Medical Image Analysis (1996/7) volume 1, number 4, pp 331–341

Interactive live-wire boundary extraction

William A. Barrett* and Eric N. Mortensen

Department of Computer Science, Brigham Young University, Provo, UT 84602, USA

Abstract
Live-wire segmentation is a new interactive tool for efficient, accurate and reproducible boundary extraction which requires minimal user input with a mouse. Optimal boundaries are computed and selected at interactive rates as the user moves the mouse starting from a manually specified seed point. When the mouse position comes into the proximity of an object edge, a 'live-wire' boundary snaps to, and wraps around the object of interest. The input of a new seed point 'freezes' the selected boundary segment and the process is repeated until the boundary is complete. Two novel enhancements to the basic live-wire methodology include boundary cooling and on-the-fly training. Data-driven boundary cooling generates seed points automatically and further reduces user input. On-the-fly training adapts the dynamic boundary to edges of current interest. Using the live-wire technique, boundaries are extracted in one-fifth of the time required for manual tracing, but with 4.4 times greater accuracy and 4.8 times greater reproducibility. In particular, interobserver reproducibility using the live-wire tool is 3.8 times greater than intraobserver reproducibility using manual tracing.

Keywords: boundary cooling and training, speed, accuracy and reproducibility, graph searching, interactive segmentation, live-wire boundary extraction

Received November 20, 1996; revised February 6, 1997; accepted February 10, 1997

1. INTRODUCTION

Due to the wide variety of image types and content, fully automated segmentation is still an unsolved problem, while manual segmentation is tedious and time consuming, lacking in precision and impractical when applied to extensive temporal or spatial sequences of images. However, most current computer-based techniques require significant user input to specify a region of interest, initialize or control the segmentation process or perform subsequent correction to, or adjustment of, boundary points. Thus, to perform image segmentation in a general and practical way, intelligent, interactive tools must be provided to minimize user input and increase the efficiency and robustness with which accurate, mathematically optimal contours can be extracted. Previous algorithms have incorporated higher-level constraints, but still use local boundary defining criteria which increases the susceptibility to noise (O'Brien and Ezquerra, 1994; Gleicher, 1995). Other researchers (Montanari, 1971; Chien and Fu, 1974; Martelli, 1976; Ballard and Brown, 1982; Pope *et al*., 1984) have incorporated global properties for robustness and to produce mathematically optimal boundaries, but most of these methods use one-dimensional (1-D) implementations which impose directional sampling and searching constraints to extract two-dimensional (2-D) boundaries, thus requiring 2-D boundary templates, as with snakes. In addition, many of the former techniques exhibit a high degree of domain dependence and usually still require extensive user interaction to define an initial set of boundary points or region of interest, or to modify results and thereby produce accurate boundaries. Finally, these methods are often very computational or make use of iterative algorithms which limit high-level human interactivity as well as the speed with which boundaries can be extracted.

More recent boundary definition methods make use of active contours or snakes (Kass *et al*., 1987; Amini *et al*., 1990; Williams and Shah, 1992; Daneels *et al*., 1993), to 'improve' a manually entered rough approximation. After being initialized with a rough boundary approximation, snakes iteratively adjust boundary points in parallel in an attempt to minimize an energy functional and achieve an optimal boundary. The

*Corresponding author
(e-mail: barrett@cs.byu.edu)

energy functional is a combination of internal forces such as boundary curvature and distance between points and external forces like image gradient magnitude and direction. Snakes can track frame-to-frame boundary motion provided the boundary has not moved drastically. However, active contour models follow a pattern of initialization followed by energy minimization; as a result, the user does not know what the final boundary will look like when the rough approximation is input. If the resulting boundary is not satisfactory, the process must be repeated or the boundary must be manually edited.

While snakes and the live-wire technique presented here both require user interaction and make use of similar boundary features and cost functions to find optimal boundaries, the two methodologies differ in several significant ways. First, snakes iteratively compute a final optimal boundary by refining a single initial boundary approximation, whereas the live-wire tool interactively selects an optimal boundary segment from potentially all possible minimum cost paths. Secondly, live-wire boundaries are piecewise optimal (i.e. between seed points), providing a balance between global optimality and local control, whereas snakes are globally optimal over the entire contour. This piecewise optimality also allows for path cooling and inter-object on-the-fly training. Thirdly, snakes are typically attracted to edge features only within the gravity of an edge's gradient energy valley, whereas the live-wire boundary can snap to strong edge features from arbitrary distances since the live-wire search window is the entire image. Fourthly, the smoothing term for live-wire boundaries is data-driven (i.e. computed from external image gradient directions), whereas the active contour smoothing term is internal, based on the contour's geometry and point positions. Finally, the Laplacian zero-crossing binary cost feature seems to have not been used previously in active contour modes.

Interactive optimal 2-D path selection is what makes live-wire boundary snapping work and what distinguishes it from all previous techniques. As a practical matter, live-wire boundary snapping typically requires less time and effort to accurately segment an object than it takes to manually input an initial approximation to the object boundary. Live-wire boundary snapping for image segmentation was introduced initially by Mortensen *et al.* (1992) and Udupa *et al.* (1992). This paper reports on the application of the live-wire technology to medical images and its performance in terms of the speed, accuracy and reproducibility with which boundaries can be extracted (Barrett and Mortensen, 1996). In addition, several significant contributions to the original technology include:

(i) A Laplacian cost function and cursor snap for localization of boundary position.

(ii) Interleaved boundary selection and wavefront expansion for uninterrupted user interaction.

(iii) On-the-fly training for dynamic adaptation of the live-wire boundary to the current edge.

(iv) Data-driven boundary cooling for automated generation of seed points.

(v) Application to full color images.

2. LIVE-WIRE BOUNDARY SNAPPING

With live-wire boundary snapping, boundary detection is formulated as a graph searching problem, where the goal is to find the optimal path between a start node and a set of goal nodes, where pixels represent nodes and edges are created between each pixel and its eight neighbors. Thus, in the context of graph searching, boundary finding consists of finding the globally optimal path from the start node to a goal node. The optimal path or boundary is defined as the minimum cumulative cost path from a start node (pixel) to a goal node (pixel) where the cumulative cost of a path is the sum of the local costs (or edge links) on the path.

2.1. Local costs

Since an optimal path corresponds to a segment of an object boundary, pixels (or links between neighboring pixels) that exhibit strong edge features are made to have low local costs. The local cost function is a weighted sum of the component cost functionals for each of the following features. Features include the Laplacian zero crossing, f_Z, gradient magnitude, f_G, and gradient direction, f_D.

Letting $l(\boldsymbol{p}, \boldsymbol{q})$ represent the local cost for the directed link (or edge) from pixel $\boldsymbol{p}$ to a neighboring pixel $\boldsymbol{q}$, the local cost function is

$$l(\boldsymbol{p}, \boldsymbol{q}) = \omega_G \cdot f_G(\boldsymbol{q}) + \omega_Z \cdot f_Z(\boldsymbol{q}) + \omega_D \cdot f_D(\boldsymbol{p}, \boldsymbol{q}) \tag{1}$$

where each ω is the weight of the corresponding feature function. Empirical default values for these weights are $\omega_G = 0.43$, $\omega_Z = 0.43$ and $\omega_D = 0.14$.

The gradient magnitude feature, $f_G(\boldsymbol{q})$, provides a 'first-order' positioning of the live-wire boundary. Since it is a strong measure of edge strength it is heavily weighted. However, so that high image gradients will correspond to low costs, the gradient magnitude, G, is scaled and inverted using an inverse linear ramp function. That is,

$$f_G = 1 - \frac{G}{\max(G)}. \tag{2}$$

The Laplacian zero-crossing feature, $f_Z(\boldsymbol{q})$, is a binary edge feature used for boundary localization. That is, it provides a 'second-order' fine-tuning or refinement of the final boundary position. The output, I_L, of the convolution of the image with a Laplacian edge operator is made binary by letting $f_Z(\boldsymbol{q}) = 0$ where $I_L(\boldsymbol{q}) = 0$ or has a neighbor with a different sign and $f_Z(\boldsymbol{q}) = 1$ otherwise. If $I_L(\boldsymbol{q})$ has a neighbor with a different sign, of the 2 pixels, the one closest to zero represents the zero crossing or the position to localize the boundary. Thus, $f_Z(\boldsymbol{q})$ has a low local cost (0) corresponding to good edges or zero crossings and a (relatively speaking) high cost (1) otherwise. This results in single-pixel wide cost 'canyons' which effectively localize the live-wire boundary at point $\boldsymbol{q}$.

The gradient direction or orientation adds a smoothness constraint to the boundary by associating a high cost for sharp changes in the boundary direction. The gradient direction is the unit vector defined by the image gradients, G_x and G_y in the x and y directions, respectively. Letting $\boldsymbol{D}(\boldsymbol{p})$ be the unit vector which is normal to the gradient direction at point $\boldsymbol{p}$ (i.e. rotated 90° clockwise such that $\boldsymbol{D}(\boldsymbol{p}) = (G_y(\boldsymbol{p}), -G_x(\boldsymbol{p}))$, the formulation of the gradient direction feature cost is

$$f_D(\boldsymbol{p}, \boldsymbol{q}) = \frac{2}{3\pi}\left\{\cos[d_{\boldsymbol{p}}(\boldsymbol{p}, \boldsymbol{q})]^{-1} + \cos[d_{\boldsymbol{q}}(\boldsymbol{p}, \boldsymbol{q})]^{-1}\right\} \quad (3)$$

where

$$d_{\boldsymbol{p}}(\boldsymbol{p}, \boldsymbol{q}) = \boldsymbol{D}(\boldsymbol{p}) \cdot \boldsymbol{L}(\boldsymbol{p}, \boldsymbol{q})$$
$$d_{\boldsymbol{q}}(\boldsymbol{p}, \boldsymbol{q}) = \boldsymbol{L}(\boldsymbol{p}, \boldsymbol{q}) \cdot \boldsymbol{D}(\boldsymbol{q})$$

are vector dot products and

$$\boldsymbol{L}(\boldsymbol{p}, \boldsymbol{q}) = \begin{cases} \boldsymbol{q} - \boldsymbol{p} & \text{if} \quad \boldsymbol{D}(\boldsymbol{p}) \cdot (\boldsymbol{q} - \boldsymbol{p}) \geq 0 \\ \boldsymbol{p} - \boldsymbol{q} & \text{if} \quad \boldsymbol{D}(\boldsymbol{p}) \cdot (\boldsymbol{q} - \boldsymbol{p}) < 0 \end{cases} \quad (4)$$

is the normalized (i.e. unit) bidirectional link or edge vector between pixels $\boldsymbol{p}$ and $\boldsymbol{q}$ and simply returns the direction of the link between $\boldsymbol{p}$ and $\boldsymbol{q}$ such that the difference between $\boldsymbol{p}$ and the direction of the link is minimized (i.e. $\leq \pi/2$).

Thus, the dot product $\boldsymbol{L}(\boldsymbol{p}, \boldsymbol{q})$ is a horizontal, vertical or diagonal link vector (relative to the position of $\boldsymbol{q}$ in $\boldsymbol{p}$'s neighborhood) and points in a direction such that $d_{\boldsymbol{p}}(\boldsymbol{p}, \boldsymbol{q})$, the angle between unit vectors $\boldsymbol{D}(\boldsymbol{p})$ and $\boldsymbol{L}(\boldsymbol{p}, \boldsymbol{q})$, is positive and $\leq \pi/2$ [see Equation (4)]. Thus, the neighborhood link direction associates a high cost to an edge or link between two pixels that have similar gradient directions but are perpendicular, or nearly perpendicular, to the link between them. Therefore, the direction feature cost is low when the gradient direction of the two pixels are similar to each other and the link between them. Finally, in Equation (3), these dot products are converted to an angular measure and summed to $\leq 3\pi/2$ to produce the gradient direction feature f_D.

2.2. Boundary detection as graph searching

Boundary finding can be formulated as a directed graph search for an optimal (minimum-cost) path. Nodes in the graph are initialized with the local costs described above. Then, a user-selected seed point is 'expanded', meaning that its local cost is summed into its neighboring nodes. The neighboring node with the minimum cumulative cost is then expanded and the process continues producing a 'wavefront' which expands in order of minimum cumulative cost such that for any node within the wavefront, the optimal path back to the seed point is known.

Unlike some previous approaches to optimal boundary detection (Montanari, 1971; Chien and Fu, 1974; Martelli, 1976; Ballard and Brown, 1982; Pope *et al.*, 1984), this approach allows two degrees of freedom in the search, producing boundaries of arbitrary complexity. Previous approaches also required a fixed-goal node/pixel which must be specified before the search begins, whereas selection of a goal node or 'free point' within the wavefront expansion allows any goal node to be specified interactively with the resulting boundary immediately available by following pointers (i.e. boundary points) back to the seed point.

In addition, we expand nodes in order of minimum cumulative cost, creating a dynamic wavefront which expands preferentially in directions of highest interest (i.e. along edges). This requires n iterations over the wavefront for paths of length n, which is far more efficient than previous approaches which require n iterations over the entire cost matrix. Furthermore, since we maintain the wavefront in a sorted list, expansion happens at interactive rates (1–2 s for an entire 512×512 image). The cost expansion algorithm is described in Figure 1.

Figure 1 illustrates the expansion algorithm that creates a minimum total cost path map with pointers (edges connecting nodes) showing the optimal path to that point. Figure 1a shows the initial local cost map with the seed point circled. Figure 1b shows a portion of the total cost and pointer map after expansion of the seed point with total costs summed into neighboring nodes. (Note that the local cost for the seed point is zero and that diagonal costs have been scaled by the Euclidean distance.) Note also that the total cost to a node may change if a lower total cost is eventually computed from neighbors that have yet to be expanded. This is demonstrated in Figure 1c where two points have now been expanded—the seed point and the next lowest total cost neighboring node. Notice how the points diagonal to the seed point have changed total cost and direction pointers. Figures 1d–f show the total cost–direction pointer map at various stages of path expansion. As can be seen, the cost expansion produces a 'wavefront' that grows out faster along directions (edges) of lower cost.

This type of wavefront expansion has several advantages: (i) it almost always allows path expansion to keep pace with path selection; so that (ii) interactively selected paths are

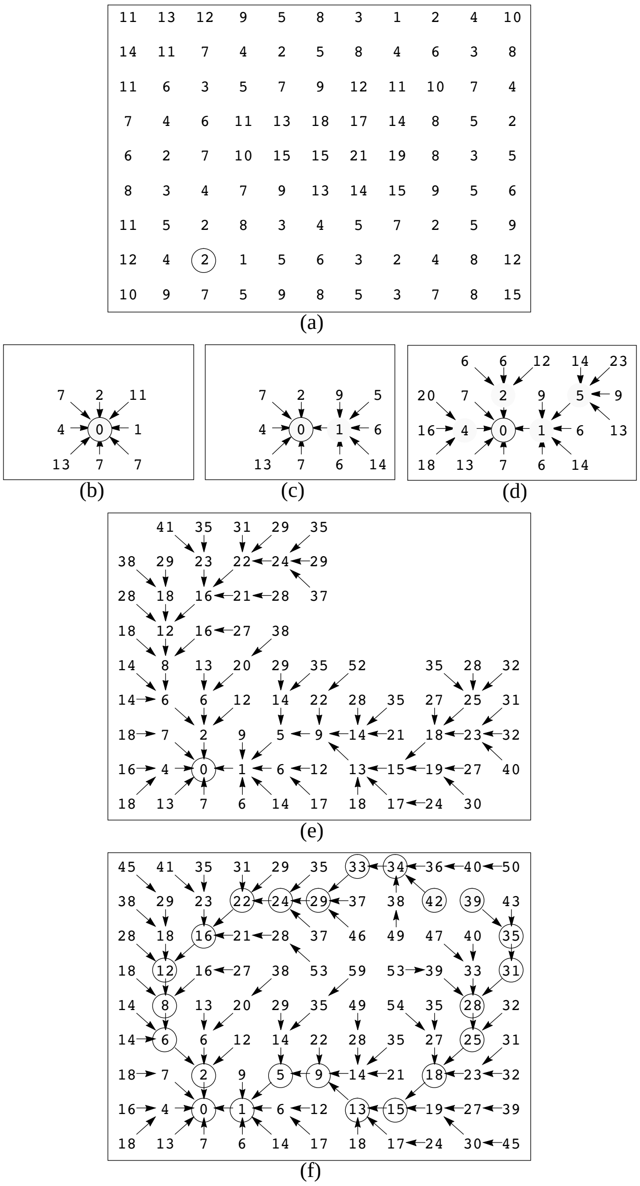


Figure 1. Cost expansion. (**a**) Local cost map, (**b**) seed point expanded, (**c**) two points expanded, (**d**) five points expanded, (**e**) 47 points expanded, (**f**) completed total cost path–pointer map with optimal paths shown from nodes with total costs of 42 and 39.

immediately available; (iii) it overcomes the limitations of previous approaches which use dynamic programming where nodes must be expanded in fixed stages and therefore require up to *n* iterations through the cost matrix for paths of length *n* and (iv) unlike many sequential edge linking programs, the optimal path can change dynamically with the movement of

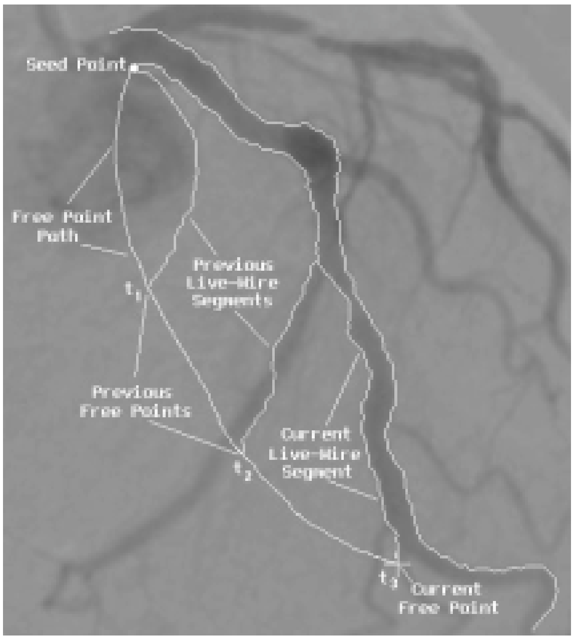


Figure 2. Continuous snap–drag of a live wire to a coronary edge (shown at times t_1, t_2 and t_3). The boundary is completed in about 2 s.

the free point. For example, note that in Figure 1f the path will change completely if the free point is moved between nodes with total costs 42 and 39.

2.3. Interactive 'live-wire' segmentation tool

Once the optimal path pointers are generated, a desired boundary segment can be chosen dynamically via the free point specified by the current cursor position. Interactive movement of the free point by the mouse cursor causes the boundary to behave like a live wire as it erases the previous boundary points and displays the new minimum cost path defined by following path pointers from the free point back to the seed point. By constraining the seed point and free points to lie near a given edge, the user is able to interactively 'snap' and 'wrap' the live-wire boundary around the object of interest. Figure 2 shows how a live-wire boundary segment adapts to changes in the free point (cursor position) by latching onto more and more of the coronary artery. Specifically, note the live-wire segments corresponding to user-specified free-point positions at times t_1, t_2 and t_3. Although Figure 2 only shows live-wire segments for three discrete time instances, live-wire segments are actually updated dynamically and interactively (on the fly) with each movement of the free point. The dynamics of the live-wire interaction are best demonstrated on the accompanying CDROM.

When movement of the free point causes the boundary to digress from the desired object edge, input of a new seed point prior to the point of departure effectively 'ties off' or freezes the boundary computed up to the new seed point and reinitiates the boundary detection starting from the new seed point. This produces a wavefront expansion emanating from the new seed point with a new set of optimal paths to every point within the wavefront. Thus, by selecting another free point with the mouse cursor, the interactive live-wire tool is simply selecting an optimal boundary segment from a large collection of optimal paths.

Since each pixel (or free point) defines only one optimal path to a seed point, a minimum of two seed points must be deposited to ensure a closed object boundary. Two seed points are sufficient to provide a closing boundary path from the free point if the path (pointer) map from the first seed point of every object is maintained during the course of an object's boundary definition. The closing boundary segment from the free point to the first seed point eliminates the need for the user to manually close off the boundary.

Placing seed points directly on an object's edge is often difficult and tedious. If a seed point is not localized to an object edge then spikes result on the segmented boundary at those seed points. To facilitate seed point placement, a cursor snap is available which forces the mouse pointer to the maximum gradient magnitude pixel within a user-specified neighborhood. The neighborhood can be anywhere from 1×1 (resulting in no cursor snap) to 15×15 (where the cursor can snap as much as 7 pixels in both x and y). So that the responsiveness of the live wire and the associated interactivity is not encumbered, a cursor snap map is precomputed by encoding the (x, y) offset from every pixel to the maximum gradient magnitude pixel in the neighborhood. Thus, as the mouse cursor is moved by the user, it snaps or jumps to a neighborhood pixel representing a 'good' static edge point.

3. ON-THE-FLY TRAINING

On occasion, a section of the desired object boundary may have a weak gradient magnitude relative to a nearby strong gradient edge. Since the nearby strong edge has a relatively low cost, the live-wire segment snaps to it rather than the desired weaker edge. This is demonstrated in the cine' CT scan of the heart in Figure 3a where the live-wire segment cuts across the corner of the ventricle and in Figure 3b where the live-wire segment snaps to the epicardial–lung boundary surface, the edge of greater strength, rather than following the endocardial edge.

Training allows dynamic adaptation of the cost function based on a previous sample boundary segment that is already considered to be 'good' (i.e. between the two red seed points in Figure 3c). Since training is performed 'on-the-fly' as part of the boundary segmentation process trained features are updated interactively as an object boundary is being defined. This eliminates the need for a separate training phase and allows the trained feature cost functions to adapt within the object being segmented (Figure 3a) as well as between objects in the image (Figure 3b). Thus, training overcomes the problems in Figures 3a and b with no additional seed points.

The idea of training is to follow the edge of current interest, rather than simply the strongest, by associating low costs with current edge features and relatively higher costs with edge features that are not characteristic of the current edge. For example, Figure 3d shows a distribution of the gradient magnitudes associated with the light-to-gray endocardial intensities in Figures 3a–c compared with a distribution of the relatively stronger gray-to-dark epicardial intensities. Since it is desirable in this case to follow the weaker gradient, we weight the gradient magnitude feature by using as the cost function the inverted distribution corresponding to that edge (Figure 3e).

This style of training was introduced in Barrett *et al.* (1980). Here, the training is dynamic, making use of the distribution corresponding to the previous segment on the edge of current interest. Hence, the term, on-the-fly training.

Another example of training is shown in Figures 3f and g where the live wire is being used to extract thin bone structure around the orbital region. Notice that without training the live wire snaps to the stronger air–tissue boundary (Figure 3f). However, with training the live wire adheres to the relatively weaker interior boundary of interest.

In general, training is facilitated by building a distribution of a variety of features (notably image gradients) from the training segment. As is done with the gradient feature itself, distributions are inverted and scaled so that features associated with the 'good' training segment have lower costs. To allow training to adapt to slow (or smooth) changes in edge characteristics, the trained cost functions are based only on the most recent or closest portion of the currently defined object boundary. A monotonically decreasing weight function (either linearly or Gaussian based) determines the contribution from each of the closest t pixels. The training algorithm samples the precomputed feature maps along the closest t pixels of the edge segment and increments the feature distribution element by the corresponding pixel weight to generate a distribution for each feature involved in training. Since training is based on learned edge characteristics from the most recent portion of an object's boundary, training is most effective for those objects with edge properties that do not change radically as the boundary is traversed (or at least change smoothly enough for the training algorithm to adapt).

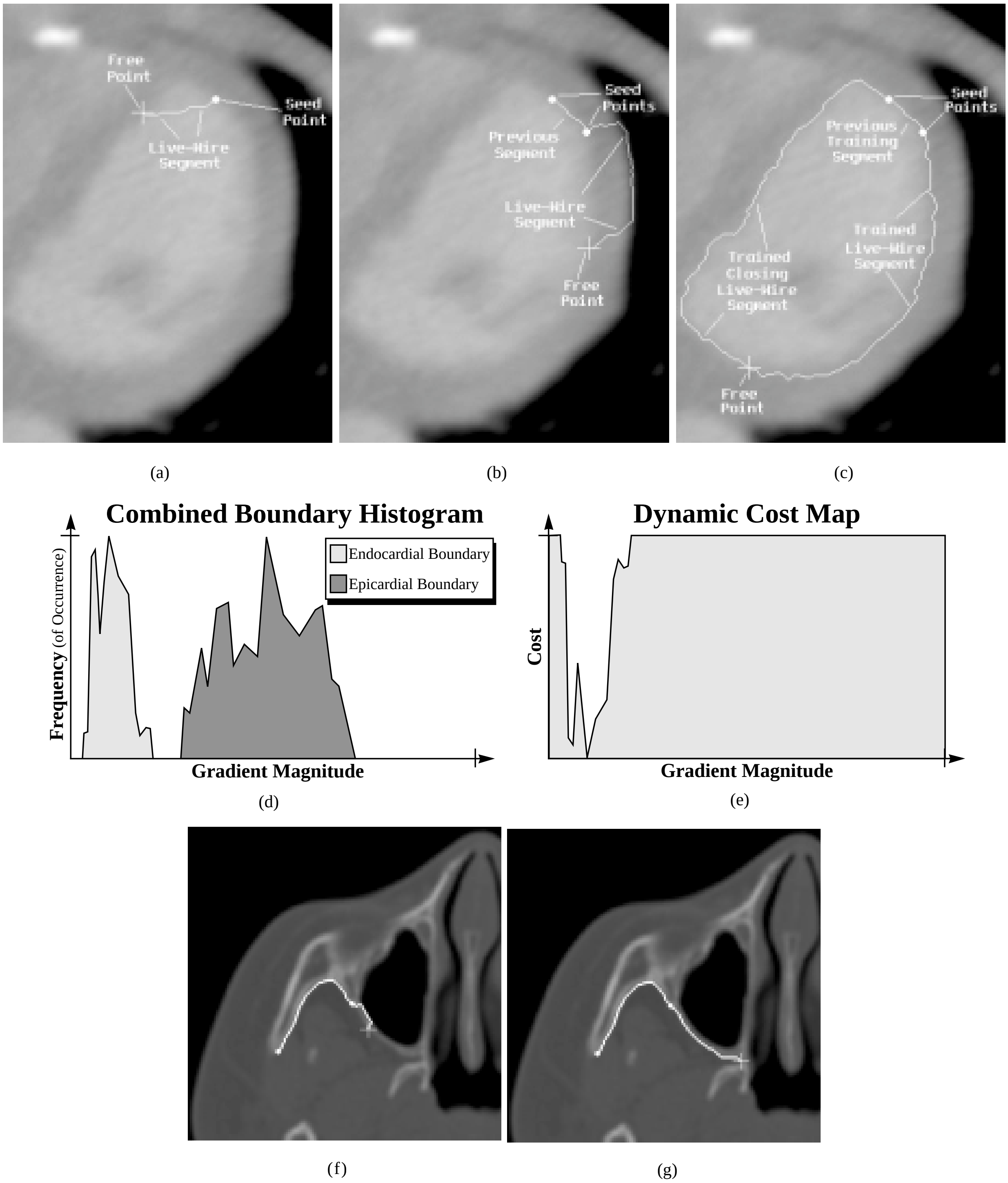
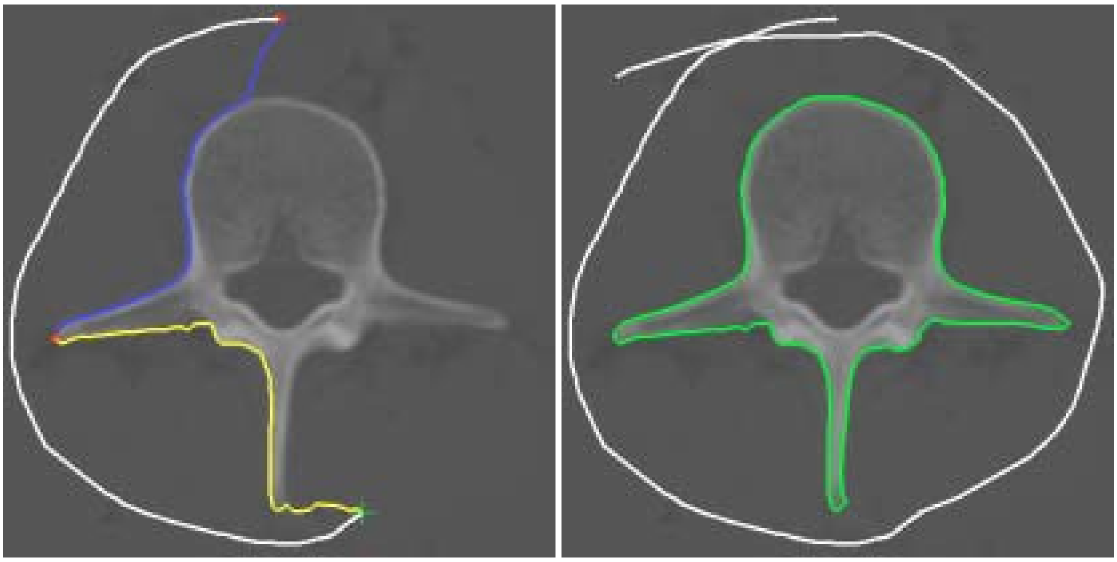


Figure 3. CT slice—mid heart. Without training live wire 'cuts' a corner, (**a**), or snaps to a lung boundary, (**b**). With training, (**c**), the correct boundary is followed. Panel (**d**) shows distributions of gradient magnitudes corresponding to the light-to-gray endocardial boundary in (**c**) and the gray-to-dark epicardial boundary snapped to in (**b**). (**e**) The cost function uses the inverse gradient magnitude to give preference to the light-to-gray endocardial boundary. (**f**) Without training the live wire snaps to the air–tissue boundary of the orbit. With training, (**g**), the live wire adheres to the inner bone–tissue boundary.



(a) (b)

Figure 4. (**a**) Cooling of a live-wire boundary (blue segment) as the cursor path (shown in white) continues to position of current free point. (**b**) Path cooling, combined with boundary clipping, allows extraction of an entire boundary with a very general cursor path (in white).

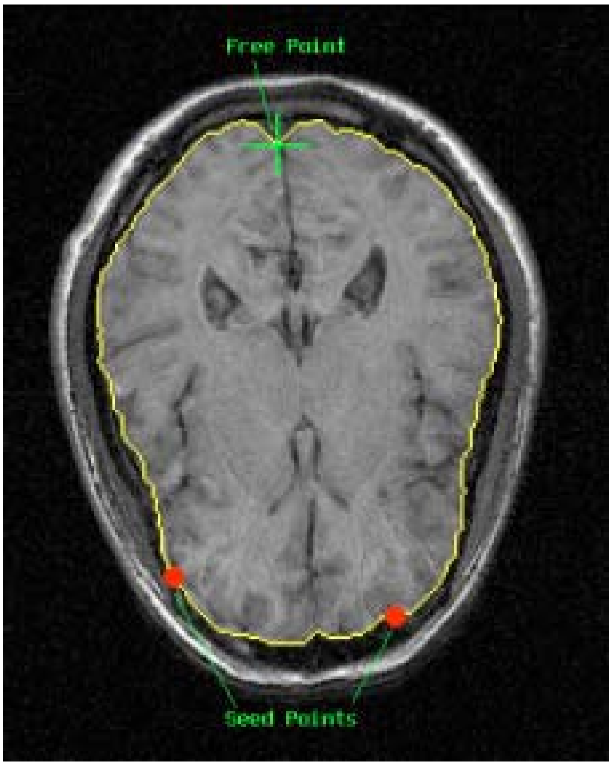


Figure 5. MRI brain boundary extracted in 2.3 s using cooling and on-the-fly training.

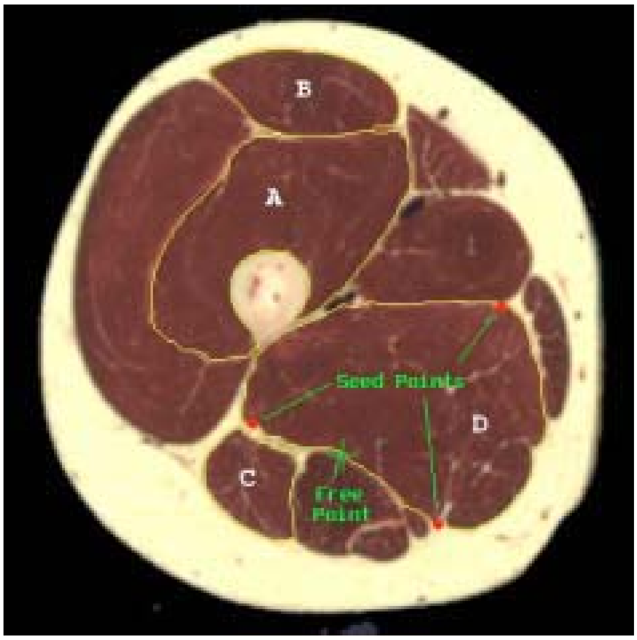


Figure 6. Mid-thigh section—Visible Human project. Live wire separates muscle groups in seconds, although they touch, are of similar color and contrast and have weak gradients at junctures.

The training length is typically short (in the range of 32–64 pixels) to permit local dependence (prevent trained features from being too subject to old edge characteristics) and thereby allow it to adapt to slow changes.

4. DATA-DRIVEN PATH COOLING

As described in Section 2.3, closed object boundaries can be extracted with as few as two seed points. However, more than two seed points are often needed to generate accurate boundaries. Typically, two to five seed points are required for boundary definition but complex objects may require many more. Even with cursor snap, manual placement of seed points can be tedious and often requires a large proportion of the overall boundary definition time.

The idea of path cooling was introduced in Section 2.3. Namely, that depositing a new seed point by fixing the free point causes the previous boundary segment to cool or freeze, meaning that the previous segment is now fixed and is no longer part of the live-wire boundary—the live-wire boundary is now rooted at the new seed point. Since this kind of path cooling requires the user to manually input new seed points with a click of the mouse, it would be preferable to have seed points deposited, and associated boundary segments freeze automatically, as a function of image data and path stationarity. Thus, automatic seed point generation is the motivation behind data-driven path cooling.

Automatic seed point generation relieves the user from placing most seed points by automatically selecting a pixel on the current active boundary segment to be a new seed point. Selection is based on ‘path cooling’ which in turn relies on path coalescence. Though a single minimum cost path exists from each pixel to a given seed point, many paths ‘coalesce’ and share portions of their optimal path with paths from other pixels. Due to Bellman’s principle of optimality, if any two optimal paths from two distinct pixels share a common point or pixel, then the two paths are identical from that pixel back to the seed point. This is particularly noticeable if the seed point is placed near an object edge and the free point is moved away from the seed point but remains in the vicinity of the object edge.

Though a new path is selected and displayed every time the mouse cursor moves, the paths are typically all identical near the seed point and only change locally to the free point. As the free point moves farther away from the seed point, the portion of the active ‘live-wire’ boundary segment that does not change becomes longer. New seed points are generated automatically at the end of a stable segment (i.e. that has not changed recently). Stability is a function of (i) time on the active boundary and (ii) path coalescence (number of times the path has been drawn from distinct seed points). This measure of stability provides the live-wire segment with a sense of ‘cooling’. Since the degree to which paths will coalesce over a given interval is a function of the underlying data (noise, gradient strength, variability in geometry and brightness of object boundary), we say that path cooling is data driven. Pixels that are on a stable segment sufficiently long will freeze, automatically producing a new seed point.

In Figure 4a we see that the first live-wire segment around the left-hand side of the vertebrae has become frozen (turning blue) causing a second seed point to be generated automatically at the tip of the left spinous process while the cursor path continues to the position of the current free point. In Figure 4b, path cooling, combined with boundary clipping, allows the entire boundary to be extracted with a very general cursor path, shown in white.

Path cooling and training were used to extract the boundary in Figure 5 while automatically generating an additional seed point.

5. RESULTS

Figures 2–6 illustrate the application of live-wire boundary extraction for a variety of medical image types: Figure 2 (X-ray projection angiography), Figure 3 (CT), Figure 4 (CT), Figure 5 (MRI) and Figure 6 (a color photograph (mid-thigh) from the Visible Human Project). Table 1 shows the times and the number of seed points required to extract boundaries in these images.

As shown in Table 1, the average time required to extract the boundaries shown in Figures 2–6 was just over 3 s with an average of a little under three seed points needed. (Recall that two seed points are required.) A more detailed study shows that, for an experienced user, the time required to extract boundaries with the live-wire tool is roughly 4.6 times less than for manual tracing. In other words, the average time required to manually trace the boundaries in Figures 2–6 would be about 13.7 s. However, although live-wire boundary extraction is only four to fives times faster, it is worth pointing out that live-wire boundaries are also much more accurate and reproducible, as shown below. In other words, to get the same kind of accuracy and reproducibility with manual tracing, many times more effort in manual tracing would need to be expended.

Figure 6 demonstrates application of the live-wire tool to a full-color image—one in which the separations between muscle groups are subtle, but visually noticeable. This is a classic example of where traditional edge-following or region-growing schemes would have difficulty due to weak gradients, touching objects and similar color or contrast. However, the live-wire tool snaps to these boundaries of separation quite easily.

Table 1. Times to extract boundaries for the anatomy contained in Figures 2–6 with a number of seed points used or automatically generated.

Figure	Anatomy	Time (s)	Seed points	Training used	Cooling used
2	Coronary (right side)	2.02	2	N	N
	Coronary (left side)	3.50	3	N	N
3	Left ventricle	3.71	2	Y	N
4	Brain	2.30	3	Y	Y
5	Lumber spine	5.90	4	N	Y
6	Thigh muscle A	6.40	5	N	N
	Thigh muscle B	1.33	2	N	N
	Thigh muscle C	1.31	2	N	N
	Thigh muscle D	3.24	3	N	N
	Average:	3.30	2.89		

Y = training/cooling used, N = training/cooling not used.

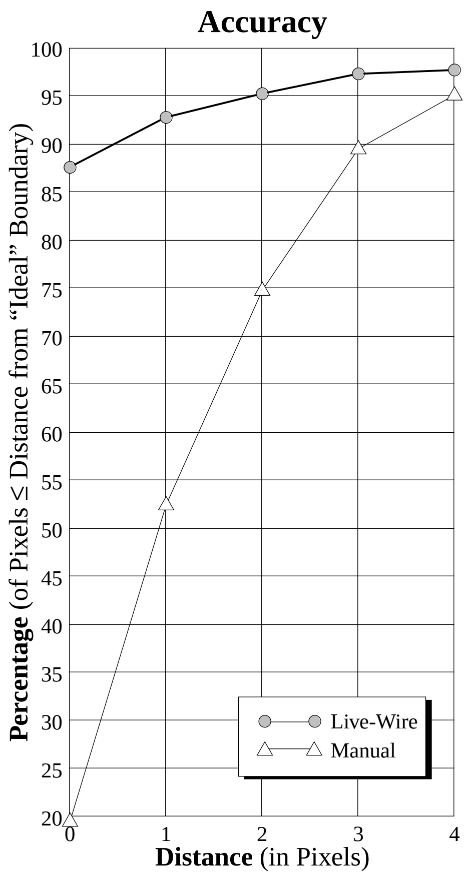


Figure 7. Average accuracy comparison between live-wire and manually traced boundaries for eight users.

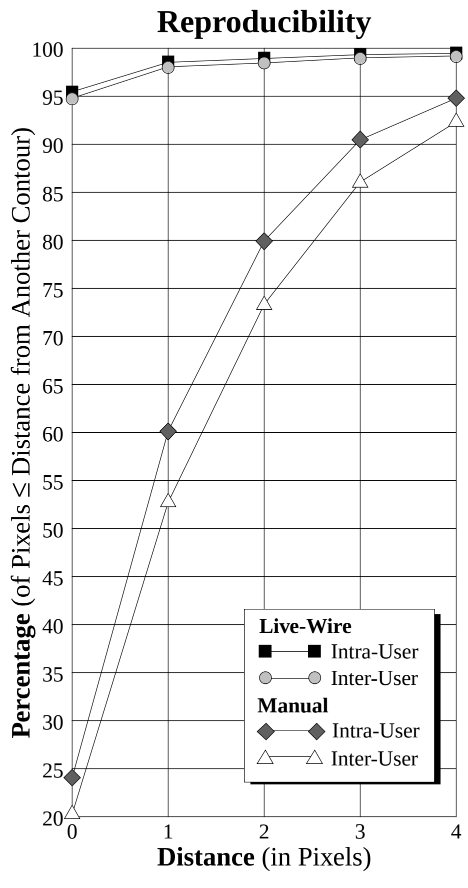


Figure 8. Average reproducibility comparison between live-wire and manually traced boundaries for eight users.

Figure 7 compares graphically the live-wire boundary accuracy with manual tracing. The graph shows the average time and accuracy from a study where eight untrained users were asked to define the boundaries of five objects. Each user spent a few minutes becoming familiar with the live-wire tool as well as a manual tracing tool before defining the boundaries of the five objects. Each boundary was defined multiple times by each user with both the live-wire and the manual tracing tool so that inter- and intra-observer reproducibility could also be measured (Figure 8).

The graph in Figure 7 shows that only 20% of the manually defined boundary points corresponded to the 'true' boundary positions as determined by an expert, whereas 87% of the boundary points determined with the live-wire tool matched those chosen by the expert. In other words, the accuracy obtained with the live-wire tool was 4.4 times (87/20) greater. On the other hand, 52% of the manually specified boundary points were within 1 pixel of the position chosen by the expert while 93% of the live-wire boundary points fell within that range. A range of up to 4 pixels was chosen since the curves asymptote and connect at that point.

A similar study was performed to compare reproducibility for both the live-wire and the manual tracing tools. Eight users extracted five object boundaries five times with the live-wire tool and the same object boundaries three times with the manually tracing tool. The results are shown in Figure 8. As the graph shows, for a given user (intra-observer), slightly more than 95% of the same points were chosen consistently over the five repeated trials using the live-wire tool. What is even more significant is that virtually the same percentage was obtained when all live-wire boundaries were pooled (inter-observer), demonstrating dramatically that the boundaries are virtually identical regardless of which user is performing the task. This is in striking contrast to the inter- and intra-observer reproducibility shown for the manual tracing tool. (High variability for manual tracing is a well known problem.) In particular, the inter-observer reproducibility is roughly 4.8 times (95/20) better for live-wire boundaries at the zero-pixel tolerance, with the curve asymptoting quickly to 100% for higher error tolerances. Perhaps the most striking result is that the inter-observer reproducibility using the live-wire tool is 3.8 times (95/25) greater than the intra-observer reproducibility at the zero-error tolerance level. This, of course, shows that we will have much better consistency with the live-wire tool regardless of who is performing the boundary extraction than we would just having a single person doing the boundary extraction manually. When taken in conjunction with the accuracy and timing results, this is significant indeed.

6. CONCLUSIONS AND FUTURE WORK

Live-wire boundary extraction provides an accurate, reproducible and efficient interactive tool for image segmentation. Across the board, the live-wire methodology is roughly four to five times as fast, as accurate and as reproducible as manual methods. Extraction of boundaries with the live-wire tool is completed in the amount of time that many semiautomated methods require just to initialize the process. In fact, and in sharp contrast to tedious manual boundary definition, or iterative compute–edit–compute... strategies, object extraction using the live wire is almost fun.

As illustrated in the results section, the technique is robust across a wide variety of imaging modalities including X-ray angiography, CT and MRI, and we believe it would be particularly effective for ultrasound images. The greatest difficulty that we have discovered with the tool has to do with the level of boundary detail the user wishes to extract and the interaction commensurate with that effort. For example, user interaction with the live-wire tool becomes greatest for objects containing hair-like boundaries where the user must finesse the live wire around the object structure. However, even in this case, the live-wire approach in far superior to manual boundary definition. Nevertheless, these types of problems invite the development of complimentary region-based interactive segmentation tools, which we are currently exploring.

The live-wire tool is intuitive to use and can be applied to black and white or color images of arbitrary content and complexity. There are many rich extensions of this work, including (i) making use of the weighted zero crossings in the Laplacian to determine boundary position at the sub-pixel position and (ii) extension of the graph search and application of the live-wire snap and training tools to spatial or temporal image sequences.

REFERENCES

Amini, A. A., Weymouth, T. E. and Jain, R. C. (1990) Using dynamic programming for solving variational problems in vision. *IEEE Trans. PAMI*, 12, 855–866.

Ballard, D. H. and Brown, C. M. (1982) *Computer Vision.* Prentice Hall, Englewood Cliffs, NJ.

Barrett, W. A., Clayton, P. D. and Warner, H. R. (1980) Determination of left ventricular contours: a probabilistic algorithm derived from angiographic images. *Comp. Biomed. Res*, 13, 522–548.

Barrett, W. A. and Mortensen, E. N. (1996) Fast, accurate and reproducible live-wire boundary extraction. *Visualization in Biomedical Computing'96*, pp. 183–192.

Chien, Y. P. and Fu, K. S. (1974) A decision function method for boundary detection. *Comp. Graphics Image Process.*, 3, 125–140.

Daneels, D., Campenout, D., Niblack, W., Equitz, W., Barber, R., Bellon, E. and Fierens, F. (1993) Interactive outlining: an improved approach using active contours. *SPIE Proc. Storage and Retrieval for Image and Video Databases*, Vol. 1908, pp. 226–233.

Gleicher, M. (1995) Image snapping. *Computer Graphics (SIGGRAPH'95 Proc.)*, pp. 191–198.

Kass, M., Witkin, A. and Terzopoulos, D. (1987) Snakes: active contour models. *Proc. 1st Int. Conf. on Computer Vision*, pp. 259–268.

Martelli, A. (1976) An application of heuristic search methods to edge and contour detection. *Commun. ACM*, 19, 73–83.

Montanari, U. (1971) On the optimal detection of curves in noisy pictures. *Commun. ACM*, 14, 335–345.

Mortensen, E. N., Morse, B. S. and Barrett, W. A. (1992) Adaptive boundary detection using 'live-wire' two-dimensional dynamic programming. *IEEE Proc. Computers in Cardiology*, pp. 635–638.

Mortensen, E. N. and Barrett, W. A. (1995) Intelligent scissor for image composition. *Computer Graphics (SIGGRAPH'95 Proc.*), pp. 191–198.

O'Brien, J. F. and Ezquerra, N. F. (1994) Automated segmentation of coronary vessels in angiographic image sequences utilizing temporal, spatial and structural constraints. *Visualization in Biomedical Computing '94*, SPIE, Vol. 2359, pp. 25–37.

Pope, D. L., Parker, D. L., Gustafson, D. E. and Clayton, P. D. (1984) Dynamic search algorithms in left ventricular border recognition and analysis of coronary arteries. *IEEE Proc. Computers in Cardiology*, pp. 71–75.

Udupa, J. K., Samarasekera, S. and Barrett, W. A. (1992) Boundary detection via dynamic programming. *Visualization in Biomedical Computing '92*, pp. 33–39.

Williams, D. J. and Shah, M. (1992) A fast algorithm for active contours and curvature estimation. *CVGIP: Image Understanding*, 55, 14–26.