Binocular correspondence problem

Last class we discussed how to remap the pixels of two images so that corresponding points are in the same row. This is done by computing the fundamental matrix, defining the epipolar lines from the null vectors of $\mathbf{F}$, and then mapping the epipolar lines to a new image so corresponding epipolar lines in the two images are in the same rows.

The problem now is to find corresponding points for all points (or as many as possible) and to do this matching automatically and accurately. This is a classic computer vision problem. We have seen a version of this problem when we discussed image registration, and indeed the Lucas-Kanade method does work well. Later today I will sketch out a more recent approach, which gives more accurate results. To set up this newer approach, though, it is helpful to review the basics, ... and a bit of history.

Random dot stereograms

The first computer vision approaches to the stereo correspondence problem (in the 1960s) were developed to help us understand how stereovision works in human vision. These early algorithms were tested on a special set of images, called *random dot stereogram* (RDS) which were invented by Bela Julesz.[1]. An RDS is a pair of images, each of which is a collection of black and white pixels. 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.

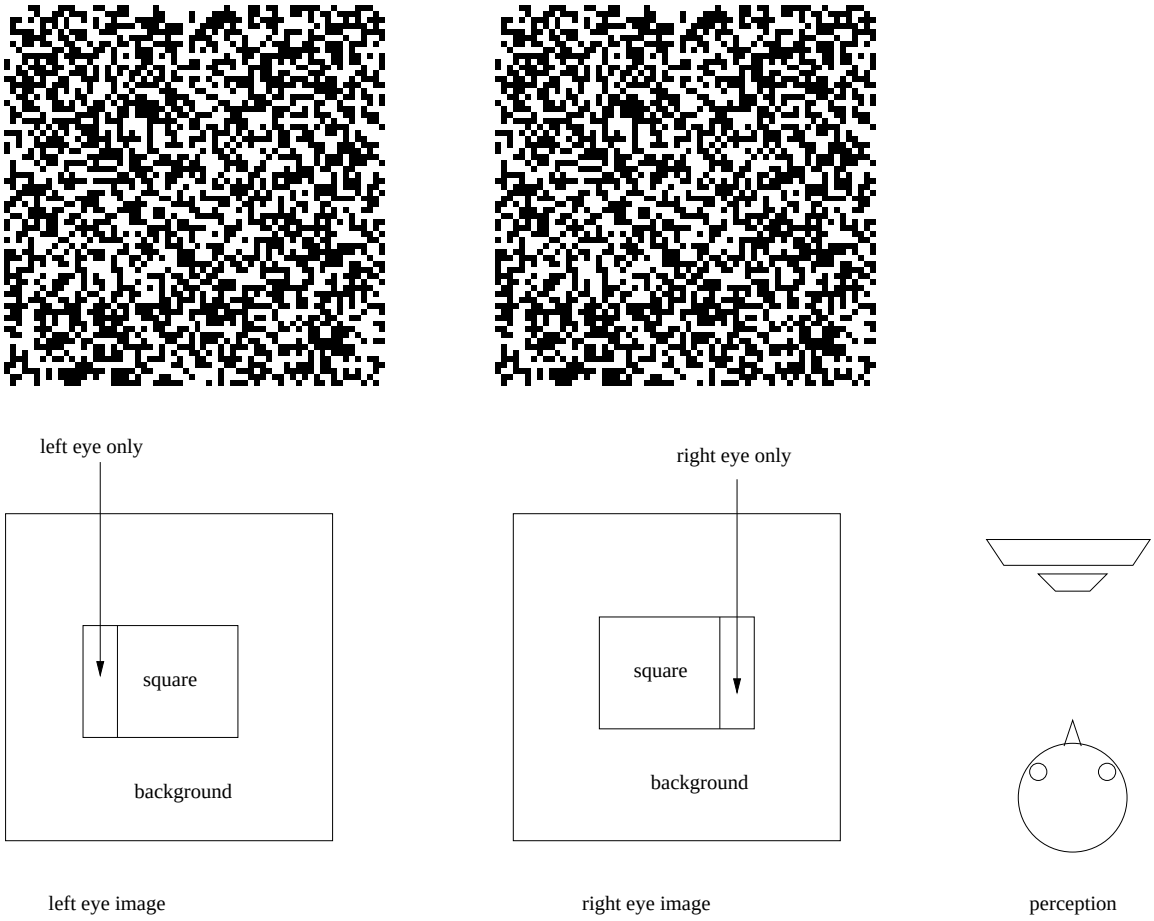
The above procedure yields four types of regions in the two images.

- 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 both two images (background, not shifted)

To view a stereogram such as below, 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, but if you do it correctly then you will see a square floating in front of a background.

[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

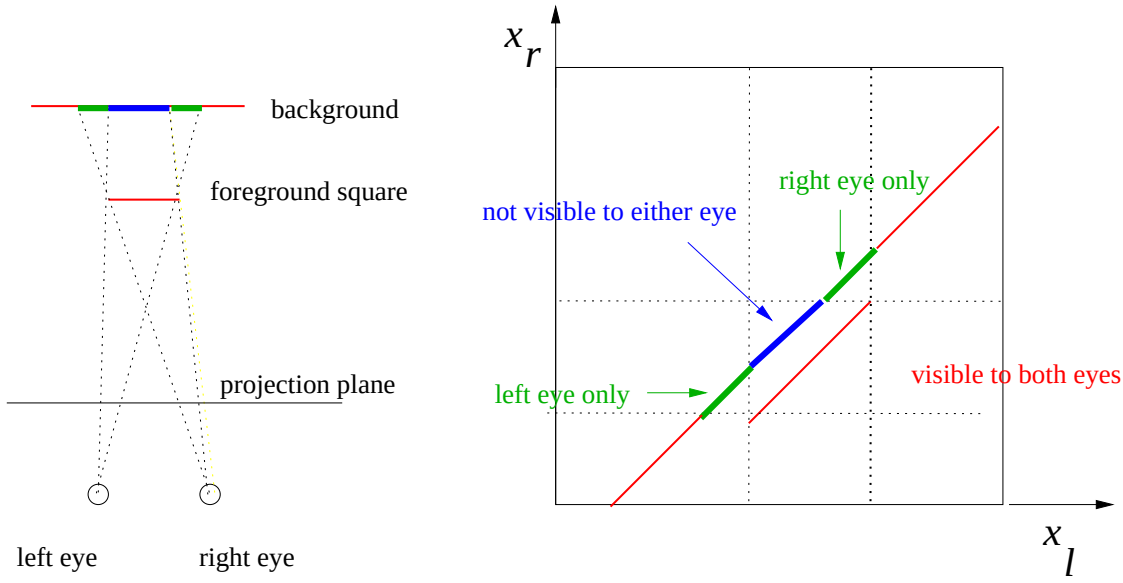
Let's relate the above example to a 3D scene geometry that could give rise to it. Assume the cameras are pointing in the same direction (Z), that the right camera is displaced relative to the left by a distance T, and that the projection plane is at $Z = f$. Corresponding points in the two images must therefore have the same y value and so, for each y, we only need to find corresponding x_l and x_r values.

The construction of the stereo pair as described above is consistent with a 3D scene, that 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

$$d = x_l - x_r = \frac{T}{Z}.$$

Note that $x_l > x_r$ for all points $Z > 0$. Thus, disparity d is positive.



In the figure below on the right, the set of rays that arrive at the left eye are represented by vertical lines and the set of rays that arrive at the right eye are horizontal lines. 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.

Because objects are opaque, *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. Recall what the monocular points were for the random dot stereograms, namely the points erased from or added to the right image when the square is shifted.

Stereo correspondence as a graph labelling problem

Many algorithms for solving the binocular correspondence problem use the (x_l, x_r) space. The main idea is to consider a 2D matrix in which each element corresponds to a pair of positions (x_l, x_r) in the left and right image, along the chosen epipolar line. Elements that satisfy $x_l - x_r = d$ for fixed d correspond to point of constant (inverse) depth.

The basic idea is to set up a graph as shown below. The vertices are the points (x_l, x_r) which are pixel positions on an epipolar line. There are vertical, horizontal, and diagonal edges. The vertical edges connect vertices of constant x_l. These correspond to line segments of rays through the pixels in the left eye. The horizontal edges are segments of rays through pixels in the right eye. The diagonal edges have constant disparity d. They correspond to small constant (inverse) depth surfaces.

Finding a correspondance between the left and right images can be defined as a labelling of the vertices in the graph. Each vertex can be either empty, a surface point seen by both eyes, a surface

point seen by one of the eyes, or a point behind a surface and hence not visible.

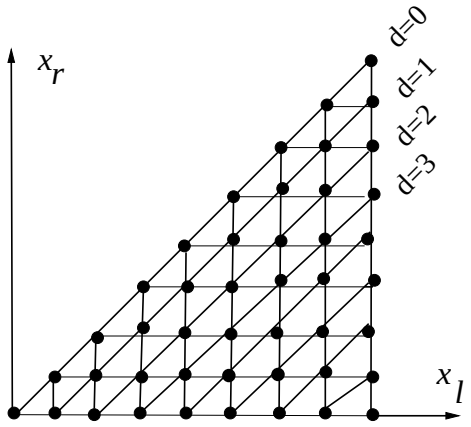
What makes a good solution i.e. a good labelling? Keep in mind that the images have intensities $I_l(x_l)$ and $I_r(x_r)$ – here we are ignoring the y coordinate – and so for any x_l, we want to choose a disparity d, i.e. $x_l = x_r + d$ such that

$$I_l(x_r + d) \approx I_r(x_r).$$

A similar constraint was used in image registration, of course.

What else makes a good solution? Since the depths Z of visible surfaces in the world are typically piecewise continuous, we don't prefer the disparity value of any x_l to be the same as that of its neighbors $x_l \pm 1$. (similarly for x_r.) This means that if a particular (x_l, x_r) is labelled as visible to both eyes, then its right neighbors $(x_l + 1, x_r + 1)$ and its left neighbor $(x_l - 1, x_r - 1)$ are likely to be visible to both eyes. Thus we should prefer the vertices of diagonal edges to have similar labels.

Finally, horizontal and vertical edges should not connect points (x_l, x_r) that are labelled as visible to both eyes, since this means that two points are visible along a line of sight which tends not to happen in real situations.



Stereo: the graph cut method

There have been hundreds of algorithms proposed for labelled disparity space graphs such as defined above. Here I sketch out one simple method that is relatively easy to explain and works rather well (and typically better than Lucas-Kanade, for example).

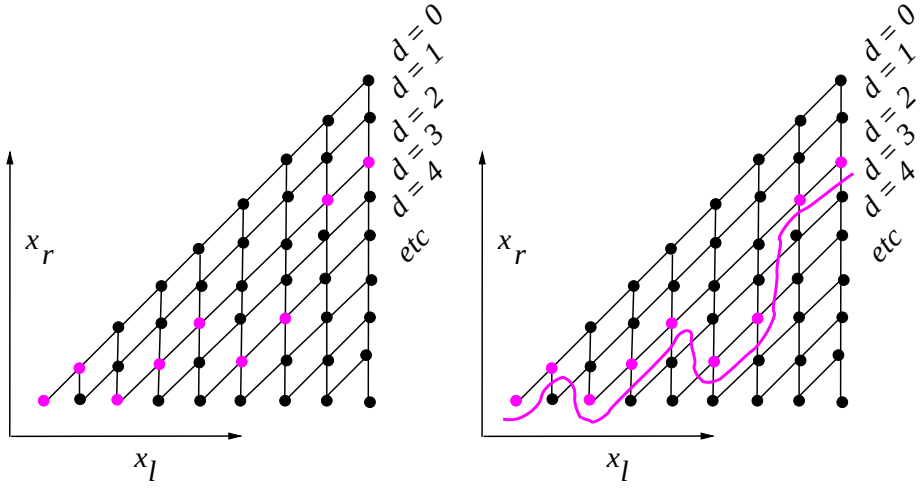
Define an undirected graph $G = (V, E)$ whose vertices V consist of pairs $\{(x_l, x_r) : \ x_l - x_r \geq 0\}$ as above, and whose edges E are a set of "neighbors" in V:

- diagonal edges: $\{ \ (x_l, x_r), \ (x_l + 1, x_r + 1)\}$
- vertical edges: $\{ \ (x_l, x_r), \ (x_l, x_r + 1)\}$

Notice that we do not have horizontal edges anymore.

We now seek a unique disparity for each x_l. That is, for each position x_l in the left image, we seek a corresponding position x_r in the right image (and hence disparity $d = x_l - x_r$). Such vertices are shown in pink in the figure below on the left.

There may be (monocular) points in the left image for which there is no corresponding visible point in the right image (and vice-versa). In the formulation I am presenting now, for these points in the left image, we would still choose a disparity d, and so the corresponding position in the right image would be $x_r = x_l - d$.



The figure above on the right shows a curve that cuts through the (x_l, x_r) space. This curve is a function $d(x_l)$, i.e. for each x_l there is a disparity d and hence a position x_r. This curve does not pass through any of the vertices V. Rather it cuts through the set of edges in E. In particular, the curve *partitions* the vertices V into two sets. These two sets correspond to the points in front of the surface (from the viewpoint of the left camera) and the points that are on the surface or behind the surface(s) as seen from the left camera's viewpoint. This partition is called a *graph cut*.

Which edges should we remove to cut the graph? The approach one takes is to define a cost on each edge, and to find a graph cut whose cut edges have minimum cost. How should we define these costs?

Diagonal edges (smoothness)

Since surfaces in the world are generally smooth (or continuous, at least), we would like the disparity d to vary as little as possible from one position x_l to the next x_l+1. This suggests that *we associate a cost for every diagonal edge that we cut*. By inspection, each diagonal edge that we cut is associated with a unit change in disparity from one position x_l to its neighbor $x_l + 1$. For example, if the disparity d differs by 2 from x_l to $x_l + 1$ then we need to cut 2 diagonal edges. This case occurs twice in the above example.

To minimize the number of disparity changes across the image, we can associate a constant cost K for each diagonal edge that we cut.

$$edgeCost(\ (x_l, x_r),\ (x_l + 1, x_r + 1)\ ) = K.$$

These is called a *smoothness* cost.

Vertical edges (data cost)

To cut the graph, we also need to cut vertical edges, namely we cut a vertical edge from a pink point (the visible surface point for a given x_l) to its neighbor immediately below it. This is an edge between (x_l, x_r) and $(x_l, x_r - 1)$. What should be the cost on such an edge?

Choosing the disparity d for a position x_l amounts to finding a position in the right image x_r which is a "good match". By this, we mean that the intensities are similar: $I_l(x_l) \approx I_r(x_r)$. This suggests that we assign a cost of cutting an edge which depends on the intensity difference. If we want to minimize the total cost of cutting edges, then we want to define small costs with good intensity matches and large costs with poor matches. One way to define the cost of a vertical edge, which is consistent with the above idea, is:

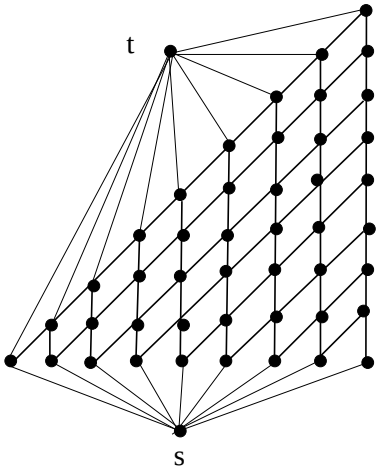
$$edgeCost(\ (x_l, x_r),\ (x_l, x_r - 1)\ ) = \ |\ I_l(x_l) - I_r(x_r)\ |$$

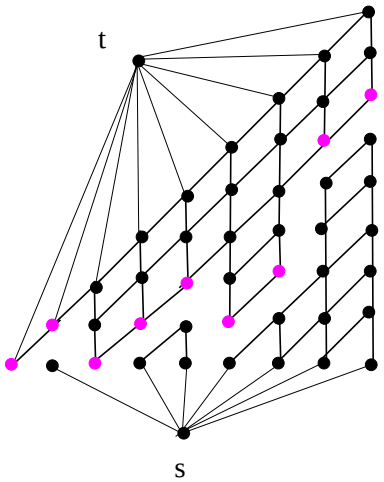
This is called the *data cost*. Note that this cost can be zero, namely if the intensities in the left and right eye are equal at the chosen disparity.

Minimum cut = maximum flow

One can solve the graph cut problem above by re-mapping it to a well known problem in graph theory. If you have a graph with weighted edges, then you can interpret the weights as *flow capacities* in a flow network. Think of a network of pipes and you want to pass as much water as possible through the pipes. In particular, suppose you have one vertex s which is a source of the flow and another vertex t which is a termination (or sink) of the flow. You then try to find out the maximum flow that you can send from s to t. There is a well-known result that the maximum flow you can attain is the equivalent to the minimum cut, namely the minimum total cost of weights of edges that cut the graph into a set of vertices (path-)connected to the source and vertices (path-) connected to the sink.

To turn our graph problem into a max-flow/min-cut problem, we need to define a source vertex and a sink vertex, along with the edges that join these vertices to the existing graph. See below.

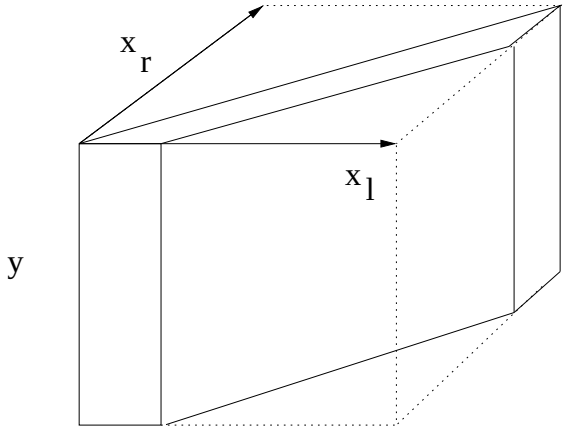


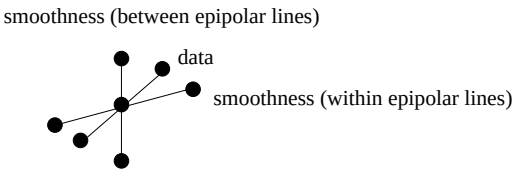


The max-flow/min-cut method was first applied to binocular stereo by Sebastien Roy[3] who is now a professor at U de Montreal. More recent methods use a different (and more complicated) graph construction[4], and give better results.

2D images

I have given you the impression that stereo is a 1D image problem, since we are using rectified images and so correspondences occur along (horizontal) epipolar lines which are rows in the image. While it is true that matching occurs along epipolar lines, good matches requires taking advantage of smoothness constraints (surfaces are piecewise smooth), and these smoothness constraint apply both within rows and between rows. Current methods in fact build a graph on a 3D grid such as shown below, and impose smoothness constraints between rows. (Details omitted.)





In addition, it is typically unnecessary to use the entire range of disparities. Typical stereo images have disparities from 0 to 50 pixels, with an image width of say 500 pixels. One can use much less memory by using only part of (x_l, x_r, y) volume, i.e. chopping off the part of the graph which corresponds to larger disparity values. See a sketch above.

[3] S. Roy, "A Maximum-Flow Formulation of the N-Camera Stereo Correspondence Problem" ICCV 98

[4] Y. Boykov, O. Veksler, R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts", PAMI 2001