Image rectification

If we want to estimate the positions of the 3D points in the scene, then it is best if we use calibrated cameras. Suppose we know the internals and externals of the two cameras, in particular, the $\mathbf{K}$ matrices and the rotation $\mathbf{R}$ and translation $\mathbf{T}$ that relate the two cameras. Rather than working with images in pixel coordinates, we will assume today that we have inversed mapped with $\mathbf{K}^{-1}$ from pixel coordinates to image projection plane coordinates, and we will ignore the pixels. We will also assume the the projection planes for the two cameras are at the same distance f.

Our goal now is to make the correspondence problem easier to solve by using homographies (rotations, specifically) to re-project the images onto a projection plane that is parallel to the line joining the two cameras, such that the epipolar lines in the first and second image coincide. We do so by mapping the epipoles of each image to a point at infinity in the direction of the x axis, i.e. $(1, 0, 0)$. When the epipole is at $(1, 0, 0)$, all epipolar lines are horizontal i.e. parallel lines meet at the point at infinity. Remapping an image an image so that all epipolar lines are parallel is called *rectification*.

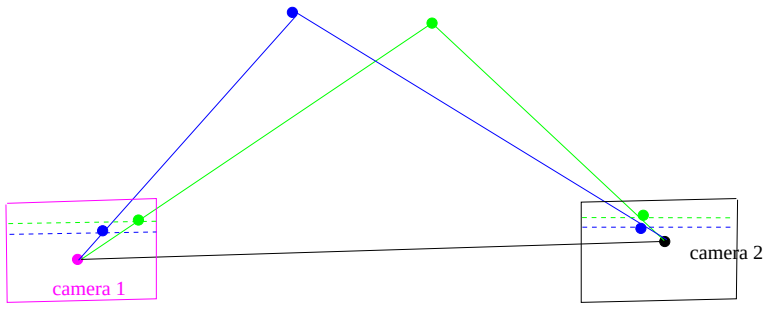
To do this remapping, we define a rotation matrix

$$\mathbf{R}_{rect} = \begin{bmatrix} unit(\mathbf{T}) \\ unit(\mathbf{T} \times \hat{\mathbf{z}}) \\ unit(\mathbf{T} \times (\mathbf{T} \times \hat{\mathbf{z}})) \end{bmatrix}$$

where, by construction, the rows are orthonormal and

$$\begin{bmatrix} |\mathbf{T}| \\ 0 \\ 0 \end{bmatrix} = \mathbf{R}_{rect}\mathbf{T}$$

and so the epipole $\mathbf{T}$ of the first camera is mapped to direction $(1, 0, 0)$ as desired. Moreover, since all epipolar lines intersect at the epipole, the epipolar lines are mapped to lines that intersect at $(1, 0, 0)$. Hence the remapped epipolar lines are parallel to the x axis. Thus, if we apply the rotation $\mathbf{R}_{rect}$ to any point $\mathbf{x}_1$ and then divide by the third coordinate and multiply by f, we get the coordinates of the point in the rectified camera 1. In the rectified camera 1, the epipolar lines are parallel to the x-axis.



We would like to do the same for camera 2. Assuming we know the matrix $\mathbf{R}_{1\leftarrow 2}$, we can rotate each $\mathbf{x}_2$ to undo the rotational difference between the two cameras, i.e. to make the two cameras' axes parallel to each other. This brings the second camera's epipole $\mathbf{e}_2$ to a vector which is parallel

to $\mathbf{T}$. (In fact, the first camera is in direction **-T**, which is parallel to $\mathbf{T}$.) We then rotate using $\mathbf{R}_{rect}$ so that the camera 2 rectified coordinate axes are parallel to the camera 1 rectified coordinate axes. Taking the two maps together, we are mapping the camera 2 epipole to infinity:

$$\mathbf{R}_{rect}\mathbf{R}_{1\leftarrow 2}(\mathbf{e}_2) = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

So the rectification for camera 2 image is

$$\begin{bmatrix} w'x_2' \\ w'y_2' \\ w' \end{bmatrix} = \mathbf{R}_{rect}\mathbf{R}_{1\leftarrow 2} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

We have now reduced the stereo problem to a simple configuration in which two cameras have the same orientation (i.e. parallel coordinate axes) and are separated by a baseline translation $(|\mathbf{T}|, 0, 0)$. A given 3D point in the scene can be represented (X_1, Y_1, Z_1) in camera 1's rectified coordinates and $(X_2, Y_2, Z_2) = (X_1 - |\mathbf{T}|, Y_1, Z_1)$ in camera 2's rectified coordinates. This point will project to image positions in the two cameras:

$$(x_1, y_1) = f(\frac{X_1}{Z_1}, \frac{Y_1}{Z_1})$$

$$(x_2, y_2) = f(\frac{X_1 - |\mathbf{T}|}{Z_1}, \frac{Y_1}{Z_1}).$$

We define the binocular disparity (for the rectified images) to be

$$d = x_1 - x_2 = f\frac{|\mathbf{T}|}{Z}.$$

If we can now find corresponding points in the two images, then we can trivially estimate the binocular disparity of these points and estimate the distance Z to the corresponding 3D scene point.

Binocular correspondence problem

Finding corresponding points automatically and accurately is an important computer vision problem, and it has a long history. We have seen a version of this problem when we discussed image registration. Indeed the Lucas-Kanade method works relatively well for solving the stereo correspondence problem. Today I will discuss another approach, which has eventually led to more accurate results than can be achieved with Lucas-Kanade type methods. For a recent survey of stereo methods, see D. Scharstein and R. Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1):7-42, May 2002." and the Middlebury database of stereo images used to compare results `http://vision.middlebury.edu/stereo/`.

Random dot stereograms

The roots of computer vision approaches to the stereo correspondence problem are over fifty years ago old, and were originally developed to help us understand how stereovision works in human vision. Up until the early 1960's, it was believed that the correspondence problem was solved by the brain by finding feature points (such as corners) in the two images, matching them, and "filling in" between these matched features. Many experiments were carried out by psychologists using stereo image pairs which tried to identify what sorts of features were used.

The main breakthrough in this line of research was the invention of the *random dot stereogram* (RDS) by Bela Julesz.[1]. The RDS is a pair of images (a "stereo pair"), each of which is a random collection of white and dark pixels. As such, it does not contain any familiar features. The key to the RDS is the relationship between the random pixels in the two images. The random pixels in the left image are related to the random pixels in the right image by a shift.

Specifically, the left image is created by setting each pixel randomly to either black or white. Then, a copy of this image is made. Call this copy the right image. The right image is then altered by taking a patch (say a square[2] and shifting that patch horizontally by d pixels to the left, writing over any pixels values. The pixels vacated by shifting the patch are filled in with random values.

The above procedure yields four types of regions in the two images.

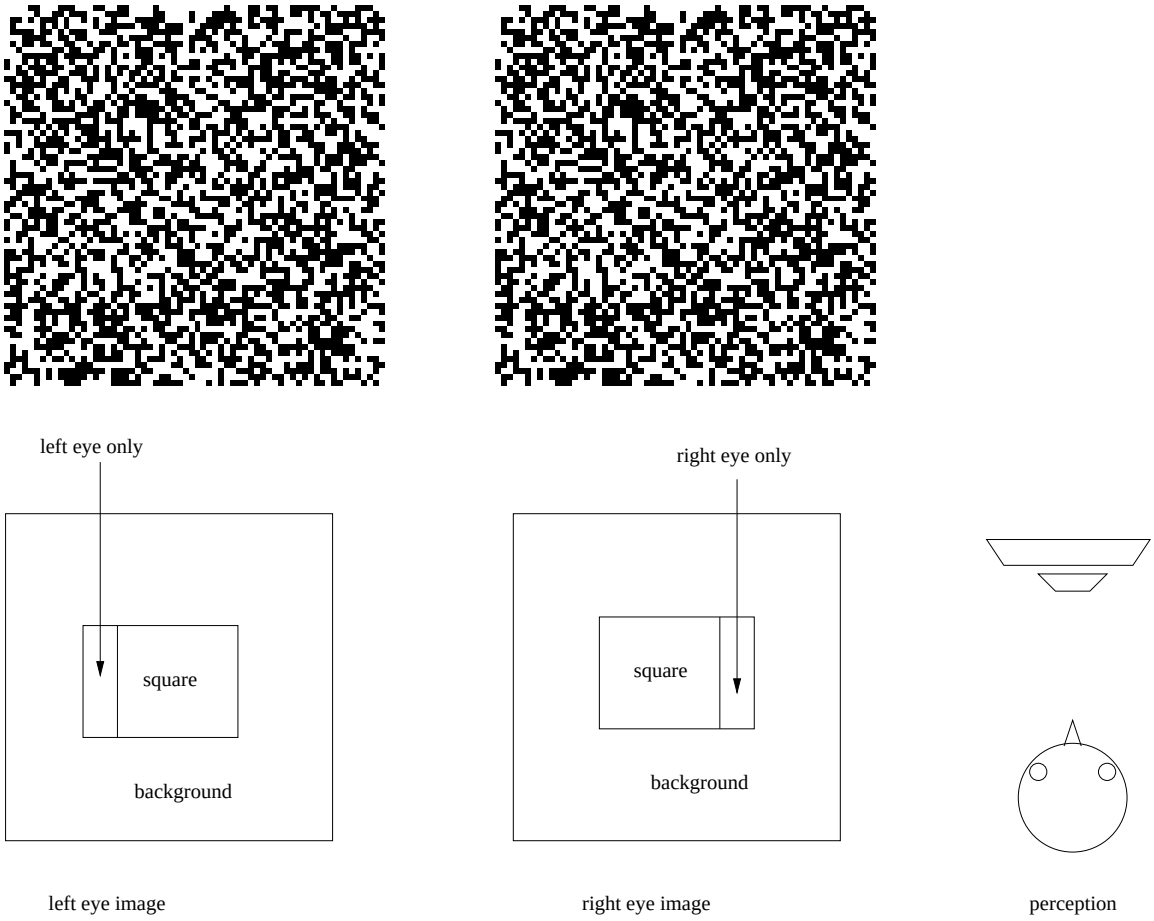
- the shifted pixels (visible in both left and right images)
- the pixels that were erased from the right image, because of the shift and write; these pixels are visible in the left image only;
- the pixels in the right image that were vacated by the shift; these are visible in the right image only
- any other pixels in the two images (both left and right)

To view a stereogram such as above, your left eye should look at the left image and your right eye should look at the right image. (This is difficult to do without training.) If you do it correctly, then you will see a square floating in front of a background. [ASIDE: a subsequent invention is the *autostereogram* which works using a single image only. See for example `http://en.wikipedia.org/wiki/Autosterogram`.]

Julesz argued that since each image of a random dot stereogram is just random pixels, there are no common "features" in the sense of edges or corners or other commonly occuring familiar patterns, so it seems implausible that the human visual system is relying on such features. Rather, he argued, a more basic type of matching must be going on.

[1] B. Julesz, "Binocular depth perception without familiarity cues", Science, 145:356-362 (1964) [ASIDE: Julesz was interested in the military application of detecting enemy tanks and other non-natural features of the landscape from aerial images. A common way to hide people, tanks, etc, is to make them have the same appearance as the background (cover them with green and brown uniforms, textures). From his experiences in the second world war, Julesz knew that humans could "break the camouflage" using stereovision, provided that there was a great enough depth range to give stereo information.]

[2] It doesn't have to be a square – it can be any simple shape or collection of shapes.

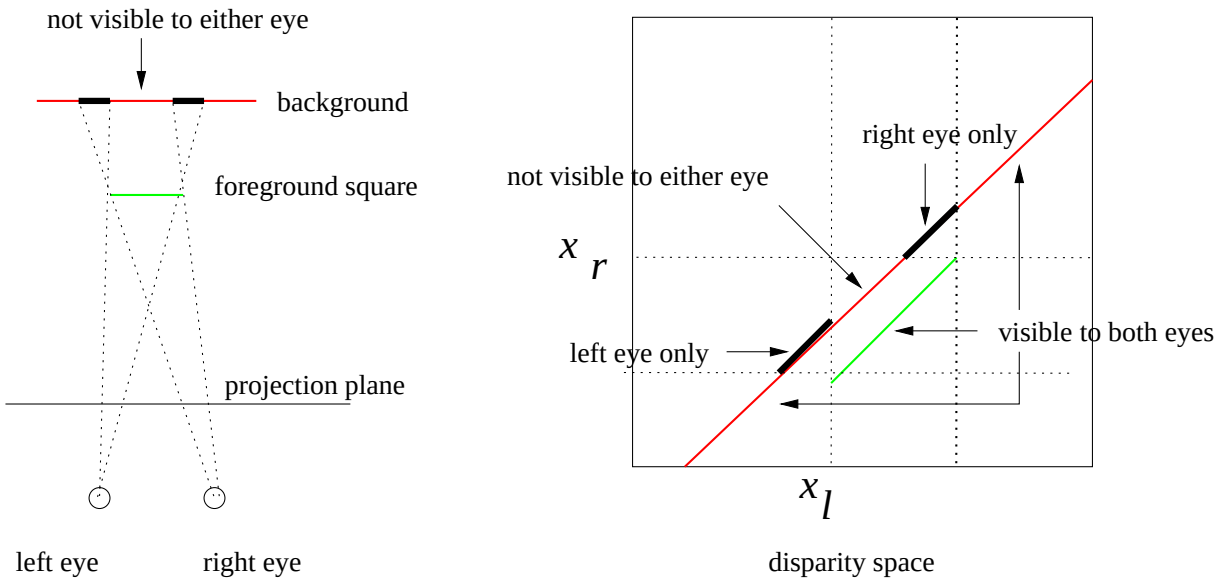


Disparity space

Let's relate the above example to a 3D scene geometry that could give rise to it. This will allow us to represent the problem in a different way (namely, geometrically rather than algebraically). Suppose the optical axes of two cameras are parallel, i.e. rectified as discussed at the beginning of this lecture. The right camera is displaced from the left in the X direction only and the two cameras have the same projection plane $Z = f$. The 3D scene is a small foreground square in front of a background. This scene yields a disparity d between the position of points on the square in the left and right images. Consider a single horizontal line $y = y_0$ in the image projection plane such that this line cuts across the displaced square. We wish to understand the disparity along this line.

The figure below represents this line in the two images using a coordinate system (x_l, x_r). For each 3D scene point that projects to this line, there is a unique x_l and x_r value. Moreover, each depth value Z corresponds to a unique disparity value, since $d = x_l - x_r = T_x/Z$. Note that $x_l > x_r$ for all points $Z > 0$. Thus, disparity d is positive.

The set of rays that arrive at the left eye are vertical lines in the figure on the right, and the set of rays that arrive at the right eye are horizontal lines in the figure on the right. Similarly, each horizontal line in the figure on the left represents a line of constant depth (constant disparity), and this corresponds to a diagonal line in the figure on the right, namely a line of constant disparity. *For any vertical or horizontal line, the scene point that is visible is the one with largest disparity,*



i.e. smallest Z value. Make sure you understand where each point on the foreground square and each point on the background square appear in the left and right figures. (I talked through it in class.) In particular, note that some points on the background surface are visible to one eye only; others are visible to both eyes; still others are visible to neither eye. Points that are visible to one eye only are called *monocular* points. These points occur for random dot stereograms too – namely they are the points erased from or added to the right image, when the square is shifted.

Solving the stereo correspondence problem

Let's review a simple algorithm for solving the stereo correspondence problem, in particular, for random dot stereograms.[3] We consider only a single epipolar line i.e. horizontal line. The algorithm is derived from two principles, also called "constraints":

- (*uniqueness:*) A given pixel in each image is typically the projection of *one* 3D point in the world, namely a point on an opaque surface. It follows that for any position the left or right image, we would like to find one disparity value for that point;
- (*continuity:*) Surfaces in the world are continuous, and since an image is the projection of a *set* of surfaces in the world, possibly at different distances from the camera, disparity is *piecewise continuous* across the left and right image.

Note that the two constraints don't always hold. The first constraint fails when we have a transparent surface such as a window, since we have objects visible at different depths and at the same image position. It also fails when we have specular reflections, which appear at different surface positions in the left and right image. The second constraint fails when the scene contains many small surfaces at different depths, for example, branches of a dense bush. In this case, there will be so many depth discontinuities that it doesn't make much sense to enforce piecewise continuity.

[3] D. Marr and T. Poggio. "Cooperative Computation of Stereo Disparity". Science, 194:282-287. (1976).

Marr and Poggio proposed an iterative algorithm to solve the correspondence problem. Assuming N pixels in each image, they discretize the space (x_l, x_r) using an $N \times N$ square matrix. Each row/column in the matrix corresponds to a single position in the left/right image respectively. The goal is to compute a binary labeling of the nodes in this $N \times N$ matrix. An `ON`-label means that the node corresponds to a visible surface at that (x_l, x_r) value, a `OFF`-label otherwise.

- The uniqueness condition implies that at most one node can have the label `ON` in any row and in any column.
- The continuity condition implies that the disparity d should be a piecewise continuous function of x_l and as a function of x_r. Any fixed value of disparity d defines a line $x_l - x_r = d$ in (x_l, x_r) space. This is a diagonal line with slope 1. Thus, the continuity constraint implies that nodes with label `ON` should cluster along such diagonal lines.

Each node in the matrix has a label, $\mathcal{L}(x_l, x_r)$ which is `ON` or `OFF`. We would like to compute a labelling such that a label is `ON` if there is a visible surface at this position and OFF otherwise. We want an algorithm to compute this labelling, given binary images $I_l(x)$ and $I_r(x)$.

The idea of the Marr and Poggio algorithm is for the nodes to locally communicate with each other. The uniqueness constraint means that nodes along the same horizontal line or vertical line *inhibit* each other (as defined below). The continuity constraint means nodes along the same diagonal line support each other. The algorithm solves the following iterative equations, where `ON` is the value 1 and `OFF` is the value 0:

$$\mathcal{L}^{n+1}(x_l, x_r) = \tau\Big(\mathcal{L}^0(x_l, x_r) + \epsilon_1 \sum_{(x_l', x_r') \text{ in excite ngd of } (x_l, x_r)} \mathcal{L}^n(x_l, x_r) - \epsilon_2 \sum_{(x_l', x_r') \text{ in inhib ngd of } (x_l, x_r)} \mathcal{L}^n(x_l, x_r)\Big)$$

Notice that the initial labelling $\mathcal{L}^0(x_l, x_r)$ is included in the update equations.

The function τ is used to keep the labels binary, and defined:

$$\tau(t) = \begin{cases} \texttt{ON}, & \text{if } t > \tau_0 \\ \texttt{OFF}, & \text{otherwise} \end{cases}$$

The constants $\tau_0, \epsilon_1, \epsilon_2$ are chosen by the user.

One way to define the *initial labelling of the nodes* is:

$$\mathcal{L}^0(x_l, x_r) = \begin{cases} \texttt{ON}, & \text{if } I_l(x_l) = I_r(x_r), \\ \texttt{OFF}, & \text{otherwise} \end{cases}$$

In the 35 years since Marr and Poggio's work, there have been many algorithms proposed for solving this problem – and the more general problem where the images have intensities in 0,..,255 (rather than binary). In the next (and final) lecture, I will sketch out some of the key ideas.