An Ultra-Fast User-Steered Image Segmentation Paradigm: Live Wire on the Fly

Alexandre X. Falcão, Jayaram K. Udupa*, and Flávio K. Miyazawa

***Abstract*—We have been developing general user steered image segmentation strategies for routine use in applications involving a large number of data sets. In the past, we have presented three segmentation paradigms: live wire, live lane, and a three-dimensional (3-D) extension of the live-wire method. In this paper, we introduce an ultra-fast live-wire method, referred to as live wire on the fly, for further reducing user's time compared to the basic live-wire method. In live wire, 3-D/four-dimensional (4-D) object boundaries are segmented in a slice-by-slice fashion. To segment a two-dimensional (2-D) boundary, the user initially picks a point on the boundary and all possible minimum-cost paths from this point to all other points in the image are computed via Dijkstra's algorithm. Subsequently, a live wire is displayed in real time from the initial point to any subsequent position taken by the cursor. If the cursor is close to the desired boundary, the live wire snaps on to the boundary. The cursor is then deposited and a new live-wire segment is found next. The entire 2-D boundary is specified via a set of live-wire segments in this fashion. A drawback of this method is that the speed of optimal path computation depends on image size. On modestly powered computers, for images of even modest size, some sluggishness appears in user interaction, which reduces the overall segmentation efficiency. In this work, we solve this problem by exploiting some known properties of graphs to avoid unnecessary minimum-cost path computation during segmentation. In live wire on the fly, when the user selects a point on the boundary the live-wire segment is computed and displayed in real time from the selected point to any subsequent position of the cursor in the image, even for large images and even on low-powered computers. Based on 492 tracing experiments from an actual medical application, we demonstrate that live wire on the fly is 1.3–31 times faster than live wire for actual segmentation for varying image sizes, although the pure computational part alone is found to be about 120 times faster.**

***Index Terms*—Active boundaries, boundary detection, graph algorithms, image segmentation, shortest-path problem, 3-D imaging.**

I. Introduction

Image segmentation is a hard problem with numerous applications in the imaging sciences [1]. It consists of two tightly coupled tasks—recognition and delineation. Recognition is the process of identifying roughly the whereabouts of a particular object in the image and delineation is the process of specifying the precise spatial extent and composition of this object. While computer algorithms are very effective in object delineation, the absence of relevant global object-related knowledge is the main reason for their failure in object recognition. On the other hand, a simple user assistance in object recognition is often sufficient to complement this deficiency and to complete the segmentation process. There are many difficult segmentation tasks that require a detailed user assistance. To address these problems, a variety of interactive segmentation methods are being developed [2]. These methods range from totally manual painting of object regions or drawing of object boundaries, to the detection of object region/boundaries with minimal user assistance [3]–[7].

In interactive segmentation, deformable boundary approaches have been actively pursued and numerous publications have resulted during the past ten years on this topic [5]. The basic principle underlying these techniques was introduced by Kass *et al.* with the concept of active contour models [8]. The idea is that the users specify an initial contour, which is subsequently deformed based on image-derived criteria (such as energy), assuming that the contour at minimum global energy should match with the desired boundary. The users' interaction is in specifying image-independent criteria, such as external forces, in addition to specifying an initial contour properly. If a method using this approach provides a solution without requiring the user to correct the result at the end, that would be ideal. Unfortunately such is not the case. In many difficult situations, a more active and tighter control provided to the user on the process of segmentation would perhaps significantly reduce the time spent by him/her in the process. In spite of the user time being one of the most important factors in assessing the goodness of an interactive segmentation method, careful evaluation of this factor, especially compared to manual tracing, does not seem to have been carried out for published deformable boundary methods. Motivated by this efficiency consideration, we have been developing and evaluating application-independent interactive boundary-based segmentation strategies with two specific aims: 1) to provide as complete a control as possible to the user on the segmentation process while it is being executed and 2) to minimize the user involvement and the total user's time required for segmentation, without compromising the precision and accuracy of segmentation. Our strategy in these methods has been to actively exploit the superior abilities of human operators (compared to computer algorithms) in object recognition and the superior abilities of computer algorithms (compared to human operators) in object delineation.

In the past, we have presented two user steered segmentation paradigms, referred to as live wire and live lane [6], [9], to segment three-dimensional (3-D)/four-dimensional (4-D) object boundaries in a slice-by-slice fashion. These methods are in routine use in several applications [10]–[13] with over 15 000 tracings done so far. Although the live-wire method has its origin in some early joint work between Barrett and Udupa [14]–[16], this method has been subsequently developed independently by the two groups [6], [7], [9], [17]–[19]. We have extended the ideas underlying the live-wire method to create new methods, live lane [6] and the 3-D extension of live wire [17]. In this paper, we introduce an ultra-fast live-wire method, referred to as live wire on the fly, with a new algorithm for significantly reducing the user's time compared to our previous work on two-dimensional (2-D) live wire.[1]

In the live-wire method [6], [7], to segment a 2-D boundary, the user initially picks a point on the boundary and all possible minimum-cost paths from this point to *all* other points in the image are computed via Dijkstra's algorithm [20]. Subsequently, a live wire is displayed in real time from the initial point to any subsequent position taken by the cursor. If the cursor is close to the desired boundary, the live wire snaps

Manuscript received April 5, 1999; revised November 1, 1999. This work was supported in part by the CNPq under Grant 300698/98-4 and in part by the FAPESP under Grants 98/06314-5, 98/12308-8, and 97/13306-6, the NIH under Grant NS 37172, the Department of the Army, Project ProNEx 107/97 (MCT/FINEP) under Grant DAMD 179717271, and the CNPq under Individual Research Grant 300301/98-7. The Associate Editor responsible for coordinating the review of this paper and recommending its publication was M. Vannier. *Asterisk indicates corresponding author.*

A. X. Falcão and F. K. Miyazawa are with the Institute of Computing, State University of Campinas, Campinas, SP, Brazil, 13083-970 (e-mail: afalcao;fkm@dcc.unicamp.br; fkm@dcc.unicamp.br).

*J. K. Udupa is with MIPG, Department of Radiology, University of Pennsylvania, 423 Guardian Drive, Philadelphia, PA 19104-6021 USA (e-mail: jay@mipg.upenn.edu).

Publisher Item Identifier S 0278-0062(00)01225-8.

[1]Readers can test two different versions of live wire on the fly by downloading the software available in www.dcc.unicamp.br/~afalcao/segmentation.html and www.mipg.upenn.edu/~Vnews, respectively.

on to the boundary. The cursor is then deposited and a new live-wire segment is found next. The entire 2-D boundary is specified via a set of live-wire segments in this fashion. A drawback of this approach is the time taken to compute all possible minimum-cost paths from each selected point on the boundary to other points in the image. This time increases with the size of the image compromising the interactivity of the method in some practical situations. For images from 256×256 to 1024×1024 pixels, for example, live wire running on a 300-MHz Pentium PC requires about 2–180 s to compute all possible minimum-cost paths from each selected point.

In the live-wire-on-the-fly method, the user-interaction process remains the same, but we have devised a linear time complexity graph-search algorithm to save a considerable amount of user time by avoiding the computation of all possible minimum-cost paths. When the user selects a point on the boundary, the live-wire segment is computed and displayed in real time from the selected point to any subsequent position of the cursor in the image. To make this feasible, we exploit the fact that by the time we have found a live-wire segment with cost value K, we have actually found all possible live-wire segments with cost value less than K in the image. Moreover, any live-wire segment with a cost value greater than or equal to K contains one of the previous live-wire segments with cost value less than K. Therefore, the computation of the live-wire segment from a selected point to the current position of the cursor uses the results of computation from the selected point to the previous position of the cursor.

The use of graph searching techniques in boundary finding has been investigated for many years now [21], [22]–[24]. Perhaps due to the lack of computer power, some of these early techniques [21] tried to simplify the problem by using heuristics that usually resulted in a graph that is deficient in some aspects. Other techniques [22], [23] put severe restrictions on the form of the contour such as, for example, that any radial line drawn outward from a point inside the object should not intersect the contour at more than one point. Several application-specific solutions have been developed [25], [26]. The work presented here together with the work reported in [6] not only brings about the use of graph searching techniques in finding optimum boundary but also demonstrates that it is possible to solve the problem in real time using a graph model that 1) does not impose any restrictions on the form, shape, or size of the objects whose boundaries are to be traced; 2) considers all possible boundaries in the entire image and is independent of the starting points selected; and 3) considers the boundary elements to be oriented and the boundaries to be oriented connected Jordan curves so that all boundaries (boundaries of even line-like structures) have a well defined inside and outside. These properties set our approach apart from previous optimum boundary detection techniques including other live-wire methods [7], [18], [19] and active contour approaches. Particularly in contrast to the latter, our aim has been to give a tighter control to the user on the segmentation process for providing whatever help is needed by the algorithm. The live-wire methods are designed for very difficult segmentation tasks. In such situations, we postulate that these methods are more efficient than the active contour approaches for the following reasons. When dealing with multiple objects with closely situated boundaries, the initial boundary specification may require careful drawing by a user. When the optimum boundary found goes astray because of noise, gaps, or other stronger boundaries nearby, postdetection correction will be required. Boundary orientedness and shortest path solutions used in live wire are able to effectively overcome these problems. Furthermore, since the user controls the detection process in live wire, there is no postdetection correction required.

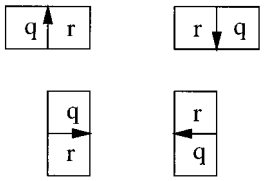


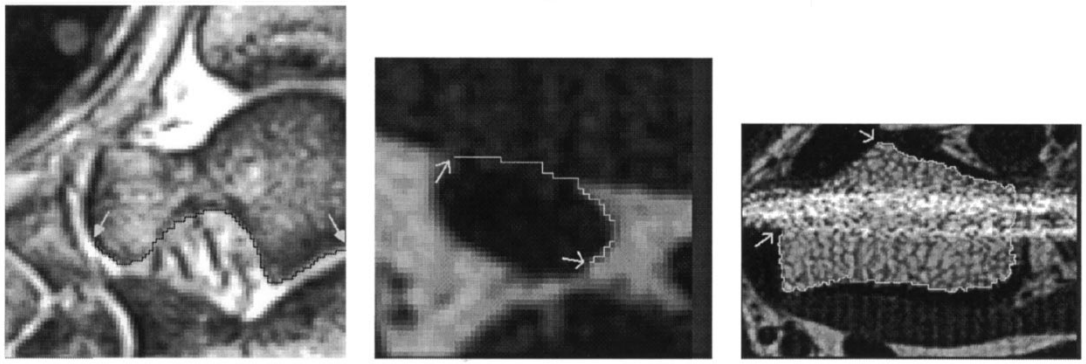
Fig. 1. A boundary element (bel) is an oriented pixel edge. The four possible types of bels in a scene are shown. The "inside" of the boundary is to the left of the bel and the "outside" is to the right.

In Section II, we present the live-wire-on-the-fly method and its algorithms. In Section III, we present the results of a careful comparison between live wire and live wire on the fly based on efficiency for segmentation in a practical application. Finally, we state our concluding remarks in Section IV.

II. Live-Wire-on-the-Fly

We start with a description of the graph model used in our live-wire approach [6]. We define a 2-D scene $\mathbf{C}$ as a pair (C, g) consisting of a finite 2-D rectangular array C of pixels and a function $g: C \rightarrow [L, H]$ that assigns to each pixel in C an intensity value lying in an interval $[L, H]$. Each oriented pixel edge in $\mathbf{C}$ is a potential boundary element b, which is called a bel for short. Every bel $b = (q, r)$ of $\mathbf{C}$ has a location and an orientation. The location of b is that of the unique edge shared by the two four-adjacent pixels q and r. We assume, without loss of generality, that its orientation is such that q is always inside the boundary, r is outside the boundary, and the inside is always to the left of b (see Fig. 1). Clearly, any bel of $\mathbf{C}$ should be in one of four orientations, as shown in Fig. 1. We associate with C a directed graph in which the vertices of the pixels represent the nodes of the graph and the oriented pixel edges represent the arcs. That is, between any two adjacent vertices in the scene, there are two arcs going in opposite directions. A 2-D boundary of interest in $\mathbf{C}$ is a closed, oriented, and connected contour made up of oriented pixel edges. To each bel b we assign a set of features whose values characterize the boundariness of b. These values are converted to a single joint cost value $c(b)$ per bel b. Thus, the problem of finding the best boundary segment (live-wire segment) between any two points (pixel vertices) specified on the boundary is translated to finding the global minimum-cost path between the corresponding two vertices of the graph. The methods of selection of features and how to convert feature values into cost values were previously addressed in [6] in detail.

Fig. 2 illustrates the power of live wire, stemming from the unique graph model, in negotiating difficult boundary segments. In our model, we assume that the boundaries have a clockwise path between the vertices indicated by the two arrows shown in each case. If the user forces live wire to trace in the opposite orientation, he/she will immediately notice that the live-wire segment will not snap onto the correct boundary, indicating the strong orientation sensitivity of the method. In Fig. 2(a), the boundary of the bone talus in an MR image of a foot is being detected. In spite of three other boundaries (of the navicular at the lower left, of the tibia at the top, and of the calcaneus at the bottom) of identical property coming close to the boundary of the talus, the attachment of ligaments in the lower part, and very uneven contrast along the boundary, the detection is not distracted. This is because in those parts where the three boundaries come close to the talus boundary, the talus boundary runs opposite to the other boundaries. The graph models used in other approaches have considered pixels themselves as the vertices in the graph. These models run into topological difficulties in imposing orientedness on



(a) (b) (c)

Fig. 2. Illustration of live-wire tracing in three different situations, all via MRI. (a) Delineating a boundary in the presence of other boundaries with similar properties. The boundary of interest is that of the bone talus in the foot. Talus is close to three other bones: the navicular at the lower left, the tibia at the top, and the calcaneus at the lower right. The boundaries traced here are assumed to have an clockwise orientation. Due to the orientedness property, the boundaries of the navicular, the tibia and the calcaneus are actually in an opposite direction in the parts where they come close to the talus boundary and therefore repel the talus boundary. (b) Jumping gaps. An MR image of a wrist, the object of interest being a vessel. (c) Passing through noisy regions. An MR image of a wrist with the internal boundary of bone being the boundary of interest. Note how the optimum boundary is not deterred by the horizontal noisy streaks present in the image. In (a)–(c), one oriented boundary segment, found as a globally optimal path between the two points indicated by the arrows, is shown.

the boundary. Consider an object region of one pixel width as an example. The definition of an oriented boundary that encloses this region is much more elegant and straightforward using pixel edges than pixels themselves as bels. Therefore, the use of pixels as bels in oriented boundary modeling and detection has not been explored. Fig. 2(b) demonstrates that the presence of a very faint signal is sufficient to complete the boundary of a vessel in an MR image of a wrist. In Fig. 2(c), the internal boundary of the cortical part of a bone in the wrist is being detected. Note that the optimum boundary is not distracted by the horizontal noisy streak. Live wire guarantees that any boundary traced by the method is a connected oriented digital Jordan curve.

The problem we wish to address in this paper is how to reduce both the time for optimum path computation and, consequently, the total user's time required for segmentation. To tackle this problem, we will exploit some known properties of graphs, particularly for the computation of shortest paths, as described in Section II-A. This leads to the algorithms presented in Section II-B.

A. Graph Properties of Shortest Paths

In the literature on shortest path algorithms [27] there are many efficient solutions for finding minimum cost paths in a weighted and directed graph. Particularly, we have adopted Dial's implementation of the Dijkstra's algorithm [28]. This algorithm computes the shortest-paths[2] to all nodes from a single node in $O(m+n\mathcal{C})$ time where m is the number of arcs, n is the number of nodes, and $\mathcal{C}$ is the maximum cost assigned to any arc in the graph. In this case, the cost assigned to each arc should be an integer in the interval $[0, \mathcal{C}]$. Dial's solution uses a circular queue with $\mathcal{C}+1$ buckets of nodes as the priority queue of the Dijkstra's algorithm. Since the bottleneck of the Dijkstra's algorithm is in maintaining the priority queue, Dial's solution uses the bucket sort algorithm to speed up this process. We will come back to this issue in Section II-B.

[2]In a weighted graph, customarily the phrase shortest-path is used to refer to the path with the minimum total weight. We shall use the phrase minimum cost path to mean the same and use the two interchangeably.

In our problem, the live-wire segment between a selected point (pixel vertex) v_s on the boundary and the point indicated by the current position v_e of the cursor in $\mathbf{C}$ is the shortest path $P = (v_s \rightsquigarrow v_e)$ from v_s to v_e in our graph where the cost of P, denoted $K(P)$, is the sum of the joint costs $c(b)$ of all bels b comprising P. In fact, the Dijkstra's algorithm returns a tree of minimum cost paths (or a tree of shortest paths) rooted at v_s [29], which consists of all minimum cost paths from v_s to all vertices in $\mathbf{C}$. We will denote this tree by $T(v_s) = \{P = (v_s \rightsquigarrow v_e)/v_e \in \mathbf{C}\}$.

For any real number k, we denote by $T_k(v_s)$ the tree of minimum cost paths rooted at v_s such that the cost of any path in this tree is less than k. That is, $T_k(v_s) = \{P = (v_s \rightsquigarrow v_e)/v_e \in \mathbf{C}, K(P) < k\}$. The algorithm reported in this paper exploits the following properties of $T(v_s)$.

1. To compute the minimum cost path $P = (v_s \rightsquigarrow v_e)$ with cost $K(P)$, there is no need to compute $T_k(v_s)$ for $k > K(P)$.
2. By the time we have found the minimum cost path $P = (v_s \rightsquigarrow v_e)$ with cost $K(P)$, we have actually found the tree of minimum cost paths $T_{K(P)}(v_s)$.
3. The tree of minimum cost paths $T_k(v_s)$ contains the tree of minimum cost paths $T_{K(P)}(v_s)$ whenever $k \geq K(P)$.

We use the first property to modify Dial's implementation of the Dijkstra's algorithm to quit optimum path computation by the time we have found the minimum cost path $P = (v_s \rightsquigarrow v_e)$. We call this algorithm DSP (see Section II-B). We use the second property to avoid optimum path computation for any path $P' = (v_s \rightsquigarrow v_e')$ with cost $K(P') < K(P)$. Thus, when the user moves the cursor to a new position v_e' such that $K(P') < K(P)$, and we have already found P, the algorithm just displays $P' = (v_s \rightsquigarrow v_e')$ without requiring its computation. We use the third property to continue optimum path computation for paths $P' = (v_s \rightsquigarrow v_e')$ with costs $K(P') \geq K(P)$ based on the previous result of algorithm DSP for computing P. For an illustration of this idea, Fig. 3 shows two regions R_1 and R_2 in $\mathbf{C}$. Region R_1 contains all minimum cost paths from v_s with costs less than $K(P)$ as a result of the previous computation for finding $P = (v_s \rightsquigarrow v_e)$. When the user moves the cursor to a new position v_e', he/she defines region R_2 and all minimum cost paths with costs K' such that $K(P) \leq K' < K(P')$ will end at a vertex v in R_2.

Thus, to find $P' = (v_s \rightsquigarrow v'_e)$, we must consider only vertices in R_2. Our new interactive algorithm with all these characteristics is presented next.

B. Algorithms

Algorithm LWOF

Input: The joint cost function c and an initial vertex v_0 selected on a 2-D boundary of interest in $\mathbf{C}$, and interactive input as described in the *repeat-until* loop below.

Output: A closed, connected, and oriented contour (made up of bels).

Auxiliary Data Structures: A 2-D cumulative cost array cc representing the total cost of the optimal paths found so far from v_s to the vertices in $\mathbf{C}$; a 2-D direction array dir indicating, for each vertex, to which of its immediate neighboring vertices the optimal path goes; a circular queue Q of vertices with $\mathcal{C}+1$ buckets; a list L of vertices which have already been processed; a current path $P(v_s \rightsquigarrow v_e)$, where v_s is the current point selected on the boundary and v_e is the current position of the cursor in $\mathbf{C}$; and a list $\mathbf{B}$ of bels which have already been identified as belonging to the boundary of interest in $\mathbf{C}$;

begin

1. set $cc(v)$ to ∞ and $dir(v)$ to *null* for all vertices v in $\mathbf{C}$, and set L and $\mathbf{B}$ to empty;
2. $v_s \leftarrow v_0$, set $cc(v_s)$ to 0, and put v_s in Q;

repeat

a. determine the vertex v_e in $\mathbf{C}$ currently pointed to by the cursor;

b. *if* v_e is not a vertex of any bel in $\mathbf{B}$ *then*

(i) compute $P \leftarrow$DSP$(v_s, v_e, Q, cc, c, dir, L)$ and display the bels in P;

(ii) *if* v_e is the vertex selected by the user *then*

a. add the bels in P to $\mathbf{B}$;

b. set $cc(v)$ to ∞ and $dir(v)$ to *null* for all vertices v in $\mathbf{C}$;

c. remove all vertices v from Q, and remove from L all vertices v which do not belong to any bel in $\mathbf{B}$;

d. $v_s \leftarrow v_e$, set $cc(v_s)$ to 0, remove v_s from L, and put v_s in Q;

endif;

endif;

until the user indicates a close operation;

4. $v_e \leftarrow v_0$ and remove v_e from L;
5. compute $P \leftarrow$DSP$(v_s, v_e, Q, cc, c, dir, L)$ and display the bels in P;
6. add the bels in P to $\mathbf{B}$ and output the bels in $\mathbf{B}$;

end

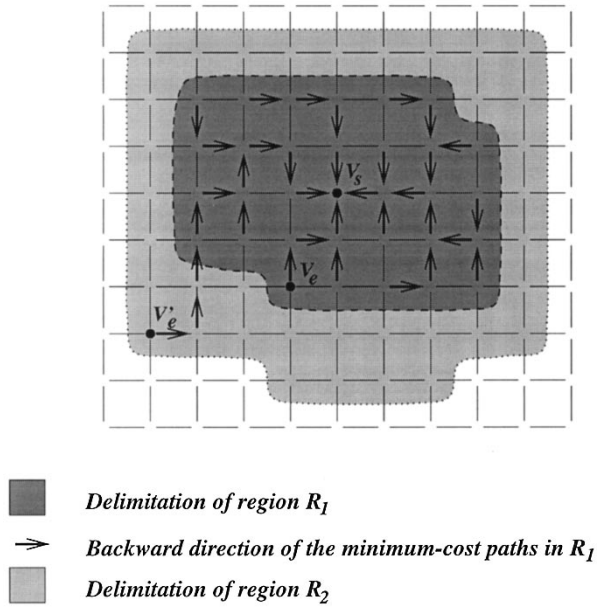


Fig. 3. Minimum cost path computation in R_2 based on the previous computation of all minimum cost paths in R_1.

Algorithm DSP

Input: an initial vertex v_s; a terminal vertex v_e; the circular queue Q; the cumulative cost array cc; the joint cost function c; the direction array dir; and the list L of already processed vertices;

Output: A set of bels forming an optimal path from v_s to v_e;

begin

1. *while* $v_e \notin L$ *do*

a. remove a vertex v from Q such that $cc(v) = \min_{v' \in Q}\{cc(v')\}$, and put v in L;

b. *for* each vertex v' such that v' is in the set of the four-adjacent neighbors of v and $v' \notin L$ *do*

(i) compute $cc_{tmp} = cc(v) + c(b')$ where b' is the bel whose direction goes from v' to v and $c(b')$ is the joint cost of b';

(ii) *if* $cc_{tmp} < cc(v')$ *then*

a. set $cc(v')$ to cc_{tmp} and $dir(v')$ to the direction from v' to v;

b. *if* $v' \notin Q$ *then* insert v' in Q *else* update v' in Q;

endif;

endfor;

endwhile;

starting from v_e, trace recursively the next vertex pointed to by the current vertex using the direction information in dir until v_s is reached, and return the bels so traced;

end

In the algorithms above, Q is a bucket represented by a circular vector with $\mathcal{C}+1$ positions from 0 to $\mathcal{C}$ (see Fig. 4). Each position

i, $i = 0, \cdots, \mathcal{C}$ has associated with it a doubly linked list which contains vertices with the same cumulative cost value. The doubly linked lists in Q are represented by an array A of pointers such that each possible vertex v in $\mathbf{C}$ has its corresponding position in A and, at this position, we indicate the next and the previous vertices from v in Q. This data structure is shown in Fig. 5. In Step 3.b.(ii).c. of Algorithm LWOF, we remove all vertices v from Q in $O(\mathcal{C})$ time since we only have to set to *null* the list associated with each position $i, i = 0, \cdots, \mathcal{C}$ in Q. An index i_0 is used to indicate the current initial position in Q (see Fig. 4). In Step 1.a. of Algorithm DSP, a vertex v in Q with the minimum cumulative cost $cc(v)$ is removed from the beginning of the doubly linked list at position i_0. If this list is empty, i_0 is incremented until the next position in which a nonempty list is found. Taking the worst case, this operation has a computational time complexity of $O(\mathcal{C})$. In Step 1.b.(ii).b. of Algorithm DSP, a vertex v' with cumulative cost $cc(v')$ is inserted in Q at the beginning of the doubly linked list at position $[cc(v') \bmod (\mathcal{C}+1)]$. This operation has a computational time complexity of $O(1)$. The difference between the maximum and the minimum cumulative costs of the vertices in Q is always less than or equal to $\mathcal{C}$. To see why this is so, consider position zero in Q to store vertices with cumulative costs $\mathcal{C}+1$ or its multiples. Since the cost assigned to any edge is in $[0, \mathcal{C}]$, a vertex with cumulative cost $\mathcal{C}+1$ can be reached only from vertices with cumulative costs in $[1, \mathcal{C}+1]$. That is, by the time a vertex with cumulative cost $\mathcal{C}+1$ is inserted at position zero, all vertices with cumulative cost zero will have already been removed from this position. Therefore, position zero will never store vertices with different cumulative costs at the same time. This observation is valid for any position in Q. In the same step, a vertex v' in Q may have its cumulative cost updated, meaning that we have found a new path from v_s to v' with a cost less than the current cost $cc(v')$. In this case, we have to remove v' from its current position in Q and insert it into a new position in Q. This process is done with a computational time complexity of $O(1)$.

In the worst case, algorithm DSP has the same computational time complexity $O(m + n\mathcal{C})$ as in the Dial's implementation of Dijkstra's algorithm, where m is the number of bels in $\mathbf{C}$, n is the number of vertices in $\mathbf{C}$, and $\mathcal{C}$ is the maximum cost $c(b)$ assigned to any bel b. Other shortest path algorithms exist with computational time complexity less than $O(m+n\mathcal{C})$ (e.g., $O(m+n \log \mathcal{C})$, $O(m+n\sqrt{\log \mathcal{C}})$, and $O(m \log\log \mathcal{C})$, see [28]). These algorithms use more complex data structures than our circular queue to reduce the time complexity for inserting and removing vertices. In our implementation, we have a time complexity of $O(1)$ for inserting and updating vertices in Q. In the worst case, we have a time complexity of $O(\mathcal{C})$ for removing a vertex from Q with minimum cumulative cost, as opposed to a logarithmic complexity obtained by these algorithms. After some experimentation, we have found that the number of increments to reach the next nonempty position in Q is usually less than 1% of $\mathcal{C}$. Actually, even $\mathcal{C}$ is not a big number. Typically, we have used 4095 and 255 for $\mathcal{C}$ in our implementation of live wire. Therefore, it is not clear that the speed improvement in live wire with other algorithms is really significant. This, however, should be investigated further.

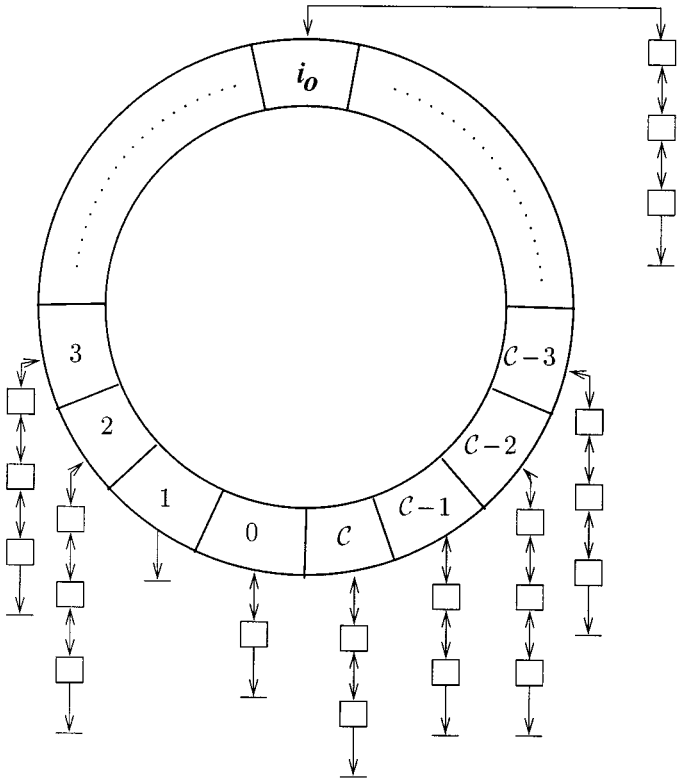


Fig. 4. Bucket structure in a circular queue. Each position in the queue has a doubly linked list associated with it which contains vertices with the same cc value.

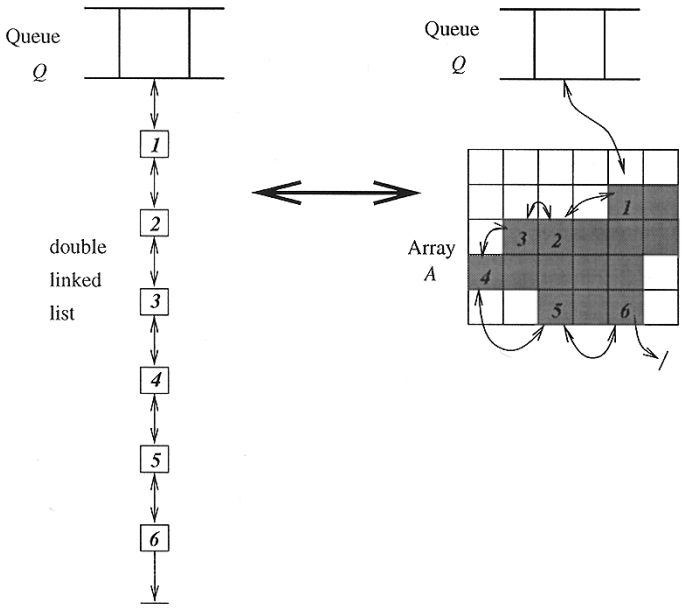


Fig. 5. The vertices with the same cc value in Q stored in a doubly linked list are represented by a pointer array A.

III. Evaluation

In [6], we suggested that the goodness of a segmentation method be assessed based on three factors—precision, accuracy, and efficiency. Precision refers to the repeatability of the method and can be measured by evaluating the variations in the result of segmentation because of subjective operator input. Accuracy refers to the degree of agreement with truth. Efficiency refers to the practical viability of the method expressed as some function of the total user's time required to complete the segmentation process. Based on 2000 tracings in a particular application and statistical analysis of the results, we have shown that the segmentations of the 2-D live-wire method, in general, agree with those of manual tracing (accuracy) and that the live-wire method is more repeatable (precision), with a statistical significance level of $p < 0.03$, and 1.5–2.5 times faster (efficiency), with a statistical significance level of $p < 0.02$, than the manual method. In this section, we will show

the results of comparing live wire and live wire on the fly, taking into account the efficiency of the methods. Since the delineation of the contours output by live wire on the fly is exactly in the same way as in live wire, the accuracy and precision of the former will be identical to those of the latter, and, therefore, they need not be assessed again.

In [6], we have introduced a feature called f_8 in live wire to constrain the search for optimal paths in the current slice to an annular region (shell) of width W centered around the projection onto the current slice of the contour(s) traced in the previous slice. With feature f_8, live wire yields a fast response even for large images. Of course, we can also use f_8 to further improve the efficiency of live wire on the fly on large images, but we will consider in this section a comparison between live wire with f_8 and live wire on the fly. Therefore, our experiments will take into account three methods.

- LW: live wire without f_8.
- LWF8: live wire with f_8 using $W = 60$ pixels. This value of W was determined to be adequate in [6].
- LWOF: live wire on the fly.

For our experiments, we have chosen one object [the talus bone of the human foot, see Fig. 2(a)] in one of our ongoing applications, the kinematic analysis of the tarsal joints of the foot based on MR images [10], [11], [12]. This was one of the objects used in the past to evaluate the previous live methods [6], [17]. The segmentation task here is difficult because of the lack of MR signal from bone, the presence of adjoining bone boundaries, and the tendons and ligaments that attach to the bones. We created a set of 67 2-D scenes from the images within our database as follows. The images (slices) in our database are all of size 256 × 256 pixels. We chose a set, denoted $\mathbf{C}_{256}$, of 30 slices from the database pertaining to the data set of one subject. By bilinear interpolation of each of these slices we created another set, denoted $\mathbf{C}_{128}$, of 30 128 × 128 slices. Analogously, we created a set $\mathbf{C}_{512}$ of five 512 × 512 slices and a set $\mathbf{C}_{1024}$ of two 1024 × 1024 slices from the original 256 × 256 slices. In all cases, we kept the size of the object region relative to the scene size roughly the same. That is, the object occupied roughly the same extent of the scene. The reason for choosing a fewer number of scenes of size 512 × 512 and 1024× 1024 is that the response time of LW in these scenes is prohibitively slow and conducting experiments on 30 slices in each of these cases would take too much time. Each of three operators O_1, O_2 and O_3 segmented the talus in each of these 67 scenes using each of the two methods LW and LWOF. They also segmented the talus in $\mathbf{C}_{256}$ using LWF8. Our evaluation study thus consists of 492 segmentation experiments in total. We used a 300-MHz Pentium PC for these experiments.

We denote the time taken to complete any segmentation experiment e by T_e (expressed in seconds). Consider any fixed $o \in \{O_1, O_2, O_3\}$, the scene type $t \in \{\mathbf{C}_{128}, \mathbf{C}_{256}, \mathbf{C}_{512}, \mathbf{C}_{1024}\}$, and method $m \in \{\text{LW, LWOF, LWF8}\}$. We define the time taken T_{otm} (in seconds/slice) for operator o to segment the talus in a 2-D scene of type t using method m to be the average of all times T_e over all segmentation experiments e involving o and m and all 2-D scenes of type t. We define the time taken T_{tm} (in seconds/slice) to segment the talus in a 2-D scene of type t using method m to be the average of all times T_{otm} over all operators involving m and all 2-D scenes of type t.

We have done three types of timing measurements. The first type measures the CPU times for computing the live wire segments independent of other supporting processes that are required to conduct live wire segmentation. This allows us to assess the difference in speed that exists purely between the old and the new algorithms. The second type measures the time taken by the user to segment one complete contour ignoring the time for other processes such as displaying the slice and the computation of the cost values $c(b)$ for all bels. The third type includes all processes, and, therefore, gives an idea of the comparative user time required for overall segmentation for the different methods in an actual application. We note here that, as in the live-wire method [6], training is required only once for an application and is not needed on a per study basis. This is typically under 5 min and is not included in any of the time measurements.

TABLE I
SEGMENTATION TIMES T_{otm} IN SECONDS/SLICE FOR ALL POSSIBLE VALUES OF o, t, AND m. THIS TABLE LISTS THE FIRST TYPE OF MEASUREMENT THAT INDICATES THE TIME TAKEN BY THE SHORTEST-PATH ALGORITHMS ONLY INDEPENDENT OF OTHER PROCESSES

		$\mathbf{C}_{128}$	$\mathbf{C}_{256}$	$\mathbf{C}_{512}$	$\mathbf{C}_{1024}$
LW	O_1	2.14	15.17	99.57	901.24
	O_2	2.86	19.13	120.44	1257.64
	O_3	2.06	13.62	72.63	546.90
LWOF	O_1	0.23	0.62	2.27	8.74
	O_2	0.25	0.66	1.56	6.07
	O_3	0.19	0.51	1.63	7.84
LWF8	O_1	–	8.25	–	–
	O_2	–	9.66	–	–
	O_3	–	6.86	–	–

TABLE II
SEGMENTATION TIMES T_{otm} IN SECONDS/SLICE FOR ALL POSSIBLE VALUES OF o, t, AND m. THIS TABLE LISTS THE SECOND TYPE OF MEASUREMENT THAT INDICATES THE TIME TAKEN BY THE USER TO SEGMENT ONE COMPLETE CONTOUR IGNORING THE TIME FOR OTHER PROCESSES

		$\mathbf{C}_{128}$	$\mathbf{C}_{256}$	$\mathbf{C}_{512}$	$\mathbf{C}_{1024}$
LW	O_1	8.37	20.93	116.20	959.00
	O_2	16.67	36.07	142.40	1312.50
	O_3	8.13	18.97	81.40	572.50
LWOF	O_1	5.67	5.33	8.00	15.50
	O_2	11.53	9.87	14.00	19.50
	O_3	4.97	5.33	6.20	16.00
LWF8	O_1	–	14.13	–	–
	O_2	–	23.50	–	–
	O_3	–	10.77	–	–

Tables I, II, and III list the values of T_{otm} for all possible values of o, t, and m for the three types of measures. Tables IV, V, and VI list the values of T_{tm} for all possible values of t and m corresponding to Tables I, II, and III, respectively. Tables I, II, and III show that operators O_1 and O_3 can finish segmentation quicker than operator O_2. The reason is that, as they become more familiar with the behavior of the algorithms and with the boundary under segmentation, they can react faster and select less points on the boundary. O_2 has considerably less experience with live wire than O_1 and O_3. In Table IV, we see that $LWOF$ can find optimum paths about a hundred twenty times faster than LW. However, Tables V and VI show that users cannot react with the same speed. Table VI shows that, from the point of view of actual segmentation, LWOF is about 1.3–31 times faster than LW for images from 128 × 128 pixels to 1024 × 1024 pixels. Even constraining optimum path computation into an annular region of width equal to 60 pixels (i.e., method LWF8), live wire on the fly is about 1.8 times faster. Note that, the advantage of live wire on the fly over live wire increases with the size of the image and with the number of points required per boundary. In our experiments, the 2-D boundaries of the talus require typically 2–5 points for segmentation.

IV. CONCLUDING REMARKS

We have presented a new user steered image segmentation paradigm, called live wire on the fly, to segment object boundaries in a slice-by-slice fashion. The method uses the previously published live wire framework [6], but utilizes a substantially faster shortest path algorithm for improving speed. Based on 492 segmentation experiments from an actual medical application, we have shown that the new method

TABLE III
SEGMENTATION TIMES T_{otm} IN SECONDS/SLICE FOR ALL POSSIBLE VALUES OF o, t, AND m. THIS TABLE LISTS THE THIRD TYPE OF MEASUREMENT THAT INDICATES THE TIME TAKEN BY THE USER FOR OVERALL SEGMENTATION INCLUDING ALL PROCESSES

		C_{128}	C_{256}	C_{512}	C_{1024}
LW	O_1	12	24	120	990
	O_2	24	44	168	1336
	O_3	11	21	82	580
LWOF	O_1	8	8	12	30
	O_2	21	16	24	35
	O_3	8	8	11	27
LWF8	O_1	–	18	–	–
	O_2	–	28	–	–
	O_3	–	13	–	–

TABLE IV
SEGMENTATION TIMES T_{tm} IN SECONDS/SLICE FOR ALL POSSIBLE VALUES OF t, AND m DERIVED FROM TABLE I

	C_{128}	C_{256}	C_{512}	C_{1024}
LW	2.35	15.98	97.55	901.93
LWOF	0.22	0.60	1.82	7.55
LWF8	–	8.26	–	–

TABLE V
SEGMENTATION TIMES T_{tm} IN SECONDS/SLICE FOR ALL POSSIBLE VALUES OF t, AND m DERIVED FROM TABLE II

	C_{128}	C_{256}	C_{512}	C_{1024}
LW	11.06	25.32	113.33	948.00
LWOF	7.39	6.84	9.40	17.00
LWF8	–	16.13	–	–

TABLE VI
SEGMENTATION TIMES T_{tm} IN SECONDS/SLICE FOR ALL POSSIBLE VALUES OF t AND m DERIVED FROM TABLE III

	C_{128}	C_{256}	C_{512}	C_{1024}
LW	16	30	123	969
LWOF	12	11	16	31
LWF8	–	20	–	–

is about 1.3–31 times faster than live wire for actual segmentation, although the pure computational part alone is about a hundred twenty times faster. For the typically encountered 256 × 256 images, the speed advantage is about 2.7.

The efficiency of the methods, as shown in Tables II and V, is reduced when we consider the results of overall segmentation, as shown in Tables III and VI. The reason for this is the additional time taken to compute edge costs $c(b)$ for every slice. This can be improved if the edge costs are precomputed for all slices and stored before segmentation. For a typical medical image data set of 50 256 × 256 slices, for example, considering 2 bytes per pixel and all four orientations for the bels of our graph, the storage of edge costs would require about 26 MB. This would bring the total RAM requirement to 96 MB which is certainly reasonable for modern PC's and workstations.

A drawback of live wire is the computation time for all possible minimum cost paths from each selected point on the boundary to all other points in the image. This time increases with the size of the image compromising the interactivity of the method in some practical situations. Tables I–VI show that live wire loses efficiency considerably for images larger than 128 × 128 pixels. Our previous solutions for this problem involved some restrictions such as to constrain live wire into a shell around the boundary (i.e., method $LWF8$) and the live lane method [6]. The former is dependent on object shape and topology. It runs into difficulty when the location and shape of contours change rapidly from slice to slice, especially when contours split and merge. Live lane requires substantially more user involvement during segmentation. LWOF offers a solution without any restrictions. Further, it is less dependent on image size and computer power. It computes and displays live-wire segments in real time, even for large images, even on low-powered computers. Combined with the application-specific training facility provided for effectively assigning costs to bels (described in detail in [6]), live wire on the fly offers a general very efficient practical solution to image segmentation for applications wherein more automatic solutions cannot be devised easily.

REFERENCES

[1] R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Comput. Vision Graph. Image Processing*, vol. 29, no. 1, pp. 100–132, Jan. 1985.

[2] S. D. Olabarriaga and A. W. M. Smeulders, "Setting the mind for intelligent interactive segmentation: Overview, requirements, and framework," *Lecture Notes Comput. Sci.*, vol. 1230, pp. 417–422, 1997.

[3] W. E. Higgins and E. J. Ojard, "Interactive morphological watershed analysis for 3-D medical images," *Computerized Med. Imag. Graph.*, vol. 17, no. 4/5, pp. 387–395, 1993.

[4] R. Adams and L. Bischof, "Seeded region growing," *IEEE Trans. Pattern Analy. Machine Intell.*, vol. 16, pp. 641–647, June 1994.

[5] T. McInerney and D. Terzopoulos, "Deformable models in medical image analysis: A survey," *Med. Image Anal.*, vol. 1, no. 2, pp. 91–108, 1996.

[6] A. X. Falcão, J. K. Udupa, S. Samarasekera, S. Sharma, B. E. Hirsch, and R. A. Lotufo, "User-steered image segmentation paradigms: Live-wire and live-lane," *Graphical Models Image Processing*, vol. 60, no. 4, pp. 233–260, July 1998.

[7] E. N. Mortensen and W. A. Barrett, "Interactive segmentation with intelligent scissors," *Graphical Models Image Processing*, vol. 60, no. 5, pp. 349–384, Sept. 1998.

[8] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Comput. Vision*, vol. 1, no. 4, pp. 321–331, 1987.

[9] A. X. Falcão, J. K. Udupa, S. Samarasekera, and B. E. Hirsch, "User-steered image boundary segmentation," in *SPIE Proc. Medical Imaging 1996*, vol. 2710, Newport Beach, CA, February 1996, pp. 278–288.

[10] J. K. Udupa, B. E. Hirsch, S. Samarasekera, H. Hillstrom, G. Bauer, and B. Kneel, "Analysis of in vivo 3-D internal kinematics of the joints of the foot," *IEEE Trans. Biomed. Eng.*, vol. 45, pp. 1387–1396, Nov. 1998 .

[11] B. E. Hirsch, J. K. Udupa, and S. Samarasekera, "A new method of studying joint kinematics from 3-D reconstructions of MRI data," *J. Amer. Podiatric Med. Assoc.*, vol. 86, no. 1, pp. 4–15, 1996.

[12] E. Stindel, J. K. Udupa, B. E. Hirsch, D. Odhner, and C. Couture, "3-D MR image analysis of the morphology of the rear foot: Application to classification of bones," *Computerized Med. Imag. Graph.*, vol. 23, pp. 75–83, 1999.

[13] R. C. Rhoad, J. J. Klimkiewicz, G. R. Williams, S. B. Kesmodel, J. K. Udupa, B. Kneeland, and J. P. Iannotti, "A new in vivo technique for 3-D shoulder kinematics analysis," *Skeletal Radiol.*, vol. 27, pp. 92–97, 1998.

[14] B. S. Morse, W. A. Barrett, J. K. Udupa, and R. P. Burton, "Trainable optimal boundary finding using two-dimensional dynamic programming," Medical Image Processing Group, Dept. Radiology, Univ. Pennsylvania, Tech. Rep. MIPG180, March 1991.

[15] J. K. Udupa, S. Samarasekera, and W. A. Barrett, "Boundary detection via dynamic programming," *SPIE Proc. Medical Imaging 1992*, vol. 1808, pp. 33–39, 1992.

[16] E. N. Mortensen, B. S. Morse, W. A. Barrett, and J. K. Udupa, "Adaptive boundary detection using live-wire two dimensional dynamic programming," *Comput. Cardiol.*, pp. 635–638, October 1992.

[17] A. X. Falcão and J. K. Udupa, "Segmentation of 3-D objects using live-wire," in *SPIE Proc. Medical Imaging 1997*, vol. 3034, Newport Beach, CA, February 1997, pp. 228–239.
[18] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition," in *Proc. Computer Graphics SIGGRAPH'95*, Los Angeles, CA, August 1995, pp. 191–198.
[19] W. A. Barrett and E. N. Mortensen, "Fast, accurate, and reproducible live-wire boundary extraction," *Visualization Biomed. Comput.*, pp. 183–192, September 1996.
[20] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1991.
[21] A. Martelli, "An application of heuristic search methods to edge and contour detection," *Commun. ACM*, vol. 19, pp. 73–83, 1971.
[22] U. Montanari, "On the optimal detection of curves in noisy pictures," *Commun. ACM*, vol. 14, no. 5, pp. 335–345, 1971.
[23] D. Pope, D. Parker, D. Gustafson, and P. Clayton, "Dynamic search algorithms in left ventricular border recognition and analysis of coronary arteries," *Comput. Cardiol.*, vol. 9, pp. 71–75, 1984.
[24] M. A. Fischler, J. A. Tenenbaum, and H. C. Wolf, "Detection of roads and linear structure in low resolution aerial imagery using a multi-source knowledge integration technique," *Comput. Graph. Image Processing*, vol. 15, no. 3, pp. 201–223, 1981.
[25] M. Sonka, X. Zhang, S. C. DeJong, S. M. Collins, and C. R. McKay, "Segmentation of intravascular ultrasound images: A knowledge-based approach," *IEEE Trans. Med. Imag.*, vol. 14, pp. 719–732, Aug. 1995.
[26] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 2nd ed. Pacific Grove, CA: PWS, 1998.
[27] N. Deo and C. Pang, "Shortest-path algorithms: Taxonomy and annotation," *Networks*, vol. 14, pp. 275–323, 1984.
[28] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[29] R. E. Tarjan, *Data Structures and Network Algorithms*, 6th ed. Philadelphia, PA: SIAM, 1996.

Computer-Aided Stereotactic Functional Neurosurgery Enhanced by the Use of the Multiple Brain Atlas Database

Wieslaw. L. Nowinski*, Guo Liang Yang, and Tseng Tsai Yeo

***Abstract*—This paper introduces a computer-aided atlas-based functional neurosurgery methodology and describes NeuroPlanner, a software system which supports it. NeuroPlanner provides four groups of functions: 1) data-related for data reading, interpolation, reformatting, and image processing; 2) atlas-related for multiple atlases reading, atlas-to-data global and local registrations, two-way anatomical indexing, and multiple labeling in two and three dimensions; 3) atlas-data exploration-related for three-dimensional (3-D) display and real-time manipulation of cerebral structures, continuous navigation, two-dimensional (2-D), triplanar, 3-D presentations, and 2-D interaction in four views; and 4) neurosurgery-related for targeting, trajectory planning, mensuration, simulating the insertion of microelectrode, and simulating therapeutic lesioning. All operations, excluding atlas and data reading, are real time. The combined anatomical index of the multiple brain atlas database containing complementary 2-D and 3-D atlases has about 1000 structures per hemisphere, and over 400 sulcal patterns.**

Neurosurgical planning with mutually preregistered multiple brain atlases in all three orthogonal orientations is novel. The approach is validated with 24 intraoperative and postoperative datasets for thalamotomies, thalamic stimulations, pallidotomies, and pallidal stimulations.

Its potential benefits include increased accuracy of target definition, reduced time of the surgical procedure by decreasing the number of tracts, facilitated planning of more sophisticated trajectories, lowered cost by reducing the number of microelectrodes used, reduced surgical complications, and the extra degree of confidence given to the neurosurgeon.

***Index Terms*—Brain atlases, MRI, pallidotomy, Schaltenbrand–Wahren atlas, stereotactic functional neurosurgery, Talairach–Tournoux atlas, thalamotomy.**

I. Introduction

We propose to enhance computer-aided stereotactic functional neurosurgery by the simultaneous use of multiple complementary mutually preregistered brain atlases in multiple orientations, along with a suitable environment and tools. For this purpose we have developed NeuroPlanner, a software system providing a rich set of functions, such as multiple atlas and actual patient's data loading; interactive three-dimensional (3-D) atlas-to-data registration; atlas-based anatomical targeting; stereotactic trajectory planning; simulation of the insertion of the microelectrode; simulation of therapeutic lesioning; mensuration; 3-D display and real-time manipulation of cerebral structures and other objects such as the stereotactic trajectory and simulated therapeutic lesions; continuous navigation in the multiple atlas-data space; two-dimensional (2-D), triplanar, and 3-D presentations; two-way anatomical indexing; multiple labeling in 2-D and 3-D; volume interpolation and reformatting; and file handling. All operations (excluding atlas and data loading) are real time. The combined anatomical index of the 2-D and 3-D atlases contains approximately 1000 structures per hemisphere and over 400 sulcal patterns.

The actual patient's data, read as DICOM MRI images, are converted into NeuroPlanner's internal format, interpolated, and their

Manuscript received December 1, 1998; revised November 13, 1999. The Associate Editor responsible for coordinating the review of this paper and recommending its publication was M. Vannier. *Asterisk indicates corresponding author*.

*W. L. Nowinski and G. L. Yang are with the Biomedical Laboratory, Kent Ridge Digital Laboratories, 21 Heng Mui Keng Terrace, Singapore 119613.

T. T. Yeo is with the Department of Neurosurgery, Tan Tock Seng Hospital, Singapore 308433.

Publisher Item Identifier S 0278–0062(00)01694-3.