CORRIDOR SCISSORS: A SEMI-AUTOMATIC SEGMENTATION TOOL EMPLOYING MINIMUM-COST CIRCULAR PATHS

Dirk Farin[1], *Magnus Pfeffer*[3], *Peter H. N. de With*[2], *and Wolfgang Effelsberg*[3]

[1] Univ. of Technol. Eindhoven
[2] LogicaCMG / Univ. of Technol. Eindhoven
5600 MB Eindhoven, Netherlands
d.s.farin@tue.nl

[3] University Mannheim
68131 Mannheim, Germany
magnus.pfeffer@bib.uni-mannheim.de

ABSTRACT

We present a new semi-automatic segmentation tool, which is motivated by the Intelligent Scissors algorithm [1], but which uses a modified concept of user-interaction. This new interface provides better capabilities for modifying previous segmentation results. The advantage of the new approach is that it enables to gradually increase the quality of the segmentation. The segmentation tool is based on a shortest circular path search within a corridor that is drawn by the user along the object boundary. For this purpose, we present a new algorithm for computing the shortest circular paths. Our algorithm is so fast that it almost achieves the speed of a regular non-circular shortest path search, while still ensuring an optimal solution.

1. INTRODUCTION

Video object segmentation plays a central role in a variety of applications, including object-based video coding, video editing, or video analysis. Since a completely manual segmentation is a very time-consuming task, automatic video-object segmentation algorithms have been proposed. However, for applications that require high segmentation accuracy, automatic algorithms still cannot provide satisfactory results in most cases. Hence, as a compromise between accuracy and efficiency, semi-automatic segmentation algorithms have been developed which use user-assistance to coarsely define the object and an automatic refinement to relieve the user from working at the pixel level.

A popular approach is a class of image segmentation algorithms known as *Intelligent Scissors* [1], of which the interactive variant is usually referred to as *Live-Wire*. The basic idea is to provide the user with a scissors tool that automatically snaps to object boundaries, reducing the need for pixel-accurate interaction. The Intelligent Scissors algorithm considers the input image as a graph, where each image pixel corresponds to one node in the graph. Graph edges connect nodes that correspond to neighboring pixels. Weights are assigned to these edges according to the inverse gradient strength between the two corresponding pixels. Strong gradients hereby induce small edge weights. After manual placement of a seed and destination position, a minimum cost path is computed between the two positions. Since the shortest path will run along boundaries with strong gradients, it corresponds well with object boundaries.

With increasing distance between seed and destination position, an annoying effect becomes apparent (Fig. 1): the minimum-cost path often snaps to background clutter if it contains stronger gradients than the object boundary, since this leads to a lower path cost. Even though the cost to reach this background clutter may be high, the lower cost along the strong background gradient outweighs a slightly higher cost along the desired object boundary when the path length increases. In the Live-Wire user-interface, this effects shows as a sudden change of the complete path or as toggling between alternative paths. Hence, to define a complete video object with the conventional algorithm, the boundary has to be drawn in segments, restarting the graph-search at the beginning of each segment. Once a segment has been fixed, a later modification of that segment is not provided.

As a solution for the problem of background clutter attraction, it is proposed in [2] to limit the search area to a rectangular area between the seed and destination position. The width of the rectangle is controlled by the user. While this approach may ease the segmentation in some difficult situations, it complicates the interaction process, since another degree of freedom (width of rectangle) has to be controlled by the user. Moreover, it does not provide a solution

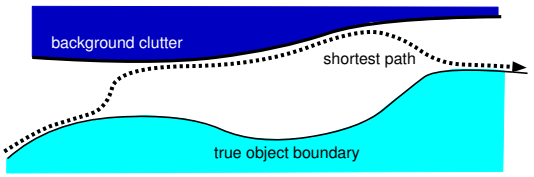


Fig. 1. The shortest path is distracted from the true object boundary because of near high-gradient background clutter.

0-7803-8554-3/04/$20.00 ©2004 IEEE.

(a) Segmentation with low object/background contrast.

(b) Path is distracted to strong gradient.

(c) Interruption of the wrong path leads to the correct segmentation.

Fig. 2. Segmentation examples. If the path is distracted to a higher contrast edge (b), the corridor can be modified to make this path impossible (c).

to the difficulty of modifying previously defined segments. We adopt the concept of a limited search-area in our algorithm, while not using a rectangular area constraint, but an arbitrarily shaped corridor along the object boundary. The new segmentation tool, which we call *Corridor Scissors*, allows the user to mark a coarse circular corridor area around the object. This can be done very quickly, since pixel-exact work is not required. After the circular corridor has been defined, the computer searches for a shortest circular path inside of the corridor. The corridor not only prevents that the path is attracted by distant background clutter, but it also reduces computation time, since the search space is reduced. If the user wants to improve the segmentation, he can do so by simply changing the shape or width of the corridor.

The underlying algorithm of Corridor Scissors is based on the same graph-search approach as the Intelligent Scissors algorithm. However, instead of searching for ordinary shortest paths, which can be easily computed using the Dijkstra algorithm [3], an algorithm for computing the shortest circular paths has to be applied. In this paper, we present a new algorithm for the shortest circular path problem, which has comparable computation time to that of the Dijkstra algorithm.

2. CORRIDOR SCISSORS TOOL

The corridor scissors tool uses a minimal, yet flexible user interface to define the desired segmentation. The user only requires to control two tools. A *corridor drawing pen*, which he uses to coarsely trace the object contour, and an *eraser* to reduce the size of the corridor if background clutter degrades the segmentation quality. The corridor may be wide when the object boundary is clear, whereas a narrower corridor might be advantageous in difficult areas. An example segmentation is shown in Figure 2.

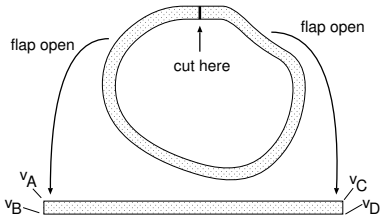


Fig. 3. The corridor is cut apart to get a non-cyclic graph.

Whenever the corridor shape is modified, a shortest circular path search is applied to the corridor area. The resulting path is taken as the object boundary. The costs that are assigned to the edges are composed of a weighted sum of gradient strength f_G and a Laplacian zero-crossing detector f_Z. If we denote the input image with I, they are defined as

$$f_G = 1 - \frac{||\nabla I||}{\max ||\nabla I||} \; ; f_Z = \begin{cases} 0 \text{ at Laplacian zero crossings,} \\ 1 \text{ otherwise,} \end{cases}$$

and the combined edge cost is defined as $f = f_G + \alpha f_Z$. The two parts of the cost function correspond to the two most important costs proposed in [1]. The original work proposes a sum of six different cost components, but the remaining four have only small weight and did not show to have much influence on the segmentation result.

3. CIRCULAR-PATH SEARCH

Searching for shortest circular paths differs from ordinary shortest path algorithms in that no seed and destination nodes are known in advance. In [4], several algorithms for the circular path search are proposed. All are based on the idea of cutting the circular graph into a lane-shaped non-cyclic graph (Fig. 3) and then solving the shortest circular path problem by building upon ordinary shortest path algorithms.

The *Multiple Search Algorithm* (MSA) uses N independent runs of the Dijkstra algorithm, where N is the width of the graph at the cutting position. Each run assumes fixed, opposing seed and destination nodes at both ends of the graph. At the end, the result that gives the minimum cost is selected. Even though this algorithm is guaranteed to always find the optimal solution, it is computationally intensive since the Dijkstra algorithm has to be executed N times independently over the full graph (N usually has a magnitude of about 15).

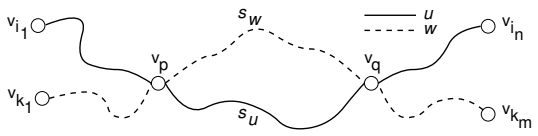


Fig. 4. Two minimum-cost paths with at least two common nodes share the whole subpath between the common nodes.

The *Image Patching Algorithm* only gives an approximate solution, but requires less computation time. Here, the non-cyclic graph is enlarged by appending part of the graph from each end to the opposite side. An ordinary shortest path is then computed through the complete, patched graph. The part of the shortest path that lies inside of the original graph is extracted and assumed to be the optimal circular path. However, even though this heuristic works in many cases, it is not assured that the optimal circular path is found. Moreover, the algorithm can even lead to non-cyclic paths. The quality of the result can be increased by enlarging the patched areas, but the required patch size is not known and the computation time increases.

In the following, we present a new algorithm to compute shortest circular paths, which has a low computation time and which guarantees to find the optimal path. Since we exploit two special properties of the Dijkstra algorithm, we briefly review them here.

Property 1: The Dijkstra algorithm always builds a complete shortest path tree, rooted at the seed node. Hence, a single run of the Dijkstra algorithm does not give only the shortest path to the specified destination, but also the shortest path to every other node.

Property 2: New nodes are inserted into the shortest path tree in the order of increasing distance from the seed node. Hence, if only the path to the specified destination node is required, the computation can be aborted as soon as the destination node is reached.

Additionally, we need the following theorem about shortest paths.

Theorem: Let $G = (V; E)$ with $V = \{v_i\}$ be a graph and let $u = v_{i_1} v_{i_2} \dots v_{i_n}$ and $w = v_{k_1} v_{k_2} \dots v_{k_m}$ be two minimum cost paths (see Fig. 4). Then we can state that if u and w have two nodes v_p and v_q in common, all nodes on the minimum paths between v_p and v_q are equal[1].

Proof: Assume that the two sub-paths s_u, s_w between v_p and v_q are not equal. Then, the cost of either s_u or s_w must be lower than the other. Let us assume that the cost of s_u is lower than the cost of s_w. Consequently, w cannot have minimum cost, because the cost can be lowered by replacing the subpath s_w with s_u. □

[1]In case of multiple paths with the same costs, there must exist one minimum cost path with common nodes between v_p and v_q.

Corollary 1: Minimum cost paths may cross at most once.

Corollary 2: If two paths share a common seed, then both paths will share a common subpath to their destination until both paths split. After the split, the paths will not cross.

In other words, paths with disjoint endpoints may cross once, while paths with one endpoint in common will never cross.

Shortest Circular-Path Algorithm

Using the above-mentioned properties allows to define a fast algorithm for computing shortest circular paths. The complete circular path search operates in four steps.

Step 1 Cut the circular corridor into a lane as shown in Fig. 3. It is important that the cut lies orthogonal to the corridor direction. Since the shortest path will only cross the cut once, placing the cut in an acute angle along the corridor may lead to bad results (see Fig. 5). A very good cut placement is obtained by starting a shortest path search at an arbitrary pixel at the outer boundary of the corridor and searching for a path to the inner boundary with uniform edge weights. As soon as the inner boundary is reached, the search can be aborted. The obtained path serves as the corridor cut.

Step 2 Perform a Dijkstra-search beginning from node v_A, which is the top-left node of the lane. This pass will compute at the same time shortest paths to nodes v_C and v_D (see Fig. 6a). Since the starting point is shared, both paths will share a subpath up to a node v_r. Shortest paths from the left side of the lane to the right side cannot traverse the area above $v_A \rightsquigarrow v_C$, since this would mean that they have to cross $v_A \rightsquigarrow v_C$ twice, which is not allowed because of the previous theorem. Hence, all nodes above the path $v_A \rightsquigarrow v_C$ can be ignored in the following steps.

Step 3 Perform a second Dijkstra-search from node v_B to node v_D. In most practical cases, this path will join the two shortest paths from the last step at some node v_l (see Fig. 6b). If this is the case, then we are sure that all shortest paths between the left side and the right side have a least the subpath $v_l \rightsquigarrow v_r$. Otherwise, we use a fallback algorithm described below.

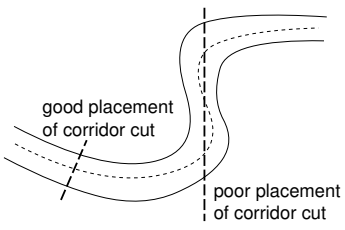


Fig. 5. Corridor cuts should be placed at narrow positions, perpendicular to the corridor. Note that the shortest path cannot cross the cut more than once.

Step 4 Since we already know that the subpath $v_l \rightsquigarrow v_r$ is part of the shortest circular path, we only have to search for a connection from v_r back to v_l to close the circle. We also know that this connection must lie between the area of the previously computed shortest paths. Hence, we perform a third Dijkstra-search from v_l to v_r over the nodes in the shaded area depicted Fig. 6c. Appending this path to the path $v_l \rightsquigarrow v_r$ gives the shortest circular path inside of the corridor.

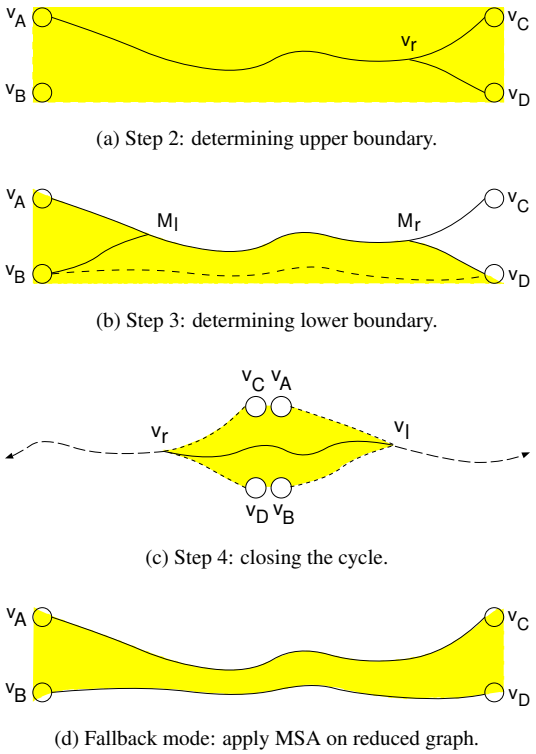
If, in Step 3, the shortest path from v_B to v_D does not join the path $v_A \rightsquigarrow v_C$, no common subpath exists and the MSA algorithm from [4] has to be applied as a fallback-algorithm. However, we can restrict the search to a reduced graph, consisting only of the nodes between $v_A \rightsquigarrow v_C$ and $v_B \rightsquigarrow v_D$ (see Fig. 6d). During our experiments, this special case was very rare, and it only happened with some very small objects. In this case, the corridor size was so small that applying the MSA algorithm did not lead to an observable slowdown. Since this special case is rare, Step 3 can be implemented more efficiently by allowing a sub-optimal solution in these cases.

Modified Step 3 Compute a full path between v_B and v_D, but stop as soon as the search reaches the path $v_A \rightsquigarrow v_C$. The connecting node is labeled v_l and the search is aborted. Under the assumption that there is a common subpath and because of the property of the Dijkstra algorithm to process nodes in the order of increasing distance to the seed, this is equivalent to the Step 3 as described above. Implementation of the MSA algorithm is not needed in this case.

The computation time of our algorithm is very low. The path search of Step 1 is aborted after N Steps (N being the width of the corridor). Step 2 has the complexity of an ordinary Dijkstra algorithm, but over the corridor area only. The search in the modified Step 3 can usually be aborted after a small number of steps, and the search in Step 4 is over a small area. Hence, the total computation time is only little more than a single Dijkstra run. The exact number depends on the length of the corridor and the image content, but usually, the computation time is only about 5% more than ordinary Dijkstra. Moreover, because of the grid-structure of the graph, a single Dijkstra-search only takes time $O(|V|)$.

4. CONCLUSIONS AND FUTURE WORK

We have presented Corridor Scissors as a new technique for semi-automatic image segmentation. It provides an intuitive user-interface which especially supports incremental modifications to the segmentation to improve its quality. Furthermore, an algorithm for computing shortest circular paths has been proposed, which is the basis for our segmentation technique. The described algorithm almost reaches the same computation efficiency as the regular shortest path computation using the Dijkstra algorithm.



(a) Step 2: determining upper boundary.

(b) Step 3: determining lower boundary.

(c) Step 4: closing the cycle.

(d) Fallback mode: apply MSA on reduced graph.

Fig. 6. Illustration of Steps 2-4 of the shortest circular path search algorithm.

Future research concentrates on enhancing the current still image segmentation to support object tracking, thereby making it easier to segment objects through a video sequence. Moreover, further research on the shortest circular path algorithm may lead to the elimination of the special MSA fallback-case, with the aim to find an optimal solution at a guaranteed low computational cost.

5. REFERENCES

[1] E. N. Mortensen and W. A. Barrett, "Interactive segmentation with intelligent scissors," *Graphical Models and Image Processing*, vol. 60, pp. 349–384, 1998.

[2] Huitao Luo and Alexandros Eleftheriadis, "Rubberband: An improved graph search algorithm for interactive object segmentation," in *Proc. IEEE International Conference on Image Processing (ICIP)*, 2002, vol. 1, pp. 101–104.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[4] Changming Sun and Stefano Pallottino, "Circular shortest path on regular grids," Tech. Rep., CSIRO Mathematical and Information Sciences, May 2001.

[5] Davi Geiger, Alok Gupta, Luiz A. Costa, and John Vlontzos, "Dynamic programming for detecting, tracking, and matching deformable contours," *IEEE Trans. on PAMI*, vol. 17, no. 3, pp. 294–302, 1995.