Fast Intersection Checking for Parametric Deformable Models

Douglas P Perrin[a] and Andrew M. Ladd[b] and Lydia E. Kavraki[b] and Robert D. Howe[b] and Jeremy W. Cannon [b]

[a]Division of Engineering Applied Sciences, Harvard University,Cambridge,MA 02138;
[b]Department Of Computer Science, Rice University, Houston, Texas 77005

Abstract

Parametric active deformable models for image-based segmentation offer a distinct advantage over level sets: speed. This paper presents an extension to active deformable models that makes real-time volume segmentation possible on mid-range off-the-shelf hardware and without the use of specialized graphics hardware. The proposed method uses region-based parametric deformable models. A region-based parametric model, represented by a polygon, must remain non-self intersecting (simple) while undergoing deformation. The simplicity constraint can be enforced by allowing topological changes or by restricting motions of the curve. In either case, intersections of curve segments must be detected otherwise catastrophic divergence results. Good performance relies on the efficiency of the intersection check operation. This paper presents a parameter-free and efficient technique for on-line simplicity checking of polygons undergoing motion. We present timing results validating our approach; in particular, we segment 3*-D ultrasound data at 20 volumes per second.*

1. INTRODUCTION

Active deformable models are a robust method for segmenting imagery and for tracking regions across sequences. Active deformable models can be divided into two groups[1]: parametric models (snakes)[1–4] and level-set models (geometric contours).[5, 6] While the limitations of parametric models are well documented in the literature, the computational efficiency of parametric models is ideal for real-time computer vision where processing cycles need to be completed in milliseconds. Although level sets provide excellent segmentation, perform well in the presence of weak gradients,[7] and can incorporate topological constraints,[8] they still remain too inefficient for real-time volume segmentation.

In this work, we enhance the performance of locally constant curvature snakes[4] to make them suitable for real-time applications. In this implementation the snake is approximated by a piecewise constant curve (a polygon). The propagation equations are only well-defined on simple polygons.[9] The efficiency of snake convergence is highly dependent on the efficiency of checking whether the contour polygon is simple.[4, 9]

Definition: *Let P be a planar polygon with vertices $p_1, ..., p_n$. s_i denotes the segment between p_i and p_{i+1}. A polygon is simple if it is not self-intersecting, in other words, all non-adjacent pairs of segments are disjoint.*

The intersection problem for a simple polygon with control points (vertices) undergoing motion at discrete time steps can be described as follows. The input consists of an initial position for the polygon, P_0, which will be simple. A data structure is built and the time taken for initialization is not considered important. Afterward, a sequence of updates is processed incrementally. Each update consists of a change in position for a single vertex; p_i becomes p'_i. If $P = p_1...p_n$ is a simple polygon before the update then the we must compute whether $P' = p'_1...p'_i...p'_n$ is simple. In the case where P' is not simple then the update does not occur and the snake simulation adjusts itself appropriately.

This paper addresses the minimization of the time required to process the control point update, modify the data structure, and determine whether the polygon is simple. We provide an intersection checking algorithm that has excellent experimental performance when applied to snake point motion. Our solution is an exact and parameter-free method which does not use any discretization of the polygon vertex/segment locations. The speed of our algorithm makes slice-wise segmentation of volumes possible in real-time. We demonstrate this with the segmentation of 3D ultrasound data at 20 volumes per second.

Further author information: (Send correspondence to Douglas Perrin E-mail:dperrin@deas.harvard.edu, Telephone: 1 617 496 9098

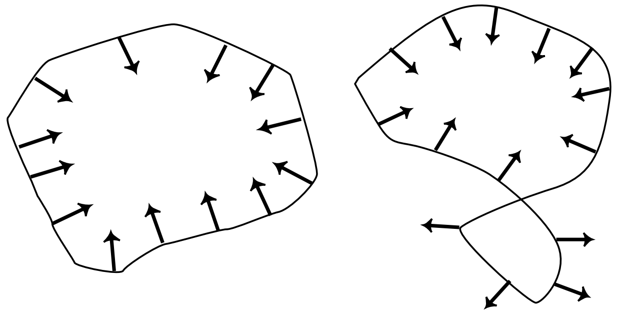
Figure 1. Simple (left) and non-simple (right) curves with normals

2. BACKGROUND

2.1. Snakes

The traditional deformable snake model was first proposed by Kass[2] and is a curve S parameterized by u of the form

$$S(u) = I(x(u), y(u))',\ u \in [0, 1], \tag{1}$$

where I is an image containing the coordinates x and y. This curve is allowed to change shape and position, minimizing an energy defined along its length. The energy function in[2] is:

$$\begin{aligned} E &= \int_0^1 \left[E_{int}(S(u)) + E_{img}(S(u), I)\right] du & (2)\\ E_{int} &= \frac{\alpha}{2}\left|\frac{\partial}{\partial u}S(u)\right|^2 + \frac{\beta}{2}\left|\frac{\partial^2}{\partial u^2}S(u)\right|^2. & (3)\end{aligned}$$

E is the sum of internal energy (tension and curvature) and image energy. α and β are free weighting parameters. E_{img} is the potential induced by the image values.

To improve the behavior in the absence of strong image energy, Ivins and Porrill's[9] proposed solution was the dynamic pressure model. This method incorporated a pressure term that computed the similarity of pixel values around the snake control points to create a force that pushed points toward region boundaries.

$$E = \int_0^1 \left[E_{int}(S(u)) + E_{pressure}(S(u))\right] du. \tag{4}$$

When a portion of the contour lies within a negative pressure region, the curve contracts. When a portion of the contour lies within a positive pressure region, the curve expands. Since the model now relies on inflation and deflation of the curve, the reversed normals (Figure 1) caused by self intersection lead to catastrophic divergence. This work uses the constant curvature dynamic pressure snakes described in Perrin.[4] The performance of these snakes is dependent on efficient simplicity checking, as are a number of other snake implimentations.[1, 3, 9] Regardless of how the snake implementation handles the crossover event an efficient method to determine curve simplicity is needed.

Previous attempts to solve this problem for snakes specifically have been the "all pairs" algorithm, the use of accumulators to "draw" the lines and scan for pending intersections,[9] and using griding.[1, 3] There are several possible techniques from the computational geometry literature for determining if a polygon is simple that could provide more efficient checking. The naïve algorithm (all pairs) verifies if all pairs of non-adjacent line segments do not intersect. This can be done in $O(n^2)$ time for all points on the polygon. A more refined and standard approach uses a scan line algorithm and succeeds in time $O(n \log n)$.[10] The scan-line approach from computational geometry requires significant modifications to adapt it to an efficient on-line algorithm for single vertex updates.

A common situation in on-line algorithms in computational geometry is that the data comes from a source such that it is continuous or, even stronger, low-degree algebraic. The framework that emerged for studying the latter is frequently referred to as kinetic data structures (KDS).[11]

Several existing KDS algorithms are applicable to simplicity checking and have been proposed in the context of planar collision detection for simple geometric objects such as triangles and segments.[12] The kinetic segment tree, however, matches our problem the most closely.[13, 14]

The algorithm, proposed by de Berg et al.,[14] yields a data structure for kinetic collision detection for line segments with disjoint interiors. It can process a swap of the x-coordinates of two segment endpoints in $O(\log n)$ expected time and $O(\log^2 n)$ worst case time. This improved the result of Agarwal et al.[13] which had $O(\log n)$ expected time and $O(n)$ worst case. The second result had an advantage in that it needed to only process a constant fraction of x-coordinate crossings. The model assumes that the endpoints are moving along fixed trajectories of bounded algebraic degree.

2.2. The Proposed Checking Algorithm

Our model differs from the one taken by de Berg et al. in that all points but one remain fixed over a time slice and the intermediate positions of the polygon can be ignored. Furthermore, in the worst case, a single update may cause $n - 1$ x-coordinate crossings. We propose a data structure better suited to this task which does $O(1)$ work for each x-coordinate, y-coordinate crossing, and segment with bounding box overlap with a segment having the updated point as an endpoint. As with the kinetic segment tree, the data structure breaks down when there are many crossings. Our experimental observation is that if the polygons are determined by locally constant curvature snakes[4] then the update cost is a small constant.

The data structures for our algorithm consists of two integer arrays A_x and A_y of length n, and an undirected graph G over a set of n segments. The inductive properties that must hold true for the data structures between each update are:

Inductive Property of A_x and A_y: *The array A_x contains the indices i of the points p_i sorted in order of increasing x-coordinate. The same holds true for A_y.*

Inductive Property of G: *The graph G contains an edge (i, j) if and only if segments s_i and s_j have overlapping bounding boxes.*

UPDATE maintains the invariants for A_x, A_y and G. It also computes whether the polygon was simple after the update.

Algorithm 1 UPDATE(i, q)

```
1: SWAP_x(i, q_x).
2: SWAP_y(i, q_y).
3: for (a, b) an edge in G such that a = i − 1 or a = i do
4:     if s_a intersects s_b then
5:         return polygon is not simple.
6:     end if
7: end for
8: return polygon is simple.
```

The subroutines SWAP$_x$ and SWAP$_y$ are identical except that they operate on different dimensions. The SWAP subroutines will be explained below.

Proposition UPDATE maintains the inductive properties of A_x, A_y and G. Furthermore, it correctly determines if the polygon is simple after the update.

Proof. We assume inductively that the properties of A_x, A_y, and G hold before the update occurred. After the point with index i is updated, i may be out of order in A_x. Lines 4 and 5 of SWAP$_x$, the corresponding lines in the implicit block at line 17, and the outer **while** loops are an insertion sort for the index i. Since the array is sorted except for index i, this produces a sorted A_x. The same also holds for A_y.

Algorithm 2 SWAP$_x(i, q_x)$

```
1:  Let $j$ be such that $A_x[j] = i$.
2:  $(p_i)_x$ is set to $q_x$.
3:  **while** $j < n - 1$ and $q_x > (p_{A_x[j+1]})_x$ **do**
4:      Let $k = A_x[j + 1]$ and swap $A_x[j]$ with $A_x[j + 1]$.
5:      $j := j + 1$.
6:      **for** $(a, b) \in \{(i - 1, k - 1), (i, k - 1), (i - 1, k), (i, k)\}$ **do**
7:          **if** $s_a$ and $s_b$ are not adjacent **then**
8:              **if** $s_a$ and $s_b$ have overlapping bounding boxes **then**
9:                  add edge $(a, b)$ to $G$.
10:             **else**
11:                 remove edge $(a, b)$ from $G$.
12:             **end if**
13:         **end if**
14:     **end for**
15: **end while**
16: **while** $j > 0$ and $(p_{A_x[j-1]})_x > q_x$ **do**
17:     Identical to lines 4-14 except $j + 1$ becomes $j - 1$.
18: **end while**
```

Moving the ith point causes segments s_{i-1} and s_i to change. This also causes changes to the bounding boxes of the segments. Two bounding boxes intersect if and only if the intervals obtained by the axis projections are overlapping on both the x and y-axes. A change in interval overlaps for all segment projections can occur when the order of the endpoints of two segments change. During an update swaps in the order of A_x and A_y correspond to potential bounding box overlap state changes. If index i and k swap position then segments s_a and s_b where $(a, b) \in \{(i-1, k-1), (i-1, k), (i, k-1), (i, k)\}$ have a state change. Lines 6 through 14 in SWAP$_x$ account for this together with the corresponding lines implicit at line 17 by recomputing the state of the edge (a, b) in the graph G.

Finally, lines 3 through 8 in UPDATE correctly compute whether the polygon is simple after the update is complete. Two segments that do not have overlapping bounding boxes cannot intersect and G contains all intersecting bounding boxes. Furthermore, only s_{i-1} and s_i need to be checked since no other segments changed. $\square$

An efficient implementation of UPDATE can be achieved with the appropriate data structures. A_x and A_y are represented as arrays and line 1 of the SWAP$_x$ call is implemented in constant time by using a hash table that inverts the mappings of A_x and A_y. Edge insertion and deletion in G is accomplished in constant time with a hashed implementation of a graph. Edge enumeration of all edges touching a vertex used on line 3 of UPDATE is implemented in $O(degree(i))$ by using hash indexed adjacency sets stored as arrays.

The overall running time of an update is linear with respect to the total number of endpoint swaps that occur in the x and y-axes and the number of bounding boxes that overlap with the bounding boxes of s_{i-1} and s_i. In the worst case, this leads to $O(n)$ work being done per update. However, the experimental performance is $O(1)$.

3. EXPERIMENTAL RESULTS

We present three sets of experiments to argue the efficiency of our technique and to motivate its application. The first is a timing test on a simple image which compares our proposed “fast” simplicity check with the naïve “all-pairs” simplicity check. The second is a test to determine the accuracy of our segmentation approach to volumetric ultrasound data. The objects being segmented are balloon-shaped phantoms made from tissue-mimicking gelatin. The phantoms have known volume and we compare this to the volume estimated by the segmentation. The third experiment used volumetric ultrasound data from a beating porcine heart from a 40 kg animal taken at 20 volumes per second. The ultrasound experiments motivate our approach by giving a concrete application in medical imaging. The experiments were performed on an Athlon 1900XP with 512Mb of main memory running Mandrake Linux 9.0 (for the first experiment) and Windows XP (for the second and third experiments).

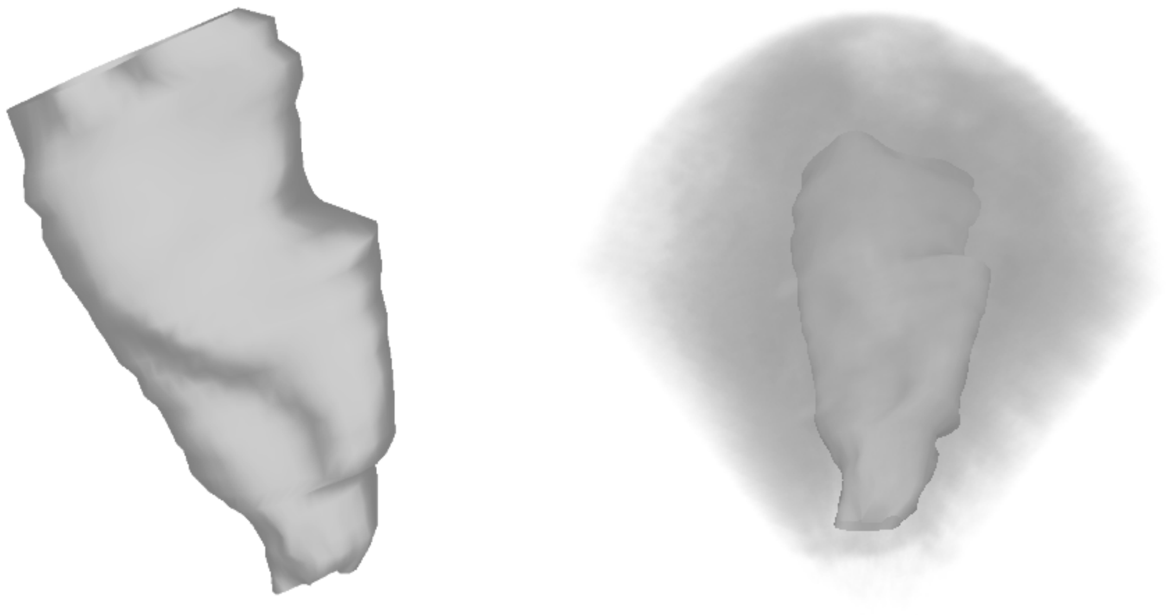
Figure 2. Segmented left ventricle of a porcine heart shown without (left) and with (right) the raw ultrasound data

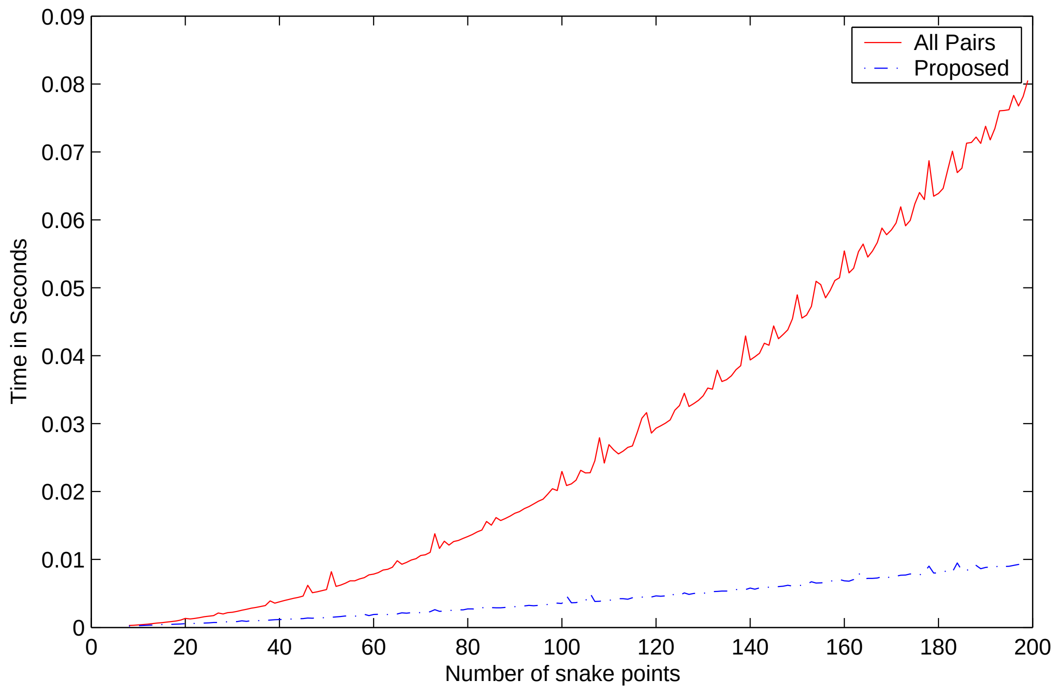

Figure 3. Timings plot for snakes running on video frames

Video For the timing experiments a simple black square was placed in front of the camera. The images were captured using a Logitech USB camera at a resolution of 640 by 480 pixels. The number of iterations required for the snake convergence, the specific parameters β, ρ, and the initial placement of the snake are fixed for the resulting timings tests. After a new image is captured, both the naïve and the proposed method are used to iterate the snake. The resulting times for each were recorded. The only difference between the two implementations is the simplicity check. Since both checks

Phantom No.	Vol. by Weight (cc)	Vol. by Imaging(cc)	% Difference
1	49.45	48.94	-1.0 %
2	54.55	55.57	1.9 %
3	37.36	38.01	1.8 %
4	42.27	42.34	0.2 %
5	37.73	38.22	1.3 %
6	46.90	46.25	-1.4 %

Figure 4. Volume comparison of balloon phantoms

are correct and both snakes are applied to the same input image, the curve shape remains the same for both through the iterations. The time taken by each algorithm is shown in Figure 3. The "all-pairs" algorithm exhibits quadratic behavior as expected while the proposed algorithm is roughly linear.

Real-time 3D Ultrasound The ultrasound experiments were executed using data from a 3-D real-time ultrasound system (Phillips Medical Systems, Andover, MA). The system uses a fully sampled 3000 element array that can acquire 20-25 volumes per second at a resolution of 160x208x144 voxels. We studied the volume estimation of the left ventricle; changes in the ventricle across the heart cycle is an important measure of cardiac function. Since the ultrasound data can be acquired in real-time, the volume estimation which is driven by the segmentation algorithm should ideally be executed in real-time as well. The segmentations were achieved with our snakes implementation. We used 40 control points on each 2-D slice and merged the resulting curves together to form the volume segmentation. Both sets of experiments used the same parameters and ran in real-time.

In the first set of ultrasound experiments, the volume of balloons filled with tissue-mimicking gelatin were estimated using the ultrasound machine and our segmentation implementation. The estimated volumes were then compared to the volumes measured by weight. Six trials were executed and the observed difference in volume was less than 2 percent. The raw data is shown in Figure 4. The confidence interval for the ground truth volumes was 95 percent by weight.

The second experiment was real-time segmentation of the left ventricle of porcine heart. Although clinical validation of the volume estimate is beyond the scope of this work, the segmentations were stable over time and the volume estimates were approximately correct. Two snapshots of these segmentations are shown in Figure 2.

4. DISCUSSION AND FUTURE WORK

In this paper, we present an efficient and exact algorithm for fast simplicity checking of a polygon. In the experiments, we demonstrate the good performance of our approach and describe an application in medical imaging that our technique enables. Although our simplicity checker may have $O(n)$ per iteration worst case performance, our experimental work suggests that it works very well with the input generated by locally curvature constrained snakes. The even spacing and curvature constraints on the control points, together with the fact that at each incremental update a control point will not move much, leads to the good performance of the algorithm. It is possible that rigorous bounds on algorithmic performance can be proved given appropriate bounds on curvature, spacing and control point motion.

The system that we presented can handle snakes that are simple closed curves. In the implementation, we described certain shortcuts that can be taken by assuming the curve topology is fixed. However, all the operations in UPDATE and SWAP can be replaced with operations of the same asymptotic complexity in order to generalize the set of simple closed curves and to support topological changes. The main modification would be to switch the arrays A_x and A_y with hash indexed doubly-linked lists. With this change, the swap and probe operations we used are still constant time and we can now support curve surgery by list surgery.

Acknowledgments

Work on this paper by A.M. Ladd and L.E. Kavraki has been supported in part by NSF ITR #0205671 and NSF #0308237. A.M. Ladd is also supported by FCAR. D.P. Perrin and R. Howe via NIH #1 R01 HL073647-01 and ONR #N00014-98-1-0669 / PR-0629. The authors would like to thank C. Smith for a number of conversations on snakes and explicitly on his ideas for color pressure.

REFERENCES

1. H. Delingette and J. Montagnat, “Shape and topology constraints on parametric active contours,” *Computer Vision and Image Understanding* **83**, pp. 140–171, 2001.
2. M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: active contour models,” *International Journal of Computer Vision, Vol. 1, No. 4* , pp. 321–331, 1988.
3. T. McInerney and D. Terzopoulos, “Topologically adaptable snakes,” in *Proceedings of IEEE International Conference on Computer Vision*, pp. 840–845, 1995.
4. D. Perrin and C. Smith, “Rethinking classical internal forces for active contour models,” in *Proceedings of of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, **2**, pp. 615–620, 2001.
5. R. Malladi, R. Kimmel, D. Adalsteinsson, G. Sapiro, V. Caselles, and J. A. Sethian, “A geometric approach to segmentation and analysis of 3D medic al images,” in *Proceedings of Mathematical Methods in Biomedical Image Ana lysis Workshop*, June 1996.
6. S. Osher and J. Sethian, “Fronts propagating with curvature dependent speed: Algorithms based o n hamilton-jaccobi fomulation,” *Journal of Computational Physics* **79**, pp. 12–49, 1988.
7. A. Yezzi, A. Tsai, and A. Willsky, “A statistical approach to snakes for bimodal and trimodal imagery,” in *Proceedings of the IEEE International Conference on Computer Vision*, (898–903), 1999.
8. X. Han, C. Xu, and J. L. Prince, “A topology preserving deformable model using level set,” in *Proceedings of IEEE CVPR 2001*, **2**, pp. 765–770, 2001.
9. J. Ivins and J. Porrill, “Statistical snakes: Active region models,” in *Proceedings of BMVC*, pp. 377–386, 1994.
10. M. de Berg, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer, Berlin, 1997.
11. L. J. Guibas, “Kinetic data structures – a state of the art report,” in *Proceedings of the 5th Workshop on Algorithmic Foundations of Robotics*, P. Agarwal, L. Kavraki, and M. Mason, eds., pp. 191–209, A.K. Peters, Wellesley, MA, 1998.
12. P. K. Agarwal, J. Basch, M. de Berg, L. J. Guibas, and J. Hershberger, “Lower bounds for kinetic planar subdivisions,” in *Symposium on Computational Geometry*, pp. 247–254, 1999.
13. P. K. Agarwal, L. J. Guibas, T. M. Murali, and J. S. Vitter, “Cylindrical static and kinetic binary space partitions,” in *Symposium on Computational Geometry*, pp. 39–48, 1997.
14. M. de Berg, J. Comba, and L. J. Guibas, “A segment-tree based kinetic BSP,” in *Symposium on Computational Geometry*, pp. 134–140, 2001.