point. This is a new kind of space, and I'll tell you how to think about it. You can take a look at this space by cutting it with a plane, for example. Then you can talk about each ray by using the point at which it cuts the plane. All rays will cut the plane except the ones parallel to it (all these lie on a plane parallel to the image plane). These parallel rays intersect the image plane at infinity. Take all these points at infinity and put them on a line, the line at infinity. So, you have an ordinary plane, the way you think of it, with

the additional property that all lines intersect.

Mel: It's hard to imagine.

Arc: Think of it this way. Cut all rays with a sphere, and take each sphere point to represent the ray. You don't need the whole sphere; take a hemisphere. The equator is the line at infinity. There is one more thing you need to do to make the space. If I give you one such ray in space, you don't know which way to move along the ray to reach the center. Whichever way you move, you will reach the center. So, in this space of rays, if I ask you where Chloe went, and you point with your finger—this way, you could have just as well pointed the other way. This simply means that if you move along a ray you will come back. To make sure this happens, take the hemisphere with its equator, and glue together antidiametric points. The new set you have is the space of rays.

Mel: That's a very cool space indeed. Now I can see that two lines always intersect. You get a line in the image by cutting the image with a plane defined by two rays. Now these lines are great circles, and they always meet. What happens when I cross the line at infinity in that space? How do I come back?

Arc: Well, when you do the gluing you must also make a twist. Otherwise you get another space; you get a donut.

Mel: So, when I cross the line at infinity I come back mirrored-reversed?

Arc: Exactly. You see, the projective plane and the sphere locally are similar, but globally quite different. They have different topology. Now you can think of images as being elements of this space, but it is not identical to a plane. To make the distinction, people call it a *projective* plane. You can do calculations of many sorts, and make many observations. First, it is a three-dimensional space. Put your usual Cartesian system somewhere. Pick a ray passing from the origin. You can represent that ray with any point on it. You can then write that a ray is (x_1, x_2, x_3), where (x_1, x_2, x_3) is any point on the ray. There are obviously infinite points that can define a ray and only the ratio $(x_1 : x_2 : x_3)$ is relevant. Mathematicians call these kind of coordinates homogeneous. You can thus write the rays as triplets of numbers, but always remember what your symbolism means.

Mel: So, a ray is really an image point; I can see that by cutting the rays with a plane. But how about a line in the image?

Arc: Two rays define a plane. Intersection of this plane with the image plane

gives a line.

Mel: Oh, so an image line corresponds to a plane in that space.

Arc: The interesting thing is that you represent that plane with the ray perpendicular to it. So, an image line also corresponds to a ray! Both image points and lines are each represented by a ray. If you want to represent the projective plane by cutting it with a plane, you can always put it in canonical position: image plane perpendicular to the Z axis at distance unit from the origin. Rays that are written as $(x_1, x_2, 0)$ make the points at infinity.

Mel: Wonderful, but what do these numbers mean? If I have (x_1, x_2, x_3) in my usual 3D Cartesian coordinate system, I know that if I start from the origin, I will go x_1 units in the x direction (east-west), x_2 units in the y direction (up-down), and x_3 units in the x direction (north-south), and I will reach point (x_1, x_2, x_3). Now, what do these numbers mean?

Arc: Your point (x_1, x_2, x_3) in our 3D space has *three* coordinates. What did you do to reach the point from the origin? You added three vectors. You first moved east, then up, then north. So, you added three vectors i, j, k, but before that you scaled them by x_1, x_2, and x_3. These three vectors are the base for our space. You can get to any point by adding these scaled vectors. These scales are the coordinates with respect to the base. If you change the base, you also change coordinates for a point. If (x_1, x_2, x_3) represents a ray, then these coordinates should be such that they don't change if we change the image plane. Since the cross ratio doesn't change, it's the only candidate. Let's see how. Take four rays. They cut the image plane at A, B, C and D. Consider the lines AB, AC and AD. Now take any ray or point on the plane, M. Lines AB, AC, AD, AM have a cross ratio which will be the same no matter how we cut the rays. Call this cross ratio x_1. Now consider another pencil of lines, BA, BC, BD, BM. Call their cross ratio x_2, and for a third pencil, x_3. Now (x_1, x_2, x_3) is your ray OM. If you have the three coordinates you can find M.

Mel: I see that. Since I know the cross ratio AB, AC, AD, AM I know the plane OAM. Similarly I know the plane OBM and so the ray is the intersection of these two planes. Interesting. In the Euclidean plane I need two points to have a base. Now, I need four.

1.1.2 Homographies Are Linear Maps

Arc: Good. Each ray OM is the sum of four vectors OA, OB, OC, OD, appropriately scaled, just as you do in the ordinary space. To get a new ray you can do it by different linear combinations of the four pyramid rays. So you can write $OM = OA + \lambda_2 OB + \lambda_3 OC + \lambda_4 OD$ for appropriate λs. These are again the coordinates, somewhat disguised, but the λs are related to the (x_1, x_2, x_3). With that base you can write each ray as a triplet of numbers. If you change the base, the triplet will change. You need four points for the base, so that you can get three lines and form cross ratios with any point M. You can choose these four rays as you like, as long as they are in a general position (no three of them co-planar). Let's say you have two bases. The same ray will be (x_1, x_2, x_3) in the first base, and (x'_1, x'_2, x'_3) in the second. How are these numbers related?

Mel: You are confusing me when you involve numbers. Can't you avoid numbers?

Arc: I can, but if you want to find out how to actually realize the building of a scene model, you have to use some numbers. Look at it this way: Say you cut the rays with two different image planes. Then we know that the map which matches image points due to the same ray is a homography and we only need to know how four points map. Then we know the whole map.

Mel: Oh, that's why I need to know how four rays move, to find how everything else moves. The four rays are the base. The homography will send these four points to four new ones. If you know how the base changes, you know how everything else changes as well.

Arc: Good. Take your base rays OA, OB, OC and OD and any ray OM. To get to OM you should move along each of the axes OA to OD by some appropriate amount. We can write this as $OM = OA + \lambda_1 OB + \lambda_2 OC + \lambda_3 OD$. Now if you map $OA \to OA', OB \to OB', OC \to OC'$ and $OD \to OD'$, how is OM mapped to OM'? Since you can write OM as an appropriately scaled sum of four rays, it is obvious that after the map the base changes. The new base is OA', OB', OC' and OD', and OM' is the (appropriately scaled) sum of the rays OA', OB', OC', OD'. So if we choose a triplet of numbers (x_1, x_2, x_3) to denote any ray, the homography will make it the new ray (x'_1, x'_2, x'_3). How are these numbers related to each other?

Mel: Hold on, you said the new ray (x'_1, x'_2, x'_3). But the rays did not change.

We just cut them with planes.

Arc: Yes, they do not change, but the way we talk about them, the way we measure them, changes. Imagine some object. It has weight. I can tell you that it is ten kilos. I can also tell you that it is twenty-two pounds. The object is the same, but the numbers change because we use different systems. Same thing here: Our ray is (x_1, x_2, x_3) with regard to some system (OA, OB, OC, OD). If you change systems, the numbers will change. We need to know how the x_is become ${x'}_i$s. As you can understand, everything depends on the two bases. I can get to ray OM' by adding the rays OA', OB', OC' and OD', but each one of the four rays OA', OB', OC', OD' can be obtained from the original pyramid $OABCD$, again as an (appropriately scaled) sum of the OAs, OBs, OCs and ODs. We only need to know how to express the new base in terms of the old base. Ultimately, ray OM' will be a sum of the original base vectors. If ray OM' is $({x'}_1, {x'}_2, {x'}_3)$, how do we get to it? By adding appropriately scaled rays. Ultimately, when you start from ray OM, or (x_1, x_2, x_3), to get to OM' you will have to scale and add rays. In the most general case, to go from (x_1, x_2, x_3) to ${x'}_1$ you may have to multiply each of the xs by a scale and add them.

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

You can also write this in the form of a matrix.

$$\begin{pmatrix} {x'}_1 \\ {x'}_2 \\ {x'}_3 \end{pmatrix} = \begin{pmatrix} h_{11} & & \\ & \ddots & \\ & & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad \text{or}$$

$x' = Hx$, and H is the homography. You see a good advantage. H is a linear transformation. It is easy and we understand how to deal with linear things, more or less. But keep in mind that this was an informal argument. A proof of this is very involved.[1]

Mel: Fascinating. The good old homographies are linear mappings.

[1] yiannis will add a note

1.2 Models: Projective, Affine and Euclidean

Arc: As you see, you need nine numbers. Actually, you only need eight because nothing would change if I multiplied the h_{ij}s by the same number. You can pick $h_{33} = 1$ and then you need eight numbers. So, take a ray $x = (x_1, x_2, x_3)$. A homography H will move x to x' and $x' = Hx$, a simple linear operation.

Mel: How does this fit with models?

Arc: Well, let's talk about simple models first, models on the plane, 2D things. As you cut the rays with different planes, you make images whose rays are related by homographies. A square in one image can be seen as any quadrilateral from other images. Whatever model I have for my rays (x_1, x_2, x_3), it is some model changed by a linear transformation H, which in general is unknown. In other words, in the projective plane if you know a number of rays—which means you have a model for some simple planar object—you really don't know them, but you know that the true ones are related to the ones you have by a homography. It is as if you have an uncalibrated camera. An uncalibrated camera is an exact projective device, and everything we discussed did not involve any angles or any parallelism, only points and lines and incidence relationships. This is what people call a projective model. You know that the real model and the projective model are related by an unknown homography.

Mel: So, this projective space is unexpectedly weird and projective models are distorted versions.

Arc: Yes, but you have an idea of how they are distorted. You know, for instance, which points are collinear in the scene. If, for instance, you can find the line at infinity in the image, you can find more. You see, vanishing points lie on the line at infinity. If you can locate it in the image you can map it back to its canonical position. You can do this with a simple homography.[2] After you do that, parallel lines in the world remain parallel in the image.

Mel: These look better.

Arc: They do, but remember we still do not have the actual thing yet. The differ-

[2](Yiannis will add a endnote.)

ence now is that the homography is simpler and the matrix looks like

$$H_{\text{aff}} = \begin{pmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_2 & t_2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} A & \mathbf{t} \\ O^\top & 1 \end{pmatrix}.$$

They call it an affine transformation. Note the last row of the matrix; it is 0, 0, 1. If you know the line at infinity, then you have a model for a ray x when you know it up to an affine transformation $H_{\text{aff}}\, x$. Parallel lines will remain parallel, but perpendicular lines will not remain perpendicular. Affine models may be more useful than projective models.

Mel: This affine transformation seems to have interesting properties. It preserves quite a few things.

Arc: Indeed. They call a property affine when it is preserved by all affine transformations, but not by all projective transformations. The property of being a segment, a vector, ray, line, angle, triangle, or conic section of a special type is affine. Also affine are parallelism, the concurrence of lines, and the ratio of distances on the same line.

Mel: How about the size of an angle.

Arc: No, angle size is not preserved. Think of affine transformations as parallel projections from one plane to another.

Mel: But how can I go from a projective model to an affine? You said you can do it if you find the image of the line at infinity.

Arc: OK, I will explain this. Take two image points, that is two rays, x and y. The line l they define is the ray perpendicular to both x and y, so $l = x \times y$. If homography H maps x and y to x', y', where does it map l?

Mel: Well, $l' = Hx \times Hy = H^{-\top}(x \times y) = H^{-\top} l$. Cute. Lines are mapped with $H^{-\top}$, the inverse transpose.

Arc: Fine. The line at infinity is $(0, 0, 1)^T$. If H is affine, then $H^{-\top} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} =$

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Mel: This means that the line at infinity is fixed under an affine transformation.

Arc: Exactly. And the converse is easy to see. Let's say $H = (h_{ij})_{3\times 3}$ does not change the line at infinity. Then, a point at infinity, say $(1, 0, 0)$ has to be mapped at a point at infinity. This requires $h_{31} = 0$. Similarly $h_{32} = 0$. So H is affine.

Mel: So, if I start from a projective model and I apply a homography that brings the line at infinity at its canonical position, then I have all of sudden an affine model. I can start measuring affine properties. If I have a line $l = (l_1, l_2, l_3)^T$, the image of the line at infinity (which I can measure from the vanishing points), I can map it with homography $H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{pmatrix}$ to (0, 0, 1). This same homography makes the model affine.

Arc: Very well said.

Mel:

So far, so good. You can get a projective model of what is out there, or an affine model. The affine model has more detail. You found a model for some rays that is one homography away from the real thing. If that homography is a general 3×3 matrix H, the model is projective. If the homography is somewhat special, like H_{aff}, the model is affine. H_{aff} has fewer unknowns (6 degrees of freedom) than the general H (8 degrees of freedom). So, an affine homography is determined by three points only; in the general case you need four.

Arc: Very good. Now, to have a model which is identical to what is in the scene up to scale, your homography has to be even more special. It is a similarity transformation

$$H_{\text{sim}} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_1 \\ s\sin\theta & s\cos\theta & t_2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ O^\top & 1 \end{bmatrix}$$

and it has four degrees of freedom, scale, rotation and translation. Remember, we are only talking about 2D objects here. This transformation preserves angles. So, if you find a model (x_1, x_2, x_3) for our rays and they are related to the real one by $x' = H_{\text{sim}}x$, you are done, because you have the exact model. It could be rotated

and translated and scaled, but it is the most useful one up to now. Usually people call it, somewhat incorrectly, the Euclidean model. The transformation H_{sim} is called Euclidean if it doesn't have a scale, so the similarity transformation is like a Euclidean one with a scale.

Mel: What is the real difference between affine and projective?

Arc: The matrix of the affine has as last row 0, 0, 1, while the projective matrix is full. With affine, the scaling of an object is the same no matter where it is on the plane, and the orientation of a transformed line only depends on its initial orientation, not its position on the plane.

Mel: I guess the same thing happens in three dimensions?

1.3 Projective 3D Space

Arc: The exact same thing. Points of this space are quadruples (x_1, x_2, x_3, x_4) . You have a few differences though. How did you get the projective plane? You went to our 3D space, picked a point and considered all the rays. To get to projective space, start from the origin of a 4D space and consider all the rays passing through.

Mel: Now it's getting hard. How does 4D space look?

Arc: That is something no one knows. We cannot easily visualize spaces of more than three dimensions. Little has been written about it and it can be read quickly.[3] One way to think of 4D space is to think what happens from 2D to 3D and seek analogies that will help you, in some way, to visualize things. If you do such things, you could think of projective space as two spheres and each point in the one sphere glued to a point in the other sphere. Now you have points at infinity, and lines at infinity, just as before, but you also have the plane at infinity. As you move through that space, when you cross the boundary of the first sphere (which is glued to the other sphere), you come back left-right reversed as before but also top-bottom reversed; you don't change at all, you just go through, but you have been rotated 180 degrees.

Mel: This is all good to hear, but I am more interested in visual space-time. Tell

[3](Yiannis will add a note here.)

me about projective, affine and similarity transformations in this space.

Arc: Nothing much to tell. Everything is the same with the exception that the dimension increased by one.

A ray in this space represents a 3D point. In the previous case a ray represented an image point. If the ray is $x = (x_1, x_2, x_3, x_4)$, the corresponding 3D point is $(x_1/x_4, x_2/x_4, x_3/x_4)$. If rays x map to rays x' through $x' = Hx$, H could be a full homography, or an affine transformation, or a similarity. If the models x' we recover are related to the real one by an H_{sim}, then we found a Euclidean model. If they are related by an affine map, we have an affine model. Otherwise, if H is a full homography, then we have a projective model. I showed how you can get these models from two views of a number of corresponding points and lines. This is far reaching because all maps involved for making the image are linear in the space of rays.

1.3.1 Equations of image making

Arc: Let's look more closely at what is involved in making an image. Take a normalized camera. Take a point (X, Y, Z) in space. Then, the image (x, y) in the image coordinate system is $x = X/Z, y = Y/Z$. The ray (X, Y, Z) represents point (x, y). I can write this as

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This 3×4 matrix P maps the rays of the 4D space to the rays of the 3D space, and you can write $x = PX$. This is the image-making map. Any corrections you make for calibration parameters, it's only a homography in general. Let's find exactly what it looks like. In the equation before, we assumed a normalized camera. Having focal length f, the image becomes $x = fX/Z$, $y = fY/Z$. In terms of matrices, it becomes:

$$\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This matrix is usually written as $\mathrm{diag}(f, f, 1)[I|O]$, with $\mathrm{diag}(f, f, 1)$ a diagonal matrix and $[I|O]$ a 3×4 matrix consisting of a 3×3 block (here the identity) plus a column vector (here O). The previous equation assumed the origin of image coordinates to be the principal point, but in general this is not true, in which case the equation becomes

$$\begin{pmatrix} fX + ZX_0 \\ fY + ZY_0 \\ Z \end{pmatrix} = \begin{pmatrix} f & x_0 & 0 \\ & fy_0 & 0 \\ & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

with (x_0, y_0) the principal point.

Mel: I can see it already. You have that a point P in 3D (homogeneous coordinates) has an image p (again in homogeneous coordinates) given by:

$$\begin{pmatrix} f & x_0 \\ & fy_0 \\ & 1 \end{pmatrix} \begin{pmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{pmatrix} P \qquad \text{or} p = H[I|O]P$$

with H the calibration homography.

Arc: Excellent. For added generality, you can add to the calibration parameters the skew parameter s, which usually is zero but can take nonzero values and depends on the angle between the image coordinate axes. You can also make the focal length different for the two directions and $H = \begin{bmatrix} fx & s & x_0 \\ & fy & y_0 \\ & & 1 \end{bmatrix}$. If you want to explicitly incorporate the angle θ between the coordinate axes, you get $H = \begin{pmatrix} fx & fx\cot\theta & x_0 \\ 0 & fy/\sin\theta & y_0 \\ 0 & 0 & 1 \end{pmatrix}$. Also, people usually denote the calibration homography by K.

Mel: Looks cute.

Arc: This is, of course, in the camera coordinate system. If your point is measured in some other coordinate system, things change slightly.

Mel: If P is in the world coordinate frame, let's assume that the camera is a rotation R and a translation $\mathbf{t}$ away from the world coordinate system. Then P, expressed in the camera system as $R(P - C)$, where C is the camera center. Then

your basic equation becomes $p = KR[I| - C]P$, well known in graphics. By setting $\mathbf{t} = -RC$, it becomes $p = K[R|\mathbf{t}]P$ explicitly incorporating the rigid motion $R, \mathbf{t}$ and the calibration homography K.

1.4 The Essential Matrix: Relating Two Views with the Epipolar Constraint

Arc: Well said. Now you can do some cool things. You first find the map relating different views. Let's say in two views you have the map E mapping points x in the first view to their epipolar lines l' in the second:

$$x \to l' = Ex$$

So, point (ray) $x' = (x_1, x_2, x_3)$ lies on the epipolar line $l' = (l_1, l_2, l_3)$ which is Ex, but as before, starting from x we'll get to l by scaling and adding rays. In other words, E is a linear mapping—I won't call it a homography because it maps points to lines.

Mel: I don't quite follow that the map E is a matrix.

Arc: Recall that we found $E = [t]_\times R$. R is a homography, so it's linear. $[t]_\times$ applied to a point (ray) x is a line connecting t and x, i.e., $t \times x$. That's linear too, you can write it as

$$\begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix} x.$$

So, E is the product of two linear maps and so it is a matrix. People call it the essential matrix. F, the map in the calibrated case, is a matrix too and people call it the fundamental matrix. So, ray $x = (x_1, x_2, x_3)$ maps to line

$$Ex = \begin{pmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

But point x' lies on l'. How do you write that?

Mel: Well, l' is the ray (l'_1, l'_2, l'_3) normal to the plane defined by the origin and l'. Ray x' lies on that plane, so $x' = (x'_1, x'_2, x'_3)$ and $l' = (l'_1, l'_2, l'_3)$

are perpendicular rays and so the inner product of the two vectors must be zero, $x' \cdot l' = 0$ or ${l'}_1{x'}_1 + {l'}_2{x'}_2 + {l'}_3{x'}_3 = 0$. You can write this as

$$\begin{pmatrix} {x'}_1 & {x'}_2 & {x'}_3 \end{pmatrix} \begin{pmatrix} {l'}_1 \\ {l'}_2 \\ {l'}_3 \end{pmatrix} = 0$$

and remembering that $l' = Ex$, we get

$$\begin{pmatrix} {x'}_1 & {x'}_2 & {x'}_3 \end{pmatrix} E \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 0.$$

This is a linear equation in the unknown E. It is known as the *epipolar constraint*. E has nine elements but as you see nothing changes if I multiply every element of matrix E with the same number. So, pick $e_1 = 1$ and find the other eight elements. If you have eight corresponding points, you get eight linear equations in the elements of E and you can solve it. After that, you can solve for the translation and rotation. Remember, $E = [t]_\times R$.

Mel: So, somehow we need to split E into its components $\mathbf{t}$ and $\mathbf{R}$.

Arc: As I explained before, there is a lot of software available for this. Since all these transformations are matrices, there are a lot of tools available for finding them. Recall that E has five degrees of freedom (rotation and translation each have 3 but there is a scale ambiguity. So, it has rank 2. The Singular Value Decomposition Theorem[4] guarantees that you can write E as the product of three matrices $U\mathrm{diag}(1,1,0)V^\top$, with the middle being a diagonal matrix. Then you can easily show that there are two possible factorizations of $E = SR$, where $S = UZU^\top$ and $R = UWV^\top$ or $UW^\top V^\top$ with $W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

Mel: I guess, since $S = [t]_\times$, you can also find $\mathbf{t}$ up to scale. But then, since you have two choices for the rotation and two choices for $\mathbf{t}$ (the sign) you will have four choices for placing the cameras.

Arc: Yes, but only one of them makes sense. For the three cases, the reconstructed

[4](Yiannis will add a note here.)

point lies behind at least one of the cameras. Similar things happen for the uncalibrated case where you have map $F = [e']_\times H$, with H some homography. In this case, you can find the epipoles and you find a model up to a homography, that is, a projective model. If you manage to find the infinite homography, that means that you can get the plane at infinity. If you know where it is from the image, you can map it to its canonical position[5] and you get an affine model of the scene. And finally, if you manage to recover a model up to a similarity transformation, you are done.

1.5 The Trifocal Constraints

Mel: I guess the same thing is happening with the constraints from three views?

Arc: Yes. Now you have a map relating corresponding points and lines in three views. Let's say you have corresponded three lines $\boldsymbol{\ell}_1$, $\boldsymbol{\ell}_2$ and $\boldsymbol{\ell}_3$ in three views. Then you can show[6] that there exist three linear maps T_1, T_2 and T_3, that is three matrices of dimensions 3×3 as before that satisfy the relation:

$$\boldsymbol{\ell}_1^{\mathrm{T}} = \boldsymbol{\ell}_2^{\mathrm{T}}[T_1, T_2, T_3]\boldsymbol{\ell}_3$$

Mel: What is this symbolism?

Arc: Think of the right side of this equation as the vector $(\boldsymbol{\ell}_2^\top T_1 \boldsymbol{\ell}_3, \boldsymbol{\ell}_2^\top T_2 \boldsymbol{\ell}_3, \boldsymbol{\ell}_2^\top T_3 \boldsymbol{\ell}_3)$.

Mel: I see. Like before , this equation becomes linear in the elements of the 3 matrices T_1, T_2, T_3. But how do you get this constraint? It looks too cute.

Arc: You can actually get constraints like this one if you have three corresponding points in the three images, or even for any point/line combination (point, line, line — point, point, line, etc.).

Mel: I can understand the usual constraints we have been any two cameras. Where is the additional knowledge that brings out the new constraint?

Arc: Assume a point P_1 undergoes rigid motion with rotation R_1 and translation T_1 and moves to P_2. So, $P_2 = R \cdot P_1 + T_1$. If $p_i = \|P_i\|$, $i = 1, 2$ the length of P_1, P_2 and P_i the unit vectors $P_i/\|P_i\|$, then

$$p_2 P_2 = p_1 R \cdot p_1 + T_1$$

[5] (Yiannis will add a note.)

[6] yiannis will add a note

and the structure—the length of P_1—is

$$P_1 = \frac{[(R_1 p_1) \times p_2]^2}{[(R_1 P_1) \times p_2]^\top (T_1 \times P_2)}$$

If you consider a third frame, then

$$P_3 = R_2 P_1 + T_2 \qquad \text{or} p_3 p_3 = p_1 R_2 p_1 + T_2$$

and the structure now is

$$p'_1 = \frac{[(R_2 \cdot P_1) \times P_3]^2}{[(R_2 P_1) \times P_3]^\top (T_2 \times P_3)}$$

But the structure did not change, so $p_1 = p'_1$. That's where you are gaining the additional knowledge.

Mel: Hmm, and how do you obtain the constraint?

Arc: Start with the two motion equations

$$\begin{aligned} p_2 P_2 &= p_1 R_1 \cdot P_1 + T_1 \\ p_3 P_3 &= p_1 R_2 \cdot P_1 + T_2 \end{aligned}$$

and eliminate p_1, p_2, p_3, the structure parameters. First, take the cross-product of the equations with P_2 and P_3, respectively.

$$\begin{aligned} 0 &= p_1 P_2 \times (R_1 \cdot P_1) + P_2 \times T_1 \\ 0 &= p_1 P_3 \times (R_2 \cdot P_1) + P_3 \times T_2 \end{aligned}$$

Rearranging terms we obtain

$$\begin{aligned} p_1 P_2 \times (R_1 \times P_1) &= -P_2 \times T_1 \\ P_3 \times T_2 &= -p_1 P_3 \times (R_2 \cdot P_1) \end{aligned}$$

and then taking the outer product of both sides

$$[P_2]_\times (R_1 \cdot P_1 \cdot T_2^\top)[P_3]_\times = [P_2]_\times T_1 (R_2 \cdot P_1)^\top [P_3]_\times$$

where, as usual,

$$[P_i]_\times = \begin{pmatrix} 0 & z_i & -y_i \\ -z_i & 0 & x_i \\ y_i & -x_i & 0 \end{pmatrix}$$

with $P_i = [x_i y_i z_i]^\top$.

If $\hat{x}, \hat{y}$ and $\hat{z}$ unit vectors along the three axes, the equation becomes:

$$[P_2]_\times (x_1(R_1 \cdot \hat{x}) + y_1(R_1 \cdot \hat{y}) + z_1(R_1 \cdot \hat{z}))T_2^\top [P_3]_\times = [P_2]_\times T_1(x_1(R_2 \cdot \hat{x})^\top + y_1(R_2 \cdot \hat{y})^\top + z_1($$

If you then define matrices K, L and M as

$$\begin{aligned} K &= T_1(R_2 \cdot \hat{x})^\top - (R_1 \cdot \hat{x}) \cdot T_2^\top \\ L &= T_1(R_2 \cdot \hat{y})^\top - (R_1 \cdot \hat{y}) \cdot T_2^\top \\ M &= T_1(R_2 \cdot \hat{z})^\top - (R_1 \cdot \hat{z}) \cdot T_2^\top \end{aligned}$$

the equation becomes:

$$[P_2]_\times (x_1 K + y_1 L + z_1 M)[P_3]_\times = [0] \qquad \text{or} \qquad [P_2]_\times [(K, L, M) * P_1] \cdot [P_3]_\times = [0]$$

with the obvious definition of $(\dots, \dots, \dots) * \dots$.

Mel: Very nice. If P_1, P_2, P_3 are corresponding points in three cameras, they satisfy this constraint, the trifocal constraint. What if you have lines?

Arc: Let's say you have three lines $\boldsymbol{\ell}_1, \boldsymbol{\ell}_2, \boldsymbol{\ell}_3$ in correspondence in three frames. You represent a line on the plane by a point and a direction. So, every point of a $\boldsymbol{\ell}_1$ can be written as $p + \lambda \mathbf{u}$, where $\mathbf{u}$ the direction vector, and P some point.

Mel: I see. Using projective coordinates the line is then $p \times u$.

Arc: Exactly. So, take a point P_1 on $\boldsymbol{\ell}_1$. It's corresponding points P_2 and P_3 on $\boldsymbol{\ell}_2$ and $\boldsymbol{\ell}_3$ are

$$\begin{aligned} P_2 &= {P'}_2 + \lambda_2 \mathbf{u}_2 \\ P_3 &= {P'}_3 + \lambda_3 \mathbf{u}_3, \end{aligned}$$

for the appropriate points ${p'}_2, {p'}_3$ and $\lambda_2, \lambda_3, \mathbf{u}_2, \mathbf{u}_3$. But then the trifocal constraint on these points says: $[{P'}_2 + \lambda_2 \mathbf{u}_2]_\times [(K, L, M) * P_1][{P'}_3 + \lambda_3 \mathbf{u}_3]_\times = [0]$. You can eliminate λ_2, λ_3 by pre- and post-multiplying with $\mathbf{u}_2^\top$ and $\mathbf{u}_3$: $({P'}_2 \times \mathbf{u}_2)^\top [(K, L, M) * P_1]({P'}_3 \times \mathbf{u}_3) = 0$ or $\boldsymbol{\ell}_2^\top [(K, L, M) * P_1] \boldsymbol{\ell}_3 = 0$ or

$$\begin{bmatrix} \boldsymbol{\ell}_2 \top K \boldsymbol{\ell}_3 \\ \boldsymbol{\ell}_2 \top L \boldsymbol{\ell}_3 \\ \boldsymbol{\ell}_2 \top M \boldsymbol{\ell}_3 \end{bmatrix}^\top P_1 = 0$$

But this equation should be true for any point of $\boldsymbol{\ell}_1$ that we can write as $P_1 + \lambda_1 \mathbf{u}_1$. Observe the above equation. It basically tells you that the first vector is normal to any vector defined by any image point. So, this vector is parallel to $\boldsymbol{\ell}_1$, that is

$$\begin{bmatrix} \boldsymbol{\ell}_2 \top K \boldsymbol{\ell}_3 \\ \boldsymbol{\ell}_2 \top L \boldsymbol{\ell}_3 \\ \boldsymbol{\ell}_2 \top M \boldsymbol{\ell}_3 \end{bmatrix}^\top \boldsymbol{\ell}_1 = 0,$$

and so $\boldsymbol{\ell}_2^\top [K, L, M] \boldsymbol{\ell}_3$.

Mel: I now can see how you can get a constraint for line $\boldsymbol{\ell}_1$, point p_2, line $\boldsymbol{\ell}_3$. If p_2 lies on $\boldsymbol{\ell}_2$, then $P_2 \boldsymbol{\ell}_2^\top = 0$ but $\boldsymbol{\ell}_3^\top [K, L, M] \boldsymbol{\ell}_1$ and so $\boldsymbol{\ell}_3^\top [(K, L, M) * P_2] \boldsymbol{\ell}_1 = 0$.

Arc: Excellent. Can you guess now what the constraint for point, point, line is?

Mel: I guess it should be $\boldsymbol{\ell}_3^\top [(K, L, M) * P_1][P_2]_\times = 0$.

Arc: That's it. You have the trifocal constraints. This object $[(K, L, M)]$ that consists of three 3×3 matrices is called a tensor. You can use tensor mathematics now to further analyze these constraints.

Mel: Have people done that?

Arc: Yes. You can write more efficient and very short programs if you phrase things in the language of tensors, but you aren't going to discover anything new.

Arc: You need 27 equations. These 3 matrices together form something called a tensor. This tensor $T = [T_1, T_2, T_3]$ is known as the trifocal tensor. It only depends on the rigid motion between the views. Like before, given T you can find the essential E (or the fundamental matrix F) between any two views and you are done.

Mel: Is it that easy?

Arc: It is very easy indeed. Line $\boldsymbol{\ell}_2$ with the center of camera 2 define a plane Π. Using this plane you can make a homography map H between the first and the third view, so $x_3 = H x_1$. But then $\boldsymbol{\ell}_3 = H^{-\top} \boldsymbol{\ell}_1$ or $\boldsymbol{\ell}_1 = H^\top \boldsymbol{\ell}_3$. But you know from the trifocal constraint that $\boldsymbol{\ell}_1^\top = \boldsymbol{\ell}_2^\top [T_1, T_2, T_3] \boldsymbol{\ell}_3$. This means

$$H^\top = \boldsymbol{\ell}_2^\top [T_1, T_2, T_3] \text{ or } H = [T_1^\top, T_2^\top, T_3^\top] \boldsymbol{\ell}_2$$

So, look at points x_1 and x_3. We just saw that $x_3 = [([T_1^\top, T_2^\top, T_3^\top]) \boldsymbol{\ell}_2] x_1$. Thus, the epipolar line in view 3 corresponding to x_1 can be found by connecting the

epipole e_3 to x_3, that is

$$[e_3]_\times[(T_1^\top, T_2^\top, T_3^\top)\boldsymbol{\ell}_2]x_1$$

which means that the fundamental matrix F between 1 and 3 is $[e_3]_\times \ [T_1^\top, T_2^\top, T_3^\top]\boldsymbol{\ell}_2$, for any $\boldsymbol{\ell}_2$. Be careful not to choose $\boldsymbol{\ell}_2$ in the null space of the T_i's because that way you get a degenerate situation. A good choice is the epipole e_2.[7]

Mel: I understand the principle. From points and lines you find the epipolar constraint (the map E) and the trifocal tensor $T = [T_1, T_2, T_3]$. If you have different combinations instead of three lines, does the same thing happen?

Arc: Yes, you get similar constraints for point, point, point or point, line, line, etc.

Mel: I guess that's it then. This provides the placement of the cameras.

1.6 Bundle Adjustment

Arc: Perfect. All this of course exists in good software today. People have performed amazing work on the subject. Very highly developed and good science. The software has an additional feature, a sort of final step.

Mel: Oh, there is more?

Arc: Well, not really. You see, the software assumes that another step will provide the correspondence between points and lines in the different views. This problem is hard and when you obtain an estimate for it, it will have errors. Because of these errors, when you place the cameras, you will have errors as well. They could be small or large and as a result the 3D model will have errors. This final step has as its goal to reduce these errors, to smooth things out.

Mel: How is it done? New constraints?

Arc: No. Consider two images of some scene containing points m_i and m'_i in correspondence. If G is the rigid transformation between the two views, one can finally obtain the corresponding 3D points M_i as a function of m_i, m'_i and G, i.e., $M_i = f(G, m_i, m'_i)$. So, the projection of M_i on the first camera is $P(M_i) = m_i$ and on the second $P' = (M_i) = m'_i$. Bundle adjustment amounts to adjusting

[7] (Yiannis will add a note.)

the bundle of rays between each camera center and the set of 3D points so that the distance between the reprojected point $P(M_i) = P(f(G, m_i, m'_i))$ and the detected (measured) point m_i is minimized, that is,

$$\min \sum_i \|P(f(G, m_i, m'_i)) - m_i\|^2 + \|P'(f'(G, m_i, m'_i)) - m'_i\|^2$$

It is quite a complicated optimization usually starting after a solution is produced. It keeps, in effect, changing all relevant parameters until a solution is achieved that provides a (local) minimum for the reprojection error.

Mel: I see. In some sense, minimizing the reprojection error is sort of the best thing one can do.

Arc: It appears so. But, as you already know, the results are not perfect. In simple terms, this existing theory will not give you an answer for the geometry of visual space-time. Small errors in the input of the software package create problems in the output, camera placement and 3D model. Up to now, no one has a crisp idea what is the exact nature of these errors.[8] The practitioners of this field do not really understand the correspondence problem.

Mel: So, this theory that you described basically shows the laws that relate points and lines in different views. I see; at least I have something to start with. But I need to systematize this knowledge. To have the whole thing in front of me. All this body of knowledge, it seems to me, was developed by different people and I cannot fit it together with a few formulas.

1.7 Multiview Geometry in a Nutshell

1.7.1 Coordinates

Arc: OK, I will tell you a simple formalism that will help you do this. The trick is to use homogeneous coordinates for the image but Euclidean coordinates for the 3D points. You see, we have a very good understanding of the space of rays, that is the projective plane. Usually people consider homogeneous coordinates for 3D points as well. But these are 4-dimensional and although we know how to use them, we don't have much intuition. The basic projection equation (image point) $x = P \cdot X$ (world point), where P is a 3×4 projection matrix denoting the camera

[8] (Yiannis will add a note.)

is valid when both x and X are in projective coordinates. But, I have something easier. Something that allows you to work with large numbers of views.

I will use homogeneous coordinates for the image points and lines. Remember that if for any object $s \in S$ with coordinate $\mathbf{s}$, the coordinate $\lambda\mathbf{s}$ also refers to s for any $\lambda \in \mathbb{R}$, then the coordinates are said to be homogeneous. The coordinate 0 does not represent any object $s \in S$.

It turns out that the equations are much easier if we use vectors, so we'll stick with those. All vectors used will be column vectors and I will transpose them as necessary. I will use now and then the identities shown in figure **??**. Both $.^{\top}(. \times .)$ and $|...|$ are used to denote the triple product.

Mel: I guess homogeneous coordinates will allow you to use linear algebra in projective spaces. Good choice.

Arc: Let's take a world point $P \in \mathbb{R}^3$. We can represent such a point in a particular coordinate system as

$$\mathbf{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Consider now the ray OP, from the camera center O to point P. It creates the image point p. Geometrically, I will represent image point p with the ray OP, as you know. That way, the image points exist in the projective plane $\mathbb{P}^2$.

Mel: I see your motivation. This projective space $\mathbb{P}^2$ has the advantage of duality between points and lines, which makes it extremely easy to consider both and transform formulas which consider one into formulas which consider the other.

Arc: Exactly! So, an image point can be represented in a particular coordinate system as $\mathbf{p} = [x\ y\ z]^{\mathrm{T}}$ with these coordinates being homogeneous.

Mel: One moment. I notice that the coordinates of the world and image points are both 3-vectors, but one is a Euclidean space of dimension 3 and the other is in the projective plane. Can you mix them?

Arc: Of course, as long as you know what you are doing. You can think of a camera as a device for considering the coordinates $\mathbf{P}$ of a point $P \in \mathbb{R}^3$ to be coordinates of a point $p \in \mathbb{P}^2$.

Mel: That's really cool. If I have a coordinate system in 3D with its origin at the camera center, then the image of a point $P \in \mathbb{R}^3$ with coordinates $\mathbf{P} = [X\ Y\ Z]^{\mathrm{T}}$

is the ray $OP \in \mathbb{P}^2$ with coordinates $[X\ Y\ Z]^{\mathrm{T}}$. So what is a system in which it will be easy to talk about large numbers of cameras?

Arc: As you recall, the coordinates of a 3D point become the coordinates of the image point (e.g., the ray). However, our world points exist in one fiducial coordinate system, while the image points exist in the particular camera coordinate systems. Therefore our camera is defined in relation to this fiducial coordinate system as follows:

1.7.2 Projecting points

A **camera** C is a map $C : \mathbb{R}^3 \rightarrow \mathbb{P}^2$ from world points to image points. Given a fiducial coordinate system, we may represent this map with a pair $(B, \mathbf{T})$, where $B : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is a linear function (represented by a 3×3 matrix), and $\mathbf{T}$ is a 3-vector representing the **camera center**. The action of the map on a world point coordinate $\mathbf{P}$ is:

$$C(\mathbf{P}) = B \cdot (\mathbf{P} - \mathbf{T})$$

where $C(\mathbf{P})$ is considered as a member of $\mathbb{P}^2$.

Mel: I see. $\mathbf{T}$ is the translation and B is the rotation between the camera coordinate system and the fiducial world coordinate system. So, you put the rigid transformation in the definition of the camera. Not bad! (Fig. 1.1)

Arc: Well, B is more general than that because it can also hide the calibration information. B was defined as a linear function, not necessarily an orthogonal rotation matrix.

Mel: I have seen cameras defined in terms of projection matrices P. If you know P, then you know the camera. Why do you use your definition instead?

Arc: Good question! We have defined the camera transformation as first a translation and then a matrix multiplication on the world point. This allows us to easily undo the matrix multiplication on the image point by applying B^{-1}. Each camera will then be only a translation away from the fiducial coordinate system (Fig. 1.2), which we will see allows easier derivation of our constraints. B also does not necessarily have to be a linear function. We can remove the linear constraint and our cameras can model nonlinear distortion as is common with real-world cameras.

Mel: You talked about this B being a function rather than an rotation matrix. This

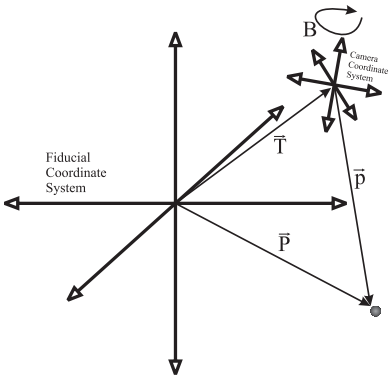

Figure 1.1: The rotation B and translation $\mathbf{T}$ define a camera

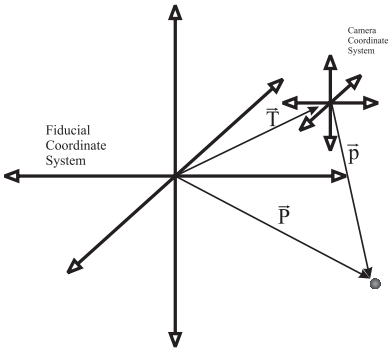

Figure 1.2: Cameras with no rotation B are a translation away from the fiducial coordinate system

seems strange in that you're convoluting the 3D transformation of the camera with the 2D transformation on the image.

Arc: But this can be more natural. Think of your camera as being represented, like we talked about before, as a center of projection and a plane cutting the rays. The only thing we can do with the center of projection is to translate it to different places. The only thing we can do with our plane is to place it in relation to our center of projection. So our formula is naturally geometric in that the $\mathbf{P} - \mathbf{T}$ places our center of projection and our B chooses the plane with which we cut the rays. The rigid rotation of the camera is chosen by the placement of the ray cutting plane.

B may be considered to be a transformation on the world points $\mathbb{R}^3$ or of the image points $\mathbb{P}^2$. In the literature, B is usually split apart using a QR decomposition[9], with the orthogonal matrix representing a rotation of the camera (a transformation on $\mathbb{R}^3$) and the residual matrix representing a linear transformation of the image (a transformation on $\mathbb{P}^2$) (calibration homography). Since the coordinates are the same, we ignore such distinctions and just talk about the B.

Mel: But if you have a single camera moving through space, the calibration matrix, as you call it, will remain the same while the rotation will change.

Arc: Sure, for many purposes it is advantageous to separate our the calibration matrix. For simplicity, we just consider the B matrix as a whole.

Mel: Okay, now we see how points project. I have a fiducial coordinate system somewhere. With regard to that, a point P is seen by camera $(B, \mathbf{T})$ as image point (ray) $p = B(\mathbf{P} - \mathbf{T})$. p is the ray from the camera center to P. How about lines?

1.7.3 Projecting Lines: The Plücker code

Arc: Finding an appropriate coordinatization for lines can be a tricky business. Fortunately, in the case of projection an reconstruction of lines, there is an extremely natural coordinate system, called the Plücker coordinate system. The definition I state now.

A **world line** L is the set of all the points $P \in \mathbb{R}^3$ such that $\mathbf{P} = (1 - \lambda)\mathbf{Q}_1 + \lambda\mathbf{Q}_2$ for two points $\mathbf{Q}_i$, and some scalar λ. If we consider $\lambda = 1$ and $\lambda = 0$, we see that the line L contains both Q_1 and Q_2. The Plücker coordinates of this line

[9] (Yiannis will add a note)

are $\mathbf{L} = \begin{bmatrix} \mathbf{L}_d \\ \mathbf{L}_m \end{bmatrix}$, where:

$$\begin{aligned} \mathbf{L}_d &= \mathbf{Q}_2 - \mathbf{Q}_1 && \text{direction of } L \\ \mathbf{L}_m &= \mathbf{L}_d \times \mathbf{P} && \text{moment of } L \end{aligned}$$

Note that regardless of the choice of λ to define $\mathbf{P}$, the definition of $\mathbf{L}_m$ is the same. Also, the coordinates of $\mathbf{L}$ are homogeneous, and $\mathbf{L}_m^\top \mathbf{L}_d = 0$.

Mel: That seems like a strange definition.

Arc: It may seem strange at first, but when you see the simplicity of our formulas, you will see why this coordinatization was chosen.

Mel: I'll go along with you. What exactly are image lines in this system?

Arc: An image line is a line in $\mathbb{P}^2$, and you may give it coordinates $\boldsymbol{\ell} = [l_1 \; l_2 \; l_3]^{\mathrm{T}}$. A point $\mathbf{p}$ is incident on line $\boldsymbol{\ell}$ if and only if $\mathbf{p}^\top \boldsymbol{\ell} = 0$. As you can easily see, the line $\boldsymbol{\ell}$ connecting points p_1 and p_2 has coordinates $\boldsymbol{\ell} = p_1 \times p_2$; similarly, if point p lies on both $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ then $p = \boldsymbol{\ell}_1 \times \boldsymbol{\ell}_2$. So, if you have two rays p_1 and p_2 they define the line $\boldsymbol{\ell} = p_1 \times p_2$, the intersection of the plane defined by the two rays and the image. This line is expressed as the ray perpendicular to the plane defined by the two rays. That's the cool thing about the projective coordinates. Both points and lines on the image plane are expressed as rays.

Mel: How about the projection? It seems like it would be difficult to go from Plücker lines to these projective lines.

Arc: That's the great thing. The projection of a world line onto a camera is as simple as the projection of a point onto a camera. If you look at the definitions, you see that the coordinates of $\mathbf{L}_m$ are created by a cross product of two points. This is mirrored in the projective plane in that the coordinates of the line incident on two points is the cross product of the coordinates of the points. This is why we chose the Plücker coordinates. If we have a line L and a camera $(B, \mathbf{T})$, then the image line associated with L is

$$\hat{\boldsymbol{\ell}} = B^{-\mathrm{T}}(\mathbf{L}_m - \mathbf{T} \times \mathbf{L}_d)$$

Mel: That's easy to see. If we have two points on the world line P_1 and P_2, the coordinates of their image points on a camera $(B, \mathbf{T})$ are just $B(\mathbf{P}_1 - \mathbf{T})$ and $B(\mathbf{P}_2 - \mathbf{T})$, considered as members of $\mathbb{P}^2$. Then the image line containing them

must be just

$$\begin{aligned}\hat{\boldsymbol{\ell}} &= (B(\mathbf{P}_1 - \mathbf{T})) \times (B(\mathbf{P}_2 - \mathbf{T})) \\ &= B^{-\mathrm{T}}(\mathbf{P}_1 \times \mathbf{P}_2 - (\mathbf{T} \times (\mathbf{P}_2 - \mathbf{P}_1))) \\ &= B^{-\mathrm{T}}(\mathbf{L}_m - \mathbf{T} \times \mathbf{L}_d)\end{aligned}$$

I see why you say projection is just as simple for lines as it is for points. If we ignore the B and $\mathbf{T}$, then our image line $\boldsymbol{\ell}$ is just $\boldsymbol{\ell} = \mathbf{L}_m$, while for points $\mathbf{p} = \mathbf{P}$. It's just that in the case of lines, we ignore the $\mathbf{L}_d$.

Arc: Good. You should note that when the image point coordinates $\mathbf{p}$ are transformed by a map B to $\hat{\mathbf{p}}$ with $\hat{\mathbf{p}} = B\mathbf{p}$, then the image line coordinates $\boldsymbol{\ell}$ are transformed to $\hat{\boldsymbol{\ell}} = B^{-\top}\boldsymbol{\ell}$. If we have a world coordinate system, and a camera in that coordinate system with parameters $(B, \mathbf{T})$, then we consider the $\hat{\mathbf{p}}$ and $\hat{\boldsymbol{\ell}}$ to be the actual point and line coordinates measured in the image. For most of the derivations, we use the normalized image lines/point coordinates:

$$\begin{aligned}\mathbf{p} &= B^{-1}\hat{\mathbf{p}} \\ \boldsymbol{\ell} &= (B^{-\mathrm{T}})^{-1}\hat{\boldsymbol{\ell}} \\ &= B^{\mathrm{T}}\hat{\boldsymbol{\ell}}\end{aligned}$$

Whenever we assume that we already have the B, we will use the normalized image lines/points for the calculations. We multiply the appropriate B or $B^{-\mathrm{T}}$ back later. $\hat{\mathbf{p}}$ and $\hat{\boldsymbol{\ell}}$ are the real points and lines in the image. $\mathbf{p}$ and $\boldsymbol{\ell}$ are the derotated ones and normalized. So, I can develop relationships for points are lines when the cameras are just a translation away from each other and then substitute for $\mathbf{p}$ $B^{-1}\hat{\mathbf{p}}$ and for $\boldsymbol{\ell}$ $B^{\mathrm{T}}\hat{\boldsymbol{\ell}}$ and I get the relation for the real thing. The step of going from $\mathbf{p}$ and $\boldsymbol{\ell}$ to $\hat{\mathbf{p}}$ and $\hat{\boldsymbol{\ell}}$ and vice versa, we'll call calibration, for lack of a better term.

It is also useful to remember the line intersection property for Plücker coordinates (Fig. 1.3), which is easy to prove.

Mel: Now that we know all about projection, we need to find the camera coordinate systems, so let's talk about the constraints given projected points.

1.7.4 Reconstructing points and lines

Arc: Actually, it's a lot easier if we do it in the wrong order, and talk about reconstruction first, assuming we already know the camera parameters.

Mel: If you say so.

Arc: We need to know how we can reconstruct a world line and a world point. Given two arbitrary cameras, it is not in general possible to reconstruct a world point from two image points, unless they satisfy some condition, which is the epipolar constraint. A moment's thought will confirm this, since the two world lines formed by the image points with their respective centers of projection do not in general intersect. It is possible to form a joint reconstruction/constraint, but this complicates matters, and yields no benefit. So, let's see how we reconstruct a line. If we have a line L in space which projects to two image lines $\hat{\boldsymbol{\ell}}_1$ and $\hat{\boldsymbol{\ell}}_2$ in cameras $(B_1, \mathbf{T}_1)$, and $(B_2, \mathbf{T}_2)$, then we can calculate the coordinates for $\mathbf{L}$ if $|\boldsymbol{\ell}_1 \times \boldsymbol{\ell}_2| \neq \mathbf{0}$ as in figure 1.4.

This not hard to prove, but to prove it you will have to assume that the translation between the cameras cannot be perpendicular to the moment vector of the world line. Translating in this plane leaves the image line the same in both cameras, so that there is no depth information in the images, and no reconstruction is possible.

Mel: But we could still calculate the formula in the case that $|\boldsymbol{\ell}_1 \times \boldsymbol{\ell}_2| = 0$, couldn't we? We would just get the zero vector for the answer.

Arc: I hadn't thought of that. Actually, this is even more general than that. Consider if instead of being zero, that cross product has a fairly small magnitude. Our $\mathbf{L}$ will have a small magnitude. But this is just the case where our reconstruction is likely to be errorful. The magnitude of $\mathbf{L}$ is a confidence measurement of our reconstruction.

Mel: And no divisions! But why do you start with lines and not reconstruct points?

Arc: I will tell you about points, although it is not so useful, in order to explain the concept of camera collapse. While it is not in general possible to reconstruct a world point from two arbitrary cameras and image points, it is possible to reconstruct a world point from three arbitrary cameras using image lines which are

If we have two lines $\mathbf{L}_1$ and $\mathbf{L}_2$, they intersect if and only if

$$\mathbf{L}_{d,1}^{\mathrm{T}}\mathbf{L}_{m,2} + \mathbf{L}_{d,2}^{\mathrm{T}}\mathbf{L}_{m,1} = 0$$

Figure 1.3: Line intersection property

We are given world line $\mathbf{L}$ projected to two lines $\hat{\boldsymbol{\ell}}_i$, $i \in \{1, 2\}$ by cameras $(B_i, \mathbf{T}_i)$. If we set $\boldsymbol{\ell}_i = B_i^{\mathrm{T}}\hat{\boldsymbol{\ell}}_i$, then the Plücker coordinates of the world line are:

$$\mathbf{L} = \begin{bmatrix} \boldsymbol{\ell}_1 \times \boldsymbol{\ell}_2 \\ \boldsymbol{\ell}_1\mathbf{T}_2^{\mathrm{T}}\boldsymbol{\ell}_2 - \boldsymbol{\ell}_2\mathbf{T}_1^{\mathrm{T}}\boldsymbol{\ell}_1 \end{bmatrix}$$

Figure 1.4: Line reconstruction

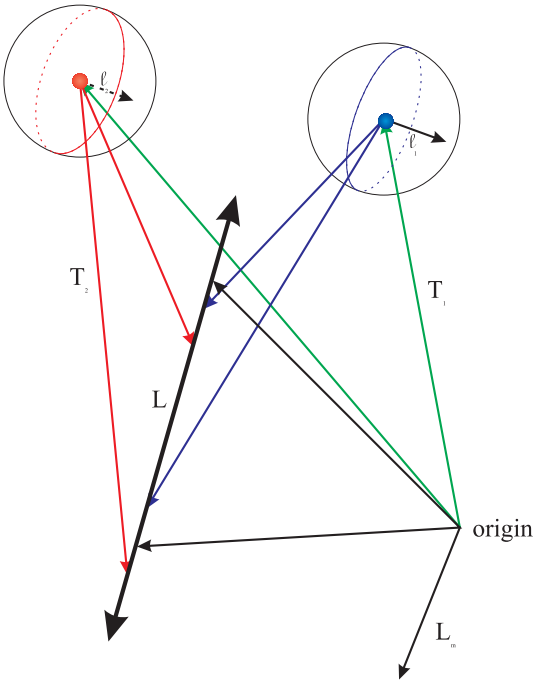

Figure 1.5: Reconstruction of a world line

incident on the world point's image in each of the three cameras.

Mel: What do these arbitrary image lines have to do with anything? They aren't measured, so how do you pick them?

Arc: We just pick three lines which go through image points. Is there a problem with that?

Mel: But as I understand it, lines are usually formed by finding corners, which are just the intersection of points. Why introduce this reconstruction at all if it doesn't concern our basic measurements?

Arc: You're right, and that's why this reconstruction is just for show, so to speak.

If we project a world point P into three cameras with parameters $(B_i, \mathbf{T}_i)$, and we have measured image lines $\hat{\boldsymbol{\ell}}_i$ which go through the image points $\hat{\mathbf{p}}_i$, then the coordinates of P are

$$\begin{aligned}\mathbf{P} &= \begin{bmatrix}\boldsymbol{\ell}_1^{\mathrm{T}}\\ \boldsymbol{\ell}_2^{\mathrm{T}}\\ \boldsymbol{\ell}_3^{\mathrm{T}}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{\ell}_1^{\mathrm{T}}\mathbf{T}_1\\ \boldsymbol{\ell}_2^{\mathrm{T}}\mathbf{T}_2\\ \boldsymbol{\ell}_3^{\mathrm{T}}\mathbf{T}_3\end{bmatrix}\\ &= \frac{(\boldsymbol{\ell}_2\times\boldsymbol{\ell}_3)\boldsymbol{\ell}_1^{\mathrm{T}}\mathbf{T}_1 + (\boldsymbol{\ell}_3\times\boldsymbol{\ell}_1)\boldsymbol{\ell}_2^{\mathrm{T}}\mathbf{T}_2 + (\boldsymbol{\ell}_1\times\boldsymbol{\ell}_2)\boldsymbol{\ell}_3^{\mathrm{T}}\mathbf{T}_3}{|\boldsymbol{\ell}_1\ \boldsymbol{\ell}_2\ \boldsymbol{\ell}_3|}\end{aligned}$$

If we have a point in one camera and a line in the other, we can consider $\mathbf{T}_3 = \mathbf{T}_2$, $B_3 = B_2$ and $\mathbf{p} = \boldsymbol{\ell}_2 \times \boldsymbol{\ell}_3$, and obtain

$$\mathbf{P} = \frac{\mathbf{p}_2\boldsymbol{\ell}_1^{\mathrm{T}}(\mathbf{T}_1 - \mathbf{T}_2)}{\boldsymbol{\ell}_1^{\mathrm{T}}\mathbf{p}_2} + \mathbf{T}_2$$

This is easy to prove.

The second result is essentially the same as the first, except with cameras 2 and 3 considered as identical. We call this process "camera collapse". Because we can consider a point as the cross product of two lines, we get an equation in terms of a point (which can be considered the intersection of two lines) and another line. We will use this principle throughout our discussion by really only proving constraints for lines, and then collapsing pairs of cameras in order to obtain constraints on points.

Let me point out, that in most cases, we will not have isolated points which we must reconstruct, since most points are located as the intersection of lines. Even if we have isolated points, if there are at least three, then we may as well consider the lines joining them rather than the points themselves. We choose to operate with

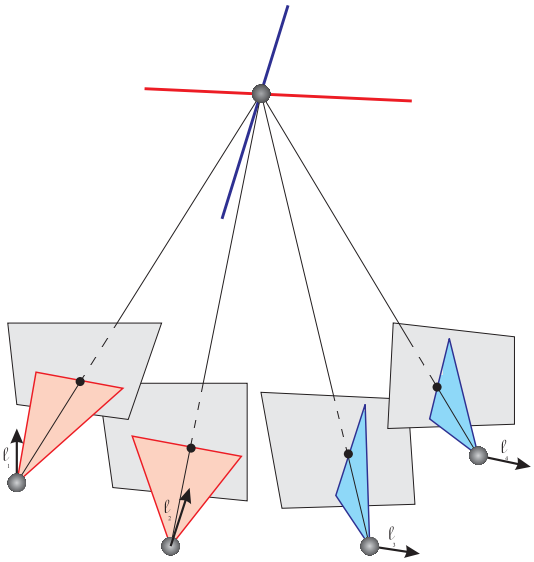

Figure 1.6: Quadrilinear Constraint with Virtual Red and Blue Lines

$$|\boldsymbol{\ell}_4\ \boldsymbol{\ell}_3\ \boldsymbol{\ell}_2|\boldsymbol{\ell}_1^{\mathrm{T}}\mathbf{T}_1+|\boldsymbol{\ell}_3\ \boldsymbol{\ell}_4\ \boldsymbol{\ell}_1|\boldsymbol{\ell}_2^{\mathrm{T}}\mathbf{T}_2 +|\boldsymbol{\ell}_2\ \boldsymbol{\ell}_1\ \boldsymbol{\ell}_4|\boldsymbol{\ell}_3^{\mathrm{T}}\mathbf{T}_3+|\boldsymbol{\ell}_1\ \boldsymbol{\ell}_2\ \boldsymbol{\ell}_3|\boldsymbol{\ell}_4^{\mathrm{T}}\mathbf{T}_4 = 0$$

Figure 1.7: Quadrilinear constraint equation

the lines rather than their intersection, so we will not use the point reconstruction later in our discussion.

1.7.5 Multilinear constraints

Mel: OK, now are we are now ready for the multi-linear constraints?

Arc: Yes, and let's start with the quadrilinear constraint, shown in figure 1.6. If we have a world point $\mathbf{P}$ which projects to cameras 1 through 4 with parameters $(B_i, \mathbf{T}_i)$, and the image points are intersected by lines $\hat{\boldsymbol{\ell}}_i$, then if we set $\hat{\boldsymbol{\ell}}_i = {B_i}^{-\mathrm{T}}\boldsymbol{\ell}_i$, the equation in figure 1.7 holds.

You can easily see it by reconstructing the world lines from the image lines in the camera pairs (1, 2) and (3, 4) to the red and blue lines in figure 1.6, respectively. These are virtual world lines, not necessarily having any reality, but even so they can be expressed as:

$$\mathbf{L}_{1,2} = \begin{bmatrix} \boldsymbol{\ell}_1 \times \boldsymbol{\ell}_2 \\ \boldsymbol{\ell}_1 \mathbf{T}_2^{\mathrm{T}} \boldsymbol{\ell}_2 - \boldsymbol{\ell}_2 \mathbf{T}_1^{\mathrm{T}} \boldsymbol{\ell}_1 \end{bmatrix}$$
$$\mathbf{L}_{3,4} = \begin{bmatrix} \boldsymbol{\ell}_3 \times \boldsymbol{\ell}_4 \\ \boldsymbol{\ell}_3 \mathbf{T}_4^{\mathrm{T}} \boldsymbol{\ell}_4 - \boldsymbol{\ell}_4 \mathbf{T}_3^{\mathrm{T}} \boldsymbol{\ell}_3 \end{bmatrix}$$

Both these lines must contain the world point $\mathbf{P}$. Therefore the lines must intersect. If you write down this condition for the intersection of two lines in 3D (1.3) you get the constraint.

Mel: How about the trifocal constraint?

Arc: If we have three cameras with parameters $(B_i, \mathbf{T}_i)$, and a world point P which projects to $\hat{\mathbf{p}}_i$, then if we have image lines $\hat{\boldsymbol{\ell}}_1$ and $\hat{\boldsymbol{\ell}}_3$ which intersect their respective image points, then if we make the usual calibration:

$$\mathbf{T}_1^{\mathrm{T}} \boldsymbol{\ell}_1 \boldsymbol{\ell}_3^{\mathrm{T}} \mathbf{p}_2 - \mathbf{T}_3^{\mathrm{T}} \boldsymbol{\ell}_3 \boldsymbol{\ell}_1^{\mathrm{T}} \mathbf{p}_2 - (\mathbf{T}_2 \times \mathbf{p}_2)^{\mathrm{T}} (\boldsymbol{\ell}_1 \times \boldsymbol{\ell}_3) = 0$$

This is known as the **point trilinear constraint**. You get it by collapsing cameras in the previous constraint, as in figure 1.8. If you set $\mathbf{T}_4 = \mathbf{T}_2$, you get:

$$\mathbf{T}_1^{\mathrm{T}} \boldsymbol{\ell}_1 \boldsymbol{\ell}_2^{\mathrm{T}} (\boldsymbol{\ell}_4 \times \boldsymbol{\ell}_3) + \mathbf{T}_2^{\mathrm{T}} \boldsymbol{\ell}_2 \boldsymbol{\ell}_1^{\mathrm{T}} (\boldsymbol{\ell}_3 \times \boldsymbol{\ell}_4) + \mathbf{T}_3^{\mathrm{T}} \boldsymbol{\ell}_3 \boldsymbol{\ell}_4^{\mathrm{T}} (\boldsymbol{\ell}_2 \times \boldsymbol{\ell}_1) + \mathbf{T}_2^{\mathrm{T}} \boldsymbol{\ell}_4 \boldsymbol{\ell}_3^{\mathrm{T}} (\boldsymbol{\ell}_1 \times \boldsymbol{\ell}_2) = 0$$

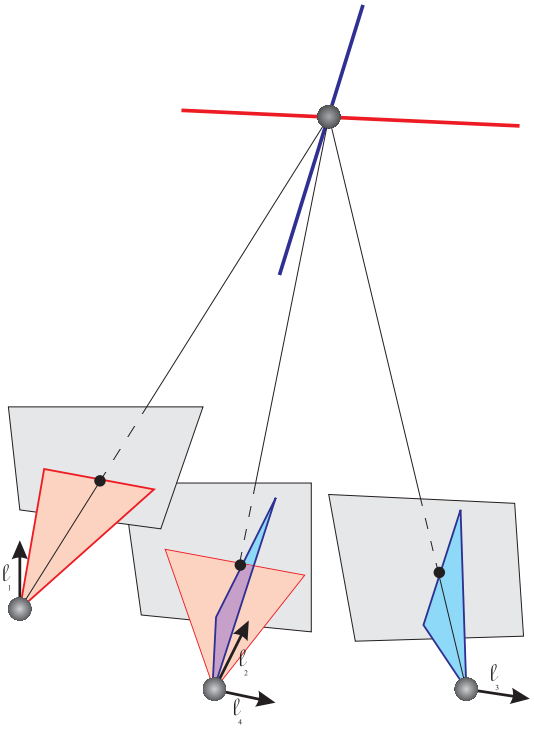

Figure 1.8: Trilinear Constraint with Virtual Red and Blue Lines

or by simplifying

$$\mathbf{T}_1^{\mathrm{T}}\boldsymbol{\ell}_1\boldsymbol{\ell}_3^{\mathrm{T}}(\boldsymbol{\ell}_2\times\boldsymbol{\ell}_4) - \mathbf{T}_3^{\mathrm{T}}\boldsymbol{\ell}_3\boldsymbol{\ell}_1^{\mathrm{T}}(\boldsymbol{\ell}_2\times\boldsymbol{\ell}_4) + (\mathbf{T}_2\times(\boldsymbol{\ell}_4\times\boldsymbol{\ell}_2))^{\mathrm{T}}(\boldsymbol{\ell}_1\times\boldsymbol{\ell}_3) = 0$$

Note that whatever $\boldsymbol{\ell}_2$ and $\boldsymbol{\ell}_4$ we choose, we will end up with the same $\mathbf{p}_2 = \boldsymbol{\ell}_2\times\boldsymbol{\ell}_4$.

Mel: I guess now you will collapse cameras and go from the trilinear to the epipolar constraint?

Arc: Exactly. If we have a world point P projected onto two cameras with parameters $(B_i, \mathbf{T}_i)$ at image points $\hat{\mathbf{p}}_i$, then if we make the usual calibration assumption, we have the **epipolar constraint**:

$$(\mathbf{T}_1\times\mathbf{p}_1)^{\mathrm{T}}\mathbf{p}_2 + (\mathbf{T}_2\times\mathbf{p}_2)^{\mathrm{T}}\mathbf{p}_1 = 0$$

If we set $\mathbf{T}_3 = \mathbf{T}_1$ in equation (1.7.5), we obtain:

$$\mathbf{T}_1^{\mathrm{T}}\boldsymbol{\ell}_1\boldsymbol{\ell}_3^{\mathrm{T}}\mathbf{p}_2 - \mathbf{T}_1^{\mathrm{T}}\boldsymbol{\ell}_3\boldsymbol{\ell}_1^{\mathrm{T}}\mathbf{p}_2 - (\mathbf{T}_2\times\mathbf{p}_2)^{\mathrm{T}}(\boldsymbol{\ell}_1\times\boldsymbol{\ell}_3) = 0$$

or by simplifying

$$-(\mathbf{T}_1\times(\boldsymbol{\ell}_1\times\boldsymbol{\ell}_3))^{\mathrm{T}}\mathbf{p}_2 - (\mathbf{T}_2\times\mathbf{p}_2^{\mathrm{T}}(\boldsymbol{\ell}_1\times\boldsymbol{\ell}_3) = 0$$

Again, it does not matter which $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_3$ we choose, because we will end up with the same $\mathbf{p}_1 = \boldsymbol{\ell}_1\times\boldsymbol{\ell}_3$.

Mel: To me, while the epipolar constraint makes sense, the quadrilinear and trilinear constraints leave me unsatisfied. They have this artificial selection of arbitrary image lines. This seems especially strange considering that we usually construct points by intersecting lines. Is there a way we can use the lines directly?

Arc: Indeed, there is a constraint based directly on lines. Strangely enough, it uses exactly the same equation as the trilinear constraint.

The previous constraints we got by considering lines intersecting in space. Now consider lines coinciding in space. Here is another constraint if you consider lines. If we have three cameras with parameters $(B_i, \mathbf{T}_i)$, and a world line L which projects to $\hat{\boldsymbol{\ell}}_i$, then if we have an image point $\hat{\mathbf{p}}_2$ which is on $\hat{\boldsymbol{\ell}}_2$ and we make the usual calibration:

$$\mathbf{T}_1^{\mathrm{T}}\boldsymbol{\ell}_1\boldsymbol{\ell}_3^{\mathrm{T}}\mathbf{p}_2 - \mathbf{T}_3^{\mathrm{T}}\boldsymbol{\ell}_3\boldsymbol{\ell}_1^{\mathrm{T}}\mathbf{p}_2 - (\mathbf{T}_2\times\mathbf{p}_2)^{\mathrm{T}}(\boldsymbol{\ell}_1\times\boldsymbol{\ell}_3) = 0 \quad (1.1)$$

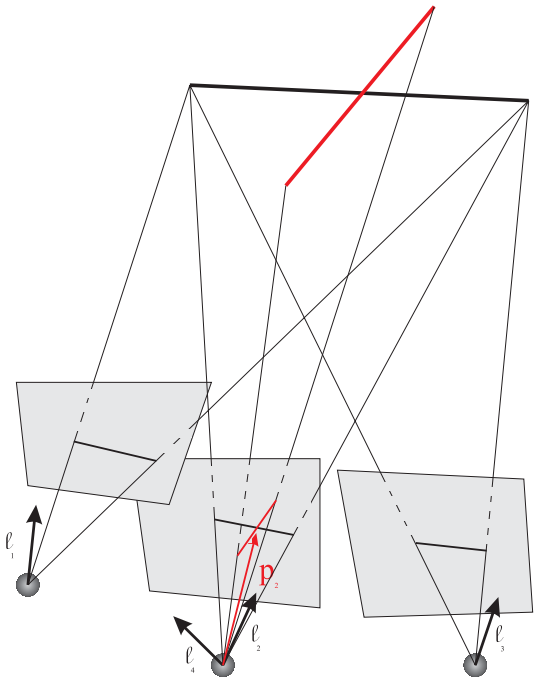

Figure 1.9: The Line Trilinear Constraint

which is the same equation, but with different measurements, and is known as the **line trilinear constraint**.

Again, it is easy to prove. We may reconstruct our line as:

$$\mathbf{L}_d = \boldsymbol{\ell}_3 \times \boldsymbol{\ell}_1$$
$$\mathbf{L}_m = \boldsymbol{\ell}_3 \mathbf{T}_1^{\mathrm{T}} \boldsymbol{\ell}_1 - \boldsymbol{\ell}_1 \mathbf{T}_3^{\mathrm{T}} \boldsymbol{\ell}_3$$

If we project this line into the second camera, we obtain

$$\boldsymbol{\ell}_2 = \boldsymbol{\ell}_3 \mathbf{T}_1^{\mathrm{T}} \boldsymbol{\ell}_1 - \boldsymbol{\ell}_1 \mathbf{T}_3^{\mathrm{T}} \boldsymbol{\ell}_3 - \mathbf{T}_2 \times (\boldsymbol{\ell}_3 \times \boldsymbol{\ell}_1)$$

Therefore, if we consider any point $\mathbf{p}_2$ on $\boldsymbol{\ell}_2$, we must have the constraint $\mathbf{p}_2^{\mathrm{T}} \boldsymbol{\ell}_2 = 0$, i.e.:

$$\mathbf{p}_2^{\mathrm{T}} \boldsymbol{\ell}_3 \mathbf{T}_1^{\mathrm{T}} \boldsymbol{\ell}_1 - \mathbf{p}_2^{\mathrm{T}} \boldsymbol{\ell}_1 \mathbf{T}_3^{\mathrm{T}} \boldsymbol{\ell}_3 - (\mathbf{p}_2 \times \mathbf{T}_2)^{\mathrm{T}} (\boldsymbol{\ell}_3 \times \boldsymbol{\ell}_1) = 0$$

Note that, while this equation has the same form as the point trilinear constraint, it operates on different objects (Fig. 1.9). Indeed, I have understood that the only constraint that is relevant for points is the epipolar constraint, while the only constraint that matters for lines is the above line trilinear constraint. The quadrilinear and point trilinear are redundant with these two constraints if we consider three or more points.[10]

Mel: So, given many views of the world the only thing you can I assume if you make the coordinates the same as before, these constraints are equivalent to the ones you showed me before.

Arc: These are the same constraints indeed. It is a simple exercise to bring them to any form. One of the nice things about the form they are in is that the positions of the cameras is explicit for *all* cameras, which means that they are easier to integrate into a many camera system. When people use two or three cameras, they model their locations through the rigid motions from camera 1 to 2, from 2 to 3, etc. If you have hundreds of cameras this scheme makes no sense. What I just described is the best for this case.

Mel: So, given many views of the world the only thing you can tell me about corresponding points and lines is that they satisfy these two line constraints, the

[10] yiannis will add a note here

epipolar and the line trilinear?

Arc: Yes, that's it!

Mel: Well, with just these two constraints I expect to get 3D models, 3D video and photography?

Arc: Probably not. But people have developed them further and they have introduced a computational framework[11] for recovering camera location and then reconstruction, as I told you. Remember, you are looking for $6(N-1)-1$ numbers, expressing the rigid transformation between any two views for N cameras, up to an overall scale ambiguity, which accounts for the -1. The 6 comes from 3 are for the rotation and 3 for the translation. The community has built several interesting optimization procedures that take as input corresponding points and lines and provide as output camera placement. They cannot do it perfectly yet, but they are getting there.

Mel: I can now sort of see the problem. All these procedures make errors that if we translate them to image quantities amount to a few pixels, which is a very small error indeed but unacceptable when it comes down to 3D models. When you have 3D models of something from several views and you want to put them together, a few inaccuracies here and there create misregistrations which create problems. Besides, quite often the errors are large.

Another problem is that correspondence is only among individual points. While this can be sufficient to compute some rigid motions, it is insufficient to calculate a full 3D model to the degree that we desire.

Arc: Very well said.

Mel: I need to study this stuff better. Maybe you can point me to some references?

Arc: Certainly. One thing to remember is that this field in computer vision grew out of photogrammetry. Of course researchers found out about photogrammetry as they worked on multiview geometry; so, they replicated some results. But they also found new things.

Mel: You mentioned this field before.

Arc: These are the map makers. After photography was invented, they started taking pictures of the world and making detailed maps of cities, landscapes, moun-

[11] The framework basically amounts to the estimation of the fundamental (or essential) matrix and the trilinear tensor. It is described in a large collection of papers and in recent surveys [1, 2].

tains, etc. That's why you see all this effort in finding how to place the cameras using as few points and as few images as possible. Needless to say that photogrammeters picked corresponding points by hand.

Mel: I was wondering about that. A video camera gets 60 frames per second and there are thousands of features in the images.

Arc: When photogrammetry started, there were no computers. Cameras and film were scarce too. But this specific need set foundations in the field. We now know what is possible to do with the minimum amount of information. So, if you are interested in this field, get at least a hold of the *Photogrammetry Manual*, and understand the meaning of the terms they use, because they are not identical to the terms used in computer vision.

Mel: Good. How about computer vision?

Arc: They breakthrough came when it was realized by Longuet-Higgins (1982) that there is a linear map relating two views. That's the essential matrix E. Because of the linearity, people could study the uniqueness properties of the problem. That was done first by Tsai and Huang (1984) and subsequently by several others who also discovered a wealth of information in the photogrammetric literature. It was, for example, discovered that you need a minimum of seven points. With eight points it is clear because you can solve for the elements of E. Are there degenerate cases, that is, cases where even if all your correspondences are perfect, you still get non-unique solutions for camera placement, i.e., for matrix E? It turns out that degeneracy occurs when both camera centers and the 3D points lie on a (ruled) quadric surface, known as the critical surface. The quadric can be non-degenerate—a hyperboloid of one sheet, or degenerate (e.g., two planes, cones, or cylinders), but it cannot be a hyperboloid of two sheets or an ellipsoid (see Maybank, 1993).

Mel: Seven points are enough? But seven points and the two camera centers make nine points and a general quadric has nine points and a general quadric has nine degrees of freedom. So I can always make a quadric to pass from all these points. Does that mean that I cannot have a unique solution?

Arc: Well, if the quadric happens to be not ruled, then you have a unique solution. If the quadric is a ruled quadric, then it will be a critical surface and there will be

three possible solutions.

Mel: OK.

Arc: Then, the trilinear trifocal constraint, the constraint for points and lines in three views was developed by Spetsakis and Aloimonos (1987, 1989, 1990). The projective generalization of the essential matrix was introduced as the fundamental matrix by Faugeras (1992) and Hartley (1993), following a paper on affine structure from motion by J. Koenderink and A. van Doorn (1992), and the work of R. Mohr on projective geometry and vision (1994). The projective generalization of the trifocal constraint was done by Shashua (1995). Since then many groups have built on top of these mathematics and now a beautiful theory is in place.

Mel: Now that you are at it, I must say that I did not fully understand how from projective or affine models we can upgrade to Euclidean models. I understood how you go from projective to affine through the line at infinity, but I missed the rest. Can you tell me, just for 2D models?

Arc: I will tell you, but you need to know a few things first. Recall for a moment the conics we talked about. In homogeneous coordinates a conic is written as $x^\top C x = 0$, where C is a 3×3 symmetric matrix.

Mel: You mean the points x satisfying $x^\top C x = 0$ lie on the conic?

Arc: Yes. You see that C is a homogeneous representation. If you multiply its entries by some scale, nothing changes.

Mel: So, since C is symmetric it has five degrees of freedom. No wonder you need five points to define a conic.

Arc: Very good. Because of the duality, you can change points with lines. You still get a conic; that is, the equation $\boldsymbol{\ell}^\top C^* \boldsymbol{\ell} = 0$ expresses the conic that is tangent to lines $\boldsymbol{\ell}$, where C^* is a 3×3 symmetric matrix.

Mel: Why do you write it as C^*?

Arc: Oh, to denote that it is the dual of $x^\top C x = 0$. It turns out that the matrix of the dual conic is the adjoint of the original matrix C^*, which happens to also be C^{-1}, up to scale of course. You can also have degenerate conics if the matrix C is not of full rank. For example, the conic $C = \boldsymbol{\ell}_1 \boldsymbol{\ell}_2^\top + \boldsymbol{\ell}_2 \boldsymbol{\ell}_1^\top$ consists of two lines $\boldsymbol{\ell}_1$

and $\boldsymbol{\ell}_2$.

Mel: OK, what is your point?

Arc: Recall how we upgraded a projective model to an affine one?

Mel: Yes, using the image of the line at infinity.

Arc: Good. Now we need to upgrade to a Euclidean model (with scale). That means that our homography will be a similarity transformation. So, we have to find something that does not change under a similarity transform.

Mel: I see, just like in the affine case you found the line at infinity as being unchanged by an affine transformation.

Arc: Exactly. These things for our case are the circular points.

Mel: What are those?

Arc: They are the points where every circle intersects the line at infinity $(0\,0\,1)^\top$. In homogeneous coordinates, if $x = (x_1\, x_2\, x_3)^\top$ the equation of a circle is $x_1^2 + x_2^2 + ax_1x_3 + bx_2x_3 + cx_3^2 = 0$. Its intersection with the line at infinity is $x_1^2 + x_2^2 = 0$, which amounts to two points $I = (1, i, 0)$ and $J = (1, -i, 0)$. These are the circular points at infinity and any similarity transformation will not change them.

Mel: Wow! It sounds weird.

Arc: Well, look at I. You can write it as $I = (1, 0, 0)^\top + i(0, 1, 0)^\top$. So, inside I you have the two orthogonal directions (1, 0, 0) and (0, 1, 0) of Euclidean geometry. It is no wonder then that once these are found you can determine orthogonality and other metric properties.

Mel: There is something I don't quite understand. To upgrade to a Euclidean model I must have the capability to measure angles. How do I do this if I have a projective model? I know that if I have, in Euclidean geometry, lines $\boldsymbol{\ell}_1(\ell_{11}, \ell_{12}, \ell_{13})^\top$ and $\boldsymbol{\ell}_2(\ell_{21}, \ell_{22}, \ell_{23})^\top$ then their angle is given by the dot product of their normals, that is, their angle θ is such that $\cos\theta = \frac{\ell_{11}\ell_{21}+\ell_{12}\ell_{22}}{\sqrt{(\ell_{11}^2+\ell_{12}^2)(\ell_{12}^2+\ell_{22}^2)}}$. I cannot use this formula in a projective model because it is not invariant under projective transformations.

Arc: That's good! You can, however, use another formula, this one:

$$\cos\theta = \frac{\boldsymbol{\ell}_1^\top C^* \boldsymbol{\ell}_2}{\sqrt{(\boldsymbol{\ell}_1^T C^* \boldsymbol{\ell}_1)(\boldsymbol{\ell}_2^T C^* \boldsymbol{\ell}_2)}}$$

Mel: Is C^* a conic?

Arc: Yes. In actual fact it is the conic which is dual to the circular points, that is:

$$C^* = IJ^\top + I^\top J$$

Mel: Hmm, how does C^* transform under similarity transformations? Oh, since $C^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ it is fixed under similarity. Now, let's see what happens with a general projective transformation. If H maps points x to x', i.e., $x' = Hx$, then lines map as $\boldsymbol{\ell}' = H^{-\top}\boldsymbol{\ell}$. How about conics?

Arc: It's not hard.

Mel: Let's say I have conic C, with $x^\top Cx = 0$. if instead of x I use $x' = Hx$, I have $x'^\top [H^{-1}]^\top C H^{-1} x' = 0$ or $x'^\top H^{-\top} C H^{-1} x' = 0$, i.e., the new conic is $C' = H^{-\top} C H^{-1}$. So, by applying H to your formula, it doesn't change it.

Arc: Yes, when you find C^*, you can start measuring angles. Clearly, if $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ are orthogonal, you have $\boldsymbol{\ell}_1^\top C^* \boldsymbol{\ell}_2 = 0$. So, if you have two lines in the image and you know they are orthogonal in space, you can constrain C^* with this equation. If you have 5 such pairs, you can find C^* (you need five points to define a conic).

Mel: So, if I have C^* I can upgrade my projective model.

Arc: The word used is rectify.

Mel: OK, I can rectify.

Arc: Keep in mind that C^* contains all the information needed to determine both the projective and affine components of the model, leaving only the similarity component undetermined.

Mel: I don't fully follow.

Arc: Here is a good way to think about it. Take any projective transformation in 2D. You can write it in block form as $H = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{u}^\top v & \end{bmatrix}$ with $\mathbf{t}$ and $\mathbf{u}$ 2 by 1 vectors, v a constant and A a 2×2 nonsingular matrix, written as $A = SRK + \mathbf{t}u^\top$, where K an upper triangular matrix normalized as $\det(K) = 1$. It is easy to see that $H = H_{\text{sim}} H_{\text{aff}} H_{\text{proj}} = \begin{bmatrix} SR & \mathbf{t} \\ O^\top & 1 \end{bmatrix} \begin{bmatrix} K & O \\ O^\top & 1 \end{bmatrix} \begin{bmatrix} I & O \\ \mathbf{u}^\top & v \end{bmatrix}$. As you

see, inside the general projective transform, you have a similarity, an affine and a projective one. So, if you have a projective transformation $x' = Hx$ where the x coordinate frame is Euclidean and x' projective, then our conic C^* transforms as: $C^* = HC^*H^\top$ or $C^* = H_{\text{sim}}H_{\text{aff}}H_{\text{proj}}C^*(H_{\text{sim}}H_{\text{aff}}H_{\text{proj}})^\top$ or $C^{*\prime} = (H_{\text{proj}}H_{\text{aff}})(H_{\text{sim}}C^*{H_{\text{sim}}}^\top)({H_{\text{aff}}}^\top{H_{\text{proj}}}^\top)$, i.e., $C^{*\prime} = H_{\text{proj}}H_{\text{aff}}C^*{H_{\text{aff}}}^\top{H_{\text{proj}}}^\top$ since C^* is invariant to similarity transformations. So, $C^{*\prime} = \begin{bmatrix} KK^\top & K^\top\mathbf{u} \\ \mathbf{u}^\top & 0 \end{bmatrix}$ which means that the components K and $\mathbf{u}$ are determined by $C^{*\prime}$. But these are the affine and projective parts!

Mel: Very clear. I guess now I could find a homography that brings me from projective to Euclidean.

Arc: Sure. You can do directly from the $C^{*\prime}$ that you found. Because a SVD of $C^{*\prime}$ gives $C^{*\prime} = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^\top$, U is your rectifying homography.

Mel: It is pretty clear now. I assume similar things happen in 3D?

Arc: In 3D, the new concept is the plane at infinity Π. Two planes are parallel iff their line of intersection is on Π and an line is parallel to another line or plane, if the point of intersection is on Π. Again, the plane at infinity is fixed under a projective transformation iff the transformation is affine.

Mel: So, I just have to apply a homography that moves Π to its canonical position $(0,0,0,1)^\top$. That homography gives me an affine model. But how about the analog of the circular points? How can I get to Euclidean models and measure angles?

Arc: In this case you have the absolute conic. The absolute conic lies on the plane at infinity Π. A point $x = (x_1, x_2, x_3, x_4)^\top$ lies on Π when $x_4 = 0$, since $\Pi - (0,0,0,1)^\top$ and x lies on Π when $\Pi^\top x = 0$. So, the conic is $(x_1, x_2, x_3)I(x_1, x_2, x_3)^\top = 0$, where I is the unit matrix. The conic can also be written as $x_1^2 + x_2^2 + x_3^2 = 0$, $x_4 = 0$.

The absolute conic Ω is fixed under any similarity transformation. So, once you have identified Ω and Π, you can measure angles. If you have lines $\boldsymbol{\ell}_1$ and $\boldsymbol{\ell}_2$ angle is $\cos\theta = \frac{\boldsymbol{\ell}_1^\top \boldsymbol{\ell}_2}{\sqrt{(\boldsymbol{\ell}_1^\top \boldsymbol{\ell}_1)(\boldsymbol{\ell}_2 \top \boldsymbol{\ell}_2)}}$. Now, to have this formula in projective space you

should modify it as $\cos\theta = \frac{\boldsymbol{\ell}_1^\top \Omega \boldsymbol{\ell}_2}{\sqrt{(\boldsymbol{\ell}_1^\top \Omega \boldsymbol{\ell}_1)(\boldsymbol{\ell}_2^\top \Omega \boldsymbol{\ell}_2)}}$ with Ω the absolute conic.

Mel: Nice! I assume I will have to identify the absolute conic. By the way, it is just consisting of imaginary points, right?

Arc: Yes. There are many things you can do now. You can see that calibrating a camera amounts to finding the image of the absolute conic. Take a general camera $P = KR[I - C]$, and a point on the plane at infinity Π. The point would be of the form $X = [\boldsymbol{\ell}^\top, 0]^\top$, where $\boldsymbol{\ell}$ a 3×1 vector. The mapping between Π and the image is clearly a homography. Let's find the image of X.

Mel: $x = PX = KR\boldsymbol{\ell}$. So, point $\boldsymbol{\ell}$ on Π maps to image point x, with homography KR.

Arc: Good. How does the absolute conic Ω map to the image?

Mel: If x maps to x' with homography H, i.e., $x' = Hx$, then conic C maps to $H^{-\top}CH^{-1}$. The absolute conic Ω is the identity matrix I, so it maps to $(KR)^{-\top}I(KR)^{-1} = (KK^\top)^{-1}$. Wow! The image w of the absolute conic Ω depends only on the calibration parameters, it does not depend on orientation R and position C.

Arc: Excellent. You see clearly that an uncalibrated camera is like a projective device, and now you can use all these results to perform camera calibration. You need to have some knowledge about how lines are in 3D (parallel, perpendicular) and this knowledge helps you calibrate or equivalently rectify your models from projective to affine and Euclidean. If x is a ray (image point) in an uncalibrated image, the calibration homography relates the ray's direction $\boldsymbol{\ell}^*$ with x two rays is $\cos\theta = \frac{\boldsymbol{\ell}_1^\top \boldsymbol{\ell}_2}{\sqrt{(\boldsymbol{\ell}_1^\top \boldsymbol{\ell}_1)(\boldsymbol{\ell}_2^\top \boldsymbol{\ell}_2)}}$ or $\cos\theta = \frac{x_1^\top (K^{-\top}K^{-1}) x_2}{\sqrt{x_1^\top (K^{-\top}K^{-1}) x_1}\sqrt{x_2^\top (K^{-\top}K^{-1}) x_2}}$ or $\cos\theta = \frac{x_1^\top \omega x_2}{\sqrt{x_1^\top \omega x_1}\sqrt{x_2^\top \omega x_2}}$.

Mel: This connection between these projective structures and calibration is really elegant. Where can I learn more about it?

Arc: It started with the work of Faugeras (199?). The book by Hartley and Zisserman (2000) is an excellent source of ideas, concepts and techniques around the general problem of calibration.

Mel: What about work on geometry? Are these ideas coming from projective

geometry?

Arc: Of course. I recommend the books . . . and . . . for a start.

Mel: Where can I learn more about visualizing (getting some intuition) about projective spaces or higher-dimensional spaces?

Arc: The books . . . and . . . are probably among the best. It is also worth studying *Flatland* and other books similar to it such as *Sphereland* by . . . and the books of Rudy Rucker, . . . and . . . .

Mel: The final thing I wanted to ask you is about specific algorithms. How do you really place the cameras, how do you apply the constraints, how do you find some correspondences?

Arc: There is a lot written on this subject. First, you go in the images and you apply an operator to find points. There are many ways to do that. These points are corners, points of high curvature. There are quite a few "interest point" operators (the Harris operator (19??), the Frei-Chen operator (19??), and others). The problem is that these operators are not perfect, they are based on local variations in the images, and as you know, images change when you change viewpoint. So, if your favorite operator gives n_1 points in the first image and n_2 in the second, there are many points in each image whose corresponding one has not been detected in the other.

Mel: So, many algorithms will have problems.

Arc: Yes. Let's look now at the correspondence algorithms. They look at a patch around a point in one image and collect the statistics of that neighborhood. Then they look in the other image for a point whose neighborhood has the "same" statistics as the first. Correlation, sum of the squared differences are at the top of possible candidates [REFS].

Mel: But then you don't expect to get all correspondences right!

Arc: Of course not, but you will get some of them. And then you can apply robust statistics, especially the RANSAC technique, to find the map E (or F). You see, your constraint is $x^\top E x' = 0$ for corresponding points x and x'. So, you want to fit a linear model to the data (correspondences). It's a problem in statistics.

Mel: What is RANSAC?

Arc: It stands for RANdom SAmple Consensus [REF]. It's a good way to fit

functions to data. Let me show you how it works for a simple case, when you fit a straight line to points on the plane; pick any two points at random. They define a line. Next, pick a threshold, and count all the points that lie within a distance threshold from the line. This is the support of the line. The points within the distance threshold are the inliers, the others are the outliers. Now, repeat this many times and choose as the solution the line with the strongest support. People have studied many aspects of this problem and we have a solid framework to work with. It is not a panacea, but it's a good technique.

Mel: I see. Your alternative would be to do some optimization like least squares, where you find the line such that some function is minimized (like the distance from the points to the line). And such procedures are sensitive to outliers.

Arc: Good. Of course, to be sure that you get the best solution you should try all possible combinations, which is not so good. But you can choose a large enough sample so that you can ensure with some probability p that one of the selected random samples is free from outliers. Suppose $p = 0.99$ and $\epsilon = 1 - \alpha$ the probability that a selected point is an outlier (and thus α the probability that it is an inlier). If m is the number of points you need to define the model, then you need at least N selections so that $(1 - \alpha^m)^N = 1 - p$ or $N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^m)}$. You can see from this formula that you need to try many things. When your proportion of outlier is 50% and $m = 8$ you need at least 1177 trials.

Mel: That can be a lot! I guess this number rises fast with m. That's why, I guess, you would want to find E using 7 points. You mentioned it but you never told me how to do it.

Arc: It's not so important. You can read about it in [REF]. It amounts to solving higher order algebraic equations. In any case, there are now good ways to pick all these parameters in the problem [REF]. With regard to your problem, you need 8 points. You get a linear system which gives you E. It makes sense to normalize the data first.[12] After you fit an E, you can then find the distance of the rest of the correspondences from this E, and then you are in business with RANSAC.

Mel: What is this distance? I guess some form of error that some correspondence produces for the E you fit.

Arc: Exactly. People have tried a few different alternatives. For example, it could be the value of $x^\top E x'$, for some correspondence (x, x'). Others have considered

[12] yiannis will add a note

the distance from the epipolar line. After you have an estimate for E, you know the epipolar geometry. So, given x, you know its epipolar line in the other image. So, the distance of x' from that epipolar line is a good measure. You should also do the symmetric case. There are a few different error measures [REFS] that have been developed over the years. They basically amount to appropriately scaling the distance from the epipolar line, to get better statistical performance. Other approaches develop an error criterion in 3D, as opposed to 2D, i.e., the image.

Mel: So, what is the result?

Arc: As I told you, if I get the correspondences by hand, I can do quite well. But, as you have already realized, things cannot be perfect. This procedure provides reasonable solutions. Sometimes very close to the actual solution, sometimes far away. When you are done, you can continue improving the solution. Bundle adjustment is one way. There are others as well. The books ... and ... contain an excellent survey of the various techniques published over the years.

Mel: So, where are we? Can we really do it automatically?

Arc: Well, I would say we are close, but not there yet. As you realize, this problem amounts to an optimization. From your data and the basic constraints from two or more views, you make a function whose minimum corresponds to the solution you are seeking, the 3D transformation. I would say, we don't fully understand this optimization. We make errors.

Mel: I see. Making small errors here and there creates difficulties when you put together 3D models from many views.

Arc: On top of this, correspondence is like a chicken-egg problem, as I told you. You need to know the correspondence to find the camera and scene geometry, but you also need to know something about the camera and scene geometry to even think about finding correspondence. This problem is not for me.

Mel: Maybe Socrates is wrong. It's possible that this process is compositional. You cannot solve any part of the problem before others. Somehow, you need to solve them all at the same time. Maybe I have no hope of finding 3D models.

Arc: Oh, don't despair. That simply means that you need to model your problem differently.

Mel: But do you think concentrating on the 2D and 3D transformation and the 3D

shape would be enough? Or would I need to involve recognition.

Arc: I doubt that. But then, what do I know? It just feels to me that the whole thing is simply geometric.

Bibliography

[1] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[2] G. Xu and Z. Zhang. *Epipolar Geometry in Stereo, Motion and Object Recognition: A Unified Approach*. Kluwer Academic Publishers, 1996.