

## CS221 Lecture notes #10

# Geometric vision

In these notes, we will examine the geometry of **image formation**. How does a 3-D scene in the world translate into the 2-D image that is picked up by the human retina or a camera? An understanding of the geometry of image formation will lead to some very useful algorithms.

## 1 Perspective camera model

Our model for image formation will be the **perspective camera**, where all of the points in the world are projected onto an imaginary 2-D plane. As shown in Figure 1, we have an **image plane**  $\pi$ , and a **camera center**, labeled  $O$ . The **optical axis** is the line which passes through the camera center  $O$  and is perpendicular to the image plane  $\pi$ . We call the distance from  $O$  to  $\pi$  the **focal length** and denote it as  $f$ . The point  $o$  where the optical axis intersects  $\pi$  is called the **principal point**, or **image center**. Typically, we choose a Cartesian  $(x, y, z)$  coordinate system such that the camera center  $O$  is at the origin  $(0, 0, 0)$ , and the  $z$ -axis is the optical axis.

The image is generated by projecting points in 3-space onto the image plane  $\pi$ . More specifically, consider the point  $P$  shown in Figure 1. To project  $P$  onto  $\pi$ , we take the line connecting  $O$  to  $P$ , and let the **projection**  $p$  of  $P$  be the point where that line intersects  $\pi$ . (In general, we will denote points in 3-space with capital letters, and projections onto the image plane with lowercase letters.) We can express projection algebraically as follows. Let

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

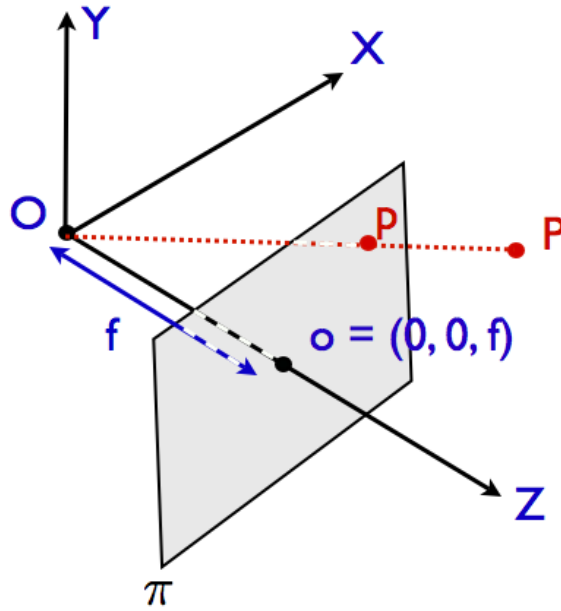


Figure 1: The perspective camera model.

Since we took the camera center  $O$  to be the origin  $(0, 0, 0)$ , all the points on the line connecting  $O$  and  $P$  will be scalar multiples of  $P$ . In other words, we have

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}.$$

To determine the value of  $\alpha$ , note that  $\alpha Z = z$ , and that  $z = f$ , because the image plane  $\pi$  is at  $Z$  coordinate  $f$ . This implies that  $\alpha = f/Z$ , and so,

$$x = f \frac{X}{Z}, \quad y = f \frac{Y}{Z}. \quad (1)$$

We will use the two equations (1) as our definition of projection for the rest of this lecture.

Note that projection is a **non-linear** function of  $P$  because of the factor  $\frac{1}{z}$ . (Recall that a function  $g$  is linear if for any vectors  $u$  and  $v$  and scalars  $\alpha$  and  $\beta$ ,  $g(\alpha u + \beta v) = \alpha g(u) + \beta g(v)$ .) Also, projection does not preserve distances between points or angles between lines. (The projections of two trees far off in the distance might be very close together in the image plane, and the projection of a chessboard might not have  $90^\circ$  angles in the image

plane.) However, as you will show in Problem Set 3, projection does preserve straight lines. Specifically, if there are three points  $P_1, P_2, P_3$  in 3-D that lie in a straight line, then the projection of these three points onto the image plane will also lie on a straight line (in the image plane).

## 2 Camera calibration

Now we see how to apply our understanding of how images are formed to problems in geometric vision. Ultimately, we will want to use the images from a camera to deduce information about the 3-D world. Before we can do so, we first need to know certain parameters of our perspective camera model. More specifically, we need to know:

- **Extrinsic parameters:** the position and orientation of the camera.
- **Intrinsic parameters:** the mapping between pixels in an image and points  $(x, y)$  in the image plane.

Fortunately there is off-the-shelf software<sup>1</sup> which can solve both of these problems quite well. Typically, the process goes as follows. You take a series of pictures of a chessboard and mark by hand the four extreme corners of the chess board in each image. Because a chess board is extremely regular, an algorithm can automatically detect all of the corners in the chess board. Once you know how these particular points in the world map to points in the image, an algorithm can deduce the location and orientation of the camera.

## 3 Stereopsis

We are often interested in recovering 3-D information about the world, but cameras only give us 2-D images. How do we extract 3-D information from 2-D images? One common technique is **stereopsis**, which relies on the fact that two cameras in slightly different positions will “see” slightly different images. Each point  $P$  in the world will appear at a slightly different location in the image planes of the two cameras, and we can use the difference in the two projections, called the **disparity**, to estimate the distance to  $P$ .

More precisely, consider the 2-D setup in Figure 2(a). We have two projective cameras, each one like the one described earlier in this lecture. The origins of the two cameras are at locations  $O$  and  $O'$ . Suppose first that

---

<sup>1</sup>Such as [http://www.vision.caltech.edu/bouguetj/calib\\_doc/](http://www.vision.caltech.edu/bouguetj/calib_doc/).

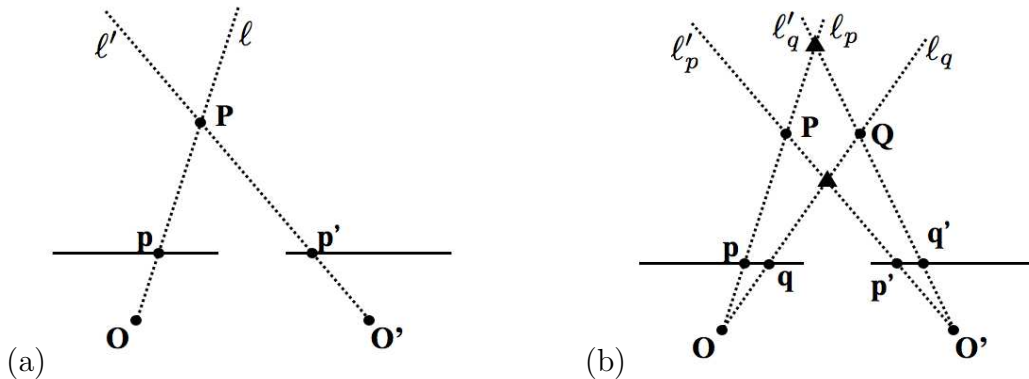


Figure 2: (a) If two cameras observe a point  $P$  in the world, we can resolve its location unambiguously. (b) If there are two points,  $P$  and  $Q$ , the correspondence problem creates ambiguity. If we choose the wrong correspondences, we might mistakenly believe we’re seeing the two points marked by triangles.

we are trying to estimate the location of one particular point  $P$  in the 2-D world, and its projections onto the two image planes are denoted  $p$  and  $p'$ . If the only information we are given is  $p$ , the projection onto the first image plane, we know that  $P$  must lie on the line  $\ell$ , which gives the set of points in the world which project to  $p$ . However, we also know  $p'$ , the projection onto the second plane. This means  $P$  also has to lie on the second line,  $\ell'$ . Hence,  $P$  must be located at the intersection of the lines  $\ell$  and  $\ell'$ .

Of course, the real world contains more than just one point. To perform the sort of reasoning described above, we would first have to figure out which points in one image correspond to which points in the other. This is known as the **correspondence problem**. Let’s add another point  $Q$  to the world, to get the setup in Figure 2(b). If we choose the correct correspondences, we can correctly identify  $P$  and  $Q$  in the 2-D plane. But if we choose wrong, we intersect the wrong pairs of lines, and therefore believe we see the two points marked by triangles.

How can we actually find correspondences? One simple method is to search for regions in the two images that look “similar” to each other. More specifically, we search for a match where the pixel intensities are close, as measured by the sum-of-squares of their differences, as follows:

For each pixel  $(i, j)$  in the left image:

For each possible disparity  $[d_1, d_2]$ , compute the cost

$$c(d_1, d_2) = \sum_{k=-w}^w \sum_{\ell=-w}^w [I_{\text{left}}(i+k, j+\ell) - I_{\text{right}}(i+k-d_1, j+\ell-d_2)]^2.$$

Choose the minimum cost disparity for  $(i, j)$ :

$$\arg \min_{d_1, d_2} c(d_1, d_2).$$

But what is the set of disparities  $[d_1, d_2]$  which we have to consider? It might seem like we have to consider every single point in the right-hand image as a possible match for  $(i, j)$ . Fortunately, we can usually narrow it down to a small number of possible disparities. Assume we know the geometries of the two cameras are as shown in Figure 3 and we are trying to determine which point in the right-hand image plane  $\pi'$  corresponds to the point  $p$  in the left-hand image plane. Knowing only the projection point  $p$ , we can narrow down the location of the actual point  $P$  to a single line in 3-space, namely the one which passes through  $O$  and  $p$ . The set of points we must consider as possible matches in  $\pi'$  are the projections of all of the points which lie on this line, as shown in the figure. Because projection preserves straight lines (as you will prove in PS3), we know these projections must all lie on a single line. This line on the image plane  $\pi'$  is called the **epipolar line**. By limiting ourselves to points which lie on the epipolar line, we vastly reduce the number of disparities which must be searched.

In theory, stereopsis can precisely pin down the location of any point in 3-space given only the projections onto two image planes. But how accurate is it in practice? It turns out that stereopsis works quite well for nearby objects, but is ineffective for far away objects. When a point is far away, changing the location of the point by the same amount in 3-space causes a much smaller change in the disparity between the two cameras. This means it's harder to pin down the precise location of points which are far away. You will have a chance to quantify this effect precisely in Problem Set 3.

It is also worth mentioning a variant on stereopsis, namely **structure from motion**. Suppose the camera is moving through a static 3-D scene, and it takes two snapshots at slightly different times. (In practice, we would usually take more than two images.) The result is as if the two snapshots came from two different cameras at slightly different locations, and we can compute disparity using techniques similar to those for stereopsis. Of course, using structure from motion the way we describe here requires assuming a static scene, which is sometimes not true. Furthermore, we need to estimate

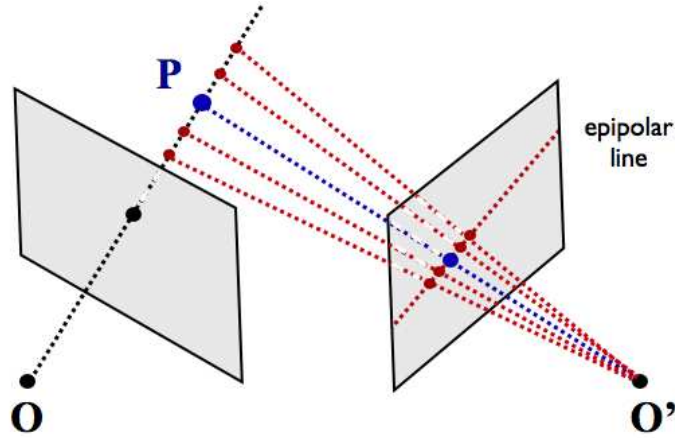


Figure 3: The set of all points in 3-space consistent with the left camera's observation forms a straight line (the epipolar line) in the right camera's image plane.

the relative positions of the camera at the two points in time when the images were taken. We won't discuss this precise method for computing structure from motion (i.e. computing the disparity between points). Instead, we will discuss another algorithm called optical flow.

## 4 Optical flow

So far, we have assumed a static environment and dealt with the problem of determining the locations of objects within that environment. Now we'll consider the problem of inferring motion. Suppose we are given a series of frames of video. What could cause the image to change from one frame to the next? There are two possibilities: motion in the image can be due either to the motion of the objects within the scene, or to the motion of the camera itself. In general, it will be some combination. We're not going to talk about how to distinguish these two cases; rather we will discuss the simpler problem of estimating motion in the image itself.

In particular, we discuss how to compute the **optical flow**, the directions of motion at particular locations in an image. For instance, Figure 4 shows some cartoon examples of the optical flow we might expect for some particular kinds of motion.

To motivate our derivation of optical flow, let  $(x(t), y(t))$  denote the po-

sition of a particular object at time  $t$ , and let  $E(x, y, t)$  denote the brightness (pixel intensity) of the image point  $(x, y)$  at time  $t$ . We suppose that the object is translating in some direction in the image. In computing optical flow, we make the simplifying assumption that the object appears equally bright at all times. Thus, if we follow the object's location  $(x(t), y(t))$  at each time  $t$ , that intensity will be constant. This can be expressed with the following differential equation, called the **brightness constancy equation**:

$$\frac{\partial}{\partial t} E(x(t), y(t), t) = 0.$$

Applying the chain rule for partial derivatives, we get:

$$\frac{\partial E}{\partial x} \frac{\partial x}{\partial t} + \frac{\partial E}{\partial y} \frac{\partial y}{\partial t} + \frac{\partial E}{\partial t} = 0. \quad (2)$$

The partial derivatives  $\frac{\partial E}{\partial x}$  and  $\frac{\partial E}{\partial y}$  encode how the image intensity values vary with respect to the location in the image, and the partial derivative  $\frac{\partial E}{\partial t}$  encodes how the intensity at a particular location varies with time. Note that we know the quantities

$$\frac{\partial E}{\partial x}, \quad \frac{\partial E}{\partial y}, \quad \frac{\partial E}{\partial t}$$

from the video sequence itself, and we are interested in estimating the terms corresponding to the object's motion, which we will denote by

$$v_x = \frac{\partial x}{\partial t}, \quad v_y = \frac{\partial y}{\partial t}.$$

Can we use this equation to solve for  $v_x$  and  $v_y$ ? No, because in general there will be more than one solution to this equation in two unknowns. So, we will instead take a **window** of size  $5 \times 5$  (say), and find a single optical flow vector  $(v_x, v_y)$  which will describe the motion in the entire window. Of course, we probably can't find a vector which satisfies (2) exactly for all 25 points in the window. Rather, we will search for a flow vector  $(v_x, v_y)$  which minimizes the following:

$$\sum_{k=-2}^2 \sum_{\ell=-2}^2 \left[ \frac{\partial E(x+k, y+\ell, t)}{\partial x} v_x + \frac{\partial E(x+k, y+\ell, t)}{\partial y} v_y + \frac{\partial E(x+k, y+\ell, t)}{\partial t} \right]^2.$$

Remember that the partial derivative terms are simply constants we can pull from the video sequence itself. Therefore, we are simply trying to solve a

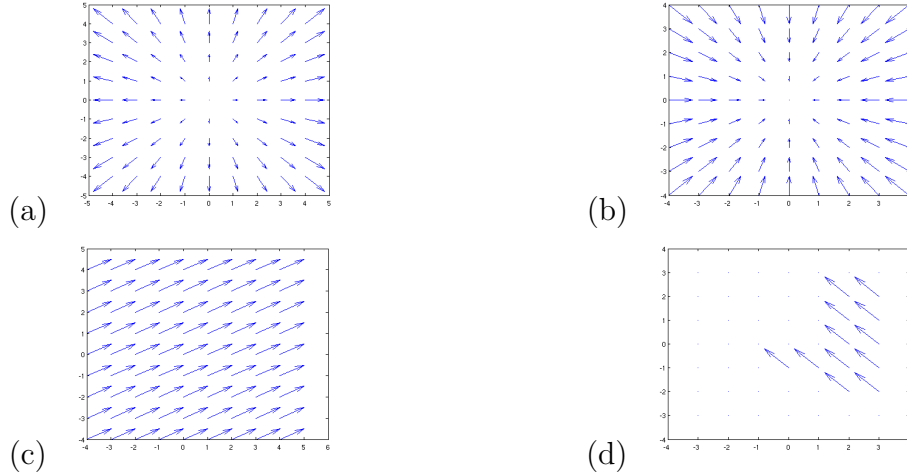


Figure 4: Some cartoon examples of optical flow. (a) The camera is moving forwards in a static environment. (b) The camera is moving backwards in a static environment. (c) The camera is moving down and to the left in a static environment. (d) The camera is fixed, and an object is moving up and to the left.

least-squares problem (just like linear regression), where the variables are the two real numbers  $v_x$  and  $v_y$ .

One way to solve this is with gradient descent, but it turns out there's a closed-form solution we can use instead. The derivation is beyond the scope of this class, but for the sake of completeness, we will present the closed-form solution here. Define the matrix

$$A = \left[ \begin{array}{cc} \frac{\partial E(x-2,y-2,t)}{\partial x} & \frac{\partial E(x-2,y-2,t)}{\partial y} \\ \frac{\partial E(x-2,y-1,t)}{\partial x} & \frac{\partial E(x-2,y-1,t)}{\partial y} \\ \vdots & \vdots \\ \frac{\partial E(x+2,y+2,t)}{\partial x} & \frac{\partial E(x+2,y+2,t)}{\partial y} \end{array} \right] \left. \vphantom{\begin{array}{c} \\ \\ \\ \end{array}} \right\} 25 \text{ rows}$$

and the vector

$$b = - \left[ \begin{array}{c} \frac{\partial E(x-2,y-2,t)}{\partial t} \\ \frac{\partial E(x-2,y-1,t)}{\partial t} \\ \vdots \\ \frac{\partial E(x+2,y+2,t)}{\partial t} \end{array} \right] \left. \vphantom{\begin{array}{c} \\ \\ \\ \end{array}} \right\} 25 \text{ rows}$$

(Note the minus sign for  $b$ .)  $A$  and  $b$  contain 25 rows, one for each position

in the  $5 \times 5$  window. The least-squares solution, then, will be given by

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} \approx (A^T A)^{-1} A^T b.$$