Stereo: the graph cut method

Last lecture we looked at a simple version of the Marr-Poggio algorithm for solving the binocular correspondence problem along epipolar lines in rectified images. The main idea was 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. We noticed that elements that satisfy $x_l - x_r = d$ for fixed d correspond to point of constant depth Z. These are diagonal lines in the matrix.

There are several problems with the Marr Poggio algorithm. First, it uses only binary images. Second, it is difficult to characterize what the algorithm does. Is the solution optimal in any sense? More recent stereo methods use a similar representation (namely (x_l, x_r) space) but they pose the correspondence problem in a very different way. Today I will sketch out one such method which has been show to perform very well.

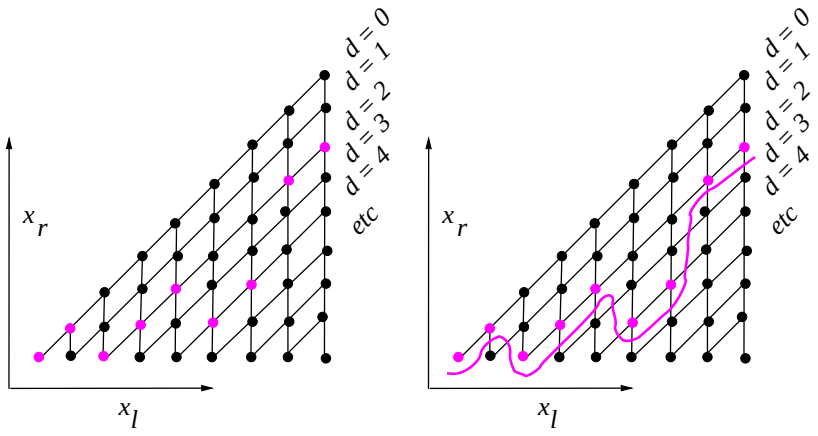
For any corresponding pair of epipolar lines in the rectified images, define a graph $G = (V, E)$ as follows. The vertices V consist of pairs $\{(x_l, x_r) : \ x_l - x_r \geq 0\}$. The edges E are a set of "neighbors" in V:

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

The edges are *undirected*, and each vertex belongs to at most four edges. Those on vertices (x_l, x_r) on the boundary of the (x_l, x_r) space belong to fewer than 4 edges.

One way to define the stereo correspondence problem is to find, for each position x_l in the left image, a corresponding position x_r in the right image (and hence disparity $d = x_l - x_r$). So, we assign a *unique* disparity value to each x_l. Such vertices are shown in pink in the figure below on the left.

Note, as we saw with our example last class, there may be 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.

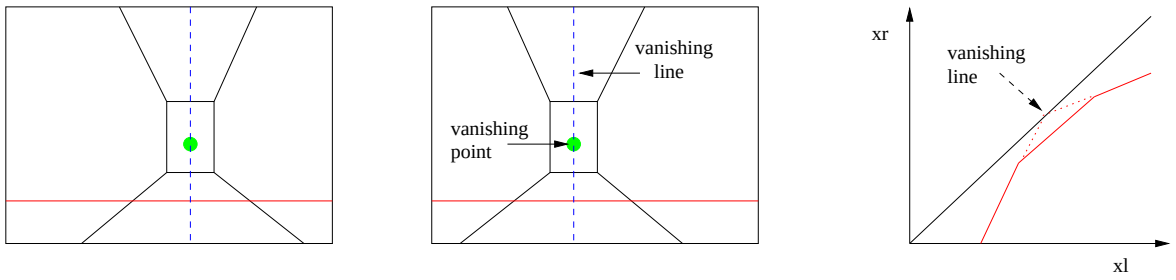
Note that in the example discussed last class (and in the figure above) each surface lies at a constant depth plane and hence the surface points lie on diagonals in (x_l, x_r) space. This is not the case in general, though. As we saw in lecture 2 and several other times in the course, a slanted plane in (X, Y, Z) gives rise to a slanted plane in $(x, y, \frac{1}{Z})$ space. But note that

$$\begin{bmatrix} x_l \\ x_r \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -fT & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_l \\ \frac{1}{Z} \\ y \end{bmatrix}$$

and so a plane in XYZ space (the 3D scene) produces a plane in (x_l, x_r, y) space too. This plane would be diagonal ($x_l - x_r = dconstant$) only if the scene plane were of constant depth Z. Otherwise disparity $d(x_l)$ would be a linear (and non-constant) function of x_l. Thus, there would be a small step in d for each step from x_l to $x_l + 1$.

The example below illustrates this effect. *(Here I elaborate on this example a bit more than I did in class.)* A corridor is shown. The two walls of the corridor are parallel planes in 3D. Their image projections share a vanishing line which is the vertical dashed line. This vanishing line is not "visible" though, since it corresponds to a (vertical) horizon which is at infinity and the scene is finite (closed).

The edges of the floor and ceiling along the corridor would, if extended to infinity in the scene, terminate in the image at a vanishing point which is marked in the image by a green dot. This green dot is not shown on the (x_l, x_r) plot on the right, because (1) the row containing the vanishing point is different from the red row and (2) the vanishing point is marking a point at infinity but this point at infinity is not visible in the image since the corridor is finite. Instead, on the right, I



have pointed to a point on the $d = 0$ plane ($Z = \infty$). This is the point where the walls would meet if there were no floor and if the walls extended to infinity (no end to the corridor).

Consider how this epipolar plane (in red) would map to the *discretized* (x_l, x_r) space – i.e. to the discrete grid. We would not get diagonal lines for the two walls. Rather, these red lines would be approximated with a set of diagonal lines which occasionally jump from one diagonal (disparity) to the next. These small jumps in disparity cannot be avoided since we are working on a square grid. Thus, surfaces that are slanted in depth Z would give rise to a small number of smoothness costs.

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.

Other costs (not discussed in class)

There are many ways to set up the graph cut problem for stereo, and different ways of setting up the graph allow one to define different costs. For example, if there are two x_l values that have the same x_r value, then it might be better only to associate a data cost with one of the matches, namely the one with the larger disparity – the reason being that if the two disparities are indeed correct, the match with the smaller disparity would not be visible to the right eye and it would make no sense to penalize such matches according to the data cost since we would not expect a good intensity match for if a point is only visible to the left camera. (Find the three examples of this in the figure on page 1.)

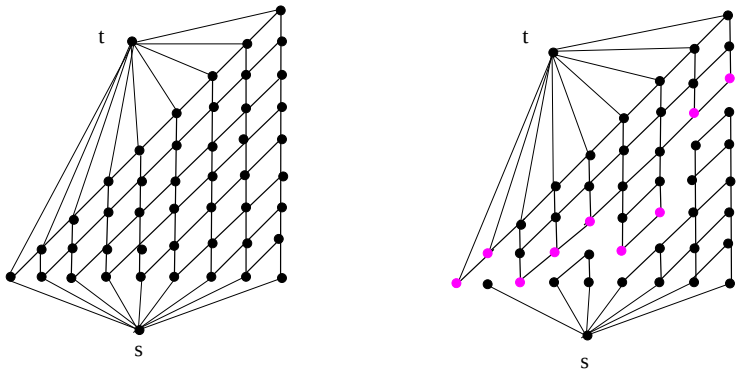
Note, however, that if we want to decide whether a vertex (x_l, x_r) is visible to the left camera (and hence whether to assign a data costs to the edge), we would need to know the solution (the cut) beforehand! This requirement is not met in the way we have posed the problem.[1]

Wrapup - a few technical points

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[2] and independently by Ishikawa and Geiger in 1998. More recent methods use a different (and more complicated) graph construction[3], and give better results.

2D images

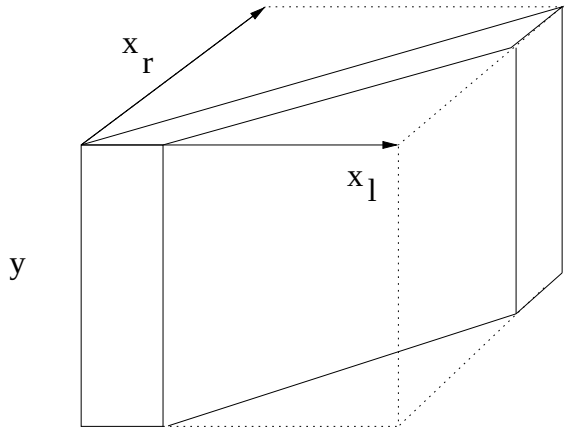
I have given you the impression that stereo is a 1D image problem, since we are using rectified images and so correspondences occur along (horizontal) epipolar lines which are rows in the image. While it is true that matching occurs along epipolar lines, good matches requires taking advantage of smoothness constraints (surfaces are piecewise smooth), and these smoothness constraint apply

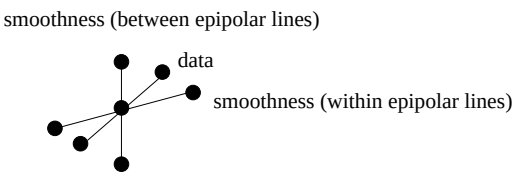
[1] See the Cornell University graph cuts page `http://www.cs.cornell.edu/~rdz/graphcuts.html`

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

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

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.