GRAPHICAL MODELS AND IMAGE PROCESSING **60,** 233–260 (1998)
ARTICLE NO. IP980475

User-Steered Image Segmentation Paradigms: Live Wire and Live Lane

Alexandre X. Falcão, Jayaram K. Udupa, Supun Samarasekera, and Shoba Sharma

Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, Pennsylvania 19104-6021

Bruce Elliot Hirsch

Department of Biomedical Sciences, Pennsylvania College of Podiatric Medicine, Philadelphia, Pennsylvania

and

Roberto de A. Lotufo

Faculty of Electrical Engineering, State University of Campinas, Campinas, SP, Brazil

Received January 18, 1996; revised March 16, 1998; accepted March 26, 1998

In multidimensional image analysis, there are, and will continue to be, situations wherein automatic image segmentation methods fail, calling for considerable user assistance in the process. The main goals of segmentation research for such situations ought to be (i) to provide *effective control* to the user on the segmentation process *while* it is being executed, and (ii) to minimize the total user's time required in the process. With these goals in mind, we present in this paper two paradigms, referred to as *live wire* and *live lane*, for practical image segmentation in large applications. For both approaches, we think of the pixel vertices and oriented edges as forming a graph, assign a set of features to each oriented edge to characterize its "boundariness," and transform feature values to costs. We provide training facilities and automatic optimal feature and transform selection methods so that these assignments can be made with consistent effectiveness in any application. In live wire, the user first selects an initial point on the boundary. For any subsequent point indicated by the cursor, an optimal path from the initial point to the current point is found and displayed in real time. The user thus has a live wire on hand which is moved by moving the cursor. If the cursor goes close to the boundary, the live wire snaps onto the boundary. At this point, if the live wire describes the boundary appropriately, the user deposits the cursor which now becomes the new starting point and the process continues. A few points (live-wire segments) are usually adequate to segment the whole 2D boundary. In live lane, the user selects only the initial point. Subsequent points are selected automatically as the cursor is moved within a lane surrounding the boundary whose width changes

1077-3169/98 $25.00

as a function of the speed and acceleration of cursor motion. Live-wire segments are generated and displayed in real time between successive points. The users get the feeling that the curve snaps onto the boundary as and while they roughly mark in the vicinity of the boundary.

We describe formal evaluation studies to compare the utility of the new methods with that of manual tracing based on speed and repeatability of tracing and on data taken from a large ongoing application. The studies indicate that the new methods are statistically significantly more repeatable and 1.5–2.5 times faster than manual tracing.

Key Words: image segmentation; interactive segmentation; boundary detection; medical imaging; visualization; evaluation of segmentation.

1. INTRODUCTION

In many areas of science and engineering, multidimensional image data are generated via an imaging device or a simulation process that captures information about certain physical phenomena. In medical imaging, for example, tomographic scanners routinely produce two-, three-, and four-dimensional digital image data pertaining to human internal structures. The physical phenomenon in this instance may be the anatomical form or shape of a static or dynamic structure system. It may often also be some physiologic function of the structure system. There are certain "objects" in these images (for example, the anatomical organs) whose form, shape, and functional process need to be visualized, manipulated, and analyzed [1] to fulfill the main objectives of imaging the phenomenon. The most fundamental requirement for the successful visualization, manipulation, and analysis of such objects is a method for the repeatable, accurate, and efficient extraction of object information from given multidimensional images. This process is commonly referred to as *image segmentation.*

Approaches to image segmentation may be broadly classified into two groups: automatic and interactive. Automatic methods [2–8] avoid user intervention, but their complete success cannot always be guaranteed. Interactive methods [9–15] range from totally manual painting of object regions or drawing of object boundaries to the detection of object region/boundaries with minimal user assistance. While automatic methods are currently in use in an application-specific and tailored fashion, they fail when new situations due to different imaging modality protocol or object type are presented. To make them work effectively in a repeated fashion on a large number of data sets often requires considerable research and development. Consequently, in such situations, interactive (mostly manual) methods are used. There are, and will continue to be, situations wherein available techniques fail or require considerable further development and so manual segmentation will be the only immediate alternative. From this consideration, interactive methods which provide complete control to the user are more attractive since they are always guaranteed to work, albeit possibly becoming time consuming in some instances. This naturally leads us to conclude that two of the main goals of research in interactive segmentation methods ought to be (i) to provide as *complete* a *control* as possible to the user on the segmentation process *while* it is being executed and (ii) to minimize the user involvement and the total user's time required for segmentation.

The aspect of complete and tight user control while the process is underway is crucial for ensuring accuracy as determined by the expert user. The minimization of the user's time

requirement is obviously paramount to making the segmentation method practically viable. These goals and the attendant issues seem to have not caught much attention in the past in interactive segmentation research. Deformable boundary approaches are actively pursued currently [10, 13–17], and numerous publications have resulted during the past five years on this topic. 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 situations, a more active and tighter control provided to the user on the process of segmentation would significantly reduce the time spent by him/her in the process. The methods presented in this paper are designed with the aforementioned goals in mind. They operate slice-by-slice in their present form and permit segmentation of object boundaries of the type shown in this paper in a few seconds of user time per slice.

The goal of provision of tight control is motivated by the following consideration. The entire segmentation process can be thought of as consisting of two tasks: *recognition* and *delineation*. Recognition consists of determining roughly "where" the object (boundary) is and distinguishing it from other object(boundary)-like entities. Delineation consists of precisely defining the spatial extent of the object region/boundary in the image. Human operators (application experts) usually outperform computer algorithms in most recognition tasks. The inability to translate relevant global object-related knowledge needed for recognition to computable local operations has been one of the major obstacles to automatic image segmentation. On the other hand, computer algorithms usually outperform human operators in the delineation task. Simple user assistance is often sufficient to complete the segmentation process. Attempts to avoid this with the aim of total automation may call for considerable research and development without producing a tangible and fool-proof practical solution at the end. An effective strategy therefore appears to be to actively exploit the synergy between the human operator and the computer algorithms during segmentation. This strategy forms the basis of the segmentation methods presented here.

We present two paradigms, called *live wire* and *live lane*, both are boundary-based and consistent with the motivation described above. For both approaches, we think of the pixel array as constituting a directed graph with the vertices of the pixels representing the vertices of the graph and the oriented pixel edges representing the arcs. To each oriented pixel edge, we assign a set of features whose values characterize the "boundariness" of the oriented edge. These values are converted to a single cost value per oriented edge. The problem of finding the best boundary segment (as a sequence of oriented pixel edges) between any two points (pixel vertices) specified on the boundary is translated to finding the minimum-cost path between the two vertices and solved via dynamic programming. The entire 2D boundary is identified as a set of consecutive boundary segments, each specified and detected in this fashion. The two approaches differ in the mechanism by which the successive points are specified and in the underlying process of human–machine interaction.

In the live-wire approach, described in Section 2, the user initially specifies a point on the boundary using the cursor of the pointing device of the machine. For any subsequent position of the cursor, as the pointing device is moved, a curve—a globally optimal path—connecting the initial point to the current cursor position is computed and displayed in real time. As the current cursor position moves close to the boundary, the live wire snaps onto the boundary. If the live-wire segment adequately describes the boundary, the user deposits the cursor whose location now becomes the new starting point. A complete 2D boundary is specified via a set of live-wire segments in this fashion.

In the live-lane approach, described in Section 3, the user involvement is even more active and more tightly integrated with the machine's actions. To extract a boundary in a given slice, the user selects only an initial point and subsequently steers the cursor in the vicinity of the boundary within a lane of certain width (hence the name "live lane"). During the user's drawing action, points are selected intermittently by the computer automatically and live-wire segments between successively selected points are computed and displayed in real time. The lane width is varied adaptively along the boundary, making it smaller in weak and uncertain regions and larger in regions with strong boundaries. Again the user is the best judge as to the type of boundary presented in different parts of the image (recognition task) and the computer "reads" his/her actions to infer the boundary type. Since the users usually steer the cursor faster in strong parts than in weaker and uncertain parts, this reading is naturally done in terms of the speed and acceleration of the motion of the cursor. When the cursor movement is fast, the lane width is large and the method approaches live wire, becoming almost manual tracing when the movement is very slow. Live lane thus becomes a process which operates within a continuum between live wire and manual tracing.

The assessment of the performance of live-wire and live-lane methods as segmentation tools on a comparative basis is the topic of Section 4. We statistically assess the utility of four methods—live wire, live lane with fixed width, adaptive live lane, and manual tracing—based on speed and repeatability of segmentation. We state our conclusions in Section 5.

In finding optimum boundary segments, dynamic programming [18, 19] is used extensively in our methods. The use of dynamic programming and other graph searching techniques in boundary finding has been investigated for many years now [20–23]. Some of these early studies [21, 22] put severe restrictions on the form of the contour—that any radial line drawn outward from a point inside the object should not intersect the contour at more than one point. Some of these restrictions were overcome in [23] by using a graph, in which pixels are the nodes and 8-adjacency represents the arcs, to represent potential boundary lines. The aim in this work was detecting line-like structures, namely roads, in aerial images and not closed boundaries of object regions. Similar formulations were also used in [24, 25]. There are definite topological advantages in using pixel edges, rather than pixels, as primitive boundary elements in tracing boundaries of thin, or closely running objects as closed, connected, nonoverlapping and nonintersecting curves. The formulation in [20] did use pixel edges as boundary elements but did not model the graph of all possible boundary paths in the image. Instead it models boundary searching as a tree, which is rooted at a given "start" pixel edge and in which nodes represent (groups of) pixel edges and the arcs correspond to possible emanating boundary lines. The graph model used in the present paper seems to be unique in that (i) it does not impose any of the above restrictions on the form, shape, or size of the objects whose boundaries are to be traced; (2) it considers all possible boundaries in the entire image and is independent of starting points selected; (3) it considers boundary elements and boundaries to be oriented so that all boundaries (boundaries of even line-like structures) have a well-defined "inside" and "outside."

The genesis of the "live" methods presented in this paper has its origin in the early interactions between one of the authors of this paper (Udupa) and Barrett of Brigham Young University. Udupa introduced some of his graph model and optimum boundary formulations [26] to Barrett during his first visit to BYU in 1989. During Udupa's visit to BYU in 1992, Barrett and Mortensen demonstrated to Udupa their implementations of some of Udupa's formulations and their own extensions. In particular, they showed how by bringing human interaction to specify an initial point and to subsequently display optimal boundary segments

from the initial point to the current cursor position can potentially enhance the practical utility of optimal boundary definition techniques. They also coined the phrase "live wire" to such a strategy. The initial interaction between the two groups resulted in some joint publications [27–29]. Subsequently the groups pursued further development of these ideas independently [30, 31].

In summary, the methods presented in this paper differ from published works on optimum boundary finding and other "live" methods in five essential ways: (1) in the model of the underlying graph, (2) in considering boundaries as "oriented" curves, (3) in how costs are assigned, (4) in the ideas underlying live-lane method, and (5) in the method of evaluation. We will elaborate on these aspects at appropriate points in our presentation.

2. LIVE WIRE

A *2D scene* **C** is a pair (C, g) consisting of a finite 2D rectangular array C of pixels called the *scene domain* and a function $g(p) : C \rightarrow [L, H]$ called *scene intensity* that assigns to each pixel p in C an intensity value lying in an interval $[L, H]$.

The boundaries we seek in **C** are made up of oriented edges of pixels. They should satisfy certain conditions in order for them to become "legitimate." The most important properties are "closedness," "orientedness," and "connectedness." These properties for digital spaces have been rigorously defined in the literature (see for example [31]). Closedness here means that the boundary partitions the space into two disjoint components such that any "path" starting from a pixel in one component and ending in the other "meets" the boundary. Orientedness means that one of the two components can be unequivocally identified as constituting the "interior" of the boundary and the other as representing the "exterior." Connectedness guarantees that the boundary is a single connected curve.

There are numerous possible closed, oriented, and connected contours (made up of pixel edges) in any given scene **C**. Each such contour is a potential boundary. Every pixel edge in **C** is thus a potential boundary element. We define a *boundary element, bel* for short, of **C** as an ordered pair of 4-adjacent pixels. Every bel $b = (p, q)$ of **C** has a location and an orientation. The location of b is that of the unique edge shared by p and q. We assume, without loss of generality, that its orientation is such that p is always "inside" the boundary, q is "outside" the boundary, and the "inside' is always to the left of b (see Fig. 1). Clearly, any bel of **C** should be in one of four orientations as shown in Fig. 1.

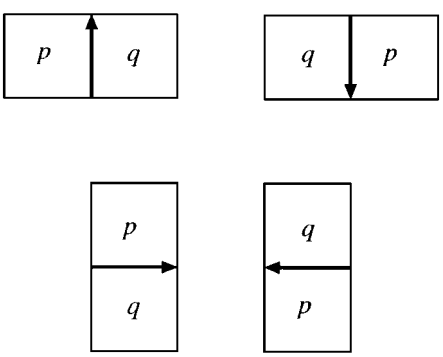

FIG. 1. A boundary element is an oriented pixel edge. The four possible types of bels in a scene are shown. The inside of the boundary is to the left of the bel and the outside is to the right.

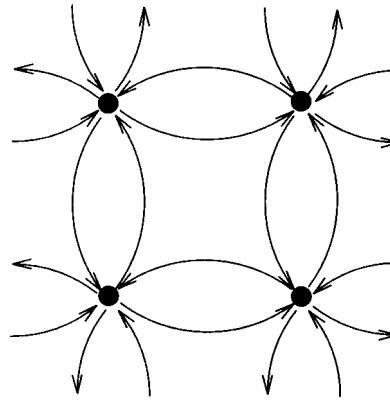

FIG. 2. The graph model used in finding optimum boundaries. The nodes in this graph are the pixel vertices and the directed arcs are oriented pixel edges. Only four nodes are shown in the figure.

To every bel $b = (p, q)$ of **C**, we assign a set of features. The features are intended to express the likelihood of the bel belonging to the boundary that we are seeking in **C**. The features describe certain properties of the object (the "interior" of the boundary), of the background (the "exterior" of the boundary), and of the boundary itself. The feature values are converted to a single joint cost value which describes the cost of having b as part of the boundary we are seeking. To every closed, oriented, connected contour that can be defined in **C**, we assign a cost which is simply the sum of the costs of all bels comprising the contour. Our aim is to find a contour with the smallest total cost. This enormous combinatorial optimization problem is greatly simplified if we take the user's help and indicate a bel b_0 of **C** that should be part of the boundary that is detected. The problem now becomes one of finding a closed, oriented, connected contour of minimal cost in **C** that contains b_0. This problem can be solved elegantly via dynamic programming [18, 19] if we translate this to a problem of finding a minimum-cost path in a graph as follows. Let v_s and v_e denote the vertices bounding the oriented edge represented by b_0. We define a directed graph (see Fig. 2) whose vertices are the vertices of all pixels of **C** and arcs are the bels of **C**. The problem now becomes one of finding a minimum-cost path from v_s to v_e that does not include b_0. This path together with b_0 forms the optimum boundary we are seeking. We point out that the graph model considered here differs from the models previously suggested in two respects. First, the basic boundary elements considered are oriented pixel edges. This makes it possible for us to trace boundaries of thin (even one-pixel wide) and subtle parts of the object as closed boundaries without distractions from closely running boundaries. Second, the directed nature of the graph allows us to distinguish between boundary segments that have opposite orientations but that otherwise have very similar properties.

Unfortunately, the optimum boundary detected in this fashion with this simple form of user help does not usually match with the actual boundary of the object. The main reason for this failure is that it is very difficult to find effective features and their effective transform into cost values such that the optimum boundary matches with the desired boundary except in an ideal situation such as in a scene with only two gray levels. Consequently, the user needs to specify more points along the desired boundary for its successful detection. In the live-wire method, the user initially selects a vertex v_s on the desired boundary in **C** using a cursor. For any subsequent location of the cursor, an optimum path from v_s to the

vertex v_e currently pointed to by the cursor is computed and displayed in real time. The user can verify the appropriateness of the optimum path in describing a segment of the desired boundary by moving the cursor and hence by changing v_e. To minimize the user's involvement in the segmentation process, he/she should attempt to get as large a segment of the desired boundary as possible through this optimum path. If this optimum path is acceptable, he/she deposits vertex v_e which subsequently becomes the new starting vertex v_s, and a new optimum segment that *does not include* the segments already detected should be found by moving the cursor to pick a new v_e. The process continues in this fashion. For the last segment on the desired boundary, v_e is taken to be the first starting vertex.

The issues we need to address then in completing the description of the method just outlined are: the selection of features, how to convert feature values into cost values, and how to rapidly find an optimum path between any two vertices v_s and v_e specified in **C**. These are addressed in the following sections.

2.1. *Feature Selection*

We have incorporated the following features in our current implementation. We consider each feature to be a function that assigns to every bel $b = (p, q)$ of a given scene $\mathbf{C} = (C, g)$ an integer representing a property value.

1. *Intensity on the positive side* (*"inside"*) *of the boundary* (f_1): A unit vector in the direction of a gradient of g determined at the center of b in the scene indicates the direction in which scene intensity increases most rapidly across b. Except when the direction of b is parallel to the oriented edge corresponding to b (Figs. 3a and 3b), one of p and q, denoted s, is in the positive direction of the gradient vector and the other, denoted s', is in the negative direction. In the case of Figs. 3a and 3b, $f_1(b)$ is a value randomly chosen from $g(p)$ and $g(q)$. In the case of Fig. 3c,

$$f_1(b) = g(s). \tag{1}$$

2. *Intensity on the negative side* (*"outside"*) *of the boundary* (f_2): This feature's definition is analogous to that of f_1, except that the value of the pixel in the negative direction of the gradient vector is considered. That is,

$$f_2(b) = g(s'). \tag{2}$$

Features f_1 and f_2 characterize "typical" scene intensities in the immediate interior and the immediate exterior of the boundary we are seeking in those regions where there is some evidence of boundariness.

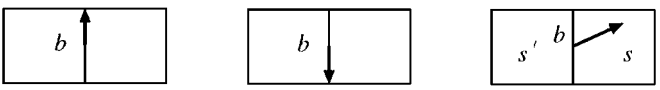

FIG. 3. (a, b) A unit gradient vector computed at the center of b aligns along the edge represented by bel b with a downward orientation. (c) s is in the positive direction of the unit gradient vector and s' is in the negative direction.

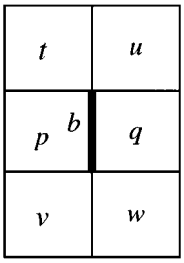

FIG. 4. Neighborhood of b used for defining f_3, f_4, f_5, and f_6.

3. *Gradient magnitudes* (f_3, f_4, f_5, f_6): These features represent gradient magnitude of g computed at the center of b using the following digital approximations (see Fig. 4):

$$f_3 = |g(p) - g(q)| \tag{3}$$

$$f_4 = \frac{1}{3}|g(p) + g(t) + g(v) - g(q) - g(u) - g(w)| \tag{4}$$

$$f_5 = \frac{1}{2}\left|g(p) + \frac{1}{2}g(t) + \frac{1}{2}g(v) - g(q) - \frac{1}{2}g(u) - \frac{1}{2}g(w)\right| \tag{5}$$

$$f_6 = \frac{1}{4}(|g(p) - g(u)| + |g(t) - g(q)| + |g(p) - g(w)| + |g(v) - g(q)|). \tag{6}$$

These features are clearly not independent. However, we find some of them better suited than others in different applications.

4. *Orientation-sensitive gradient magnitude* (f_7): Typically, at points on the boundary that we are seeking, the gradient of g is directed either from the interior of the boundary to its exterior or vice versa. If this orientation is known for the application at hand, the information can be used to discriminate more accurately among bels that actually belong to the boundary and bels that do not but have otherwise values of other features (especially gradient magnitude) "consistent" with those of the desired boundary. The orientation of each bel b can be used to ascertain locally whether or not the gradient of g evaluated at the center of b is consistent with the desired boundary, since we know to which side of b lies the interior of the boundary and to which side lies its exterior. "Consistent" here means, for example, that while tracing the contour say in the counterclockwise sense, the gradient of g goes from the left (interior) of the contour to the right (exterior) of the contour. In case of consistency, we define $f_7(b) = f_6(b)$, otherwise $f_7(b) = -f_6(b)$.

5. *Distance from previous boundary* (f_8): $f_8(b)$ represents the shortest distance of the center of b from the boundary that is defined on the previous slice and that is projected onto the current slice. This feature is undefined if boundary detection has not been done for the previous slice. The idea behind this feature is that the boundaries usually do not change significantly from slice-to-slice if the distance between slices is sufficiently small. Therefore, boundary elements in the current slice that are closer to the boundary in the adjoining slice are more likely to belong to the boundary being sought than those that are farther. This feature is utilized in two ways: (i) to modify the cost assigned to bels and (ii) to delimit subsets of the scene domain that are used for optimal boundary search. A similar feature was used earlier for optimal boundary finding [33] in 2D angiographic images.

The ability to discern favorable orientation via f_7 allows us to assign "low" costs to bels with positive values of $f_7(b)$ and "high" costs to those with negative values of $f_7(b)$. Note

that each of features f_1 and f_2 is independent of the gradient features considered in (3) above and provides vital information about the immediate vicinity of the boundary. Feature f_8 has considerable relevance in practical segmentation involving many slices. The ideas underlying f_1, f_2, f_7, and some aspects of f_8 have not been utilized in previous efforts on optimum boundary finding.

2.2. *Cost Assignment*

The feature values $f_i(b)$, $1 \le i \le 8$, associated with each bel b of $\mathbf{C}$ are converted to cost values $c_j(f_i(b))$ using functions c_j which we refer to as *feature transforms*. The range of these functions is [0, 1] and their domain is the set of integers. Any of the following feature transforms can be used with any of the features.

1. *Linear* (c_1): c_1 is a linear mapping within an interval $[l_1, h_1]$ of feature values. Values outside this interval are mapped to 1.
2. *Inverted linear* (c_2): c_2 is an inverse linear mapping within an interval $[l_2, h_2]$ of feature values. Values outside this interval are mapped to 1.
3. *Gaussian* (c_3): c_3 is a Gaussian function with mean l_3 and standard deviation h_3.
4. *Inverted Gaussian* (c_4): c_4 is an inverted Gaussian function with mean l_4 and standard deviation h_4.
5. *Modified hyperbolic* (c_5): Only the part of a hyperbolic function with nonnegative values is considered:

$$c_5(x) = \begin{cases} 1, & \text{for } x \le l_5 + \frac{a^2}{2} \\ \frac{a^2}{2(x-l_5)}, & \text{for } l_5 + \frac{a^2}{2} \le x \le h_5 \\ 0, & \text{for } x > h_5. \end{cases} \tag{7}$$

Here, a, l_5, and h_5 are free parameters. a represents the distance of the focus of the hyperbola from its two asymptotes, l_5 represents the distance of the vertical asymptote $x = l_5$ from the origin, and h_5 is the upper cut-off value for x.

6. *Inverted modified hyperbolic* (c_6): c_6 is an inverted version of c_5 defined as follows:

$$c_6(x) = \begin{cases} 0, & \text{for } x \le l_6 + \frac{a^2}{2} \\ 1 - \frac{a^2}{2(x-l_6)}, & \text{for } l_6 + \frac{a^2}{2} \le x \le h_6 \\ 1, & \text{for } x > h_6. \end{cases} \tag{8}$$

The *joint cost* $c(b)$ associated with each bel b of $\mathbf{C}$ is a linear combination of the costs assigned to its features,

$$c(b) = \frac{\sum_i w_i c_{f_i}(f_i(b))}{\sum_i w_i}, \tag{9}$$

where w_i is a positive constant indicating the emphasis given to feature f_i and c_{f_i} is the cost associated with feature f_i.

It should be noted that any feature transform can be used with any feature. The combination should be chosen to minimize the joint cost associated with the bels belonging to the boundary of interest. The appropriateness of a combination depends on the application at hand. Not all features may be relevant for a given application. There may be features not considered here that are more effective for the segmentation task at hand than the combinations that can be produced from the features listed here. The point we wish to emphasize is that there are numerous ways a joint cost function can be devised. As long as these lead to

sufficiently low joint cost values for bels along the desired boundary, even in an intermittent fashion, the live methods presented here will use this as a road map with the user's assistance to quickly segment the object.

The feature transforms are devised with different types of features and their intended influence on the detectability of the boundary in mind. For instance, if we expect gradient magnitude along the desired boundary to fall within an intermediate range (not too high and not too low), then c_4 with l_4 set to the most likely value and h_4 set to a fraction of this range is an appropriate feature transform. As another example, if low values are to be favored (i.e., assigned low cost) for a feature (e.g., f_8), then c_5 with suitable values for l_5, h_5, and a will form an appropriate feature transform.

In addition to the selection of proper features and of appropriate feature transforms to go with them, the parameters of the feature transforms, l_i, h_i, a, and w_i, for $1 \leq i \leq 8$, are also under user control. Although f_1, f_2, f_7, all with c_4 (i.e., as per the notation in Eq. (9), $c_{f_1} = c_{f_2} = c_{f_7} = c_4$) produce effective performance in most situations, making effective choices of features, transforms, and their parameters can be quite taxing, especially for end users in large applications who are neither knowledgeable nor care to learn about the innards of the segmentation scheme. An effective and intuitive method is therefore needed for making these selections in routine use of this segmentation method. This forms the subject of the next section. However, the rich environment described above is useful in a research setting.

2.3. *Training*

The purpose of training is to facilitate the process of selection of features, transforms, and their parameters, obviating the need for the users to become knowledgeable about the segmentation methodology. In the following two subsections, we describe two approaches toward this goal, the first for the automatic selection of values for parameters of feature transforms and the second for the automatic selection of an optimum feature set. Training in the former sense has been reported previously in optimum boundary finding [33].

2.3.1. Automatic parameter selection We assume here that the user has decided which features to use and what the accompanying feature transforms are. The aim of training then is to determine the values to be selected for the parameters describing each feature transform.

Training consists of the user painting typical segments of the boundary. The user is allowed to select paint brushes of various sizes. For sharper boundaries, thinner brushes should be used. Segments that are nontypical should be avoided. Any number of slices or data sets may be used, although typically painting a couple of boundary segments is sufficient. At the completion of painting, the values of the following parameters are computed for all features: minimum, maximum, mean, and standard deviation. The parameters of the feature transforms are determined automatically as some fixed relationship to these feature statistics. For example, when feature f_6 is used with transform c_5, we set $l_5 = MIN$, $h_5 = MAX$, and $a = (MAX - MIN)/10$, where MIN and MAX are the minimum and maximum feature values, respectively. The user may accept this automatic assignment or may vary the values for fine tuning. To illustrate how the sensitivity of the live wires to boundaries can be changed dramatically by this method of training, we present one example.

Figure 5a is a display of a CT slice of a patient's knee. The default settings of the parameters in our implementation are as follows: feature f_6, transform c_3, $l_3 = 0$, $h_3 =$ (maximum of f_6)/10, all other features are turned off (not used). Evidently, high gradient

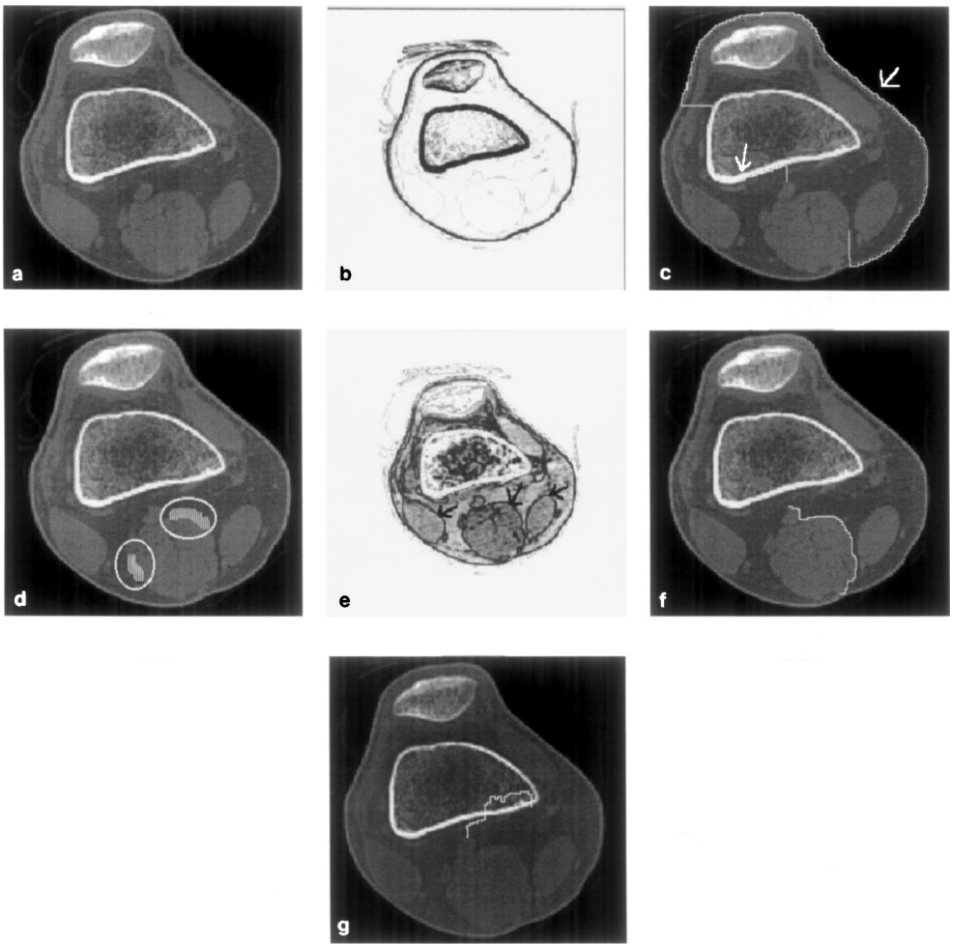

FIG. 5. (a) A CT slice of a knee; (b) an image showing the joint cost of all bels of the scene (a), using f_6, c_3, $l_3 = 0$, and $h_3 =$ (maximum of f_6)/10; (c) live wire being attracted by the high-contrast boundaries; (d) training for low-contrast boundaries; (e) joint cost function after training for low-contrast boundaries; (f) live wire being attracted by the low-contrast boundaries; (g) live wire being repelled by the high-contrast boundaries.

magnitudes are assigned low costs and low gradient magnitudes are assigned high costs. Figure 5b shows the joint cost function c as an image in which brightness is proportional to cost. If we try to segment the lower-contrast soft-tissue boundaries via live wire as in Fig. 5c (the two points v_s and v_e are specified on the low-contrast boundary in the lower-middle part of the image), the live wire is forcefully attracted by the high-contrast bone and skin boundaries as indicated by the arrows. The optimal path actually takes a circuitous route although the two points specified are close by. The live wire can be made to be attracted by the lower-contrast soft-tissue boundaries by training. After the user paints, as indicated in Fig. 5d, the resulting joint cost image is shown in Fig. 5e. Note how the low-contrast boundaries (indicated by arrows) get a significantly lower cost compared to their cost in Fig. 5b and the reverse situation with high-contrast boundaries. The live wires are now attracted remarkably well (Fig. 5f) by the low-contrast boundaries and are actually repelled by the high-contrast boundaries (Fig. 5g). The live-wiring process is dynamic and unfortunately cannot be adequately illustrated in static hard copies.

Note that, in segmentation tasks involving a large number of data sets, training needs to be done only once, perhaps on one or two slices, for each of the objects to be segmented, and not on every data set. Therefore, the time spent in training is negligible.

2.3.2. Optimum feature selection In spite of the training facilities available to the users, as described in the previous section, sometimes it is difficult to determine the features and their transforms to be used to obtain a good, object-representative joint cost function. We show in this section that it is possible to generate a consistently good operational joint cost function automatically.

Let B denote the set of bels of a scene $\mathbf{C}$ which are indicated (via painting) in training, let N denote the set of all bels within a small neighborhood of the bels in B, let $\bar{B} = N - B$, and let $\min_{f_i}$ and $\max_{f_i}$ denote the minimum and maximum values for feature f_i for bels in B.

Let $|X|$ denote the cardinality of any set X, let $F = \{f_1, f_2, \ldots, f_m\}$ (in our case $m = 7$, since we do not use feature f_8 for training), and let $X = \{f_{i_1}, f_{i_2}, \ldots, f_{i_j}\}$ be any nonempty subset of F. Let H_X be the hypercuboid defined by the j intervals $[\min_{f_i}, \max_{f_i}]$:

$$H_X = \left[\min_{f_{i_1}}, \max_{f_{i_1}}\right] \times \cdots \times \left[\min_{f_{i_j}}, \max_{f_{i_j}}\right]. \tag{10}$$

Our aim is to select a subset X of features such that the number of bels in $\bar{B}$ whose feature values fall within H_X is as small as possible. That is, we wish to select that combination of features which results in the set of bels selected to be as close as possible to B (ideally equal to B). To facilitate this optimum selection, we maximize the following function over all subsets X of F:

$$P_B(X) = \frac{|B|}{|B| + |\bar{B}_{H_X}|}, \tag{11}$$

where

$$\bar{B}_{H_X} = \{b \in \bar{B} \mid (f_{i_1}(b), f_{i_2}(b), \ldots, f_{i_j}(b)) \in H_X\}. \tag{12}$$

There are $2^m - 1$ possible nonempty subsets of F (since the power set of F has 2^m elements). Given B, F, and a scene $\mathbf{C}$, the range of $P_B(X)$ therefore has $2^m - 1$ elements. For small values of m, the set X that maximizes $P_B(X)$ can be determined by exhaustive check. In our implementation, this can be done almost instantaneously after the user paints selected boundary segments for training.

Once the optimum combination of features is determined, each of these features is converted to a cost value using transform c_4 with l_4 set to the mean of the feature and h_4 to the standard deviation of the feature and identical weight for all selected features. We emphasize that, even in this automatic selection, the users may accept the recommendation, drop some of the features, fine tune some of the parameters, or discard the recommendation altogether. In our experience, the optimum selection consistently provides an excellent joint cost function that is very effective for live-wire tracing. Figure 6a shows an MR cross-sectional image of the foot of a normal subject. Figure 6b shows the joint cost function resulting from training for the boundary of the talus. Clearly, the resulting joint cost function is very decent. Figure 6c shows a live-wire tracing for this example. The points selected are indicated by arrows.

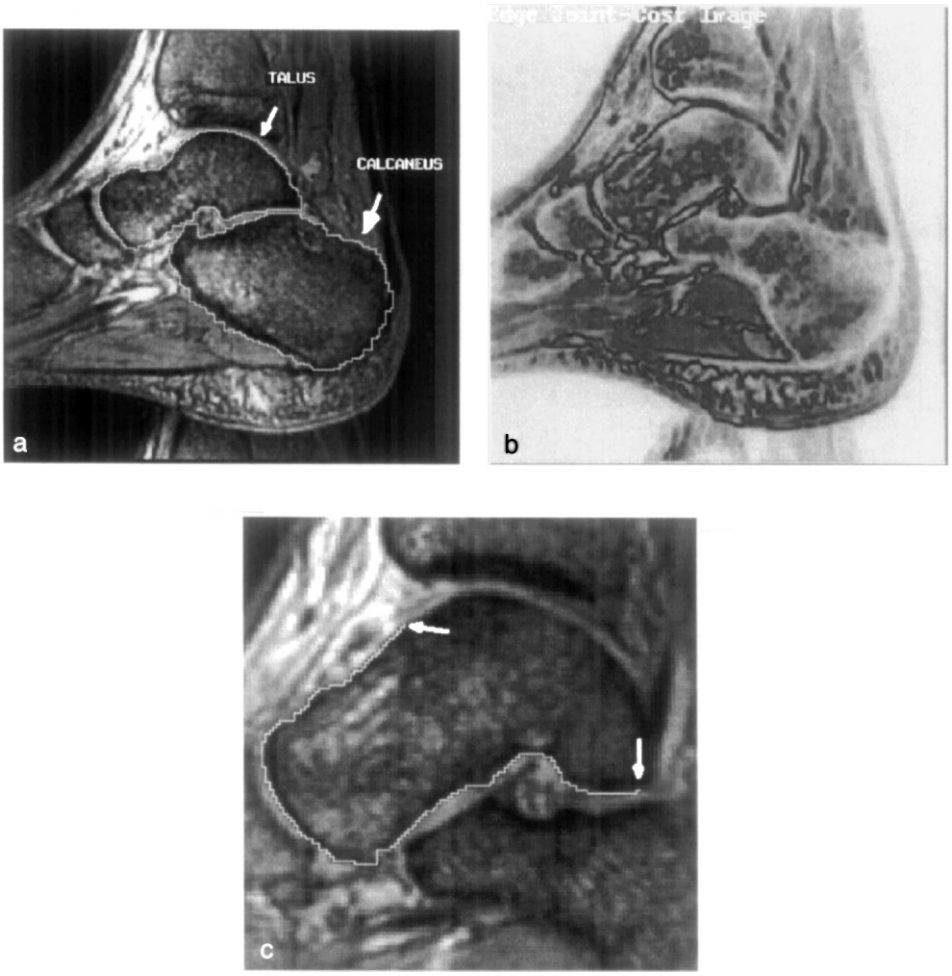

FIG. 6. (a) An MRI slice of the foot of a subject showing the bones talus and calcaneus. (b) Joint cost function obtained after training for the boundary of talus and after optimum feature selection. (c) A live-wire tracing on the boundary of the talus using the joint cost function of (b). The points (vertices) selected are indicated. Although the boundary characteristics change considerably between these points along the boundary, and other distracting boundaries (of the calcaneus and of other bones) of similar properties run closely, a single live-wire segment is adequate to detect this boundary segment between the arrows.

Some comments regarding feature f_8 are in order. It is quite different from other features, in the sense that it is preferable to use it as a "structural" feature as opposed to others which are used as "image" features. Although it can be included in training for optimum feature selection, we found a more effective way of utilizing this feature. This will be described at the end of Section 2.4.

2.4. *Finding Optimum Path*

In this section, we describe an algorithm, based on dynamic programming [18, 19], for finding an optimum path between any two vertices v_s and v_e in a given scene **C**. The path is optimum in the sense that the total joint cost (Eq. (9)) of bels in this path is the smallest of the total joint cost of bels in any path from v_s to v_e.

The algorithm is iterative and works as follows.

ALGORITHM LW

Input: The joint cost function c; an initial vertex v_s; a terminal vertex v_e; and a threshold T_{cc} on cumulative cost.

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

Auxiliary data structures: A 2D "cumulative cost" array cc representing the total cost of the optimal path found so far from v_s to each vertex of **C**; a 2D "direction" array *dir* indicating, for each vertex, to which of its immediate neighboring vertices the optimal path goes; a queue Q of vertices; and a list L of vertices which have already been processed.

begin

1. set $cc(v)$ to ∞ for all vertices v in **C** and set $cc(v_s)$ to 0;
2. put v_s in Q;
3. *while* Q is not empty *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 4-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* $cc_{tmp} < cc(v')$ and $cc_{tmp} < T_{cc}$ *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;
      endif;
   endfor;
   endwhile;
4. 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 output the bels so traced;

end

We emphasize that, in Step 3b(i), the bel connecting v' to v is not the same as the bel connecting v to v'. In fact, the latter is not considered in this step. Generally, these two types of bels may have significantly different costs associated with them.

Note that, within the do-while loop of Step 3, v_e does not come into picture. In fact, when this loop is completed, we have the optimal path information available in *dir* for paths between v_s and every vertex in **C**. Each vertex in **C** therefore becomes a potential v_e. As the user moves the cursor to indicate which vertex of **C** is to be considered as v_e, the optimal path between v_s and the selected v_e is determined in real time in Step 4 and displayed over the scene for user verification.

The algorithm can be used to output an entire closed, connected, oriented contour, rather than a boundary segment, by having v_s and v_e to be the vertices of a bel b_0. In this case, the user specifies b_0 instead of v_s and v_e. The set of bels output in Step 4 together with b_0 is now guaranteed to represent a closed, connected, oriented contour. Unfortunately, in practice, we have found that this contour does not correspond to the boundary we are seeking. There are two main reasons. First, optimality is based on total cost, and hence, the algorithm inherently favors short paths. Second, feature and feature transform specifications are rarely perfect. Consequently, the algorithm finds small loops in the vicinity of b_0 (see Fig. 7). However, there may be applications wherein it may be possible to specify the cost functions in such

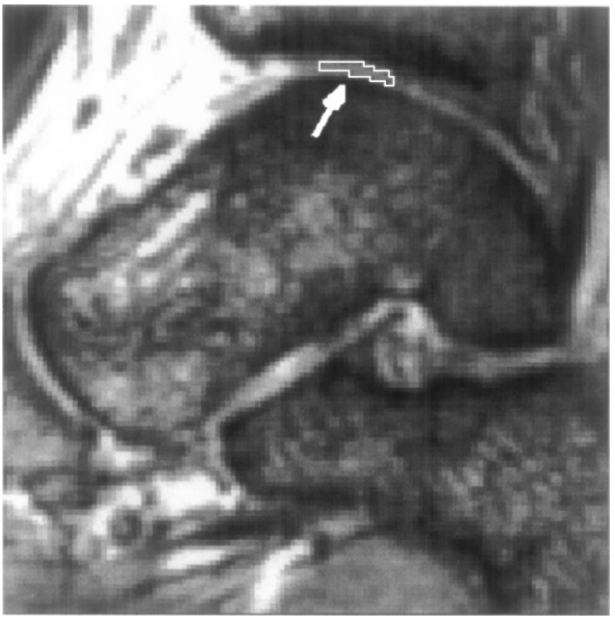

FIG. 7. Algorithm LW might fail trying to output an entire closed, connected, oriented contour, rather than a boundary segment, by having v_s and v_e be the vertices of a bel b_0. The optimality criterion leads the algorithm to find small loops in the vicinity of b_0, in spite of considering boundary orientation.

a way that the contour found with b_0 as the specified bel matches the desired boundary. Clearly, the algorithm as presented above (where v_s and v_e may be any vertices) does not mitigate this advantage of finding the boundary automatically given b_0. The user has to simply pick two points v_s and v_e and indicate a "close" operation to find the optimum path from v_e to v_s. Therefore, in general, for reasons of tight user control described previously, finding live-wire segments between any two vertices v_s and v_e is practically more useful.

To minimize computation, we set a threshold T_{cc} on the cumulative cost $cc(v)$ such that, if $cc(v) \geq T_{cc}$, then the optimum path estimation does not go beyond v. The result of setting this threshold is that potentially a large number of uninteresting paths are pruned at an early stage so that computations are not wasted on them. We set T_{cc} to the total cost of the largest object contour encountered in the application. Interesting effects are created when the value of T_{cc} is chosen to be less than this value. We will come back to this point in Section 5.

We wish to emphasize that the vertices through which the previously found live-wire segments pass are excluded for the determination of the current live-wire segment. It is readily seen that situations may be created by carefully tracing live-wire segments in a contorted fashion, wherein, for the next vertex v_e chosen, no paths exist (leave alone optimal) between the two current v_s and v_e. In such situations, the implemented software gives a warning to this effect. The user now has to backtrace to get out of the trap. In practice, we have never encountered such a situation. Therefore, when live-wire tracing is successfully completed, we are always guaranteed to have created a closed, connected, oriented contour, in other words, a digital Jordan curve.

To illustrate the power of feature f_7, we present one example. Figure 8a shows an MR slice of a foot in which our aim is to segment the boundary of bones. Since hard tissues do not give MR signals, bone appears dark in these images. Figure 8b shows a live-wire

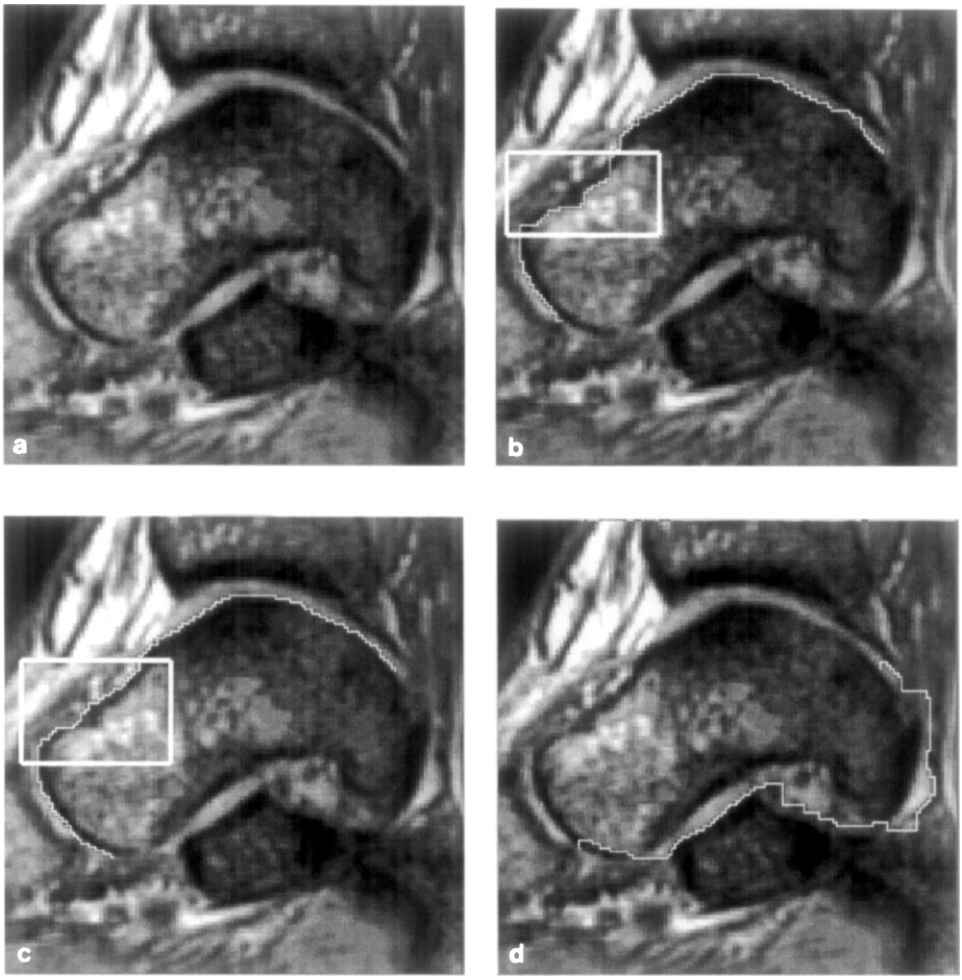

FIG. 8. Illustration of the power of feature f_7 and of the sensitivity to orientation of the tracked live-wire segments. (a) An MRI slice of the foot of a subject showing the talus. (b) A live-wire segment, obtained excluding f_1, f_2, and f_7 from the feature set, being attracted by spurious segments that have properties similar to those of the boundary segments of interest. (c) A live-wire segment between the same two vertices as in (b), obtained with f_7 added to the same feature set as in (b). The spurious boundary segment in the box does not attract the live wire since its orientation is not favorable. (d) Live-wire segments of a tracing in the clockwise sense, between the same two points as in (c) but with opposite orientation. Boundary tracing in the opposite orientation is clearly wrong, illustrating orientation sensitivity of the live-wire segment. Features f_1 and f_2 have properties similar to those of f_7. To understand why this is so, note that for the correct (counterclockwise) orientation, intensity f_1 inside (i.e., to the left of) the boundary is much lower than intensity f_2 outside (to the right of) the boundary. For the wrong part of the boundary in the box, this relationship between f_1 and f_2 does not hold. For the same reason, when we force the contour to go clockwise it takes the path shown in (d).

segment obtained for this scene via the optimum feature selection method of Section 2.3.2 but excluding f_1, f_2, and f_7. Figure 8c shows the live-wire segment obtained between the same two vertices as in Fig. 8b but with f_7 included in the feature set. Note that, in Fig. 8c, unlike in Fig. 8b, the live-wire segment is not attracted by spurious boundary segments. In general, with feature f_7, live-wiring not only requires fewer points but also proceeds faster since there are fewer distractions. Similar remarks apply to f_1 and f_2.

Figure 8d illustrates the sensitivity to orientation of the tracked live-wire segments. If tracing is forced in the clockwise sense as in Fig. 8d between the same two points as in 8c (as opposed to in the counterclockwise sense as in Fig. 8c), it is impossible to get good boundary segments. While tracing, the user immediately recognizes that live wire strongly favors one direction while appearing almost ineffective in the opposite direction. Orientedness of boundaries has not been exploited in previous optimum boundary detection research.

Returning to the issue of how to effectively use structural information available from feature f_8, we first observe that usually the object boundary contours do not change much from slice to slice. The idea is to constrain the search for optimal path in the next slice to an annular region (shell) of width W centered around the projection onto the next slice of the contour(s) traced in the current slice. Given any finite, closed, connected, oriented 3D digital surface, it is possible to determine the smallest value for W such that the boundary to be traced in the next slice is guaranteed to fall within this annular region. The advantages of this approach are twofold: (i) Live-wire segments appear instantaneously upon selection of v_s; (ii) many otherwise distracting boundary structures are removed. Both contribute toward saving user time without compromising the power of live wire. The effect of (ii) is that longer live-wire segments can be extracted. In many situations, just two points (i.e., two live-wire segments) are adequate to capture a contour. In our implementation, we leave parameter W under user control. In segmentation tasks involving a large number of 3D data sets acquired with a fixed imaging protocol, the smallest acceptable value W needs to be determined experimentally only once on one 3D data set. This value can be subsequently used for segmenting all other data sets.

Algorithm LW and those reported in [25–30] are all based on dynamic programming. LW differs from others in the underlying graph model (see Fig. 2) and hence in the neighbors that are checked in Step 3. It achieves real-time response using threshold-based pruning, structural feature f_8, and four neighbors instead of the eight that are used in other approaches.

3. LIVE LANE

Given a scene $\mathbf{C} = (C, g)$, the methods of feature selection, cost assignment, training, and optimal path finding for the live-lane process are as for live wire. To begin the process, the user selects an initial vertex v_s on the boundary of interest and subsequently steers the cursor roughly in the vicinity of the boundary. We determine a square pixel array $S(t) \subset C$ such that the vertex indicated by the cursor at any time t is that of a pixel within $S(t)$ during this process. Vertices are selected intermittently (as described below) during this process automatically, and live wire segments confined to the region defined by $S(t)$ are computed between each pair of successive vertices in real time using Algorithm LW. When the cursor eventually comes back to the vicinity of v_s, the user indicates a "close" operation to initiate the computation of the live-wire segment between the most recently selected vertex and v_s. Live lane is thus a more dynamic process than live wire. It differs from live wire in (i) the mechanism by which the end points of the live-wire segments are specified, (ii) the directed graph used for finding optimal paths, and (iii) the underlying process of human–machine interaction. Below we describe two strategies for live lane. These have quite different influences on these three factors, particularly on the third.

Let $T = \langle t_0, t_1, \ldots, t_n \rangle$ be a sequence of time instances representing a discretization of the time interval $[t_0, t_n]$, where t_0 and t_n indicate the time instances at the beginning and end

of the live-lane process, respectively. T represents a sequence of time instances at which the positions of the cursor are read during the process. For any t_i in T, let $v(t_i)$ and $S(t_i)$ denote, respectively, the vertex of the pixel in C pointed to by the cursor and the square array to which optimal path finding is confined at time t_i. We say that a vertex v is in $S(t_i)$ if v is a vertex of a pixel in $S(t_i)$.

3.1. *Tracking with Lane of Fixed Width*

For any t_i in T, the square array $S(t_i)$ defines a *subscene* $\mathbf{S}(t_i)$ in $\mathbf{C}$, defined by $\mathbf{S}(t_i) = (S(t_i), g_S(t_i))$, where $g_S(t_i)$ is the restriction of g to $S(t_i)$. We assume in this method of tracking that the size of $S(t_i)$ is fixed for all t_i.

Upon selection of the first vertex v_s in $\mathbf{C}$ at time t_0, $S(t_0)$ is defined to be a square array centered at v_s. As the user moves the cursor within $S(t_0)$, each position of the cursor $v(t_i)$ defines a vertex in $S(t_0)$ for which an optimal path from v_s is computed and displayed in real time using Algorithm LW. When the cursor crosses $S(t_0)$, the optimal path computed in $\mathbf{S}(t_0)$ from v_s to the last vertex indicated in $S(t_0)$, denoted v_e, is automatically selected as a boundary segment. This v_e now becomes the next starting point v_s, and if t_{i_1} is the time at which v_e was selected, v_e becomes the center of $S(t_{i_1})$ and $\mathbf{S}(t_{i_1})$ becomes the next subscene for optimal path computation. The process continues until at some time $t_{i_n} = t_n$ the first v_s selected falls within the subscene $\mathbf{S}(t_{i_n})$ and the user indicates a "close" operation. At this time, the last boundary segment is computed and a closed, connected, oriented contour is output. In our implementation of this method, the size of the array $S(t_i)$ is left as a free parameter to be selected by the user within the range 5 to 100 pixels.

There are several advantages of this approach over live wire. First, on computers without adequate computational power, the live-wire method may not achieve adequate speed of response, especially if the desired boundary occupies a large area (for example, a 2000×2000 pixel array as in a digitized radiograph). In contrast, live lane will reach real-time response on any modern computer, including PCs, since dynamic programming is confined to the graph associated with $\mathbf{S}(t_i)$, which is significantly smaller than that associated with $\mathbf{C}$. In other words, in live lane, unlike in live wire, the reduction of the user's time requirement in the segmentation process is more-or-less independent of the power of the computer. Second, when the boundary has many weak, undefined, and uncertain segments, the users will need to select points closely situated in such regions. Unfortunately, although the optimal paths are not likely to go too far from these points, live wire considers a substantial part of scene $\mathbf{C}$ for optimum path computations. Live lane's speed is therefore not compromised for poor boundaries. Clearly, live lane approaches manual tracing as the lane width (the size of $S(t_i)$) approaches 1.

The above discussion leads us naturally to the following question: Why not determine lane width automatically during the tracing process so that it is greater in the vicinity of well-defined boundary segments and smaller in weak and uncertain regions?

3.2. *Tracking with Adaptive Lane*

In this strategy, a crucial decision to be made is how to determine where the weak and strong boundary parts are during tracing. Any automatic computational methods (such as those based on scene intensity gradients) will face the dilemma of having to know the region to which this computation should be confined, which is precisely the information

we are seeking in the first place. Because of the users' superior ability (compared to that of algorithms) to make this decision, we decided to take their help under the following pretext. We assume that it is natural for users to negotiate clear and sharp boundaries quickly and to steer the cursor slowly for careful delineation in the vicinity of weak boundaries. This postulate leads us naturally to a rule for determining lane width as an increasing function of the cursor's speed.

For any $t_i \in T$, we consider $v(t_i)$ to be a 2-component vector indicating the two coordinates of the vertex in a fixed Cartesian coordinate system. The speed $s(t_i)$ and acceleration $a(t_i)$ are defined as

$$\begin{aligned} s(t_0) &= a(t_0) = 0, \\ s(t_i) &= \frac{\|v(t_i) - v(t_{i-1})\|}{t_i - t_{i-1}}, \quad i > 0, \\ a(t_i) &= \frac{s(t_i) - s(t_{i-1})}{t_i - t_{i-1}}, \quad i > 0, \end{aligned} \tag{13}$$

where $\|\cdot\|$ denotes vector magnitude. Let $lw(t_i)$ denote the number of columns in array $S(t_i)$ which we call *lane width* at time t_i. Adaptive live lane differs from fixed live lane in two aspects: (i) The lane width $lw(t_i)$ may vary with t_i, and (ii) end vertices v_e are selected for optimal boundary segment generation whenever the lane width changes.

The process begins when the user selects the initial vertex $v_s = v(t_0)$ with a small value for $lw(t_0)$ (10 pixels in our implementation). As the user now moves the cursor, an optimal path to each vertex $v(t_i)$ pointed to by the cursor within $S(t_0)$ is computed and displayed in real time. For $t_0 \le t \le t_i$, $lw(t) = lw(t_0)$ (i.e, $S(t) = S(t_0)$) until one of two events takes place: (i) the cursor moves out of $S(t_0)$ (meaning that $v(t_i)$ is not in $S(t_0)$ but $v(t_{i-1})$ was), and (ii) $|a(t_i)| > A$, A being a fixed threshold, but $|a(t_{i-1})| \le A$. At this time, denoted t_j, $lw(t_j)$ is defined as

$$lw(t_j) = Ks(t_j), \tag{14}$$

where $K > 0$ is a constant, and $S(t_j)$ centered at $v(t_j)$ and of size $lw(t_j)$ is determined. The optimum path from v_s to $v_e = v(t_j)$ is output as a boundary segment and v_s is set to $v(t_j)$. The process continues in this fashion until at some time instance t_n the user indicates the "close" operation and $v(t_0)$ is within $S(t_n)$. At this time, the optimal path between the current v_s and $v(t_0)$ is computed and output as the last boundary segment, and the process terminates.

Although the lane width is computed as a linear function of the speed of the cursor, the acceleration of the cursor is also necessary to interpret the actions of the user during the process. As transitions between strong and weak regions are abrupt along the boundary, high positive (or negative) values of acceleration are read in these regions. This indicates to the algorithm that the current cursor position should be selected as a new starting point v_s and the lane width should be updated even if the point is away from the boundary of $S(t_i)$. This accounts for the actions of the user who steers the cursor with high speed and low acceleration in clear regions, low speed and low acceleration in weak regions, and high acceleration/deceleration in transition regions.

The relative efficacy of live-wire and live-lane methods is studied in more detail in the next section.

4. EVALUATION

We have presented three methods for the user-steered segmentation of boundaries in 2D scenes. The aim of this section is to gain some quantitative and objective insight into the effectiveness of these methods in a comparative fashion. There are many parameters involved in each of these methods: the features, their transforms, and the weights to be used for all methods; the size and shape of the boundaries, the value of W, and the power of the machine for live wire; the lane size for fixed live lane; the values of K and A and the degree of interactive response of the input device for adaptive lane. The task of comparison becomes unmanageable if all or even the most important among these parameters were to be taken into account. Our approach, therefore, is to eliminate all those parameters which, based on our observation, are unlikely to play a competitive role in performance. For example, we have observed that automatic optimal feature and cost selection (described in Section 2.3.2) consistently outperforms manual selection although we do not rule out the existence of application situations where this may not hold true. By this process, we arrived at the following methods for comparison:

- LW: live wire with automatic optimal feature and cost selection and $W = 20$.
- LL: live lane with fixed lane width and feature and cost selection as in method LW; the fixed lane width was chosen based on preliminary experiments.
- AL: adaptive lane with feature and cost selection as in method LW, $K = 5$, $A = 9$.
- MN: manual tracing used as a basis for comparison.

The values of the parameters mentioned above were determined based on the particular application from which the test data came. All methods were implemented in an internal version of 3DVIEWNIX [34] and executed on a SGI Indy workstation.

In evaluating user-steered segmentation methods, three factors seem to be of prime importance: speed, repeatability (precision), and accuracy. Speed expresses the extent of user involvement required by the process. Repeatability expresses the variations in the segmentation outcome as a result of the subjective actions taken by the user in the steering process. Accuracy expresses the degree of agreement of the segmentation result with truth. These factors may have to be given different emphasis in assessing the utility of a method in a given application as a function of these factors.

Unfortunately, for image data acquired for live subjects, true (ideal) segmentation cannot be established. Consequently, usually compromises are made. For example, often the result of manual segmentation is considered to be a good approximation of true segmentation. In our experiments (described below), the use of manual tracing as a true segmentation became questionable since we observed that its repeatability is worse than that of live paradigms. As we pointed out in Section 1, it is in the recognition task that the live methods need, and therefore take, user's help, but delineation is done more consistently by these methods than by the human operator. Given this and the fact that the algorithms have better repeatability, assessing accuracy based on manual segmentation does not seem to have much virtue. Therefore, we decided to assess the utility of the methods as a function of only speed and repeatability.

Instead of testing on a few sample images as is commonly done, for getting real insight into the behavior of these methods we have chosen data from one of our ongoing applications, namely the kinematic analysis of the tarsal joints of the foot [35]. In this application, we study the in vivo internal 3D kinematics of the tarsal joints via MR imaging for understanding

and quantifying normal and abnormal motion, with the long-term goal of surgery planning taking into account joint motion. We have acquired over 10,000 2D scenes constituting data for over 25 live joints each in eight positions. Most of the ideas expressed in this paper evolved as a result of our attempts to devise practical and effective methods to segment bones in these images. 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 have segmented all these data using live methods for defining the boundaries of four bones—the talus, calcaneus, navicular, and cuboid.

We chose a set Σ of 30 2D scenes from this data set for evaluation. Each of three operators O_1, O_2, and O_3 segmented in three separate trials, denoted E_1, E_2, E_3, the talus (T_a), and the calcaneus (C_a) in these scenes using each of the four methods LW, LL, AL, and MN. The reason for considering these two bones is that they represent quite different conditions for the segmentation task. The talus consists of relatively well-defined boundary segments with occasional poorly defined segments. The calcaneus consists of mostly poorly defined segments. The contour resulting from each segmentation of each input 2D scene $\mathbf{C} = (C, g) \in \Sigma$ was converted to a binary scene $\mathbf{C}_e = (C_e, g_e)$, where $C_e = C$, and for any $p \in C$, $g_e(p) = 1$ if p was inside the contour and $g_e(p) = 0$ if p was outside. Note that a binary scene resulting from an experiment contains information about only one bone. For any fixed $o \in \{O_1, O_2, O_3\}, r \in \{E_1, E_2, E_3\}$, $m \in \{\text{LW, LL, AL, MN}\}$, and $\mathbf{C} \in \Sigma$, we refer to the act of operator o of tracing the contour of bone b using method m in trial r as a *segmentation experiment* involving o, r, m, b, and $\mathbf{C}$. Our evaluation study thus consists of 2160 segmentation experiments in total.

We define the *speed of segmentation* SP_e of any segmentation experiment e (expressed in number of 2D scenes (slices)/min) as

$$SP_e = \frac{1}{T_e}, \tag{15}$$

where T_e is the time in minutes that is taken to complete e. Consider any fixed $o \in \{O_1, O_2, O_3\}, r \in \{E_1, E_2, E_3\}$, $m \in \{\text{LW, LL, AL, MN}\}$, and $b \in \{T_a, C_a\}$. We define the *speed of segmentation* SP_{ormb} for operator o in trial r using method m for object b to be the average of all speeds SP_e over all segmentation experiments e involving o, r, m, b, and all scenes. We define the *speed of segmentation* SP_{omb} for operator o using method m for object b to be the average of all speeds SP_e over all segmentation experiments e involving o, m, b, and all trials and scenes.

Table 1 lists the values of SP_{ormb} for all possible values of o, r, m, and b. Table 2 lists the difference in speeds between method MN and each of LW, LL, and AL for each operator o and object b. The second (bottom) entry in this table indicates the statistical significance (the p value) of the difference in speeds obtained by conducting a Student's t-test [36]. Table 3 shows the values of SP_{omb} for all o, m, and b.

Let $\mathbf{C}_e$ be the binary scene resulting from a segmentation experiment e. We use the notation $|\mathbf{C}_e|$ to denote the number of pixels of $\mathbf{C}_e$ with value 1. The exclusive OR operation EOR between any two binary scenes $\mathbf{C}_{e_1} = (C_{e_1}, g_{e_1})$ and $\mathbf{C}_{e_2} = (C_{e_2}, g_{e_2})$ such that $C_{e_1} = C_{e_2}$, written as $\mathbf{C}_{e_1} EOR \mathbf{C}_{e_2}$, results in a binary scene $\mathbf{C}_e = (C_e, g_e)$ where $C_e = C_{e_1} = C_{e_2}$, and, for any $p \in C_e$, $g_e(p) = 1$ if $g_{e_1}(p) \neq g_{e_2}(p)$, and $g_e(p) = 0$ if $g_{e_1}(p) = g_{e_2}(p)$. Let e_1 and e_2 be any segmentation experiments involving the same fixed object $b \in \{T_a, C_a\}$, and scene $\mathbf{C} \in \Sigma$ and let $\mathbf{C}_{e_1}$ and $\mathbf{C}_{e_2}$ be the binary scenes resulting from these experiments. We define the *repeatability of segmentation*, $RP_{e_1e_2}$, for object b in scene $\mathbf{C}$ in experiments e_1

TABLE 1
Segmentation Speeds (slices/min) SP_{ormb} for All Possible Values of o, r, m, and b

		T_a			C_a		
		E_1	E_2	E_3	E_1	E_2	E_3
LW	O_1	7.51	5.87	6.17	5.41	5.73	5.32
	O_2	2.90	3.28	3.34	4.19	3.93	3.14
	O_3	5.50	5.65	6.66	5.96	6.18	6.16
LL	O_1	5.94	10.28	7.72	7.69	7.19	7.96
	O_2	3.87	4.19	3.18	3.68	3.69	4.30
	O_3	6.32	7.70	7.37	7.96	6.98	7.83
AL	O_1	8.21	7.65	8.93	7.11	7.31	7.31
	O_2	2.52	3.35	3.19	3.11	3.40	3.99
	O_3	7.11	7.53	7.98	8.75	7.70	8.61
MN	O_1	2.97	3.07	2.78	3.86	3.44	3.37
	O_2	3.50	4.19	3.39	4.04	3.71	5.40
	O_3	3.44	4.22	3.74	4.35	4.77	5.09

TABLE 2
Difference in Speeds $|SP_{omb} - SP_{om'b}|$ (top) and its level of statistical significance (bottom) for all operators o and objects b and $m' =$ MN and $m \in \{$LW, LL, AL$\}$

	T_a			C_a		
	LW,MN	LL,MN	AL,MN	LW,MN	LL,MN	AL,MN
O_1	3.51	4.66	5.29	1.94	4.06	3.70
	0.000	0.000	0.000	0.000	0.000	0.000
O_2	0.50	0.03	0.68	0.58	0.41	0.81
	0.111	0.914	0.024	0.119	0.279	0.024
O_3	2.12	3.30	3.75	1.39	2.85	3.61
	0.000	0.000	0.000	0.001	0.000	0.000

TABLE 3
Segmentation Speeds SP_{omb} (slices/min) for Operator, Method, and Object

		SP_{omb}	
		T_a	C_a
LW	O_1	6.52	5.49
	O_2	3.17	3.75
	O_3	5.94	6.10
LL	O_1	7.98	7.61
	O_2	3.75	3.89
	O_3	7.13	7.59
AL	O_1	8.26	7.24
	O_2	3.02	3.50
	O_3	7.54	8.35
MN	O_1	2.94	3.56
	O_2	3.69	4.39
	O_3	3.80	4.74

TABLE 4
Segmentation Repeatabilities $RP_{oor_1r_2mb}$ for All Possible Values of o, r_1, r_2, m, and b

		T_a			C_a		
		E_1, E_2	E_2, E_3	E_1, E_3	E_1, E_2	E_2, E_3	E_1, E_3
LW	O_1	0.993	0.994	0.992	0.984	0.986	0.981
	O_2	0.992	0.992	0.991	0.985	0.966	0.966
	O_3	0.995	0.994	0.995	0.993	0.992	0.993
LL	O_1	0.992	0.995	0.991	0.966	0.970	0.982
	O_2	0.992	0.990	0.990	0.979	0.971	0.969
	O_3	0.994	0.993	0.993	0.981	0.984	0.986
AL	O_1	0.990	0.993	0.992	0.970	0.970	0.978
	O_2	0.987	0.985	0.987	0.977	0.974	0.973
	O_3	0.990	0.993	0.991	0.970	0.978	0.967
MN	O_1	0.977	0.978	0.977	0.972	0.974	0.972
	O_2	0.970	0.965	0.967	0.966	0.959	0.958
	O_3	0.975	0.978	0.975	0.968	0.969	0.966

and e_2 as

$$RP_{e_1e_2} = 1 - \frac{|\mathbf{C}_{e_1} EOR\, \mathbf{C}_{e_2}|}{|\mathbf{C}_{e_1}| + |\mathbf{C}_{e_2}|}. \tag{16}$$

For any fixed $o_1 \in \{O_1, O_2, O_3\}$, $o_2 \in \{O_1, O_2, O_3\}$, $r_1 \in \{E_1, E_2, E_3\}$, $r_2 \in \{E_1, E_2, E_3\}$, $m \in \{\text{LW, LL, AL, MN}\}$, and $b \in \{T_a, C_a\}$, the *repeatability of segmentation* $RP_{o_1o_2r_1r_2mb}$ by operators o_1 and o_2 in trials r_1 and r_2 using method m for object b is the average of all $RP_{e_1e_2}$ for all segmentation experiments e_1 and e_2 which are such that $e_1 \neq e_2$, e_1 and e_2 are conducted on the same input 2D scene, e_1 involves operator o_1 and trial r_1, e_2 involves operator o_2 and trial r_2, and both involve method m and object b. $RP_{o_1o_2r_1r_2mb}$ measures interoperator repeatability when $o_1 \neq o_2$, and it measures intraoperator repeatability when $o_1 = o_2$. We define *repeatability of segmentation* $RP_{o_1o_2mb}$ by operators o_1 and o_2 using method m for object b as the average of the repeatabilities $RP_{o_1o_2r_1r_2mb}$ over all combinations of trials r_1 and r_2. Finally, the *repeatability of segmentation* RP_{mb} using method m for object b is the average of all $RP_{o_1o_2r_1r_2mb}$ over all combinations of trials r_1 and r_2 and operators o_1 and o_2 and method m and object b.

In Table 4, we list segmentation repeatabilities $RP_{oor_1r_2mb}$ for all possible values of o, r_1, r_2, m, and b . Table 5 shows the difference in segmentation repeatabilities between MN and

TABLE 5
Difference in Repeatabilities $|RP_{oomb} - RP_{oom'b}|$ (top) and Its Level of Significance (bottom) for Each Operator o, Object b, and $m' = \text{MN}$ and $m \in \{\text{LW, LL, AL}\}$

	T_a			C_a		
	LW,MN	LL,MN	AL,MN	LW,MN	LL,MN	AL,MN
O_1	0.016	0.015	0.014	0.011	0.000	0.000
	0.000	0.000	0.000	0.038	0.935	0.995
O_2	0.025	0.023	0.019	0.011	0.012	0.014
	0.000	0.000	0.000	0.083	0.036	0.008
O_3	0.019	0.017	0.015	0.025	0.016	0.004
	0.000	0.000	0.000	0.000	0.003	0.588

TABLE 6
Listing of Segmentation Repeatability $RP_{o_1o_2mb}$ by Operator (o_1, o_2), Method (m), and Object (b)

		T_a			C_a		
		O_1	O_2	O_3	O_1	O_2	O_3
LW	O_1	0.993	0.989	0.987	0.984	0.974	0.975
	O_2	0.989	0.992	0.984	0.974	0.972	0.966
	O_3	0.987	0.984	0.995	0.975	0.966	0.993
LL	O_1	0.993	0.987	0.986	0.973	0.967	0.965
	O_2	0.987	0.990	0.980	0.967	0.973	0.965
	O_3	0.986	0.980	0.993	0.965	0.965	0.984
AL	O_1	0.992	0.984	0.985	0.973	0.965	0.963
	O_2	0.984	0.986	0.977	0.965	0.975	0.958
	O_3	0.985	0.977	0.991	0.963	0.958	0.971
MN	O_1	0.977	0.961	0.967	0.973	0.947	0.957
	O_2	0.961	0.967	0.959	0.947	0.961	0.944
	O_3	0.967	0.959	0.976	0.957	0.944	0.968

each of methods LW, LL, and AL for each operator and object. The second (bottom) entry in this table indicates the statistical significance of the difference in repeatability obtained by conducting a Student's t-test. Table 6 shows segmentation repeatabilities $RP_{o_1o_2mb}$ for all values of o_1, o_2, m, and b. Finally, Table 7 shows the values of RP_{mb} for all values of m and b.

The following observations may be made from the results presented in Tables 1 through 7.

There is considerable variation in speeds among operators for any given method and object. O_2's speeds differ significantly from that of O_1 and O_3. For O_2, there do not seem to be any statistically significant speed advantage for the live methods over MN. On the contrary, for both O_1 and O_3, all live methods provide statistically significant ($p < 0.0005$) speed improvement over the manual method. The speed improvement achieved for O_1 and O_3 varies between 1.5 to 3 fold. For these operators, LL and AL seem to provide definite speed improvement over LW. There is variation in speed between trials for a given operator, method, and object. This perhaps reflects the aspect of learning that is inherent in this experiment. To minimize this effect, we took two precautions. First, an experienced anatomist reviewed each slice with the operators before they started so that they understood the boundary shapes and locations. Second, the sequence of experiments was randomized so that the same object was never segmented in successive experiments (E_1, E_2, E_3 do not indicate any order in which the trials were undertaken). Learning is an interesting

TABLE 7
Listing of Segmentation Repeatability RP_{mb} by Method (m) and Object (b)

	RP_{mb}	
	T_a	C_a
LW	0.989	0.975
LL	0.987	0.969
AL	0.985	0.966
MN	0.966	0.955

phenomenon which is difficult to handle objectively. We do not know what influence learning and experience will have on the relative speeds of methods.

Repeatability for the live methods is generally better than for the manual method. Intra-operator repeatability is generally better than interoperator repeatability for all operators, methods, and objects. Repeatability is better for T_a (better defined object) than for C_a. For T_a, repeatability improvement (1.5–2.5%) is statistically significant ($p < 0.0005$) for all operators and methods compared to the manual method. This improvement is not as striking for the less well defined C_a. Repeatability decreases from LW to LL to AL to MN as the methods become less automatic (Table 7).

5. DISCUSSION AND CONCLUSIONS

We have presented a novel user-steered image segmentation paradigm and several methods under this paradigm with emphasis on reducing the time spent by the user during the process. In live-wire approach, the user initially specifies a point on the boundary using the cursor of a pointing device of the workstation. For any subsequent position of the cursor, as the user moves the cursor, a curve—an optimal path—connecting the initial point to the current cursor position is computed and displayed in real time. Optimal paths are determined by assigning a variety of features and their associated costs to every boundary element and finding minimum-cost paths via dynamic programming. As the cursor moves close to the boundary, the live wire snaps onto the boundary. If the live-wire segment adequately describes the boundary segment, the user deposits the cursor whose location now becomes the new starting point. A complete 2D boundary is specified via a set of live-wire segments in this fashion. In the live-lane approach, the user involvement is more tightly integrated with the machine's actions. To define a boundary in a given image, the user selects only an initial point and subsequently steers the cursor in the vicinity of the boundary within a lane of certain width. During the user's drawing action, points are selected intermittently by the computer and displayed in real time. Lane width is varied as a function of the cursor's speed and acceleration. We have described a novel evaluation study to objectively determine the utility of these approaches compared to manual tracing based on speed and repeatability. In those segmentation tasks where manual tracing is currently employed, these methods can substantially reduce the user involvement in the process. While the actual amount of time saved depends on the application, with the proper parameter settings and user's experience, two- to three-fold reduction seems possible. Our evaluation studies indicate that the live approaches are more repeatable than manual tracing.

We use live wire in applications where the proper parameter settings permit to segment boundaries in slices selecting a few points. In situations where a larger number of points is necessary to segment such boundaries, compromising the good performance of live wire, we use live lane. We have observed that as the users become more familiar with the behavior of the algorithms, they are able to finish segmentation quicker. When the number of data sets to be segmented is large, some time spent in fine tuning parameters and in understanding the strengths and weaknesses of the algorithms (from an operational standpoint) saves considerable time for the user in the actual segmentation process. The imaging modalities/objects for which the live methods may be most usefully applied may be characterized by any combination of the following: ultrasound, PET, SPECT, and MR images; images wherein object regions are highly textured and/or do not exhibit region homogeneity; images wherein boundaries are weak or intermittantly absent or they exist amidst strong nonobject boundaries.

There are some aspects of human–machine interaction which are difficult to objectively quantify which nevertheless influence the utility of the methods. We already mentioned learning. Another important factor is fatigue. In our experience, a particular version of live lane that may be considered to be a hybrid between LL and AL causes the least amount of fatigue on extended use. In this version, we switch between two lane widths with a click of the mouse button while tracing. One lane width is "high" (typically set to 40 pixels) and the other is "low" (typically set to 1 pixel). The "high" setting is for quick segmentation of clear boundary segments and the "low" setting is for poorly defined segments.

If the value of T_{cc} in Algorithm LW is chosen to be less than the total cost T_c of the largest actual object contour encountered in the application, the live wire computation becomes even faster. However, this constrains how far v_e can be from v_s and still guarantee that the path found between v_s and v_e is optimum (without causing saturation of the cumulative cost). If T_c is chosen such that it roughly equals the total cost of the largest live-wire boundary segments that are actually found in the application, then this strategy gives the best compromise between the speed of live wire computation and the number of points that need to be specified per contour. The behavior of this method is now somewhat like that of live lane. This may be the best strategy for live wire on computers with modest power.

A question naturally arises as to the relationship between the live paradigms and active contour methods [10, 13–16]. The principles underlying these two approaches are very different. In active contours, the users specify an initial contour which is subsequently deformed based on image-derived criteria (such as energy). Users' interaction is in specifying image-independent criteria such as external forces. In contrast, live-wiring is a tightly user-controlled process. It is up to the user as to how a "live field" is set up, through a combination of features, transforms, and training and as to how the live-wire segments are selected. The more interesting question is which method has more utility (i.e., is more time-saving and repeatable) from the users' point of view, to which we do not know the answer at present.

The methods presented in this paper differ from published works on optimum boundary finding in five aspects, all of which go toward saving user time and improving their practical utility. (1) The graph model: Instead of using pixels as nodes and pixel adjacency as undirected arcs, we use pixel vertices as nodes and each pixel edge as two directed arcs going in opposite directions. Each arc represents an oriented pixel edge. Instead of using pixels as boundary elements, we use oriented pixel edges to represent the boundary. This model allows us to track boundaries of thin and subtle parts of objects without distractions from other closely running boundaries which would be problematic in the other model. (2) Orientedness of the boundary: Because of the graph model and the method of cost assignment, we can usually assign significantly different costs to boundary segments that have opposite orientations but that otherwise have very similar features (Fig. 8). This significantly reduces distractions, improves automation, and reduces user time. (3) Cost assignment and training: Features denoted f_1, f_2, f_7, and f_8 are novel ideas. Typically, features designed around only intensity gradients are utilized. Features f_1 and f_2 take into account typical intensity inside and outside the boundary. Boundaries that have similar gradients may have very dissimilar inside and outside intensity. The idea of optimum cost assignment and of different feature transforms to convert features to costs are novel and have proven to be practically very useful. (4) Live-lane methods: These novel ideas have led to the most preferred user-steered methods among those we have evaluated, particularly the adaptive version with two lane widths—one set to "high" and the other set to "low" as described above, where the two states are controlled with a mouse button while tracing. (5) Method of evaluation: We have

conducted a rigorous statistical evaluation based on thousands of tracings on images from a real ongoing application to compare the precision and efficiency of the new methods with those of manual tracing.

Recently, we have extended these methods to facilitate the segmentation of 3D boundary surfaces [37]. In this extentsion, the user initially specifies a few 2D contours via the live methods described in this paper on several strategically selected slices that are orthogonal to the natural slices of the 3D scene. Every natural slice will now contain sufficient number of points so that, once the order in which these points occur on the natural slice is determined, the 2D boundaries in it can be completed by computing automatically the live-wire segments between points in each successive pair in the sequence. Our evaluation studies indicate that the speed of segmentation is improved by a factor of 2–5 over 2D live methods, resulting in an overall speed advantage of 3- to 15-fold over manual tracing.

ACKNOWLEDGMENT

The first author (A.X.F.) is supported by CNPq, the Brazilian Council for Technological and Scientific Development.

REFERENCES

1. J. K. Udupa and G. Herman (Eds.), *3D Imaging in Medicine*, CRC Press, Boca Raton, FL, 1991.
2. O. Kübler, J. Ylä-Jääski, and E. Hiltebrand, 3-D Segmentation and real time display of medical volume images, in *Proceedings, Computer Assisted Radiology* (H. U. Lemke et al., Eds.), pp. 637–641, Springer-Verlag, Berlin, 1987.
3. M. W. Vannier, C. M. Speidel, D. L. Rickman, L. D. Schertz, L. R. Baker, C. F. Hildeboldt, C. J. Offutt, J. A. Balko, R. L. Butterfield, and M. H. Gado, Multispectral analysis of magnetic resonance images, in *Proceedings, 9th International Conference on Pattern Recognition*, pp. 1182–1186, IEEE Comput. Soc. Press, Washington, DC, 1988.
4. M. Bomans, K. H. Höhne, U. Tiede, and M. Riemer, 3D-Segmentation of MR-images of the head for 3D-display, *IEEE Trans. Med. Imaging* **9**(2), 1990, 177–183.
5. H. E. Cline, W. E. Lorensen, R. Kikinis, and F. Jolesz, Three-dimensional segmentation of MR images of the head using probability and connectivity, *J. Comput. Assist. Tomography* **14**(6), 1990, 1037–1045.
6. M. B. Merickel, T. Jackson, C. Carman, J. R. Brookeman, and C. R. Ayers, A multispectral pattern recognition system for the noninvasive evaluation of atherosclerosis utilizing MRI, in *Proceedings, 3D-Imaging in Medicine: Algorithms, Systems, Applications* (K. H. Höhne et al., Eds.), pp. 133–146, Springer-Verlag, Berlin, 1990.
7. S. P. Raya, Low-level segmentation of 3-D magnetic resonance brain images—a rule-based system, *IEEE Trans. Med. Imaging* **9**(3), 1990, 327–337.
8. H. S. Stiehl, 3-D Image understanding in radiology, *IEEE Eng. Med. Biol. Magazine* **9**(4), 1990, 24–28.
9. J. K. Udupa, Interactive segmentation and boundary surface formation, *Comput. Graphics and Image Process.* **18**, 1982, 213–235.
10. M. Kass, A. Witkin, and D. Terzopoulos, Snakes: active contour models, *Int. J. Comput. Vision* **1**(4), 1987, 321–331.
11. S. M. Pizer, T. J. Cullip, and R. E. Fredericksen, Toward interactive object definition in 3D scalar images, in *Proceedings, 3D-Imaging in Medicine: Algorithms, Systems, Applications* (K. H. Höhne et al., Eds.), pp. 83–105, Springer-Verlag, Berlin, 1990.
12. K. H. Höhne and W. A. Hanson, Interactive 3D-segmentation of MRI and CT volumes using morphological operations, *J. Comput. Assist. Tomography*, **16**(2), 1992, 285–294.
13. S. Lobregt and M. A. Viergever, A discrete dynamic contour model, *IEEE Trans. Med. Imaging* **14**(1), 1995, 12–24.

14. W. Neuenschwander, P. Fua, G. Szekely, and O. Kübler, Initializing snakes, in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94), 1994*, pp. 658–663.
15. L. Cohen and R. Kimmel, Global minimum for active contour models: a minimum path approach, in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96), San Francisco, CA, June 1996.*
16. L. Cohen and R. Kimmel, Fast marching the global minimum of active contours, in *Proceedings of the IEEE International Conference on Image Processing, Lausanne, September 1996*, pp. 473–476.
17. I. Carlbom, D. Terzopoulos, and K. Harris, Computer-assisted registration, segmentation, and 3D reconstruction from images of neuronal tissue sections, *IEEE Trans. Med. Imaging* **13**(2), 1994, 351–362.
18. E. W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* **1**, 1959, 269–271.
19. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, Chap. 25, MIT Press, New York, 1991.
20. A. Martelli, An application of heuristic search methods to edge and contour detection, *Commun. Assoc. Comput. Mach.* **19**, 1971, 73–83.
21. U. Montanari, On the optimal detection of curves in noisy pictures, *Commun. Assoc. Comput. Mach.* **14**(5), 1971, 335–345.
22. D. Pope, D. Parker, D. Gustafson, and P. Clayton, Dynamic search algorithms in left ventricular border recognition and analysis of coronary arteries, *Proc. IEEE, Comput. Cardiol.* **9**, 1984, 71–75.
23. 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. Graphics Image Process.* **15**, 1981, 201–223.
24. A. A. Amini, T. E. Weymouth, and R. C. Jain, Using dynamic programming for solving variational problems in vision, *IEEE Trans. Pattern Anal. Mach. Intelligence* **12**(9), 1990, 855–867.
25. D. Geiger, A. Gupta, L. A. Costa, and J. Vlontzos, Dynamic programming for detecting, tracking and matching deformable contours, *IEEE Trans. Pattern Anal. Mach. Intelligence* **17**(3), 1995, 294–302.
26. J. K. Udupa, Informal research notes on optimal boundary detection, *Medical Image Processing Group, Department of Radiology, University of Pennsylvania*, Philadelphia, PA, February 1989.
27. B. S. Morse, W. A. Barrett, J. K. Udupa, and R. P. Burton, Trainable optimal boundary finding using two-dimensional dynamic programming, *Technical Report MIPG180*, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, March 1991.
28. J. K. Udupa, S. Samarasekera, and W. A. Barrett, Boundary detection via dynamic programming, *Proc. SPIE* **1808**, 1992, 33–39.
29. E. N. Mortensen, B. S. Morse, W. A. Barrett, and J. K. Udupa, Adaptive boundary detection using live-wire two dimensional dynamic programming, in *IEEE Proceedings of Computers in Cardiology, October 1992*, pp. 635–638.
30. E. N. Mortensen and W. A. Barrett, Intelligent scissors for image composition, in *Proceedings of Computer Graphics (SIGGRAPH'95), Los Angeles, CA, August 1995*, pp. 191–198.
31. A. X. Falcão, J. K. Udupa, S. Samarasekera, and B. E. Hirsch, User-steered image boundary segmentation, in *Proceedings of SPIE on Medical Imaging, Newport Beach, CA, 2710, February, 1996*, pp. 278–288.
32. J. K. Udupa, Multidimensional digital boundaries, *CVGIP: Graphical Models Image Process.* **56**(4), 1994, 311–323.
33. W. A. Barrett, P. D. Clayton, and H. R. Warner, Determination of left ventricular contours: A probabilistic algorithm derived from angiographic images, *Comput. Biomed. Res.* **13**, 1980, 522–548.
34. J. K. Udupa, D. Odhner, S. Samarasekera, R. J. Goncalves, K. Iyer, K. Venugopal, and S. Furuie, 3DVIEWNIX: An open, transportable, multidimensional, multimodality, multiparametric imaging software system, *Proceedings of SPIE* **2164**, 1994, 58–73.
35. J. K. Udupa, B. E. Hirsch, S. Samarasekera, and R. J. Goncalves, Joint kinematics via three-dimensional MR imaging, *Proc. SPIE* **1808**, 1992, 664–670.
36. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge Univ. Press, Cambridge, UK, 1986. [reprinted 1992]
37. A. X. Falcão and J. K. Udupa, Segmentation of 3D objects using live wire, *Proc. SPIE* **3034**, 1997, 228–239.