CVGIP: IMAGE UNDERSTANDING
Vol. 55, No. 1, January, pp. 14–26, 1992

A Fast Algorithm for Active Contours and Curvature Estimation*

DONNA J. WILLIAMS AND MUBARAK SHAH

Department of Computer Science, University of Central Florida, Orlando, Florida 32816

Received March 8, 1991; accepted May 15, 1991

A model for representing image contours in a form that allows interaction with higher level processes has been proposed by Kass *et al.* (in *Proceedings of First International Conference on Computer Vision, London, 1987,* pp. 259–269). This active contour model is defined by an energy functional, and a solution is found using techniques of variational calculus. Amini *et al.* (in *Proceedings, Second International Conference on Computer Vision, 1988,* pp. 95–99) have pointed out some of the problems with this approach, including numerical instability and a tendency for points to bunch up on strong portions of an edge contour. They proposed an algorithm for the active contour model using dynamic programming. This approach is more stable and allows the inclusion of hard constraints in addition to the soft constraints inherent in the formulation of the functional; however, it is slow, having complexity $O(nm^3)$, where n is the number of points in the contour and m is the size of the neighborhood in which a point can move during a single iteration. In this paper we summarize the strengths and weaknesses of the previous approaches and present a greedy algorithm which has performance comparable to the dynamic programming and variational calculus approaches. It retains the improvements of stability, flexibility, and inclusion of hard constraints introduced by dynamic programming but is more than an order of magnitude faster than that approach, being $O(nm)$. A different formulation is used for the continuity term than that of the previous authors so that points in the contour are more evenly spaced. The even spacing also makes the estimation of curvature more accurate. Because the concept of curvature is basic to the formulation of the contour functional, several curvature approximation methods for discrete curves are presented and evaluated as to efficiency of computation, accuracy of the estimation, and presence of anomalies. © 1992 Academic Press, Inc.

1. INTRODUCTION

The problem of how to represent a set of points which have been determined to lie on an edge is a challenging one. Kass *et al.* [3] have proposed a model called active contours (snakes) which has the advantage that the final form of a contour can be influenced by feedback from a higher level process. The contour is initially placed near an edge under consideration, then image forces draw the contour to the edge in the image. As the algorithm iterates, the energy terms can be adjusted by higher level processes to obtain a local minimum that seems most useful to that process. However, there are some problems with the minimization procedure used. Amini *et al.* [1] pointed out some of these, including instability and a tendency for points to bunch up on a strong portion of an edge. They have proposed a dynamic programming algorithm for minimizing the energy functional that allows addition of hard constraints to obtain more desirable behavior of the snakes.

In this paper we present some of the problems in both methods and propose a further algorithm which is stable, is flexible, allows hard constraints, and runs much faster than the dynamic programming method. The energy functional being minimized has a continuity term and a curvature term in addition to the image energy and external energy terms. In the method presented here a different formulation is used for the continuity term so that the points will be more evenly spaced on the contour, rather than minimizing distance between points as in the previous methods. Discrete approximation of curvature in an accurate and efficient manner is necessary for the curvature term, so a number of different approximations are examined and evaluated.

In the next section we will discuss the previous methods of controlling active contours and point out some of the strengths and weaknesses of the approach as a whole, and of the particular methods used. Section 3 presents curvature estimation methods for discrete curves. Computational efficiency is discussed and anomalies occurring with some of the methods, especially when points are not evenly spaced, are pointed out. In Section 4 a new method is presented which uses a greedy approach. The pseudo-code for this method is given in Section 5. All three methods were applied to real images and the results are given in Section 6.

* The research reported here was supported by DOD/PMTRADE under Contract N61339-88-G-0002/6, and by Florida High Technology Council under Contract 11-28-718. The content of the information herein does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

2. MINIMUM ENERGY CONTOURS

At present it is common to use more than one scale to detect edges or represent contours [2, 4, 8]. Rather than combine the information derived at the different scales into a unified "best" representation of the information, another approach is to attempt to keep the information at different scales available so that higher level processes can use the most meaningful representation. This was one of the goals of Kass, Witkin, and Terzopoulos [3] when they developed their Active Contour Models (called snakes). They developed a controlled continuity spline which can be operated upon by internal contour forces, image forces, and external forces which are supplied by an interactive user, or potentially by a higher level process.

In their work, Kass *et al.* represented a contour by a vector, $\mathbf{v}(s) = (x(s), y(s))$, having the arc length, s, as parameter.[1] They defined an energy functional of the contour and described a method for finding contours which correspond to local minima of the functional. The energy functional is written as

$$\begin{aligned} E^*_{\text{snake}} &= \int_0^1 E_{\text{snake}}(\mathbf{v}(s))\,ds \\ &= \int_0^1 E_{\text{int}}(\mathbf{v}(s)) + E_{\text{image}}(\mathbf{v}(s)) + E_{\text{con}}(\mathbf{v}(s))\,ds. \qquad (1) \end{aligned}$$

E_{int} represents the internal energy of the contour due to bending or discontinuities, E_{image} is the image forces, and E_{con} is the external constraints. The image forces can be due to various events. The ones presented by Kass *et al.* are lines, edges, and terminations. The internal spline energy is written

$$E_{\text{int}} = (\alpha(s)|\mathbf{v}_s(s)|^2 + \beta(s)|\mathbf{v}_{ss}(s)|^2)/2. \qquad (2)$$

The above equation contains a first-order term which will have larger values where there is a gap in the curve, and a second-order continuity term which will be larger where the curve is bending rapidly. The values of α and β at a point determine the extent to which the contour is allowed to stretch or bend at that point. The relative sizes of α and β can be chosen to control the influence of the corresponding constraints. For instance, a large value of β would make the second-order continuity term larger than the other terms; thus the minimum value of E* would occur when the curve was smoother, approaching a circle for a closed contour, and a straight line for a contour which was not closed. If α is 0 at a point, a discontinuity can occur at that point, while if β is 0, a corner can develop, because large values of these terms would not be included in the total. The minimum energy contour was determined using techniques of variational calculus.

[1] Lower-case, bold letters like **v** will be used to denote vectors when they are interpreted as points, while lower-case, bold letters with an arrow above ($\vec{\mathbf{u}}$) will be used when the quantity represented is a vector from one point to another.

Amini, Tehrani, and Weymouth [1] point out some of the problems involved in this method of solution and propose that the contour having minimum energy be determined using dynamic programming rather than variational calculus. This allows the introduction of constraints that cannot be violated, called *hard* constraints, as well as the first- and second-order continuity constraints which are inherent in the problem formulation. These latter are known as *soft* constraints because they are not satisfied absolutely, only to a certain degree.

At this point it would be meaningful to examine the advantages and disadvantages of the problem formulation itself, and of the two proposed methods of solution. A "+" by an item on the list indicates a positive feature, while a "−" indicates a drawback. First we will consider advantages and disadvantages which apply to the statement of the problem and to both methods of solution.

+ A closed contour which is placed around an object can span gaps in the edge map. Similarly, if an object with texture has edges which make it appear as several smaller objects, the contour can outline the object as a whole, giving a continuous edge contour for the entire object. See Fig. 1 for an example of this.

+ Information from a higher level process can be used to determine values of the external constraint term and the values of α and β. For example, corners could be allowed at certain points and the effect on the contour examined.

− No guidelines are given in either method for determining the values of α and β. Also, both methods apparently use the same value for α and β at every point, and no discussion or examples are given explaining how

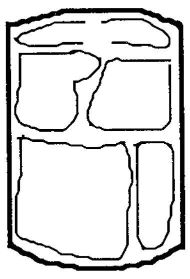
FIG. 1. Contour outlines entire object, rather than following texture edges on the surface of the object.

changing these values affects the contours. It happens that the values are critical, and must be chosen carefully to obtain meaningful results.

− Related to the previous item, if β is constant, corners will not be well defined. There is also a problem if points are far apart and a corner falls between two points on a contour.

− The first derivative term in Eq. (2) is approximated by a finite difference, $|\mathbf{v}_s|^2 \approx (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2$. This is equivalent to minimizing the distance between points, and has the effect of causing the contour to shrink.

− Points can move along the contour as well as perpendicular to it, thereby allowing points to bunch up in segments of the contour where the image forces are higher. The hard constraints provided for in the method of Amini *et al.* can be used to minimize this problem.

The following list applies to the Kass method only.

+ Forces can travel large distances along the contour in one iteration, allowing faster convergence.

− Image forces and constraints need to be differentiable in order to guarantee convergence. Thus it is not possible to include hard constraints, such as minimum distance between points.

− Intermediate results are not meaningful. The contour does not smoothly approach the minimum value. It was for this reason that the name snakes was given to the contours.

The next list gives characteristics of the Amini method.

+ Hard constraints can be introduced into the method.

+ Points are moved on the discrete grid, as opposed to the Kass method which computes point coordinates as real numbers, allowing points to fall between the discrete coordinates.

+ This method is numerically stable.

− Memory requirements are large, being $O(nm^2)$, where n is the number of points on the contour and m is the number of possible locations to which a point may move in a single iteration.

− The method is very slow, being $O(nm^3)$.

In this paper we resolve most of the difficulties with both previous methods. The inclusion of hard constraints, use of the discrete grid for point positions, and stability that were achieved by the dynamic programming method are preserved in this method. In addition the choices of values for α, β, and γ (a new parameter) are easily determined to balance the relative strengths of the terms in the functional. Reformulation of the first-order continuity term causes the points to be evenly spaced on the contour, removing the shrinking behavior of the contour and making the estimation of the second-order continuity term more accurate. Furthermore, the implementation is efficient in both time and space requirements.

3. CURVATURE ESTIMATION

Both Kass *et al.* and Amini *et al.* approximate the derivatives in Eq. (2) by finite differences. If $\mathbf{v}_i = (x_i, y_i)$ is a point on the contour, the following approximations are used:

$$\left|\frac{d\mathbf{v}_i}{ds}\right|^2 \approx |\mathbf{v}_i - \mathbf{v}_{i-1}|^2 = (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \quad (3)$$

and

$$\left|\frac{d^2\mathbf{v}_i}{ds^2}\right|^2 \approx |\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}|^2$$
$$= (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2. \quad (4)$$

Note that two assumptions have been made here. The first assumption is that the points are spaced at unit intervals. If the points are evenly spaced, then Eq. 3 should be divided by d^2, where d is the distance between points, and Eq. 4 by d^4. This is not a major problem since the values of α and β can be chosen to include these factors. In that case d will have to be made available to any higher order process which is attempting to assign values to α and β automatically.

If the points are *not* evenly spaced, the first derivative term will be incorrect by a factor of d_i^2, where d_i is the distance between points i and $i - 1$. This will cause the first-order continuity term in the energy expression to be larger for points which are farther apart. In addition, the second derivative term will give higher estimates of curvature when the points are not evenly spaced.

The second assumption is that the parameter is arc length. When this assumption is true, then curvature is given by $|\mathbf{v}_{ss}|$. However, when the parameter is not arc length, curvature is given by

$$\kappa = \frac{|x'y'' - x''y'|}{(x'^2 + y'^2)^{3/2}} \quad (5)$$

for a parameter t where $x' = dx/dt$, $x'' = d^2x/dt^2$, $y' = dy/dt$, and $y'' = d^2y/dt^2$. The quantity $|\mathbf{v}_{tt}| = \sqrt{x''^2 + y''^2}$ does not measure curvature when the parameter t is not arc length.

It is not clear what measure of curvature is the best reflection of the geometric situation depicted by the contour. The mathematical definition of curvature is $d\theta/ds$, where θ is the angle between the positive x-axis and the

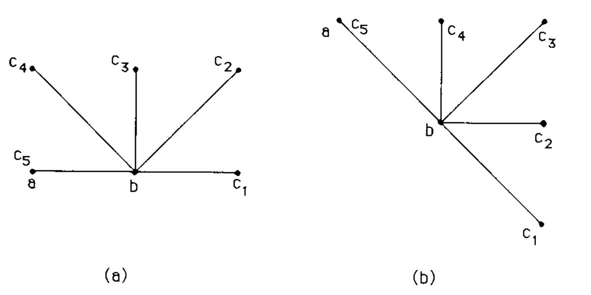

FIG. 2. Arrangement of points a, b, and c for Table 1.

TABLE 1
Comparison of Estimates of the Square of the Curvature Using Different Methods

c	$(d\theta/ds)^2$	κ^2	$\lvert\mathbf{v}_{ss}\rvert^2$	$\lvert\vec{\mathbf{u}}_i - \vec{\mathbf{u}}_{i+1}\rvert^2$	$\left\lvert\frac{\vec{\mathbf{u}}_i}{\lvert\vec{\mathbf{u}}_i\rvert} - \frac{\vec{\mathbf{u}}_{i+1}}{\lvert\vec{\mathbf{u}}_{i+1}\rvert}\right\rvert^2$
			Horizontal case		
1	0.0	0.0	0.0	0.0	0.0
2	0.42	0.512	0.40	1.0	0.59
3	2.47	8.0	2.0	2.0	2.0
4	3.80	64.0	2.34	5.0	3.41
5	9.87	∞	4.0	4.0	4.0
			Diagonal case		
1	0.0	0.0	0.0	0.0	0.0
2	0.42	0.512	0.40	1.0	0.59
3	1.23	4.0	1.0	4.0	2.0
4	3.80	64.0	2.34	5.0	3.41
5	4.93	∞	2.0	8.0	4.0

Note. The first section of the table is the horizontal situation of FIG. 2, the lower section is the diagonal case. Column one gives $(d\theta/ds)^2$, column two is κ^2 using Equation 5, and column three gives $|\mathbf{v}_{ss}|^2$. Column four is $|\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}|^2$ and column five gives $[\Delta x_i/\Delta s_i - \Delta x_{i+1}/\Delta s_{i+1}]^2 + [\Delta y_i/\Delta s_i - \Delta y_{i+1}/\Delta s_{i+1}]^2$.

tangent vector to the curve. This is a coordinate independent measure, as the same value will be obtained for $d\theta$ when any line is substituted for the x-axis; thus the measure is invariant under rotation. That is a desirable feature for model matching. Another desirable feature which is not present in curvature is scale invariance. A circle with radius r has curvature $1/r$ at each point. Thus when the radius is doubled, the curvature is halved.

The remainder of this section presents five possible measures of curvature in discrete contours, and discusses the characteristics of each. In order to demonstrate the difference in the results obtained by these different approximations, they were all applied to the two situations displayed in Fig. 2 and the results are displayed in Table 1. In each case $\mathbf{v}_{i-1}$ is point a, $\mathbf{v}_i$ is point b, while $\mathbf{v}_{i+1}$, the third point necessary in the curvature estimate, can be any one of the points $c_1 \ldots c_5$. The first section of the table is the situation in Fig. 2a where a and b are on a horizontal or vertical line. The lower section is the case where a and b lie on a diagonal line. When the external angle is 0, $\pi/2$, or π, the distances from b to its two neighbors are equal, being 1 for the horizontal case and $\sqrt{2}$ for the diagonal case. When the angle is $\pi/4$ or $3\pi/4$ the two distances are not equal, being 1 and $\sqrt{2}$.

It will be necessary to estimate the value of differentials in the following discussion. The usual convention will be to use the *backward* difference for this estimate. That is, dx at the point $\mathbf{v}_i$ is approximated by $x_i - x_{i-1}$ and is denoted Δx_i. Occasionally when the backward difference might vary substantially from the *forward* difference, $(x_{i+1} - x_i)$, the *centered* difference will be used instead. It is given by $(x_{i+1} - x_{i-1})/2$. Whenever this is done, it will be pointed out. Similar notation for finite difference estimation of differentials will be used for all variables.

The first possibility for approximating curvature is to apply the definition of curvature directly. If a discrete approximation of $d\theta/ds$ is computed for evenly spaced points, it has the property that it depends linearly on the angle $\Delta\theta$ between the two vectors, $\vec{\mathbf{u}}_i = (x_i - x_{i-1}, y_i - y_{i-1})$ and $\vec{\mathbf{u}}_{i+1} = (x_{i+1} - x_i, y_{i+1} - y_i)$. The formula for $\Delta\theta$ is given by

$$\Delta\theta = \cos^{-1}\frac{\vec{\mathbf{u}}_i \cdot \vec{\mathbf{u}}_{i+1}}{|\vec{\mathbf{u}}_i|\,|\vec{\mathbf{u}}_{i+1}|} = \cos^{-1}\frac{(x_i - x_{i-1})(x_{i+1} - x_i) + (y_i - y_{i-1})(y_{i+1} - y_i)}{\sqrt{[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2][(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2]}}.$$

Given a closed polygon and a direction, $\Delta\theta \approx d\theta$ is the external angle as the circumference is traversed. The centered difference, $(\Delta s_{i+1} + \Delta s_i)/2 = (|\vec{\mathbf{u}}_{i+1}| + |\vec{\mathbf{u}}_i|)/2$, averages the distance from point i to its two neighbors and thus gives the best estimate of ds. The smallest value for Δs is 1 and the largest value of $\Delta\theta$ is π; thus values of $\Delta\theta/\Delta s$ all fall in the interval $[0, \pi]$. This is not true for continuous curves, where rapidly bending curves can have very large curvature. However, when a curve is digitized, a limit is placed on the curvature. Although giving intuitively satisfying results, this measure requires a lot of computation. Column one of Table 1 gives values of $(d\theta/ds)^2$.

Evaluating the expression for curvature in Eq. (5)

should give results identical to that of $d\theta/ds$ for continuous curves. However, this is not the case for discrete curves. When the angle between $\vec{\mathbf{u}}_i$ and $\vec{\mathbf{u}}_{i+1}$ becomes large, Δx_i has a value near $-\Delta x_{i+1}$ and Δy_i is near $-\Delta y_{i+1}$. Thus when the centered difference is used to estimate dx and dy, these values become very small, giving a value for curvature which is unbounded, as it is for continuous curves. Column two of Table 1, the discrete approximation to Eq. 5, is comparable to the other estimates for small angles, but as $(\Delta x_i + \Delta x_{i+1})/2$ and $(\Delta y_i + \Delta y_{i+1})/2$ grow smaller, the estimate of κ^2 becomes very large.

Converting the parameter to arc length and then computing the second derivative is theoretically equal to the two previous measures for continuous curves. The discrete approximation is given by

$$|\mathbf{v}_{ss}| = \frac{1}{\Delta s}\sqrt{\left(\frac{\Delta x_i}{\Delta s_i} - \frac{\Delta x_{i+1}}{\Delta s_{i+1}}\right)^2 + \left(\frac{\Delta y_i}{\Delta s_i} - \frac{\Delta y_{i+1}}{\Delta s_{i+1}}\right)^2}, \quad (6)$$

where Δs is $(\Delta s_i + \Delta s_{i+1})/2$. The third column of Table 1 gives the square of the discrete estimate of the second derivative vector. Note in the diagonal case for column three that the curvature for c_4 is larger than for c_5, even though the path $a - b - c_5$ actually doubles back on itself, and should intuitively have higher curvature.

Another possible measure of curvature which has the advantage of being computationally efficient is given by the expression in Eq. (4). If $\vec{\mathbf{u}}_1$ and $\vec{\mathbf{u}}_2$ are the vectors defined above, this is equivalent to $|\vec{\mathbf{u}}_2 - \vec{\mathbf{u}}_1|^2$. It reflects not only the difference between the directions of the two vectors, but also the difference in length. Thus if the three points in the estimate are not evenly spaced the curvature will be larger. For example, a, b, and c may lie on a straight line and curvature will be nonzero. The fourth column in the table is the square of the curvature estimate using Eq. (4). Note that in column four, c_4 has the largest value for the horizontal case as it does for column three in the diagonal case.

Normalizing the two vectors before taking the difference removes the length differential, and the measure depends solely on relative direction. Thus it will be bounded, with values in the interval [0, 2]. The length of $\vec{\mathbf{u}}_{i+1}/|\vec{\mathbf{u}}_{i+1}| - \vec{\mathbf{u}}_i/|\vec{\mathbf{u}}_i|$ is given by $2\sin(\theta/2)$, where θ, $0 \le \theta \le \pi$, is the difference in direction of the two vectors as shown in Fig. 3. Column five of the table gives the values obtained by this formula.

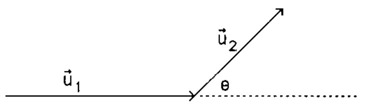

FIG. 3. Difference in direction of two vectors.

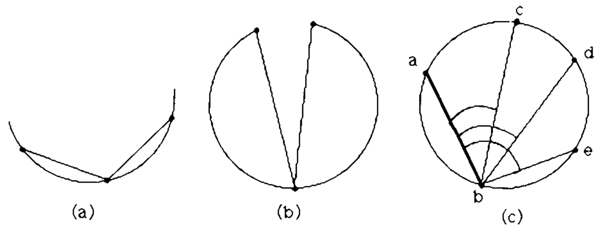

FIG. 4. Estimation of curvature by fitting a circle to three points. (a) Angle between vectors is large, hence fit is good. (b) Small angle means a circle is not a good approximation to the curve through the three points. (c) When distance between points is not equal, circles having the same radius will go through $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, b, e\}$.

It is interesting that the last three measures are closely related. Multiplying the discrete approximation of $|\mathbf{v}_{ss}|$ by Δs gives the difference of the normalized vectors (column five). When the points are evenly spaced, multiplying by Δs again gives the expression in Eq. (4) (column four).

There is a sixth method of approximating curvature at a point, that of fitting a circle through the point and its two neighbors (e.g., [5]). The radius of the circle will give a good estimate of the radius of curvature if a circle is a good approximation of the curve through the three points. However, this only gives a reasonable estimate when the angle between the two vectors is large and when the points are evenly spaced (Fig. 4a). When the angle between the two vectors is small, the circle does not give a good approximation to the curve through the three points, and the curvature estimate will be too small (Fig. 4b). If the points are not evenly spaced, very different situations, which do not seem to have the same curvature, will give a circle having the same radius. For example, the sets of points $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, b, e\}$ would have a circle of the same radius fitted through them (Fig. 4c) even though the curvature of the curve through the different sets does not appear the same. Thus, this method does not seem to have general enough application to consider here.

4. GREEDY ALGORITHM

In this section a greedy algorithm will be presented which allows a contour with controlled first and second order continuity to converge on an area of high image energy, in this case edges. This algorithm allows the inclusion of hard constraints as described by Amini *et al.* [1] but is much faster than their $O(nm^3)$ algorithm, being $O(nm)$ for a contour having n points which are allowed to move to any point in a neighborhood of size m at each iteration. While the algorithm is not guaranteed to give a global minimum, the experimental results were comparable to other methods.

The quantity being minimized by this algorithm is

$$E = \int (\alpha(s)E_{\text{cont}} + \beta(s)E_{\text{curv}} + \gamma(s)E_{\text{image}})\, ds. \quad (7)$$

The form of this equation is similar to Eq. (1). The first and second terms are first- and second-order continuity constraints and will be described in detail later. They correspond to E_{int} in Eq. (1). The last term measures some image quantity such as edge strength or intensity and is the same as the middle term of Eq. (1). No term for external constraints was included, although it would be possible to do so. The parameters α, β, and γ are used to balance the relative influence of the three terms. Their relative sizes, rather than absolute sizes, are significant.

The proposed algorithm is iterative, as are those of Kass and Amini. During each iteration, a neighborhood of each point is examined and the point in the neighborhood giving the smallest value for the energy term is chosen as the new location of the point. Only closed contours are being considered, so all index arithmetic is modulo n.

In the examples given below, $\alpha = 1$, β is either 0 or 1 (depending upon whether a corner is assumed at that location), and $\gamma = 1.2$. These were chosen so that the image gradient will have slightly more importance than either of the continuity terms in determining where points on the contour move.

Determining a suitable approximation for the first term in Eq. 7, the continuity term, presents some difficulties. Using $|\mathbf{v}_i - \mathbf{v}_{i-1}|^2$ causes the curve to shrink, as this is actually minimizing the distance between points. It also contributes to the problem of points bunching up on strong portions of the contour. These effects are even worse with a greedy algorithm where each point is moved based on local considerations. The tendency is for points to always be moved nearer the previous point, which also moves a point farther from the following point. This causes a chain reaction, moving all points toward the previous one. In observing the behavior of the given algorithms, it became apparent that a term which encouraged even spacing of the points would reflect the desired behavior of the contours more than one which caused shrinking. The original goal of encouraging first-order continuity is still satisfied. Thus the algorithm presented here uses the difference between the average distance between points, $\bar{d}$, and the distance between the two points under consideration: $\bar{d} - |\mathbf{v}_i - \mathbf{v}_{i-1}|$. Thus points having distance near the average will have the minimum value. The value is normalized by dividing by the largest value in the neighborhood to which the point may move, giving a value in [0, 1]. At the end of each iteration a new value of $\bar{d}$ is computed.

The second term in Eq. (7) is curvature. Since the formulation of the continuity term causes the points to be relatively evenly spaced, $|\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}|^2$, the formula in column four of Table 1, gives a reasonable estimate of curvature multiplied by a constant. The constant term is not significant since this term, like the continuity term, is normalized by dividing by the largest value in the neighborhood, giving a number from 0 to 1. This formula has the advantage that it is the most computationally efficient of the ones discussed in the previous section.

The third term in Eq. (7), E_{image}, is the image force, which is gradient magnitude. Gradient magnitude at each point in the image is input as an eight bit integer, with values 0–255. There is a significant difference between a point with gradient magnitude 240, and one having magnitude 255. This is not reflected when the values are normalized by division by 255. Thus, given the magnitude at a point (mag) and the maximum (max) and minimum (min) gradient in each neighborhood, (min − mag)/(max − min) is used for the normalized edge strength term. This gradient magnitude term is negative so that points with large gradient will have small values. If max − min < 5 then min is given the value max − 5. This prevents large differences in the value of this term from occurring in areas where the gradient magnitude is nearly uniform. For example, when all points in the neighborhood being examined had values 47, 48, and 49, the gradient magnitude term would be 0, −0.5, or −1.0 for points with essentially the same gradient magnitude. Thus a point would have a strong tendency to stay at a point with gradient magnitude 49, even though it is not a strong edge point. Having a minimum of 5 in the denominator would give −0.6, −0.8, or −1.0 for the gradient term, more accurately reflecting the similarity of the points. Near an edge this situation does not normally arise, but if the contour has points that begin fairly far from the final edge or span regions where there are gaps in the edge, points on the contour may resist moving without this constraint.

At the end of each iteration, a step is included which determines the curvature at each point on the new contour, and if the value is a curvature maximum, sets $\beta_i = 0$ for the next iteration. This step functions as a primitive high level process giving feedback to the energy minimization step. Curvature is computed at each of the n points by $[\Delta x_i/\Delta s_i - \Delta x_{i+1}/\Delta s_{i+1}]^2 + [\Delta y_i/\Delta s_i - \Delta y_{i+1}/\Delta s_{i+1}]^2$. This is the measure given in column five of Table 1, which is related to the angle between the vectors. This formula requires more computation than the one used in the main computation of the algorithm, but is computed fewer (n) times and is used because determining a meaningful threshold is easy. Nonmaxima suppression is then performed on curvature values along the contour, and curvature maxima points having curvature above a threshold are considered corner points for the next iteration. A further condition for designating a point as a corner is that the gradient magnitude must be above some minimum value. This prevents corners from forming until the contour is near an edge. Yuille *et al.* [9] used a similar approach in their deformable template matching. They changed the coefficients which determined the relative strengths of the terms as the convergence process entered different stages. In summary β is set equal to zero at the points satisfying the above three conditions: curvature maxima above a curvature threshold and above a gradient threshold. This allows a corner to form there, and reduces the curvature in the segments between these points.

Figure 5 demonstrates how the algorithm works. The energy function is computed for the current location of $\mathbf{v}_i$ and each of its neighbors. The location having the smallest value is chosen as the new position of $\mathbf{v}_i$. $\mathbf{v}_{i-1}$ has already been moved to its new position during the current iteration. Its location is used with that of each of the proposed locations for $\mathbf{v}_i$ to compute the first-order continuity term. The location of $\mathbf{v}_{i+1}$ has not yet been moved. Its location, along with that of $\mathbf{v}_{i-1}$, is used to compute the second-order constraint for each point in the neighborhood of $\mathbf{v}_i$. For $i = 0$, only old values are used. For this reason $\mathbf{v}_0$ is processed twice, one as the first point in the list, and once as the last point. This helps make its behavior more like that of the other points.

5. PSEUDO-CODE FOR GREEDY ALGORITHM

```
Index arithmetic is modulo $n$.
Initialize $\alpha_i$, $\beta_i$, and $\gamma_i$ to 1 for all $i$.

do
    /* loop to move points to new locations */
    for $i = 0$ to $n$                    /* point 0 is first and last one processed */
        $E_{\min} = \text{BIG}$
        for $j = 0$ to $m - 1$                 /* $m$ is size of neighborhood */
            $E_j = \alpha_i E_{\text{cont},j} + \beta_i E_{\text{curv},j} + \gamma_i E_{\text{image},j}$
            if $E_j < E_{\min}$ then
                $E_{\min} = E_j$
                $jmin = j$
        Move point $\mathbf{v}_i$ to location $jmin$
        if $jmin$ not current location, $ptsmoved += 1$     /* count points moved */
    /* process determines where to allow corners in the next iteration */
    for $i = 0$ to $n - 1$
        $c_i = |\vec{\mathbf{u}}_i/|\vec{\mathbf{u}}_i| - \vec{\mathbf{u}}_{i+1}/|\vec{\mathbf{u}}_{i+1}||^2$
    for $i = 0$ to $n - 1$
        if ($c_i > c_{i-1}$ and $c_i > c_{i+1}$      /* if curvature is larger than neighbors */
         and $c_i > threshold1$                /* and curvature is larger than threshold */
         and $mag(\mathbf{v}_i) > threshold2$     /* and edge strength is above threshold */
            then $\beta_i = 0$                    /* relax curvature at point $i$ */
until $ptsmoved < threshold3$
```

6. EXPERIMENTAL RESULTS

In order to demonstrate the performance of the algorithm described in the previous section, results are given for the greedy algorithm developed above, for the original variational calculus solution, and for the dynamic programming algorithm. These programs were run on one synthetic image, a Square (Fig. 6), and three real images: Box (Fig. 7), Bottle (Fig. 8), and Cup (Fig. 9). The Cup image tested the behavior of the algorithms

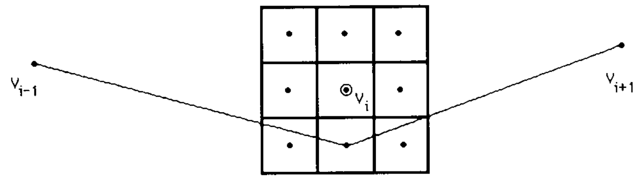

FIG. 5. The energy function is computed at $\mathbf{v}_i$ and each of its eight neighbors. The point before and after it on the contour are used in computing the continuity constraints. The location having the smallest value is chosen as the new position of $\mathbf{v}_i$.

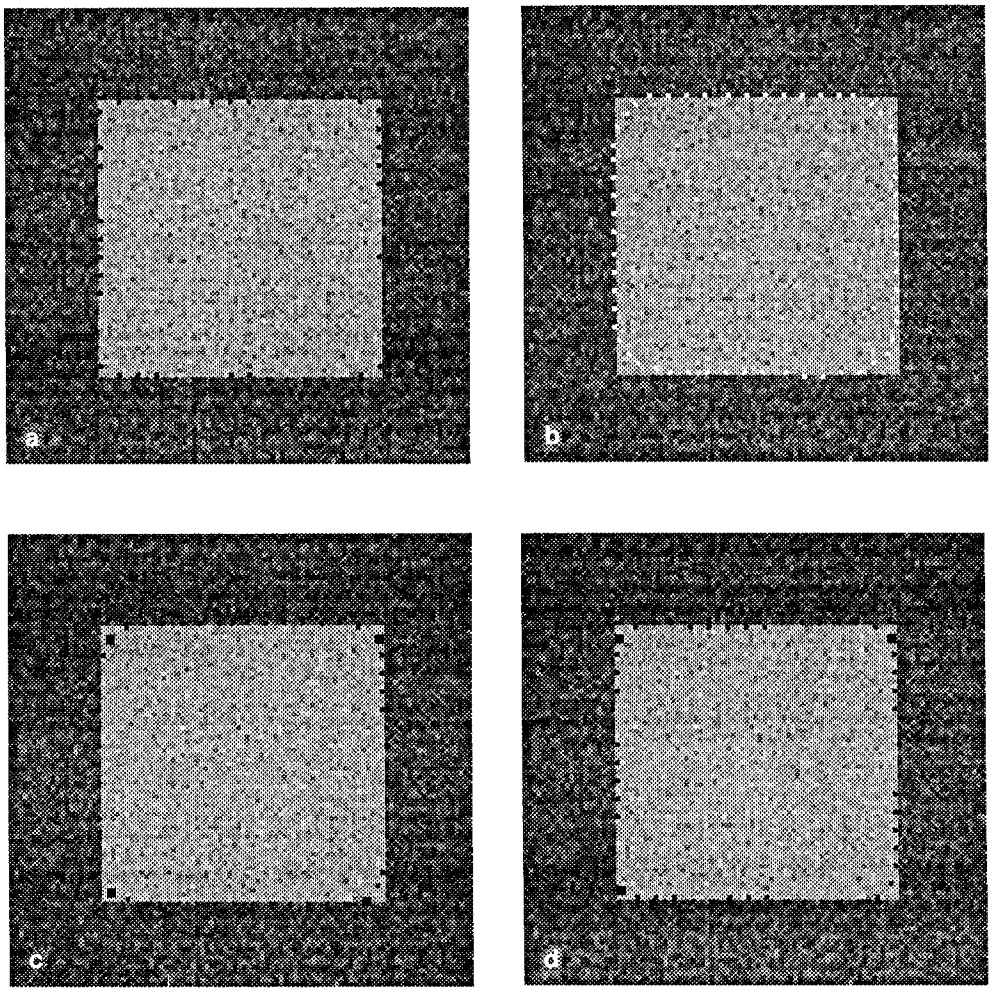

FIG. 6. Square. (a) Original contour, (b) Kass method, (c) Dynamic programming algorithm, (d) Greedy algorithm.

when the contour spanned a region where the edge was weak or missing. The initial contour for the Square was produced by the edge linking algorithm developed by the authors [7] and was quite good to start with. In all the image figures, the points on the contour that satisfied the conditions of high curvature are marked with larger squares. At these points the second-order continuity restraint was relaxed. The neighborhood examined at each point consisted of the point itself and its eight neighbors. Thus the neighborhood size, m, was 9. In the image figures, (a) shows the beginning contours, all of which had 40–60 points spaced a distance of approximately 4–6 pixels apart. The threshold for setting $\beta = 0$ was 0.25, corresponding to approximately 29°. Experimental results confirmed that this was sufficiently large to differentiate between what was perceived to be a corner and a curved line. The threshold for the minimum gradient magnitude before a corner would be marked was 100, when gradient magnitude was in the range 0–255. This is not a critical value, and a wide range of values gave similar results. The final threshold set was the number of points moved to determine that convergence had occurred. Small, nonzero values (2–5) worked quite well for this. See the third column in Table 2 for values actually used. The same method and threshold were used to determine convergence in the greedy and the dynamic programming methods.

Part (b) shows the result of allowing the original contour to converge to the edge around the object using the variational calculus method proposed by Kass *et al.*

Part (c) shows the result of the dynamic programming algorithm for the four pictures. In order to reduce the tendency to bunch up at strong points on the contour, one of the hard constraints prohibited movement perpendicular to the direction of maximum gradient. This did not prevent the points not currently on the edge from moving toward a strong edge point which was not the nearest point to the current location, but did prevent

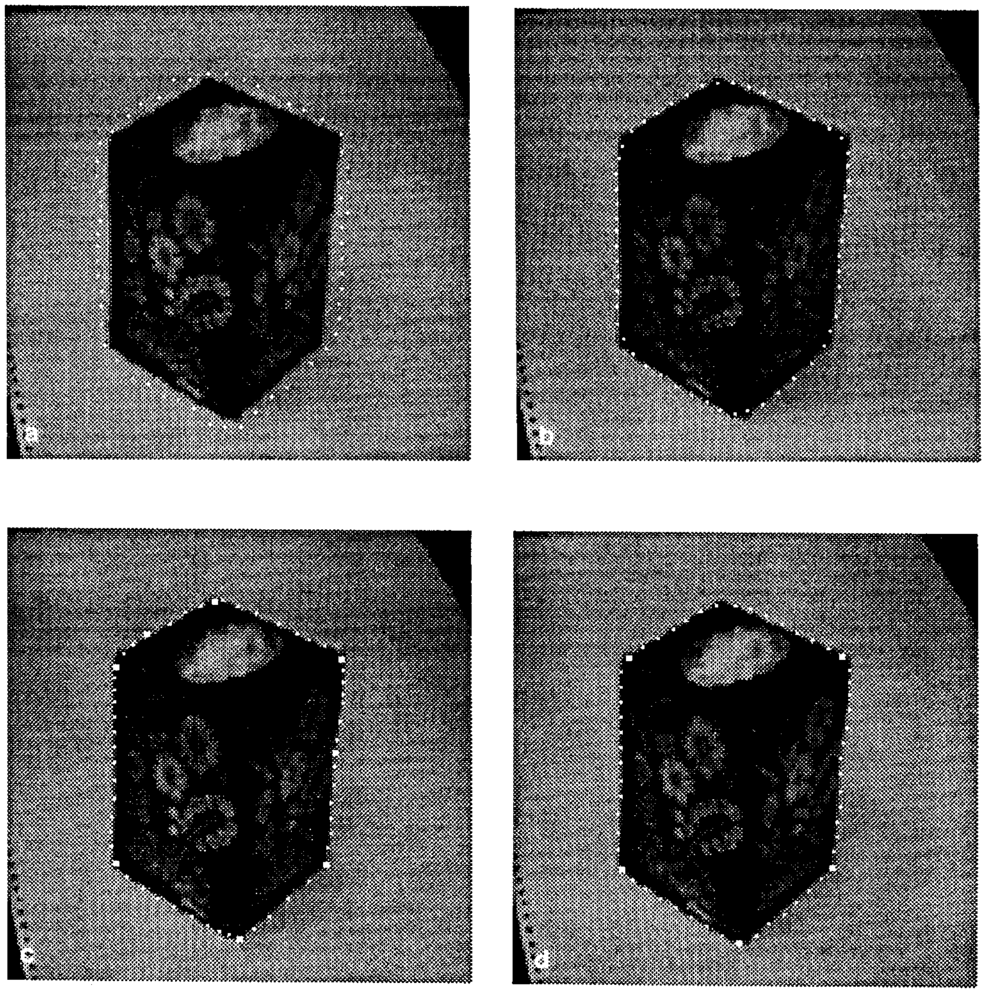

FIG. 7. Box. (a) Original contour, (b) Kass method, (c) Dynamic programming algorithm, (d) Greedy algorithm.

edges moving along the contour to higher points once they had reached the edge. Movement along the contour also extended the convergence time when this constraint was not included. The threshold given was the number of points which moved during the iteration. Usually the number of points being moved in each iteration dropped sharply when the contour approached the edge location. Note that the edge points are more closely spaced on the strong portions of the contour while in locations like the bottom of the cup handle there are no points.

Part (d) in each figure shows the results of the greedy algorithm. The results achieved by all three of the methods presented are comparable, one giving slightly better results in one image, while a different method gives bet-

TABLE 2
Comparison of Runtime in Seconds, Number of Iterations, and Number of Second-Order Discontinuities (Corners) Marked for the Greedy, Dynamic Programming, and Variational Calculus Methods

			Greedy algorithm			Dynamic prog.			Variational calc.	
Image	Size	Th.	Secs.	Iter.	Cor.	Secs.	Iter.	Cor.	Secs.	Iter.
Square	58	3	0.250	2	3	12.162	3	4	.350	4
Box	56	1	1.867	15	5	25.157	6	8	.466	4
Bottle	50	1	1.217	11	4	29.465	8	5	1.499	12
Cup	46	3	0.700	7	3	34.153	10	6	.300	4

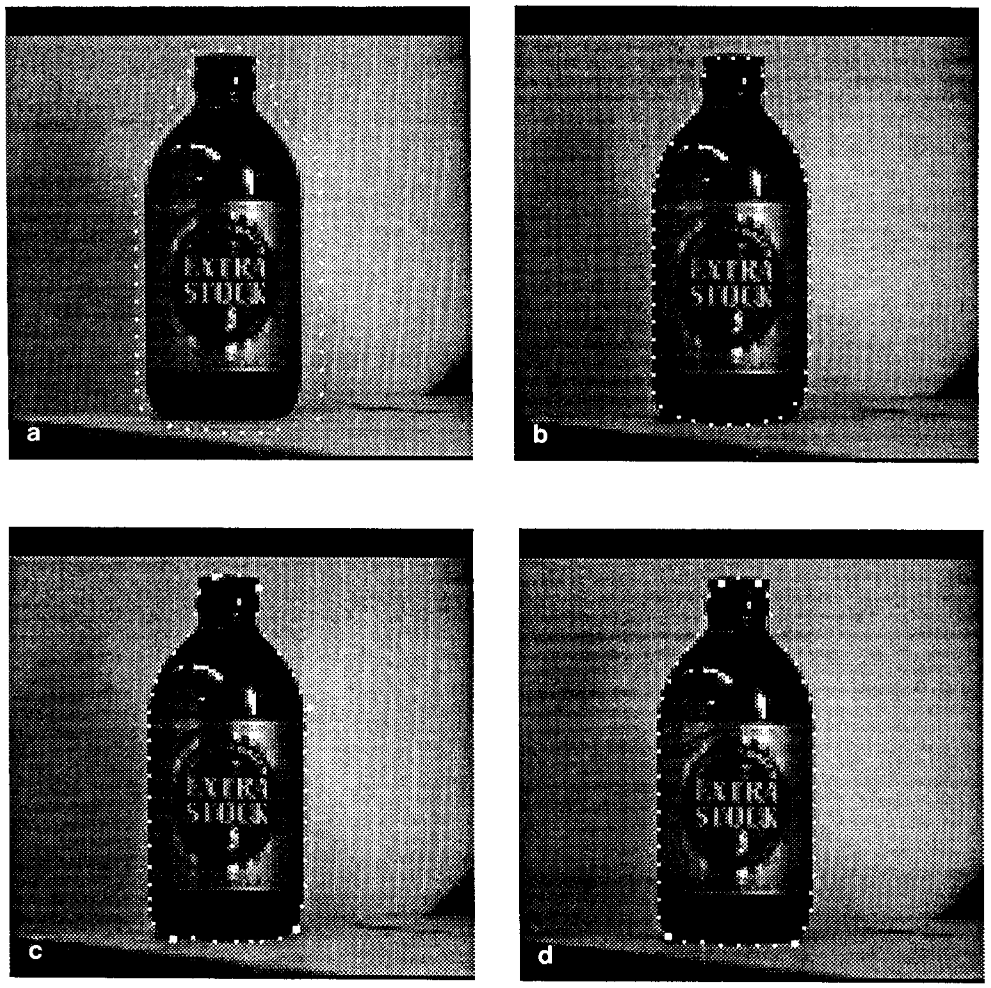

FIG. 8. Bottle. (a) Original contour, (b) Kass method, (c) Dynamic programming algorithm, (d) Greedy algorithm.

ter results in another image. The greedy algorithm has removed some of the small jogs from the inside of the square to the outside, but the dynamic programming algorithm has removed more of them. The original contour for the square was good, so there was very little change using any of the algorithms. Corners are not set with the Kass method, so it gives contours that are more rounded at the corners. The results on the Box are almost identical for the three algorithms, with the upper left edge being better with the greedy algorithm, while the upper right edge is slightly better with the dynamic programming. In the Bottle image, two points become very close together at the top and at the right-hand side with the dynamic programming, but remains evenly spaced in the greedy algorithm because of the different form of the first-order continuity constraint. The edge points do not follow the neck of the bottle as well in the greedy algorithm. As expected, the contour does not follow the right side of the cup well with any of the methods. Where the cup edges are not strong, points belonging to the background appear to be the points converged to with the greedy algorithm. All three converged to the shadow edge at the right-hand bottom corner of the cup rather than to the cup itself, since that was the first edge encountered as the contour approached the cup.

Table 2 gives the number of points in each contour, the threshold used for convergence, the user times in seconds, the number of iterations required to converge, and the number of points of high curvature at which the second-order continuity constraint was relaxed by setting β to 0. The algorithms were implemented in C on a Harris HCX9 minicomputer. Using the greedy algorithm, the speedup over dynamic programming was significant in all cases, varying from a factor of 13 for the Box, to 48 for

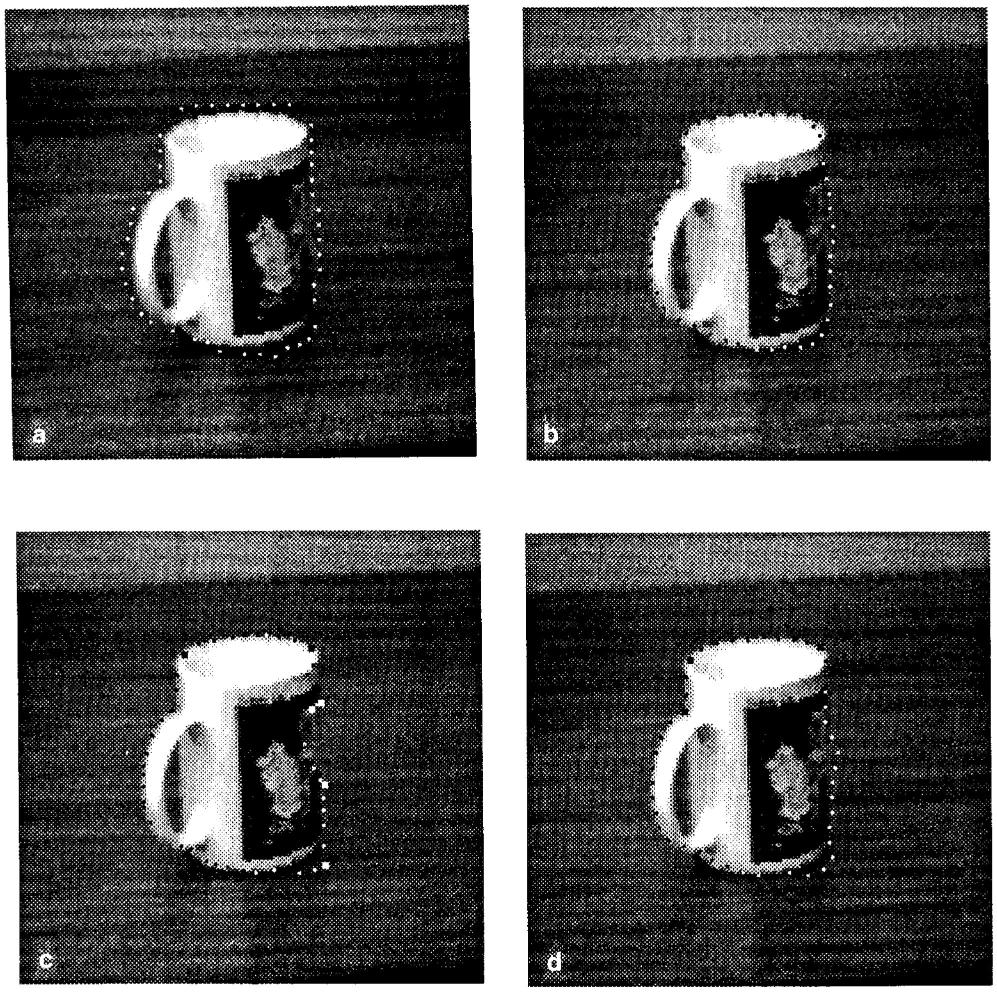

FIG. 9. Cup. (a) Original contour, (b) Kass method, (c) Dynamic programming algorithm, (d) Greedy algorithm.

the Cup and Square. Neither method was significantly better in the number of iterations required, with the Square and Cup having fewer iterations with the greedy algorithm, while the Box and Bottle required fewer iterations with the dynamic programming algorithm. The results of the contours obtained with the greedy algorithm are at least as good as those of the dynamic programming algorithm, and the run times are much better. The variational calculus approach required time comparable to the greedy method for each iteration, but usually converged in fewer iterations. If values of β were allowed to change between iterations, the inverse of a pentadiagonal $n \times n$ matrix would need to be computed, slowing its speed.

Figure 10 shows a sequence of images produced as the contour converges to the edge of the bottle, using the greedy algorithm. The edges all move smoothly toward the bottle except for one point at the right-hand bottom corner which is initially stationary, then as the curvature and continuity energy becomes large in that area it begins to move as well. In the final image the contour has settled nicely around the edges of the bottle.

7. CONCLUSIONS

A method of controlling snakes has been presented which combines speed, flexibility, and simplicity. It was compared to the original variational calculus method of Kass *et al.* and the dynamic programming method developed by Amini *et al.* and found to be comparable in final results, while being faster than dynamic programming and more stable and flexible for including hard constraints than the variational calculus approach. The introduction of the concept of curvature highlighted the prob-

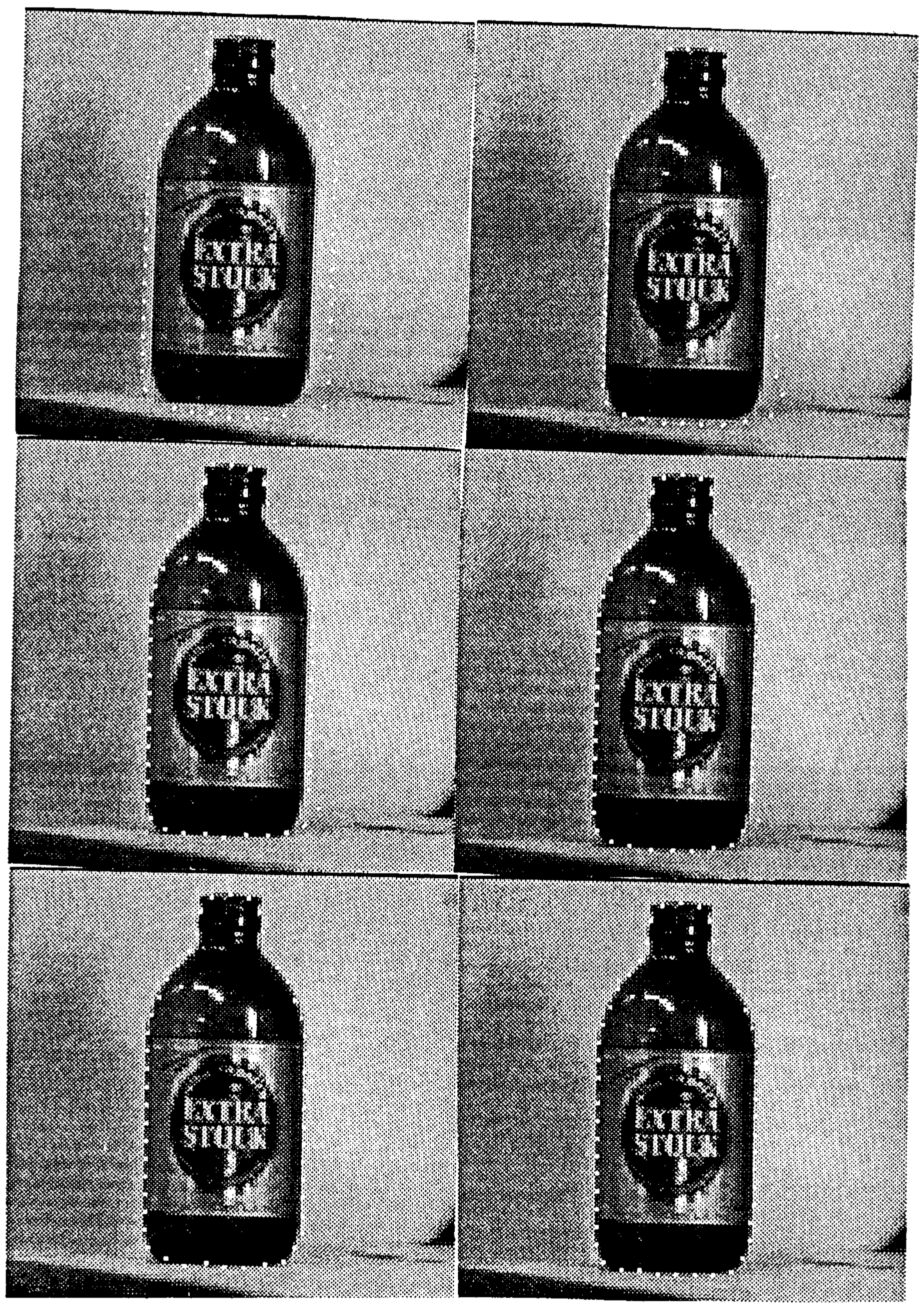

FIG. 10. Sequence showing convergence of a contour to edges of bottle using the greedy algorithm.

lem of how to approximate curvature when a curve is represented by a set of discrete points. The advantages and disadvantages of a number of different approximations of curvature were pointed out.

REFERENCES

1. A. A. Amini, S. Tehrani, and T. E. Weymouth, Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints, in *Proceedings, Second International Conference on Computer Vision, 1988,* pp. 95–99.

2. J. F. Canny, A computational approach to edge detection, *IEEE Trans. Pattern Anal. Mach. Intelligence* **PAMI-8** (1986), 679–698.

3. M. Kass, A. Witkin, and D. Terzopoulos, Snakes: Active contour models, in *Proceedings of First International Conference on Computer Vision, London, 1987,* pp. 259–269.

4. D. Marr and E. Hildreth, Theory of edge detection, *Proc. R. Soc. London B* **207** (1980), 187–217.

5. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Comput. Sci. Press, Rockville, MD, 1982.

6. D. Williams, *Edge Contours,* Ph.D. thesis, University of Central Florida, 1989.

7. D. Williams and M. Shah, Edge contours using multiple scales, *Comput. Vision Graphics Image Process.* **51** (1990), 256–274.

8. A. Witkin, Scale-space filtering, in *Proceedings Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, W. Germany,* pp. 1019–1021, IEEE Comput. Soc., 1983.

9. A. Y. Yuille, D. S. Cohen, and P. W. Hallinan, Feature extraction from faces using deformable templates, in *CVPR, 1989,* pp. 104–109.