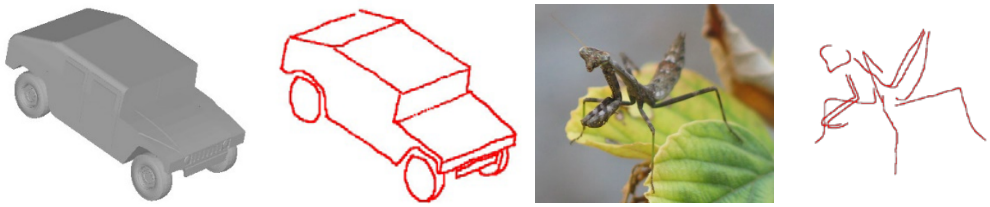
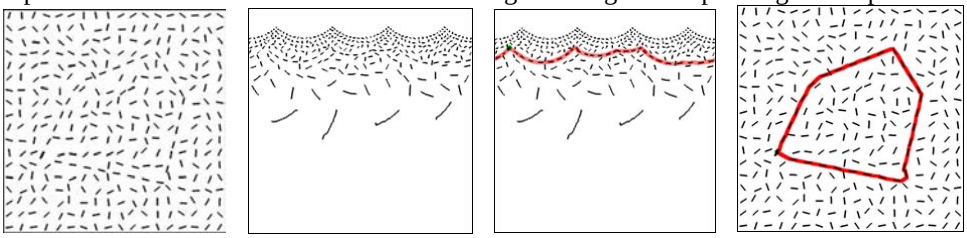
Contour Integration in Real Images

Peng Sun, Ran Duan, Doreen Hii and Zygmunt Pizlo
Department of Cognitive Sciences, UC Irvine

During the last decade, we have shown how 3D shapes can be recovered from a single 2D camera image by applying symmetry, compactness and planarity constraints. We used 2D images of real and synthetic objects like those shown below. The actual input to our algorithm was a line-drawing representing the object (see below in red). These contours had to be labeled because the algorithm had to "know" which pairs of contours were symmetrical in the 3D representation and which contours were planar. Recently, we formulated an algorithm, that performed all of these functions automatically, without any user intervention, when the input was a stereoscopic pair of images of a 3D object. The next step, was to generalize this algorithm so that it can recover 3D shapes from a single real 2D image.



Here, the first step was to find some object in the 2D image. Finding an object boils down to finding its occluding contour. Using synthetic images, like the one shown on the left below, our algorithm could integrate a closed curve by solving the shortest (least-cost) path in the complex-log representation of the image. The complex-log representation of the image, the least-cost path in that representation and the closed contour in the original image corresponding to this path are shown below.



Now, that we had an algorithm that can, in principle, find the occluding contour of an object in a 2D image, the next step was to extract meaningful contours of the object. More specifically, we wanted to extract a good 2D line-drawing to represent the object. We began with applying Line Segment Detection (LSD) to a grey-scale image. Next, we set up a complete graph K representing the line-segments and pairwise connections between the line-segments. Each line-segment L_i in the image is represented by 2 nodes V_{i1} and V_{i2} in the graph K(V,E). This redundant representation of the line-segments is needed in order to represent all 4 possible interpolations between the endpoints of each pair of line-segments L_i and L_j. These interpolations are edges E in K. Next, we selected 8 start/end-segments that are the closest to eight equally-spaced points on a circle circumscribed on the region containing the object. Then, we computed all 28 least-cost paths between pairs of start/end-points using the Dijkstra algorithm. Here, the least-cost path is computed in the Cartesian coordinates of the actual image. The cost of each edge E in the graph is a product of the cost of the turning angle between line segments and the cost of the interpolation between these line segments:

$$\text{Cost}(E) = (\text{turning angle} + c_1)\cdot(\text{distance} + c_2);\ c_1=0.3\text{rad};\ c_2=0.001$$

Note that we multiply the cost of the turning angle and the cost of interpolation, instead of summing them up. This product will assure that edges in K representing small turning angles OR short interpolations will have a small cost. A large cost will only be assigned to edges in K that have large turning angles and long interpolations. The line segments that are in the union of the 28 least-cost paths are removed from the image and the Dijkstra algorithm is applied again. The distances in the image were normalized to the geometric mean of the height and width of the image. This algorithm is able to find the most meaningful edges in the real images that contain a single object in front of a uniform background – see examples below. More importantly, the edges were represented as paths that are long curves. These curves could now be analyzed in order to determine which pairs of curves were mirror-symmetrical in the 3D representation.