Mixed Initiative Interactive Edge Detection

Eric Neufeld, Haruna Popoola, David Callele and David Mould

Department of Computer Science
University of Saskatchewan

Abstract

Interactive edge detection is used in both graphics art tools and in tools for building anatomical models from serially sectioned images. To build models, contours are traced and later triangulated. Contour tracing is time-consuming because of the fidelity and quantity of points needed, and expensive because of the background training required of individuals who do the tracing. Here we report extensions to interactive edge detection that reduce errors and effort. Our key contribution is a simple feedback interface called the *leash*, currently implemented as an extension to Intelligent Scissors, that lets the human user 'lead' the edge detection algorithm along a contour, but also helps the user to anticipate errors and provide immediate corrective feedback.

Key words: Edge detection, 3d modeling, contour data, mixed-initiative computing, intelligent graphics, Intelligent Scissors

1 Introduction

Databases of serially sectioned images such as the Visible Male and Visible Female have recently generated activity in anatomical model construction. Using tools such as Surfdriver [13], a user, usually an anatomist, traces contours on a series of cross-sectional images containing a target object. The contours are then assembled into 3D models used for classroom instruction and for distance learning. The Virtual Human Embryo Project at LSU [14] uses this process.

The contour tracing process can be time-consuming because a large number of points are required for an accurate and aesthetically pleasing model. It can also be expensive because tracing must be done by a user who is sufficiently trained to recognize the true boundaries in different types of image data. Moreover, the image data may have imperfections or flaws that require the user to interpolate. Figure 1 shows an actual set of finished contour traces.

Edge detection techniques are an obvious part of the solution, but there are four problems integrating existing techniques into our target domain. First, our typical user does not want to see all of the edges in an image (as when filters are applied to an image), but rather, edges around a specific object. Second, the user may want to trace a visually nonhomogeneous region. This limits the applicability of region-finding algorithms [6,7] that cluster similar pixels in the neighbourhood of a point selected by the user, and then outline the region. Third, the algorithm may be fooled by lighting variations or by image flaws. The user subsequently spends a lot of time identifying, and then correcting the algorithm's errors, which may be more time-consuming than simply manually tracing the boundary. Smart interactive tools that work well on ideal images also fail on images with similar variations and flaws.

Our solution to these concerns is a mixed-initiative [5] interactive contour-tracing tool called the *leash*. Like related tools [10,15], the user first selects a *seed point* on an object boundary, then drags the mouse in the general direction of a *desired contour*. The tool creates the illusion of a 'leash' leading a new contour around a boundary (Figure 7). When image flaws divert the new contour from the desired direction, information in the leash helps the user quickly correct the algorithm.

The rest of the paper is organized as follows. The next section reviews salient features of previous work on interactive edge detection [10]. Section 3 also includes descriptions of some typical feature cost functions and is provided to give readers unfamiliar with edge detection a feel for how such algorithms find features and how and why they go wrong. Later sections describe our extensions, with attention to the operation of the leash and its basic performance results.

2 Background

In an interactive edge detection approach like Intelligent Scissors *(IS)* [10] or snakes [15], the user first selects a seed point on the object boundary, then drags the mouse in the general direction of some desired contour, which generates a *live-wire*, a least-cost path from the seed point to the cursor, that makes the line appear to "snap" to the desired contour. The cost function is just the sum of several pixel-based feature costs, reviewed in the next section.

Because the live-wire runs from the seed point to the cursor, there is usually a point where the live-wire leaves the desired contour, at which point it is necessary to *recalibrate* the algorithm.

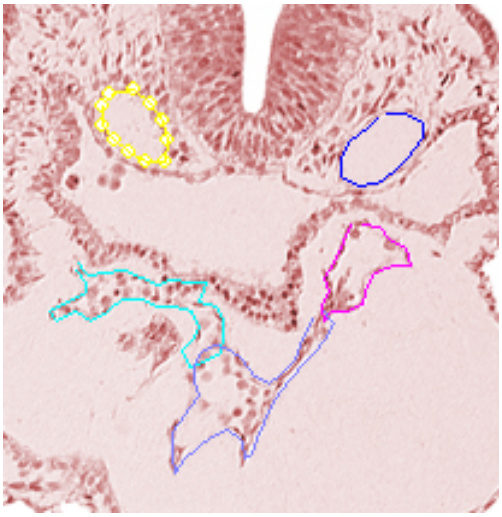
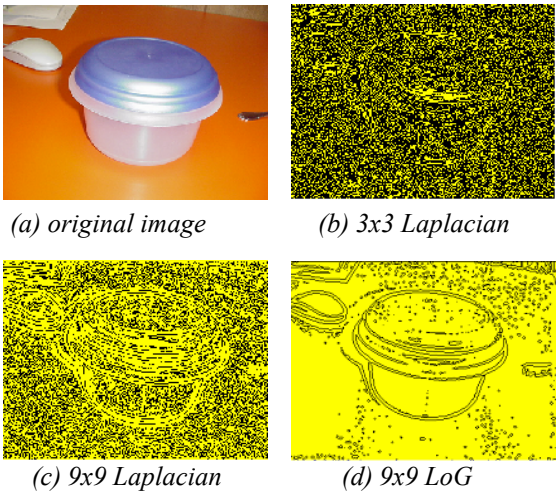


Figure 1. Contour traces, virtual human embryo

$$\begin{bmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{bmatrix}$$

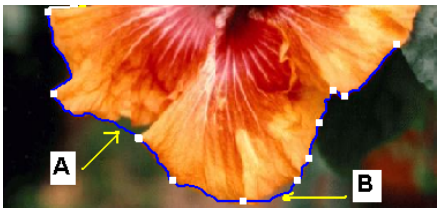
Figure 2. 9x9 LoG



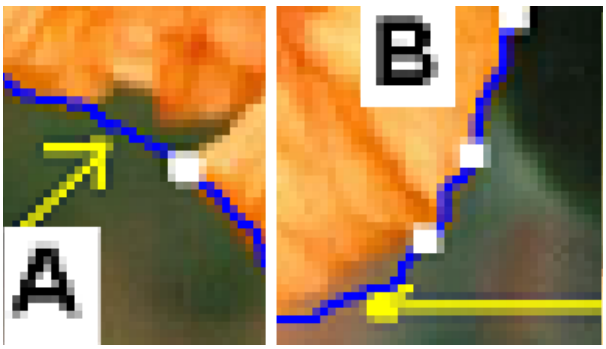
(a) original image *(b) 3x3 Laplacian*

(c) 9x9 Laplacian *(d) 9x9 LoG*

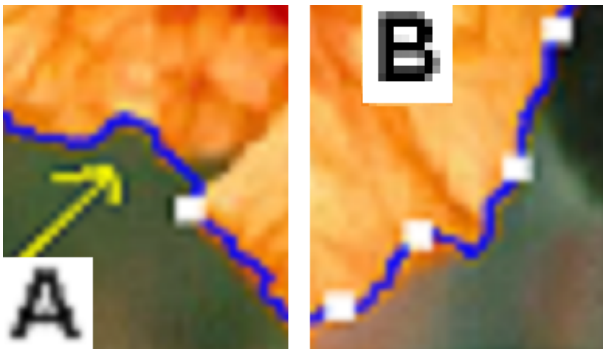
Figure 3. Effect of zero crossing detectors



(a) Bottom of hibiscus image



(b) Detail of difficult areas, using Laplacian



(c) Detail of difficult areas, using LoG

Figure 4. Detail of flower outline

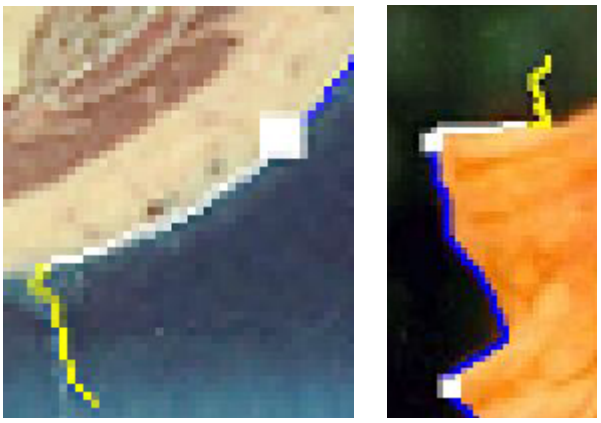


Figure 5. Detail of leash showing seed points.

Recalibration entails determining a new seed point (*re-seeding*), which implies recalculating all dynamic feature costs. This recalculation is linear in the number of image pixels, and may incur the expense of generating feature images. One simple recalibration rule is to generate a new seed point every *n* pixels along the live wire. IS also uses persistence-based heuristics to 'cool' pixels onto the desired contour. (For each pixel, IS records the total time that pixel has been on the live-wire and the number of times it has been on the live-wire. Once these values exceed some empirically determined threshold, the pixels become part of the boundary.)

On images of reasonable quality, IS performs smoothly, and without user intervention. The present paper addresses a concern that arises in domains like anatomical model construction where image quality is not always ideal. For example, Figure 1 shows an extreme actual case. The two upper contours are arteries, but there are few clues as to the structure in the image. Less extreme cases arise if the image or cross-section is damaged, incomplete, or the desired contour is partially occluded. In short, though images with such features could most benefit from smart tools, these tools are fairly easily fooled by noise. Another concern is that the recalibration trigger (pixel persistence) may be overly sensitive to vagaries of the user's hand movements.

These concerns led us to reformulate interactive edge detection as mixed-initiative computing [5], closely related to human-centred computing [3], and to cognitive prostheses in the AI community [4]. These approaches see the human and the computer as a single system comprised of two synergistic entities. In addressing the current problem domain, we spent considerable time tailoring the edge-detection technologies to our domain. In the end, we found ourselves thinking about how best to enable the human to exploit the algorithm's strengths, while avoiding its weaknesses.

Ultimately, this led us to the idea of the leash. The leash is constructed by breaking the live-wire into two differently coloured parts. The first part consists of pixels the algorithm believes are probably on the desired contour. The second part, the leash, is just the remaining pixels, pixels that fall below some probabilistic threshold of being on the desired boundary. The user has the impression of holding the leash with the mouse, with the leash leading the new boundary along. The leash also provides immediate feedback about where the algorithm might go wrong, so that the user can take appropriate action. Although the idea is simple, segmenting the live-wire optimally is also a challenging problem.

3 Basic Cost Functions

We review only the feature costs relevant here. The IS [10] least cost path sums individual *link costs* between adjacent pixels on the live-wire. To work well, the link cost aggregates three features—Laplacian zero crossing, gradient direction, and gradient magnitude.

3.1 Laplacian Zero Crossing

I_L, the image resulting from convolving an image I with a Laplacian kernel [1], approximates the second derivative of an image. Thus, if $f_Z(q)$ denotes the Laplacian zero-crossing feature cost at point q, i.e.,

$$f_Z(q) = \begin{cases} 0, & if \quad I_L(q) = 0 \\ 1, & if \quad I_L(q) \neq 0 \end{cases},$$

then this feature penalizes sudden intensity changes. Convolving an image with a Laplacian kernel yields few actual zeros; in practice a zero crossing is indicated by neighboring pixels (left/right, or up/down) of opposite signs.

3.2 Gradient Direction

This feature, written $f_D(p,q)$, penalizes sharp changes in boundary direction [2]. Adjacent boundary pixels with similar gradient direction but with a gradient direction (nearly) perpendicular to the link between the pixels have a high gradient cost. For brevity, we do not describe the calculation here, but refer the reader to [10].

3.3 Gradient magnitude

The gradient magnitude G is the vector sum of I_x and I_y, the partial derivatives of I in the x and y directions respectively. A high gradient magnitude identifies a strong edge feature [10]. In this application, it is scaled and inverted to favour high gradients. If f_G denotes the gradient magnitude feature cost, then

$$G = \sqrt{I_y^{\,2} + I_x^{\,2}},$$

and

$$f_G = 1 - \frac{G'}{\max(G')},$$

where $G' = G - \min(G)$. Gradient magnitude costs for horizontal and vertical neighbors of a pixel are scaled by $1/\sqrt{2}$ to correct for different distances between axis-aligned and diagonal pixel neighbors.

3.4 Cost Function

Many other useful features are known, but are not relevant here. The above features are scaled to the unit interval. Then, the link cost C between pixels p and q is

$$C(p,q) = \sum_X w_X f_X,$$

where the f_X range over the preceding three features and the w_X are associated empirically determined weights.

4 Interactive Edge Detection Extensions

4.1 Laplacian of Gaussian

Our first contributions were extensions to IS that become important for optimizing the performance of the leash, discussed in Section 5.

Mortensen and Barrett [10] use Laplacian filters to find zero-crossing points that indicate edge pixels in an image. It is known that an edge detector that reduces noise in an image before detecting edges in the image gives better results [12]; Marr and Hildreth's Laplacian-of-Gaussian (LoG) [8] is such a filter.

Figure 2 shows a 9x9 discrete approximation of the combination of the Laplacian operator and the Gaussian function.

Figure 3 shows the effect of applying (b), a 3 x 3 Laplacian, (c), a 9 x 9 Laplacian, and (d), a 9 x 9 LoG edge detector to an image. The lower right image in Figure 3 shows that noise is greatly reduced using a 9 x 9 LoG.

Figure 4(a) shows the bottom of the hibiscus image that we used to test the LoG. The regular Laplacian had difficulty following the outline of the flower at the points marked **A** and **B,** better seen in Figure 4(b).

The diagrams suggest that the feature calculations in the first image respond to noise by treating the lighter background area as part of the petal in areas **A** and **B**. The LoG reduces noise enough to let the feature calculation permit a noticeable detour near **A**. In both cases, the mouse was at approximately the same distance from the object boundary during tracing. (If the mouse is close to the boundary, the two perform similarly.)

4.2 Minimizing Recalibrations

The algorithm must be recalibrated reasonably often to accommodate gradual changes in pixel intensity and gradient direction, but recalibrating too often is costly because it requires recalculating a least cost path from the new seed point to every pixel in the image. Although straightforward dynamic programming solves this reasonably quickly [10], it is expensive to do for each new pixel. Regardless, a new seed point must be selected at least whenever the live-wire deviates from the desired boundary.

Because the feature cost function is biased against sharp turns even when the desired boundary actually makes such a turn, we considered ways to distinguish between genuine and spurious turns by using mouse movements to approximately infer which turns were legitimate. Because of the vagaries of hand motions, basing decisions on immediate pairs of successive positions is unwise. However, increasing the distance between pairs loses information. After some experimentation, we found that using points three cursor movements apart (i.e., every three *MouseMove* events) worked well.

This approach sometimes failed to detect turns and sometimes detected false turns. We attributed this to the method used for recording mouse motion. When a user moves too quickly while tracing an object boundary, some motions are not recorded, which subsequently affects the mouse tracker, just as the persistence approach to seed generation can be fooled by the user taking a break, or slowing down to concentrate. To avoid limiting the user to a particular tracing speed, we also initiated seed point generation whenever the length of the live-wire reached some empirically determined value. This is called *fixed rate initiation.*

However, tradeoffs remain. It may be necessary to initiate seed point generation before this specified length is reached. For example, when a user makes a turn, it may become necessary to drop a seed point before the specified length is reached if the algorithm deviates from the desired boundary (e.g., along the cat's ears, see Figure 8). This can be addressed by forcing the algorithm to re-seed, for example, with a mouse click.

Combining mouse motion and the fixed rate initiation approaches results in more re-seeding events. The fixed rate initiation approach is more reliable and also gives the user the flexibility to choose the re-seeding rate.

4.3 Optimal Seed Point Selection

Regardless of the mechanism that triggers re-seeding, the next problem is selecting a new seed point on the live-wire that maximizes the number of correct new pixels added to the contour.

We experimented with a range of techniques for optimal seed point selection. It is necessary to keep in mind that the path from the last seed point to the cursor is a least-cost path; that is, in an important sense, it is the feature cost function leading the live-wire astray. Looking at the partial path onscreen, there is a usually a visually obvious point at which it departs the partial contour and deviates towards the cursor. Because this point is an obvious discontinuity onscreen, it stands to reason that it corresponds to a feature in the path cost function. The problem then is how to find the feature without reporting false positives.

Because features are locations where the incremental cost is very high, we avoided false positives by modulating the incremental cost with a function such as $e^{-r/\lambda}$, where r is the distance from the cursor, preferring points later in the sequence.

We first considered gradient magnitude as a criterion. Since adjacent boundary pixels should have similar gradient magnitude values, there should be a noticeable jump in the gradient magnitude as we move to an area that is not on the true boundary. Figure 6 shows a sampled graph of gradient magnitudes of live-wire pixels. The *X* axis is just an ordinal numbering of the pixels on the live-wire.

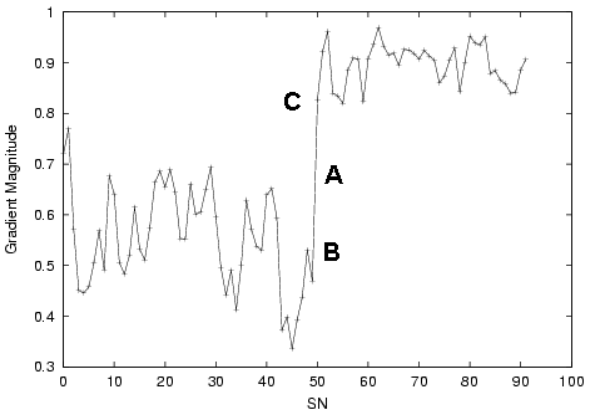


Figure 6. Sampled graph of gradient magnitudes of live-wire pixels

All pixels below point *B* belong to the object boundary. The big jump at A indicates the point where the live-wire breaks away from the object boundary. Therefore, all pixels after the point *C* are not part of the object boundary.

To handle noise, we used a Gaussian filter to smooth the gradient magnitude values before seed point selection. A one-dimensional Gaussian distribution, whose length depended on the number of pixels in the current live-wire, was convolved with the gradient magnitude values. This worked well on a range of noisy images.

As the example illustrates, a clear feature is spread over, for example, four or five pixels, so we also used longer-scale differences. A difference table generated from the gradient magnitudes of the live-wire pixels was overlaid on a Gaussian distribution with the same length as the live-wire, and the result was stored in a list, in effect a 1D convolution over the incremental differences. The maximum value in the list was chosen as the new seed point.

5 The Leash

During the course of this work, a software tool called MixEd was built to let the experimenter conveniently select combinations of algorithms, feature costs, and other parameters, and then obtain real-time feedback while actually tracing an arbitrary image. Figure 7 shows a screen capture of MixEd.

The most effective tool developed using MixEd we calld the *leash*. The leash is constructed by displaying the live-wire in two different colours, as shown in Figure 5. The white portion of the leash starts at the most recently selected seed point and indicates a contiguous sequence of pixels that MixEd believes is *probably* on the desired contour. The yellow portion of the leash simply indicates the rest of the pixels on the live-wire. The user gets the impression of a yellow leash pulling the (white) new contour. White boxes in Figure 5 indicate seed points. (Of course, the user may redefine the colour scheme as needed.) Because the user must from time to time decide when to drop a seed point, it is very effective to let the point on the live-wire where the colour changes be the new seed point. By using any of the methods in Section 4 for optimal seed point selection to select the point where leash attaches to the new contour, the user receives the additional benefit of not having to manually place the new seed point.

Figure 5 illustrates details of the leash operation. After the mouse moves some distance, the new contour is likely to 'stray away' from the desired boundary. At this point, the user can move the mouse until all of the new contour (and hence the next seed point) lies entirely on the desired boundary, and then manually generate a recalibration event. Because this event does not require the user to precisely select a new seed point (as some commercial products do, see Section 6), the recalibration can be triggered by any simple mouse motion or keystroke. Manually regenerating a recalibration event adds the white portion to the existing boundary and drop a new seed point at the junction of the white and yellow (leash) lines. At that time the algorithm recalculates feature costs and least-cost paths.

The result is an effective and simple contour tracing tool. The user begins by using the mouse to select a seed point on the object boundary. When the user (accidentally or otherwise) moves the mouse, the coloured components of the live-wire shift in a manner consistent with the cost function implementation.

Besides providing the metaphor of leading the contour, the leash also indicates to the user the direction the live-wire might take. If the user infers from the direction of the leash that the new contour is about to head in the wrong direction, the user can initiate recalibration by generating an event to make the new contour (white) part of the live-wire part of the permanent boundary. To indicate 'cooling', the permanent boundary is coloured blue, as in [10,11]. Moreover, our implementation allows the user to use both automatic and manual seed point generation, using the latter when the former fails to correctly drop a seed point. There is a case to be made that the leash even mitigates the concern of optimal seed point calculation. Obviously, a better choice

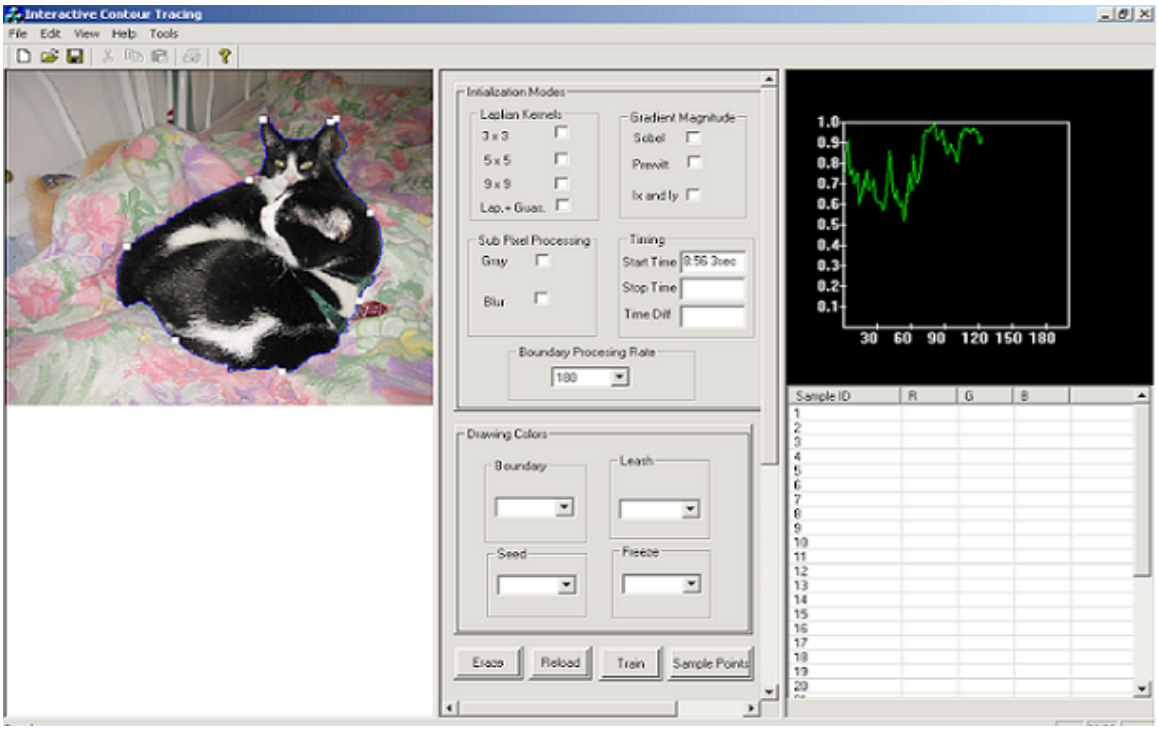


Figure 7. Screen capture of MixEd

of new seed point improves performance, but the leash seems to greatly reduce the marginal utility of fine-tuning that calculation, making reliance on good heuristics acceptable. For instance, approximating the optimal new seed point with the live-wire "midpoint", that is, the pixel midway along the length of the live-wire, still gives the user all the necessary qualitative feedback indicators—it is clear when and where the tool is about to go astray, almost independently of how accurately the seed pixel location has been calculated. (The tool must be conservative, but the midpoint is a conservative estimate.) When this happens, the user can move the mouse so that the white part of the live-wire is where the user wants it to be, and initiate a recalibration, whether or not its length is optimal. For instance, note that in both cases in Figure 5, part of the leash lies on the desired boundary.

Assuming the user can afford the cost, there is no harm in manually dropping a seed point, nor is there any harm in dragging the boundary until the white segment leaves the boundary. Having said this, it is clearly the case that the better the seed point selection technique, the less frequently recalibrations will have to be done. (MixEd also lets the user dynamically adjust the length of the leash relative to the calculated seed point.)

Occasions arise, particularly around smaller features, when it is difficult, or impossible, to force the white segment of the live-wire to lie entirely on the

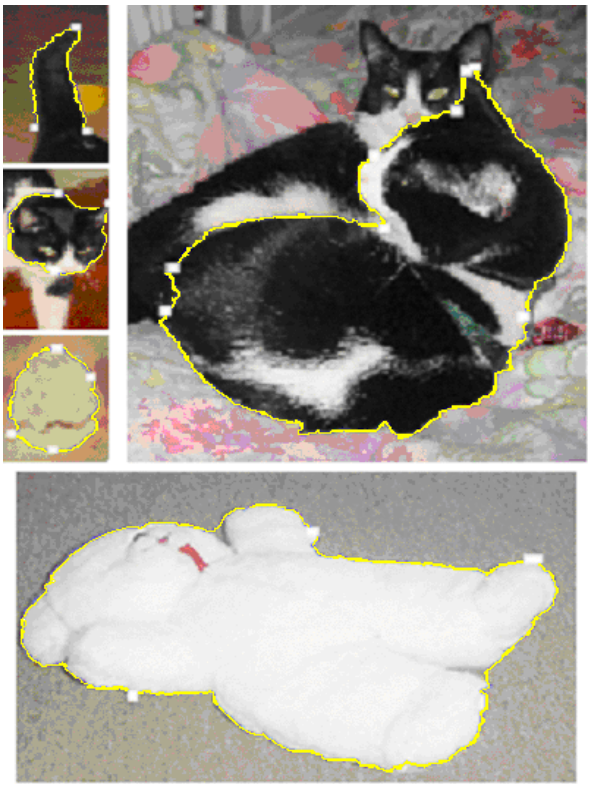


Figure 8. Comparison images. Clockwise from upper right: separating two cats, teddy, skull, cat's head, cat's tail. (Cooled edges enhanced for visibility.)

boundary. For these cases, the tool has a manual override mode where it behaves like the familiar snap tool of drawing packages. This is also useful where the image is unclear or damaged, or when the target object in the image is occluded.

In the worst case, the leash degrades to a familiar fully manual tool. However, although the leash is not fully automatic, experience thus far suggests that because the leash is interactive, the cost of postprocessing contours to find and correct errors (as occurs with a fully automatic tool such as the region-finding method described earlier) can be completely eliminated. If the total cost of producing a contour includes such post-processing costs, it may be that the leash outperforms automatic tools.

Not only does the leash mitigate the marginal value of optimal choice of a new seed point, it also makes us revisit the marginal value of extensions to the link cost function. It may be that a more cost-effective design, at least in this problem domain, would be one in which the user would quickly become adept at predicting the performance of the contour detection algorithm. This would facilitate the ability of the human to anticipate when it will be most productive to let the algorithm do the work and when it will be most productive to trace a section manually. This does not change the motivation for looking for better edge detection tools, but it changes the way we think about how smart algorithms should interact with users.

6 Comparison with Existing Tools

It is difficult to compare different pieces of software when they are deployed in different environments and with different controls. However, we found a Photoshop plug-in with similar functionality, called Extensis MaskPro, and attempted a fair comparison on a range of images. For details, on this product, see http://www.extensis.com/maskpro/.

MaskPro was chosen because its interface has a look and feel similar to our own for tracing an object's boundary. Because its live wire is a single colour, seed points in MaskPro must be exactly selected by the user. In MaskPro, all points are dropped manually while MixEd uses both automatic and manual dropping. In MixEd, seed points are dropped manually only when the algorithm fails to drop a seed point when necessary.

One performance metric used to compare MixEd to MaskPro was the number of seed points generated while tracing the object boundary. This metric is meaningful because dropping seed points takes time, and interrupts the flow of the tracing task, and indicates how often the tool diverged from the desired contour. To compare MaskPro and MixEd, we used the images shown in Figure 8. These images were chosen because they have low contrast regions that are difficult to trace automatically, but they contain objects that are easily recognized. We compared MaskPro and Mixed according to the number of seed points generated. Table 1 details the results. In the images we considered to be the most difficult, the images involving the cats and the skull, our tool generated fewer seed points. The various images generated between 5-11 seed points with MaskPro and between 3-8 with MixEd. On the teddy image, our program required 4 seed points to MaskPro's 3.

Note that this metric understates the benefits of the leash. In MaskPro, the user drags a live-wire of uniform colour. If the live-wire does not fall where the user wants, the user must think about exactly where the new seed point should go, and then laboriously place the new seed point on the desired contour. With the leash, the user drags the mouse and seeds are either dropped automatically, or dropped at the location indicated by the leash, which is usually acceptable. The user rarely has to exactly place the new seed point.

Although this comparison is somewhat simplistic, it establishes that the performance of MixEd is comparable to that of a widely available commercial tool. It is capable of dealing very well with areas of low contrast (cat's tail and skull), and areas where the intensity changes significantly (cat's face, and separating the two cats).

Table 1. Comparison of Seed Points Generated

Features	MaskPro	MixEd
Cat's Tail	5	3
Cat's Head	5	4
Brain	9	5
Skull	10	4
Two cats	11	8
Teddy	3	4

7 Design Methodology

There are several contributions to interface design for mixed-initiative edge detection. First, the results of Section 5 suggest that we might be able to modularly divide interactive edge detection into three orthogonal, but interacting components: the feature cost function, the optimal seed point calculation and the interface. Some component interactions may seem counterintuitive. For example, one might think that the best end product would simply combine the best individual components. However, there is a possibility that "less is more". (This theme also arises in the study of software 'bloat' by McGrenere et al.[9] That work takes the position that much modern software contains so many fea-

tures that productivity may be reduced. In the case of feature cost functions, it may well be that overall productivity, at least in the target domain of model construction, may be enhanced by simpler, but more predictable edge detection algorithms.) A user who can better anticipate the errors of a simpler algorithm may ultimately be more productive than one who has to struggle with a sophisticated but unpredictable algorithm.

8 Conclusions and Future Work

In conclusion, we believe this is an interesting example of the notion of cognitive prosthetic as put forward by Ford and Hayes [4]. They use the metaphor that the computer amplifies the human's abilities, the way eyeglasses improve vision. In our system, the computer contributes high level edge detection, and provides feedback via the leash. The human also provides the best sensor in the world—the human eye—to evaluate contour quality. In this domain, the software works as a cognitive prosthetic by amplifying the anatomist's drawing ability.

To date, the leash has been implemented within MixEd, as shown in Figure 7. This implementation is now robust and has been tested on a variety of scenes, well beyond those shown in the comparisons with a commercial product. The next step will be integrating it into a production environment and getting feedback from domain users. In part, this includes making decisions about how to integrate the flexibility of a control panel similar to that shown in Figure 7 with the convenience of the leash for ordinary users. Many model builders work alone or with a small team and may not want to cope with controls that require even rudimentary knowledge of image processing.

Acknowledgements

The research of the first author was supported by a University of Saskatchewan Graduate Scholarship. The research of the second author is supported by an NSERC Discovery Grant, and the last two authors by the University of Saskatchewan. Thanks to John Cork for contour data from the Virtual Human Embryo and to Scott Lozanoff for discussions about Surfdriver. Thanks also to the referees for careful and constructive comments.

References

[1] V. Berzins. Accuracy of Laplacian edge detector. *Computer Vision, Graphics and Image Processing*, (1984) 27:195-210.

[2] K. R. Castleman. *Digital Image Processing*. Prentice-Hall, 1996

[3] William Clancey and Maarten Sierhuis, Human Centered Computing, NASA Ames Research Center, http://www.arctic-mars.org/1999/SCIENCE/hcc.html

[4] Kenneth M. Ford and Patrick J. Hayes. Cognitive Prostheses. Invited talk, *FLAIRS-99*,Orlando (1999)

[5] E. Horvitz. Principles of Mixed-Initiative User Interfaces. Proceedings of *CHI '99, ACM SIGCHI Conference on Human Factors in Computing Systems*, Pittsburgh, PA, (May 1999) 159-166

[6] D.E. Lloyd. Automated target classification using moment invariants of image shapes. *Farnborough, UK, Rep*. RAE IDN AW126, Dec 1985.

[7] K.V. Mardia and T.J. Hainsworth. A Spatial Thresholding Method for Image Segmentation. *IEEE Trans., on Pattern Analysis and Machine Intelligence*, **10:6**, (November 1988) 919-927

[8] D.Marr, E. Hildreth. Theory of edge detection. *Proceedings of Royal Society of London*, **207 (**1980) 187-217

[9] J. McGrenere, R. Baecker, and K. Booth. (2002). An evaluation of a multiple interface design solution for bloated software. *ACM CHI* (2002) 164-170.

[10] E. Mortenson,. and W.A. Barrett. Interactive Segmentation with Intelligent Scissors. *Graphical Models and Image Processing* **60** (1998) 349-384

[11] Eric Mortensen and William Barrett: "Intelligent Scissors for Image Composition," *SIGGRAPH '95*, Los Angeles, CA (August 1995) 191-198

[12] A.Rosenfeld and A. C. Kak. *Digital Picture Processing*. Academic Press, INC, 1982

[13] Robert B. Trealease. Anatomical Informatics: Millennial Perspectives on a Newer Frontier. *The Anatomical Record* **269** (2002) **224-235**

[14] Virtual Human Embryo Project, Louisiana State University, http://virtualhumanembryo.lsuhsc.edu

[15] Williams, D.J. and M. Shah. A Fast Algorithm for Active Contours and Curvature Estimation. CVGIP: Image Understanding 55:1 (January 1992) 14-26