A 3D Live-Wire Segmentation Method for Volume Images Using Haptic Interaction

Filip Malmberg[1], Erik Vidholm[2], and Ingela Nyström[2]

[1] Centre for Image Analysis, Swedish University of Agricultural Sciences, Uppsala, Sweden
[2] Centre for Image Analysis, Uppsala University, Uppsala, Sweden
{filip, erik, ingela}@cb.uu.se

Abstract. Designing interactive segmentation methods for digital volume images is difficult, mainly because efficient 3D interaction is much harder to achieve than interaction with 2D images. To overcome this issue, we use a system that combines stereo graphics and haptics to facilitate efficient 3D interaction. We propose a new method, based on the 2D live-wire method, for segmenting volume images. Our method consists of two parts: an interface for drawing 3D live-wire curves onto the boundary of an object in a volume image, and an algorithm for connecting two such curves to create a discrete surface.

1 Introduction

Fully automatic segmentation of non-trivial images is a difficult problem, and despite decades of research there are still no robust methods for automatically segmenting arbitrary images. This is mainly due to the fact that it is hard to identify objects from the image data only. Often, we also need some high-level knowledge about the type of objects we are interested in. This is recognized in [1], where the segmentation process is divided into two steps: recognition and delineation. Recognition is the task of roughly determining where in the image the objects are located, while delineation consists of determining the exact extent of the object. Human users outperform computers in most recognition tasks, while computers are often better at delineation. Semi-automatic segmentation methods take advantage of this fact by letting a human user guide the recognition, while the computer performs the delineation. A successful semi-automatic method minimizes the user interaction time, while maintaining tight user control to guarantee the quality of the result.

Many semi-automatic segmentation methods exist for two-dimensional (2D) images, but in most cases it is not obvious how to extend these methods to work efficiently for three-dimensional (3D) images. One problem is that efficient interaction with a 3D image is much harder to achieve than interaction with a 2D image. In this project, we have used a system that simplifies 3D interaction. The system supports 3D input, stereo graphics, and haptic feedback through a sensing probe. The haptic feedback allows the user to feel where surfaces and structures in the volume are located, making it easier to navigate within the volume. Haptic feedback has previously been used in the context of interactive image segmentation in, e.g. [2] and [3]. Our system gives us new possibilities to create efficient user interfaces for interacting with volume images. The aim is to investigate how these new possibilities can be used to extend an existing semi-automatic segmentation method to also handle volume images efficiently.

Live-wire [1, 4] is a popular semi-automatic segmentation method that has been used successfully for many 2D problems. Various ways of extending this method to segment volume images have been proposed in the literature. Most of these methods are based on using the 2D live-wire method on a subset of 2D slices in the volume, and then reconstructing the entire object using this information. Examples of such approaches can be found in [5, 6, 7, 8]. While the reconstruction algorithms might take 3D information into account, all user interaction is performed in 2D. This restriction gives rise to problems, e.g., how to handle cases where a single object has different numbers of connected components in consecutive slices. The method presented in this paper is a step towards a more direct 3D approach that may overcome these problems.

In our method, the user still draws live-wire curves, but these curves are not restricted to planes. Pairs of such curves are then connected by a surface in the following way: using the image foresting transform (IFT) [9], a large number of live-wire curves are computed between the original curves. The areas between these curves are covered by polygons which are then rasterized to produce a tunnel-free discrete surface between the two curves that well approximates the boundary of the underlying object.

2 Environment

Our setup consists of a Reachin desktop display [10] which combines a PHANToM desktop haptic device [11] with a stereo-capable monitor mounted over a

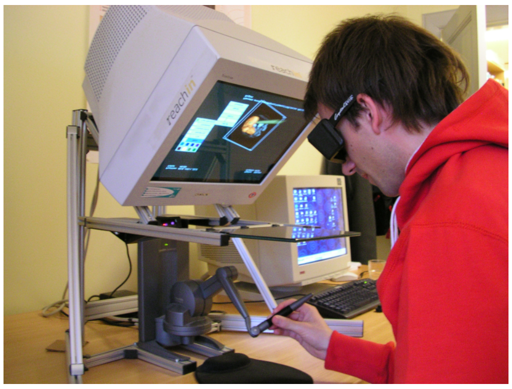


Fig. 1. The Reachin desktop display [10]. The PHANToM device [11] is positioned beneath a semi-transparent mirror. The graphics are projected through the mirror to obtain co-location of graphics and haptics.

semi-transparent mirror. See Figure 1. The PHANToM is a 3D input device that also provides the user with haptic feedback. It is designed as a stylus, and the haptic feedback is given at one point, the tip. It provides six degrees of freedom (DOF) for input and three DOF for output, i.e., the input is the position and orientation of the stylus, and the output is a force vector.

We have implemented our software using the Reachin API 3.2 [10], which is a C++ API that combines graphics and haptics in a scene-graph environment. The workstation we use is equipped with dual 2.4 GHz Pentium IV processors and 1GB of RAM.

3 The 2D Live-Wire Method

The basic idea of the live-wire method is that the user places a *seed-point* on the boundary of an object. As the user moves the cursor in the image, a path (live-wire) is displayed in real-time from the current position of the cursor back to the seed-point. The wire is attracted to edges in the image and it will snap onto the boundary of the object. If the user moves too far from the original seed-point, the wire might snap onto an edge that does not correspond to the desired object. When this happens the user can move the cursor back a little and place a new seed-point. The wire from the old seed-point to the new then freezes, and the tracking continues from the new seed-point. In this way, the entire object boundary can be traced with a rather small number of live-wire segments. See Figure 2.

The live-wire algorithm is based on graph-searching techniques. Therefore, we consider the image to be a graph where the nodes correspond to pixels and the

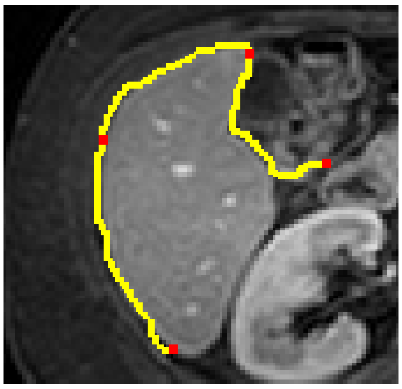


Fig. 2. In the live-wire method, the user segments an image by interactively placing seed-points along the boundary of an object. As the user moves the cursor through the image, a path from the last seedpoint to the current cursor position is displayed in real-time. The path attempts to follow edges in the image, and will thus snap onto the boundary of the object. Here, the liver in a slice of a magnetic resonance (MR) image is being segmented.

graph edges correspond to paths between pixels. For every edge, a cost is assigned to represent the "likelihood" that the edge belongs to a desired boundary in the image. How this cost function should be chosen is discussed in, e.g., [1]. When the user places a seed-point, Dijkstra's algorithm [12] is used to compute the optimal path to the seed-point from all other points in the image. Once these paths have been computed, it is possible to display the live-wire in real-time with virtually no computational cost as the user moves the cursor through the image.

4 Our 3D Live-Wire Method

In the 2D live-wire method, the user interacts with the program by placing seed-points which are then connected by curves in a 2D plane. Our 3D analogy of this is to let the user create curves, which are then automatically connected by discrete surfaces in 3D space, a process we call *bridging*. An entire object in a volume image can be segmented by drawing a relatively small number of live-wire curves on the boundary of the object.

After a tunnel-free surface representing the boundary of the desired object has been created, the entire region occupied by the object can be found by taking the complement of all connected background components that touch the border of the volume. To create this segmentation method, we need to solve two problems: (1) how to draw live-wire curves in 3D, and (2) how to connect two closed curves to form a tunnel-free discrete surface.

4.1 Drawing Live-Wire Curves in 3D

To extend the live-wire algorithm to draw closed curves in 3D, there are again two issues that we need to consider. First, we need to transform the volume image to a graph so that we can apply the shortest path calculation. This can be achieved in a number of ways, and we will see that the choice of graph structure will affect the performance of the shortest-path calculation. The second issue is how to design the user interface so that seed-points can easily be placed on the boundary of an object.

Defining a 3D Graph. In the literature on 2D live-wire, various ways of constructing a graph from the image have been used by different authors. Mortensen and Barrett [13] construct the graph by placing a node at the center of each pixel, with edges passing from each node to the nodes of all neighboring pixels. Another way of constructing the graph is proposed by Falcão *et al.* [1] where the nodes are instead placed at the pixel vertices, and the pixel edges are used as directed graph edges. The latter graph formulation has the advantage that thin (even one pixel wide) structures can be traced as a closed boundary, and the directed nature of the graph allows us to use cost functions that depend on the direction in which the edge is traversed. Both these 2D graph models are illustrated in Figure 3.

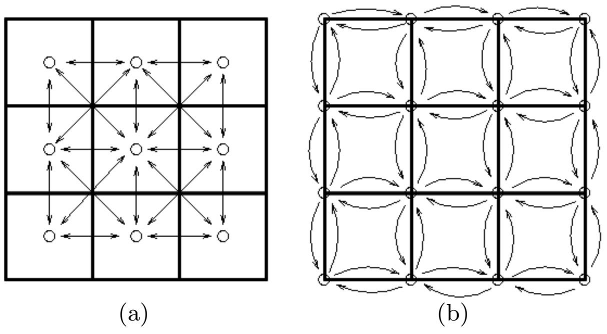


Fig. 3. Two different ways of transforming a 2D image to a graph. (a) The graph definition used in [13]. (b) The graph definition used in [1].

For our 3D graph, we have chosen to place a node at the center of each voxel and create edges to all its 26 neighbors. Although this definition does not use oriented edges, it has the advantage that the resulting graph has relatively few nodes. The bottleneck of Dijkstra's algorithm is the maintenance of a sorted heap containing all nodes that are currently examined, which means that fewer nodes lead to faster computations. Using a smaller neighborhood to define the edges of the graph would also result in faster computations, but the penalty in computational cost of increasing the number of edges is not as large as the penalty for increasing the number of nodes. Our experiments have shown that good realtime performance is achieved even if we use the full 26-neighborhood, which is advantageous since we then obtain thin and less "jagged" live-wire curves.

User Interface. We have implemented a user interface with two options for drawing live-wire curves onto the boundary of an object.

The first option is to place seed-points freely in the volume. In this case, the volume is displayed by maximum intensity projection (MIP). See Figure 4a. To help the user locate the boundary of the object, we have used a volume haptics technique described in [2]. The gradient of the volume is used as a surface normal that defines a virtual surface that can be felt with the haptic device.

The second option is to "slice" the volume with a plane that can be moved and rotated freely relative to the volume, and draw the curve onto this arbitrary plane. See Figure 4b. The plane is a haptic surface so that the user can feel where it is located and easily place seed-points onto it. Drawing a live-wire curve onto the plane feels similar to drawing on a real surface. The shortest-path computations are still performed in 3D, but since we want the curve to stay in the vicinity of the plane we also add a term to our cost function that grows quickly as we move away from the chosen plane.

The two methods have different advantages. To draw a curve around an object using the first method, the user might have to rotate the volume while drawing the curve in order to avoid twisting the wrist too much. Using the second method, however, it is easy to do this by placing the guiding plane at the desired

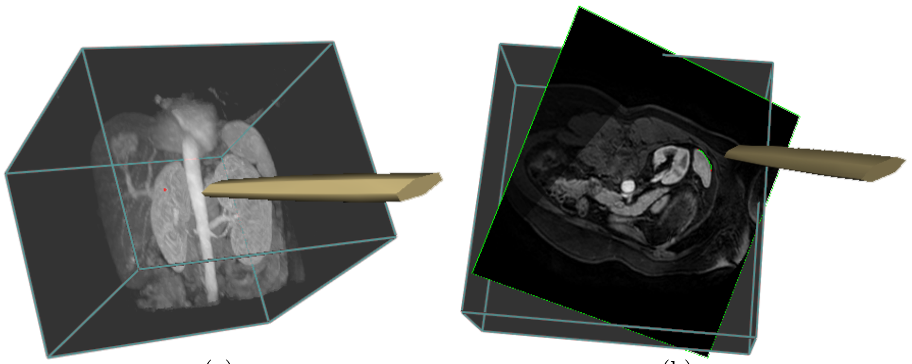

(a) (b)

Fig. 4. (a) Placing seed-points freely in the volume using volume rendering and volume haptics to locate the boundary of the object. (b) Drawing a live-wire curve onto a guiding plane.

cross-section of the object. When testing the application we have mainly used the second method for drawing curves, while the first method has been used for placing single points (which are also considered to be closed curves).

4.2 Bridging Algorithm

We have based our bridging algorithm on the image foresting transform (IFT) [9]. The IFT is essentially Dijkstra's algorithm, modified to allow an arbitrary number of seed-points as input for the computations. The result of using more than one seed-point is that we find the shortest path from all pixels in the image to *any* of the seed-points.

Using the IFT, it is quite straightforward to create a rough "wire-frame" of the surface we are looking for. To do this, we compute the optimal curves from each voxel in one of the curves, using all voxels of the other curve as seed-points. An illustration of the result is shown in Figure 5c.

Since the curves we compute are independent of each other, the result is generally not a tunnel-free surface. To fill the gaps between the curves we define a polygonal surface between adjacent curves, as shown in Figure 6. These polygons are then rasterized using the method described in [14] to obtain a discrete surface. The result is exemplified in Figure 5d. Algorithm 1. shows the pseudo-code for our bridging algorithm.

Since all points on the second curve are not necessarily the "closest" point to a point on the first curve, a surface generated using our method is unfortunately not guaranteed to fit the "goal" curve exactly. In the worst case, all points on the first curve have the same closest point on the second curve, in which case the second curve will be approximated by this point only. Our current solution to this is to apply the algorithm from both directions and return the union of the results, i.e., run the algorithm starting from one of the curves and then run

Algorithm 1. Bridging Algorithm

```
Input: Two sets of voxels, curve1 and curve2, that represent the initial curves.
Output: A set of voxels, output, that represents the generated surface.
Additional variables: Two sets of voxels, current_curve and next_curve that will be
used to store intermediate curves. A voxel next_voxel.

• current_curve ← curve1
while next_curve contains voxels that are not in curve2 do
  • next_curve ← empty
  for every voxel v in current_curve do
    • next_voxel ← v
    if v is not in curve2 then
      • calculate the optimal path from v to curve2
      • set next_voxel to be the next voxel along this path
    end if
    if next_curve is not empty then
      • rasterize the polygon between the last voxel of next_curve and the two
        last voxels of current_curve
      • add the resulting voxels to output
      • rasterize the polygon between next_voxel and the last voxels of next_curve
        and current_curve
      • add the resulting voxels to output
    end if
    • add next_voxel to next_curve
  end for
  • rasterize the polygon between next_voxel, the first voxel of next_curve and the
    last voxel of current_curve
  • add the resulting voxels to output
  • rasterize the polygon between the first voxels of current_curve and next_curve
    and the last voxel of current_curve
  • add the resulting voxels to output
  • current_curve ← next_curve
end while
```

it again starting from the other curve. The result then always contains the two original curves. However, if the shapes of the two curves differ to much "pockets" may be produced on the surface, as illustrated in Figure 7. In most cases when this happens, it is possible to reduce the error by drawing additional curves inbetween the two original curves.

4.3 Performance

For a moderately sized volume (128x128x100) our algorithm runs in approximately 1–20 seconds. The exact times depend on the length of the curves we are bridging and the distance between them. A number of things can be done to improve the performance. For example, our current implementation uses a heap data structure to manage the sorted queue when computing the IFT, but

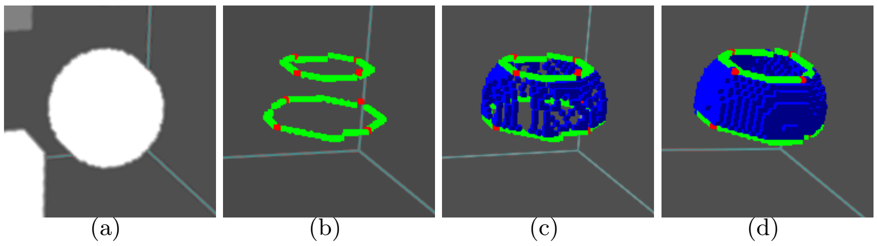


Fig. 5. (a) A synthetic object. (b) Two closed curves on the boundary of the object. (c) Result of connecting the two curves by live-wires. (d) Result of using our proposed algorithm.

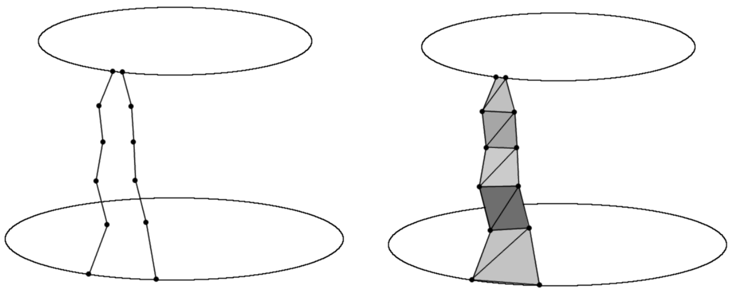


Fig. 6. To obtain a tunnel-free surface, all live-wire curves generated between the two user-defined curves are connected to their adjacent curves by triangular polygons

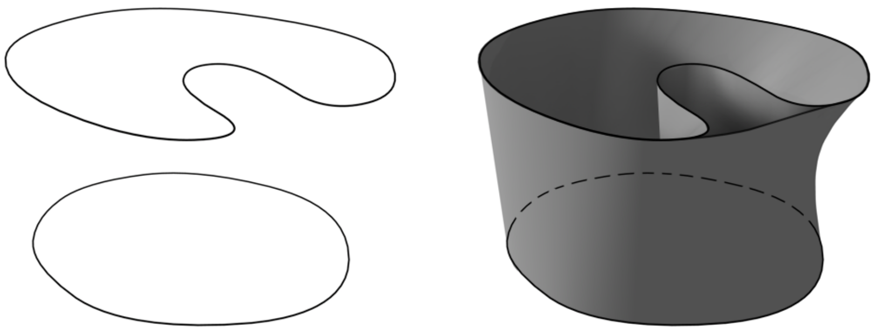
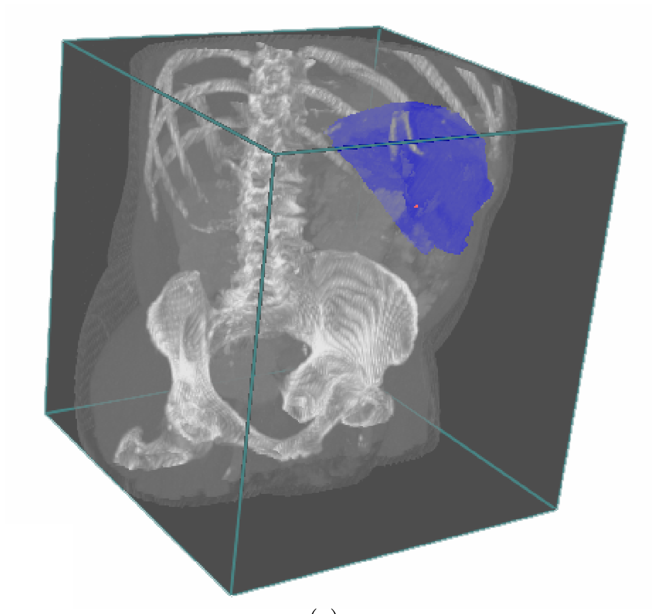
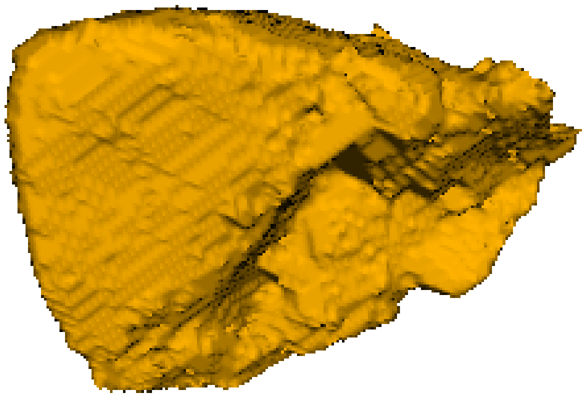


Fig. 7. If the shapes of the two curves differ too much, the surface produced by our bridging algorithm may contain pockets



(a)



(b)

Fig. 8. (a) Segmentation of the liver in a 153x153x161 CT image of a human, obtained with our method by drawing 7 live-wire curves onto the boundary of the liver. (b) Surface rendering of the segmentation result.

according to [4] better performance can be achieved by using a circular queue structure instead.

Tests on large volumes have not been possible due to the current memory-inefficient implementation where big amounts of data are stored redundantly during the shortest-path calculations. In future work, this will be made more efficient.

5 Discussion and Future Work

Despite the issues mentioned in Section 4.2, preliminary tests of our application show promising results. We have segmented objects of varying complexity using a varying number of live-wire curves. In most cases, the closed discrete surfaces obtained are satisfactory representations of the underlying object. Figure 8 shows an example of a segmentation result obtained with our method. The total segmentation time for this example, including user interaction, was approximately 3 minutes.

As stated in [1], a semi-automatic method should ultimately be measured by how much it reduces the time it takes for the user to achieve a correct segmentation result. To verify the efficiency of our method, it would be therefore be of interest to make a user study to compare it with other segmentation methods.

Acknowledgements

Prof. Gunilla Borgefors at the Centre for Image Analysis, Swedish University of Agricultural Sciences, Uppsala, Sweden is acknowledged for her scientific support.

References

1. Falcão, A.X., Udupa, J.K., Samarasekera, S., Sharma, S.: User-steered image segmentation paradigms: Live wire and live lane. Graphical Models and Image Processing (60) (1998) 223–260
2. Vidholm, E., Tizon, X., Nyström, I., Bengtsson, E.: Haptic guided seeding of MRA images for semi-automatic segmentation. In: Proceedings of IEEE International Symposium on Biomedical Imaging (ISBI'04). (2004) 278–281
3. Harders, M., Székely, G.: Enhancing human-computer interaction in medical segmentation. Proceedings of the IEEE, Special Issue on Multimodal Human Computer Interfaces **9**(91) (2003) 1430–1442
4. Falcão, A.X., Udupa, J.K., Miyazawa, F.K.: An ultra-fast user-steered image segmentation paradigm: Live wire on the fly. IEEE Transactions on Medical Imaging **19**(1) (2000) 55–62
5. Falcão, A.X., Udupa, J.K.: A 3D generalization of user-steered live-wire segmentation. Medical Image Analysis **4**(4) (2000) 389–402
6. Knapp, M., Kanitsar, A., Groller, M.E.: Semi-automatic topology independent contour based 2 1/2 D segmentation using live-wire. Journal of WSCG **20**(1–3) (2004)

7. Schenk, A., Prause, G., Peitgen, H.O.: Efficient semiautomatic segmentation of 3D objects in medical images. In: Proceedings of MICCAI 2000, Springer-Verlag (2000) 186–195
8. Hamarneh, G., Yang, J., McIntosh, C., Langille, M.: 3d live-wire-based semi-automatic segmentation of medical images. In: Proceedings of SPIE Medical Imaging: Image Processing. Volume 5747. (2005) 1597–1603
9. Falcão, A.X., Bergo, F.P.G.: Interactive volume segmentation with differential image foresting transforms. IEEE Transactions on Medical Imaging **23**(9) (2004)
10. Reachin Technologies: (http://www.reachin.se) Accessed 23 January 2006.
11. Technologies, S.: (http://www.sensable.com) Accessed 23 January 2006.
12. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
13. Mortensen, E.N., Barrett, W.: Interactive segmentation with intelligent scissors. Graphical Models and Image Processing **60**(5) (1998) 349–384
14. Kaufman, A.: Efficient algorithms for scan-converting 3D polygons. Computers and Graphics **12**(2) (1988) 213–219