COMPUTER SCIENCE TECHNICAL REPORT

Snakes and their influence on visual processing

Ken Tabb and Stella George

Report No 309 Feb 1998

Snakes and their influence on visual processing

Ken Tabb and Stella George

February 26, 1998

Abstract

Consideration is given to the use of snakes (active contour models) for tracking objects in a visual field. The basic theory of snakes is covered, and a greater depth of analysis given to both the original [1] and the Fast Snake [2] implementations. The strengths and limitations of these models, for tracking deformable objects using unaided visual systems, are discussed. Suggestions are made that their future lies in their integration with neural network visual location prediction systems.

1 Introduction

Visual object recognition is a problem that has perplexed researchers in computer vision for years. Traditionally the field of visual processing has been subdivided into different fields of research such as stereo vision, motion, edge detection, image filtering and colour [3]. Researchers have consequently tended to treat each research field in isolation and used image processing techniques specific to the areas they were studying, for example optic flow filters for studying ego-motion [4]. The most accurate vision algorithms developed tended to be the most computationally demanding, and were typically successful only in the specific problem studied.

Active contour models, or 'snakes' as they are widely known, offer the potential to integrate all of the above fields of computer vision. By changing either a small number of parameters in the energy function, or by modifying the energy function criteria itself, snakes can be adapted to suit a wide range of applications. Developed originally to assist end users identify object outlines from images, snakes have more recently been adopted by computer vision programmers as a means of object detection [5], tracking [6] and model construction [7].

This paper investigates the basic theory of snakes and discusses the relative advantages of the Fast Snake algorithm [2] which claims to address some of the problems identified in the original model [1]. Finally other mechanisms which may help overcome some of the practical problems of tracking deformable objects are outlined. Suggestions are made concerning the integration of neural networks with active contour models to solve these practical problems.

2 Generic snake properties

This section outlines those properties which are generic to all snakes, irrespective of their implementation.

A snake is an energy minimising contour, or spline, which can detect the edges of a visual object in a scene and track them during the object's motion [1]. Snakes have no 'knowledge' of the object's size, shape or location in the image, responding instead to the physical relationship of edges within a specific image.

A snake is defined as a series of consecutive (x,y) points ('control points') in an image (Figure 1), or in one frame of a movie. Collectively the control points form a partial contour in the image. Snakes are typically drawn with lines connecting adjacent control points. These lines play no part in the definition of the snake, serving only to illustrate the connectivity of control points on the snake to the viewer, and to give an impression of the shape of the snake. Whereas

the connecting lines are not important, the order of control points in the snake vector is relevant to the snake's definition, and remains constant. A given control point will always have assigned to it the same two neighbours, irrespective of where those control points move to in the image. A given control point and its preceeding and succeeding neighbours along the snake are usually termed cp_i, cp_{i-1} and cp_{i+1} respectively.

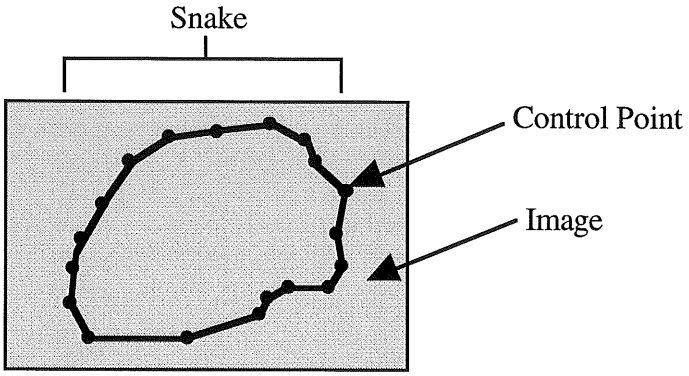

Figure 1: A snake is a series of (x,y) control points in an image. The lines connecting the control points do not form part of the snake definition. Instead they serve to indicate the contiguity of the control points.

A snake's movement is controlled through energy minimisation, where each constituent control point's position (both relative positioning to other control points and its (x,y) coordinates in the image) determines the energy function which is minimised. Snakes have an energy function that allows each control point to be given an energy level. Each control point is moved in a direction that minimises its energy level. When all control points on the snake have had their energy minimised, i.e. when the snake's list of constituent points has been iterated through once, the snake can be seen to move. This energy minimisation process is repeated until the energy difference (and therefore control point movement) between two consecutive iterations of the snake is either negligible or nil. At this point the snake is said to have locked onto a salient contour in the image.

The energy function used to minimise control point energy determines the local movement of each control point and, in turn, the meta-behaviour of the snake itself. Snakes are both able to move themselves (via energy minimisation), and be moved (via user interaction), around an image depending upon the control point energy function used in the snake. The criteria, direction and distance of movement for a control point is also determined by the snake's energy function. The control point energy function used can be customised to attract the snake (minimise its energy) toward particular features in the image. Whether the snake's control points are moved synchronously or asynchronously depends upon the particular snake model.

The energy minimisation function for a snake can comprise any or all of the following three types of energy:

- Internal energy
- Image energy
- External energy

Internal energy measures energy pertaining to the geometric shape of the snake. It takes no information from the image in which the snake resides, but is instead affected by the number and relative positions of control points along the snake.

Conversely, image energy uses the area of image immediately surrounding a particular control point to calculate its image energy value. The number and relative positioning of control points plays no part in this calculation.

External energy does not use the image or the snake itself (in terms of its constituent

control points' relative positions) as a source of information. Instead it is a measure of the user interaction that has taken place on a control point during an iteration; whether or not the user has dictated how far and in which direction a control point should be moved, for instance.

3 The original Active Contour Model

This section details the original active contour model, pioneered by Kass, Witkin and Terzopoulos [1].

3.1 Active contour models - Objectives

The original active contour model offered semi-automatic mouse-controlled object detection via user guidance. The user would click a series of points roughly around the object that he or she was interested in, forming an initial contour, and the snake would then fit itself automatically to the nearest salient image feature. If the snake did not fit the object well enough, or was unable to track it properly, the user could 'push' individual control points in the direction of the object (with their mouse), allowing the snake to overcome any local minima it may have encountered. The active contour model could track objects by continually re-minimising its energy in successive frames of a movie, subject to the user helping it.

That active contours can accurately track an object in a sequence of images, regardless of that object's shape, size and location makes them suitable for performing many computer vision image preprocessing tasks, such as edge detection, simple shape tracking and user-guided complex shape tracking. The lack of model-based knowledge regarding an object's deformability or motion behaviour, however, imposes major limitations on the power of active contour models. Without the ability to predict an object's movement (by using information pertaining to the object's patterns of motion and deformability), the model cannot track objects which become occluded. Furthermore by depending upon user interaction, the model cannot be used to automatically construct shape descriptions of objects from data without a great deal of human intervention.

In its defence, the active contour technique was not developed with these features in mind. Computer vision programmers have nonetheless adopted snakes and are continually adapting them to solve different problems, for instance cell tracking and surface reconstruction problems in the biological domain [5].

3.2 Control point energy

The energy function implemented in the original active contour models contained all three types of energy (internal, image and external energy).

Each of the three energies strive to move the control point in the direction which would most benefit (minimise) that particular energy - for example internal energy functions try to minimise a control point's internal energy. Each type of energy has its own weighting (set by the user), and the three energy values for a control point are totalled according to this weighting in a tug-of-war fashion. This total determines an overall direction to move the control point which would benefit the weighted majority of its three energy values.

Control points are moved synchronously in the original model. Each control point uses its adjoining neighbours' (x,y) locations from the previous position of the snake to calculate its energy. The control point's previous x coordinate is added to its total energy in the x axis in order to obtain it's new x coordinate, with a similar operation performed to obtain its new y coordinate, resulting in a new (x,y) coordinate (the algorithm used for this can be seen in Appendix A). All control points are actually moved to their new (x,y) locations at the end of the iteration. Consequently one control point's movement in an iteration will not affect neighbouring control points' energy values (and therefore movements) during that same iteration.

A control point's overall energy value can be defined as [1]:

$$\text{ControlPointEnergy}_{(i)} = (\text{IntEnergy}_{(i)} + \text{ImageEnergy}_{(i)} + \text{ExtEnergy}_{(i)}) \quad \text{[Equation 1]}$$

where i represents the given control point, $\text{ControlPointEnergy}_{(i)}$ represents that control point's total energy level, and $\text{IntEnergy}_{(i)}$, $\text{ImageEnergy}_{(i)}$, and $\text{ExtEnergy}_{(i)}$ represent the control point's internal, image, and external energy levels respectively. It is this energy function which is minimised.

The snake's overall energy can be defined as [1]:

$$\text{SnakeEnergy} = \sum_{i=1}^{n} \text{ControlPointEnergy}_{(i)} \quad \text{[Equation 2]}$$

where SnakeEnergy represents the overall energy of the snake, i represents each control point, and n represents the total number of control points on the snake.

In practice, the snake's overall energy value is of little use, instead the total energy levels of each individual control point (defined in Equation 1) are used. This is because all operations on the snake are made at the (local) control point level rather than at the (global) snake level.

3.3 Energy function

The three different types of control point energy collectively determine which location the given control point should be moved to in order to minimise its energy. Internal energy aims to maintain a consistent global shape for the snake by moving a control point based upon its adjoining neighbours' locations; image energy aims to lock onto strong features in the image; and external energy aims to move the control point to wherever the user dictates. This section details how each type of energy achieves these goals. The effects of each type of energy are discussed in isolation to the other energies; in practice the energy levels would be combined, and this value would then be used to move the control point to a location which would benefit most of the energy types.

3.3.1 Internal energy (IntEnergy_i)

The internal energy function moves a control point based upon the locations of the neighbouring control points in the snake, i.e. based upon the snake's overall shape. It can be regarded as being sensitive to the geometric shape of the snake. It does not use the image in which the snake resides for any of its information.

In the original active contour model, the internal energy function has two components: an elasticity function and a bending function. The total internal energy for a given control point on the snake can be defined as [1]:

$$\text{IntEnergy}_i = \text{ElasticEnergy}_i + \text{BendingEnergy}_i \quad \text{[Equation 3]}$$

$$\text{ElasticEnergy}_i = \alpha(((x_{i-1} - x_i) + (x_{i+1} - x_i)) + ((y_{i-1} - y_i) + (y_{i+1} - y_i))) \quad \text{[Equation 4]}$$

$$\text{BendingEnergy}_i = \beta((4(x_{i-1} + x_{i+1})) - 6x_i - (x_{i-2} + x_{i+2}) + (4(y_{i-1} + y_{i+1})) - 6y_i - (y_{i-2} + y_{i+2})) \quad \text{[Equation 5]}$$

where (in Equation 3) IntEnergy_i is the total internal energy for control point i on the snake, ElasticEnergy_i is the first order continuity for point i on the spline representing the snake's coordinates, and BendingEnergy_i is the second order continuity for point i on the same spline.

In Equation 4, ElasticEnergy_i is the elastic energy for control point i, α is the user-

defined elastic energy weighting parameter, x_i and y_i are the x and y coordinates for control point i (respectively), and x_{i-1}, y_{i-1}, x_{i+1} and y_{i+1} are the x and y coordinates for the previous and next adjacent control points on the snake (respectively). Equation 4 is discussed in greater depth later in this section.

In Equation 5, $BendingEnergy_i$ is the bending energy for control point i, β is the user-defined bending energy weighting parameter, x_i and y_i are the given control point's x and y coordinates in the image, and x_{i-2}, y_{i-2}, x_{i-1}, y_{i-1}, x_{i+1}, y_{i+1}, x_{i+2}, and y_{i+2} are the x and y coordinates of the given control point's four neighbours, ranging from the previous-but-one control point to the next-but-one control point (respectively). Equation 5 is discussed in greater depth later in this section.

The user-defined parameters α and β serve the purpose of adjusting the ratio of elastic to bending energy. By adjusting α and β, the user can make the snake more (or less) elastic and more (or less) bendy, allowing different snake characteristics to dominate in different tasks. For instance, if the objects to be detected are all approximately circular in shape then the bending parameter β should be small. Conversely, in a situation where the target objects contain corners, β should be larger to allow the snake to rest along the peninsulas which form the object's outline, as is shown in Figure 2.

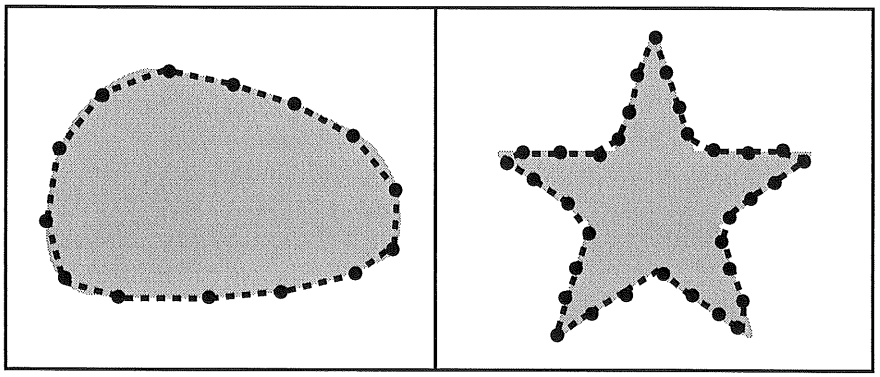

Figure 2: The type of object being detected should be considered when setting the internal energy parameters α and β. When detecting objects with no corners (left) the bending parameter β should be small (to avoid corners from forming in the snake). Conversely, when detecting objects which contain many corners (right), the bending parameter β should be large to enable corners to develop in the snake, allowing a closer match between the snake and the actual outline of the target object.

Elasticity term ($ElasticEnergy_i$)

The elastic energy term measures how much each control point should be pulled towards its neighbours. By pulling each control point on the snake closer to its neighbours, the snake's overall effect is that of shrinking (provided the snake forms a closed loop, where the final control point connects to the first). This allows the user to click an initial snake roughly around the target object, and for that snake to then 'shrink-wrap' itself onto the target image. The opposite effect can also be achieved, that of expanding the snake from within the target object to the edges, by setting the snake's elastic weighting term α to be negative. On its own, elastic energy would cause the snake to continually contract or expand beyond the target object's outline. When used with image energy and external energy (discussed in sections 3.3.2 and 3.3.3 respectively) however, the control points can be stopped from moving when they reach an edge in the image.

The elastic energy for a control point is defined in Equation 4 (above). Each control point along the snake is moved according to the magnitude of its own elastic energy value: the bigger the energy, the further that control point is moved. The favoured movement which

would most minimise the control point's elastic energy (assuming no other types of energy were also pulling the control point in their favoured directions) is returned by the equation as different numbers of pixels in the X and Y axes. The direction of movement is also returned by means of the polarity of each number; positive X and negative Y values together would suggest moving the control point up and to the right in the image (the origin of the image being at the top left). The elastic energy of a given control point is multiplied by α during its calculation; the elastic energy level (and the control point's subsequent movement) is therefore determined to some degree by the magnitude and polarity of α. Changing the value of α determines the snake's rate of contraction (or expansion), whereas changing the polarity of α determines whether the snake will contract (positive) or expand (negative).

Movement based on Equation 4 follows Pythagoras' rule. Control point **i** is pulled towards both of its neighbours at once, becoming equidistant to both. If α is large then the distance between the control point and its neighbours (both neighbours are the same distance away) will be minimised more than if α were smaller, causing the control point to lie not only equidistant to, but geometrically between, its neighbours (Figure 3).

Increasing α magnifies the effect the snake's elasticity will have on its movement. If α is set too large, the snake's elasticity will totally dominate its future movement, causing the slightest inequality in distance between control points to oscillate the snake. If α is set to 0.0, the snake's elasticity will not affect its future movement.

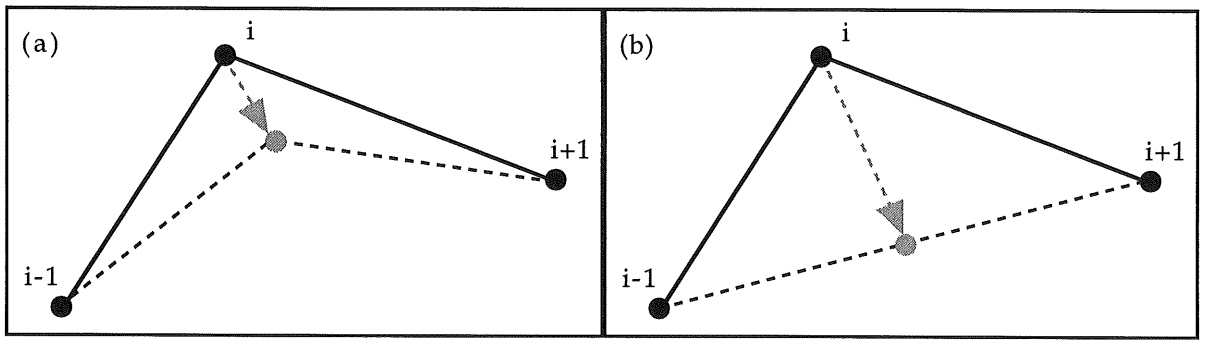

Figure 3: **(a)** If α is small (less than 0.01) control point **i** will be positioned equidistant to its two neighbours but **i** will not end up lying geometrically between its neighbours i-1 and i+1. **[b]** Setting the elastic weighting term α to be large (though usually less than 0.2) results not only in control point **i** being situated equidistant to its two neighbours, but also in those (equal) distances being minimised to a greater degree. The result is that **i** is pulled into line to exist geometrically between its two neighbours. Consequently, larger values for α result in the snake shrinking more rapidly than smaller values for α.

If α is negative, the distance between the control point and its neighbours will be maximised, not minimised. The control point is consequently moved further away from, although still equidistant to, its neighbours. The greater the magnitude of negative α, the faster the control point will 'run away' from its neighbours. When all control points have run away from each other, the snake can be seen to expand. Expanding snakes can be useful in situations where for example the target object has a flat interior (i.e. has few internal edges and is of uniform colour) and where the background environment is complex (many edges and colour changes). In this situation, it is more reliable to start the snake inside the target object's boundary and to then expand it, as there are fewer misleading edges which it could lock onto.

Bending term (BendingEnergy$_i$)

The bending energy function calculates how flexible the snake is; that is, whether or not a given control point is allowed to form a corner in the snake. If the bending energy term β is

small (typically less than 0.05) then the snake is not allowed to become too flexible, and will instead try to converge on a smooth curve.

Bending energy measures the second order continuity of the snake (Equation 5), the curvature being measured between five adjacent control points along the snake (the given control point being the central of the five points, with two neighbours either side). The various control points' coordinates are given weightings as to how much they govern control point **i**'s movement. Control point **i**'s coordinates are given the largest weighting, followed by its immediate neighbours, followed by the more distant neighbours, in the ratio 6:4:1 respectively. If these weightings were not present, closed loop snakes would oscillate more dramatically than they do, irrespective of whether the elastic energy was also contracting (or expanding) the snake.

The smaller the curvature (i.e. the more jagged a corner) between a given control point and its four immediate neighbours, the bigger the resulting bending energy value for that control point. Minimising the bending energy therefore involves trying to obtain 180° curvature between the given control point and its neighbours (i.e. forming a straight line).

As with the elastic energy, the control point is moved according to the magnitude of its bending energy (as returned by the equation); the larger the bending energy value, the further it will be moved. The movement of the control point in this way minimises the curvature that exists between the control point and its four neighbouring control points, therefore minimising its bending energy. The overall effect is that the snake is encouraged to exist either as a straight line (if the snake does not form a closed loop) or as a smooth curve (if the snake forms a closed loop) (Figure 4).

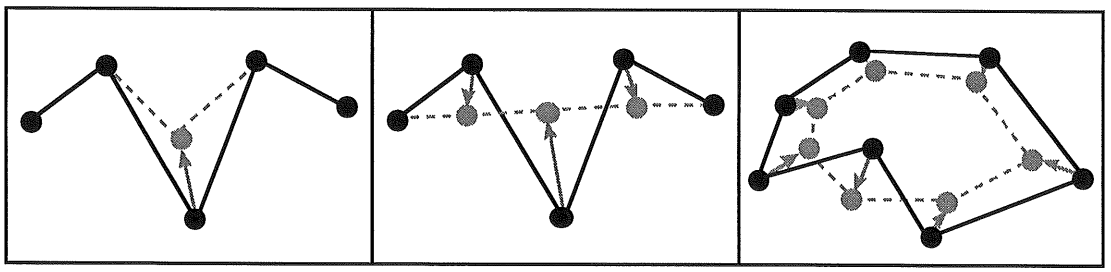

Figure 4: The bending energy serves to enable (or disable) corners from forming along the snake by pulling the given control point to a location directly between its four neighbours. [Left] The effect of bending energy on one control point in the snake. [Middle] The effect on an open-ended snake following several iterations along the snake; the snake is seen to straighten out. [Right] The effect on a closed loop snake following several iterations around the snake; the snake shrinks slightly, but fits a smoother spline curve.

It is probably easiest to envisage the effects of bending energy upon a snake as that of dropping a thin metal strip [8]. If the two ends aren't joined up the strip will open out into a straight line, whereas if the two ends of the strip are joined together (forming a closed loop) it becomes a smooth curve with no sudden corners.

Usually the value for β is set as being less than 0.1. Setting β to be too large encourages every control point along the snake to become a corner, whereby the snake oscillates off the image. Conversely if β is set to 0.0, the snake's curvature will not affect its future movement. In cases where the target object contains many corners, it may be necessary to allow the snake to form corners, so that it may lock onto the target object's contour as closely as possible.

The more control points the snake contains, the more irregular a contour it can lock onto. With a small number of control points (as in the right-most illustration in Figure 4), any corners forming in the curve would result in a large curvature value between control points; therefore the bending energy will tend to produce snakes with no corners. A 'lumpy' shape can still be achieved if more control points are used. The snake as a whole may contain significantly curved sections, but the curvature between adjacent control points may still be relatively small,

satisfying the bending energy criteria (Figure 5).

Together, the two internal energy forces (elastic and bending) govern the global shape of a snake, based upon the coordinates of neighbouring control points.

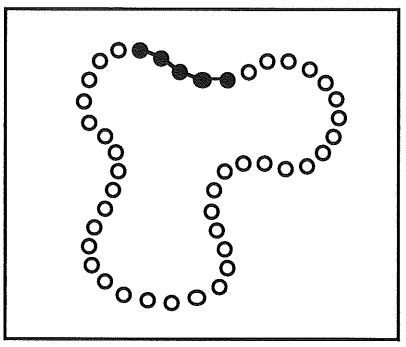

Figure 5: Even with weak bending energy acting to smooth out the 'lumps' in a snake, a complex shape (and therefore accurate detection of complex-shape target objects) can still be achieved if the snake contains enough control points that the curvature between any set of five adjacent control points is not large.

3.3.2 Image energy ($ImageEnergy_i$)

The image energy function consists of algorithms which move a control point based upon information obtained from the area of the image in which that control point resides. Neither the number nor positions of the other control points in the snake affect the given control point's image energy value.

The original snakes have three image energy algorithms: a line detection algorithm, an edge detection algorithm, and a line termination detection algorithm. A control point's image energy can be defined as [1]:

$$ImageEnergy_i = \gamma\,((\lambda\ LineEnergy_i) + (\mu\ EdgeEnergy_i) + (\nu\ TermEnergy_i)) \qquad \text{[Equation 6]}$$

$$LineEnergy_i = I_{(x,y)} \qquad \text{[Equation 7]}$$

$$EdgeEnergy_i = -\left|\nabla I_{(x,y)}\right|^2 \qquad \text{[Equation 8]}$$

$$TermEnergy_i = \frac{\delta\theta}{\delta n_\perp} \qquad \text{[Equation 9a]}$$

$$= \frac{\delta^2 C\ /\delta n_\perp^2}{\delta C\ /\ \delta n} \qquad \text{[Equation 9b]}$$

$$= \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2 + C_y^2)^{3/2}} \qquad \text{[Equation 9c]}$$

where, in Equation 6, $ImageEnergy_i$ represents the total image energy value for control point i, γ represents the user-defined image energy weighting term, λ represents the user-defined line energy weighting term, $LineEnergy_i$ represents the control point's line energy value, μ represents the user-defined edge energy weighting term, $EdgeEnergy_i$ represents the control point's edge energy value, ν represents the user-defined line termination energy weighting term, and $TermEnergy_i$ represents the control point's line termination value.

In Equation 7, x and y represent the control point's X and Y coordinates in the image,

and $I_{(x,y)}$ represents the greyscale (or colour) intensity value at location (x,y) in the image. Equation 7 is discussed in greater depth later in this section.

In Equation 8, $EdgeEnergy_i$ is the edge energy value for control point i, x and y are the control point's coordinates in the image, and $\nabla I_{(x,y)}$ is the greyscale gradient at location (x,y) in the image with respect to the pixels below and to the right of it. Equation 8 is discussed in greater depth later in this section.

In Equations 9a, b and c, $TermEnergy_i$ represents the line termination energy for control point i, C is the original image following a standard Gaussian smoothing of the greyscale pixel values, θ is the gradient angle (equal to $\tan^{-1}(C_y/C_x)$), n is the unit vector running along the gradient direction (the vertical arrow in Figure 6), equal to ($\cos\theta$, $\sin\theta$), and $n_\perp$ represents the unit vector running perpendicular to the gradient direction (the horizontal arrow in Figure 6), equal to ($-\sin\theta$, $\cos\theta$). Equations 9 a, b and c are discussed in greater depth later in this section.

γ is used to adjust the internal to image energy ratio. This is useful in situations where for example the image is noisy (or contains many misleading edges) and the shape of the target object is geometrically regular. In this instance, the image energy's impetus in moving the control point can be 'watered down' so that the snake doesn't necessarily stick to every black dot it finds in the image, due to the relative increase in the internal energy's contribution. Conversely, in a situation where image resolution isn't an issue, but the target object's shape is extremely irregular, it may pay to increase the relative contribution of the image energy to guide the snake.

Line detection ($LineEnergy_i$)

The simplest information that can be obtained from the image is the image intensity itself, i.e. the greyscale (or colour) value of a particular pixel in the image. Using this as a component of the image energy attracts the snake towards light or dark areas in the image structure; changing the polarity of the line detection term determines whether the snake is attracted towards light or dark areas. If a snake has been initialised within an area of smooth greyscale gradient (i.e. ramped shading) this feature is used to keep the snake moving up or down the gradient, until it reaches the end of the ramp. Once there the edge and line termination energies will lock it onto any nearby edges. If it weren't for the line detection energy term, the snake could become trapped in the local minimum consisting of equal greyscale gradient all around.

The line energy for a control point is defined in Equation 7, above. The larger the greyscale value at (x,y) in the image (i.e. the brighter the pixel), the larger the control point's line energy value will be. Consequently the snake will be attracted towards darker areas of the image, as it is aiming to minimise its energy. If the snake is required to be attracted toward brighter areas of the image, it is a simple matter of negating the $LineEnergy_i$ energy value by supplying a negative Line Energy weighting term.

Edge detection ($EdgeEnergy_i$)

The second of the three image energy algorithms is the edge energy value. A control point's edge energy aims to guide it towards the strongest salient image feature (edge). It uses a simple image gradient function to calculate this: if there is a large difference in the greyscale (or colour) value between a pixel and the neighbouring pixels below and to the right of it, an edge is detected.

A control point's edge energy value is defined in Equation 8, above. The value of the squared gradient $|\nabla I_{(x,y)}|$ is negated to form the edge energy for the control point; the larger the gradient (edge) that the control point lands on, the more negative the resulting edge energy. Since energy is to be minimised, this helps keep the control point on strong edges. If the negation were not present, strong edges would result in larger energy values, which would in turn mean that control points would be repelled from strong edges.

Here it can be seen why there is a need for the user defined energy terms (α, β, and γ) to balance the internal and image energies: it allows the control point to lock onto a strong edge once it has found one, rather than risk being pulled off it because the internal energies are trying to make the snake circular, for instance. However the appropriate values to use for α, β, and γ are difficult to guess and require comparison to set correctly (see later).

Line termination detection ($TermEnergy_i$)

The third of the three image energy algorithms is that of the line termination energy. This aims to guide a control point onto nearby ends of lines. There are typically two reasons why a line ends: either because that line/edge in the object has become occluded (by another of the object's edges or by another object), or because that part of the object's contour has drastically changed direction, forming a corner in the edge. Either way, it is useful for control points to be attracted towards these terminations, as it signals the end of one object and the start of another. As such it marks the boundary between the two objects, irrespective of which object the rest of the snake is trying to detect.

The line termination energy of a control point is defined in Equations 9a, b and c, above. The algorithm works on a smoothed version of the image. Smoothing reduces noise in the image. In image processing terms, noise reduction can be crucial when looking at lines or edges. For example, if a white line running across the original image for some reason has a black dot on it, the one line may appear to be two lines positioned one after the other. If the dot is small, a smoothing algorithm (such as Gaussian smoothing) will merge the two halves of the line together, removing the noisy dot from the image and allowing the line to be treated as one long line. If the dot is not small (the threshold can be set in smoothing algorithms to change the definition of how small a small dot is), the algorithm will not join the lines up; they may, after all, be two lines which happen to be positioned one after the other. If the line termination algorithm were not working on a smoothed image, it would be detecting a lot more line terminations (the apparent ones caused by noise in the image) than are actually present, preventing the snake from accurately mapping onto the target object's contour.

The algorithm detects large changes in the primary and secondary greyscale (or colour) gradients at the control point's location in the smoothed image (Figure 6). Finding the primary and secondary gradients at point (x,y) in an image identifies any lines that the point may lie on. If there is a large change in the secondary gradient (the horizontal arrow in Figure 6), then that line has terminated. The larger the gradient, the more defined the termination is, and the smaller the resulting line termination energy, causing control points to be attracted towards strong line terminations.

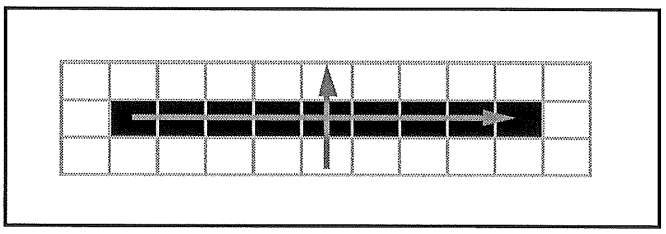

Figure 6: The primary (vertical arrow) and secondary (horizontal arrow) greyscale gradients along a line in an image.

One of the abilities of a snake incorporating the line termination energy function is the detection of subjective contours, where the arrangement of line terminations and other edges give rise to an apparent 'invisible' object in the image (Figure 7).

Combining the edge energy and the line termination energy allows a snake's control points to detect the subjective triangle in the above image. Control points would either be attracted toward one of the triangle's intersections with the circles (edge energy) or to the ends

of one of the three lines, where the triangle has occluded their continuation (line termination energy). Additional control points would be held in place along the subjective edges of the triangle (in the white space) by means of the spline properties of the snake (internal energy).

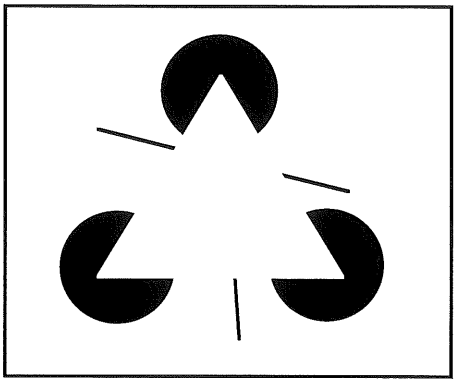

Figure 7: The line termination component of a snake's image energy algorithm can help attract it to subjective contours by locking it onto the ends of lines [1]. Typically when a line ends, it signifies the end of one object and the start of another; either the line has genuinely ended (marking the boundary between the line's parent object and the next object in the image), or it has been occluded by another object (again marking the boundary between two objects).

3.3.3 External (user) energy constraints

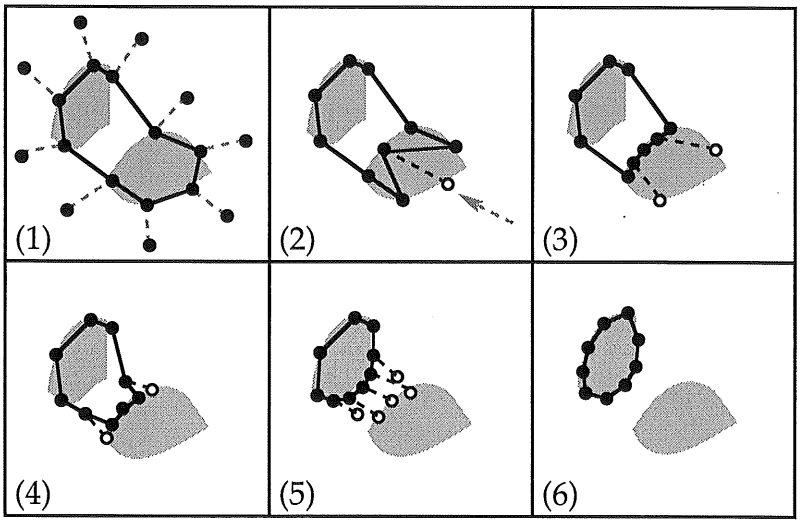

Figure 8: If some of the snake has locked onto the wrong edge, the user can 'push' the incorrect part of the snake in the direction of the correct edge. Solid lines denote the snake; dashed lines denote control point migration. (1) An initialised snake has relaxed onto two objects. (2) The user 'pushes' one of the control points in the direction of the target object. (3) The neighbouring control points move as a result of the internal spline characteristics of the snake, and of the image energy. (4 & 5) The control points again move due to the overpowering internal spline properties of the snake; note the control point originally moved by the user continues to move as its neighbours pull it in the same direction it was travelling. (6) Once around the target object, the control points space themselves out due to the elastic energy (spacing them evenly) and the need to minimise the bending energy (which is minimised with smoother curves). The control points stay around the target object's edge as a function of the edge energy term.

External energy functions are constraints imposed by the user on the system. In effect, the user can move a snake from one local minimum to another by 'pushing' the snake (Figure 8). Alternatively the user can attach a spring to a given control point. The other end of the spring can either be fixed or free. It can be fixed to a certain part of the image structure or to another control point on the snake (or any other snake in the image). If the other end is free then the spring can be used as a 'dog's lead'. The latter technique enables the snake to latch onto various contours as the user drags it around the image by its lead.

Pushing control points moves them in a certain direction, but does not constrain them to that direction; it is quite possible that the other energies may continue its movement in that direction, halt its movement, or even change the direction slightly.

Using springs not only biases the control point's movement; they also allow certain parts of the snake to become permanently locked onto certain parts of the image, imposing hard constraints which the other control points on the snake must honour. If the internal energy tries to minimise large curvature between a locked control point and its neighbours for instance, the neighbours will be moved rather than the locked control point. In this way, user interaction has the highest precedence of the three energies; an important prerequisite for a user-guided tool.

The algorithm for the external energy function employed in the original snake model is not documented, but involves moving a control point in the direction it is being 'pushed' (by the user) or pulled (by a spring) until it settles upon the next image energy minimum (the next salient image feature). If a spring is used, the control point is then locked onto that salient image feature. Neither the internal nor image energies can move it once it has been permanently positioned in the image; instead only further user interaction can move it. The only exception to this is if the snake converges and enters the next frame of a movie, when all controls points are unlocked to allow energy minimisation in the new image.

The external energy function could theoretically be substituted by a higher level cognitive process rather than user interaction [1]; snakes were originally implemented as an interactive tool however, and so avoided the need to perform cognitive skills.

3.4 Strengths and limitations of the original active contour model

The original active contour model has several advantages and several shortcomings when applied to unaided vision systems.

The model allows snakes to be placed near the target object, and to then close in on that object, allowing its detection and tracking. Several snakes may be placed in the same image, allowing several objects to be tracked simultaneously.

A modular energy function is used. This allows developers of specific vision systems to create new energy functions and 'plug them in' [8] instead of, or as well as, the internal, image, and external energy functions presented. In this manner it would be possible to create snakes which latch onto red objects, for instance, by subtracting the red component of the pixel at location (x,y) (where x and y are the control point's coordinates) from the control point's total energy. In this way, the control point would be attracted to areas of high red colouring. With this particular aim, it is relatively simple to construct an energy function which generates such a behaviour. For more complex behaviours however, the developer may not be able to construct an appropriate energy function.

The model as it stands requires user guidance in both initialising and correcting the snake [2]; the user must correct the snake by pushing and/or pulling it around the image. Whilst this was the intention of the model, it is a problem which must be addressed if the model is to be used in unaided vision systems.

It is unclear how best to initialise a snake [8]. It is known that if a snake is not initialised close enough to the target object for its energy function to guide it there, then it will not necessarily end up completely on the target object [1]. It is difficult for the user to know how to improve the snake's chances, or how to work out where a certain energy function will move the snake.

Kass, Witkin and Terzopoulos, the pioneers of the active contour model, do not document how to set α, β or γ [2]. Furthermore these values are identical for every control point on the snake: every control point has the same elastic to bending to image energy ratio [2]. Consequently snakes will not easily lock onto target objects which contain both smooth curves and jagged corners because the bending energy would either allow every control point or none of the control points on the snake to form a corner.

The snake has no knowledge of the target object's motion, either where it has been in the past (further than the previous frame) or its pattern of motion. Subsequently, the model is without predictive powers for the object. If snakes were to have such predictive powers, not only could they converge in fewer iterations in successive frames, but they may also be able to estimate the object's whereabouts when it is occluded. The active contour model cannot track an object once it has become occluded, as it relies upon the object's edges being visible in the image in order to lock onto them.

The original model is far too slow to be implemented in any real-time vision domain [2]. The algorithm itself is not mathematically intensive, but snakes take many iterations before they converge on an energy minimum.

The method for measuring elasticity is limited. This algorithm attempts to minimise the distances as well as the imbalance in distances between the various successive control points [2]. The resulting behaviour is that the snake can be seen to always shrink (or expand if α is negative). While this can be desirable when attempting to find the target object's outline, it is counter-productive as even when the edge has been locked onto by all of the control points, the elasticity algorithm is still trying to pull the snake off the object to contract (or expand) it even more.

Finally, both of the internal energy functions (elastic and bending) aim to move a control point by an amount proportional to the control point's respective energy value. If the control point has a large elastic energy value, the control point can be seen to jump across the image to its new location. The snake's image energy focuses only on the pixel underneath the control point, rather than the immediate patch of image surrounding the control point. Hence a strong edge in the image may not be detected due to the control point jumping over it (Figure 9).

4 An improved active contour algorithm - Fast Snakes

Williams & Shah [2] proposed an algorithm for implementing active contours which claims to be both faster and more accurate than the original Kass implementation. It shares much of the same approach as its predecessor in that it separates out internal and image energies, although the model developed incorporates no external energy (user interaction) term. Within its energy definition it too measures the continuity (elastic energy) and curvature (bending energy) of the snake's spline (in terms of the control point coordinates), and attempts to find the best edge in the patch of image immediately surrounding each control point. Whilst the model does not implement external energy (user constraints), this could be 'plugged in' to the modular energy function if desired, in much the same way as it is in the original algorithm.

The fast snake algorithm takes a different approach to computing where to move a given control point, and uses different component algorithms. Rather than mathematically defining how bunched up the control points are and then moving them proportionally to that amount, the fast snake algorithm takes the surrounding neighbourhood of a control point and determines whether the control point would have a lower overall energy value if it were located elsewhere within its neighbourhood. By considering every location in the immediate neighbourhood, rather than jumping the control point from one location in the image to another, the fast snake model overcomes the limitation of the original model in so much as it cannot overlook strong edges in the locality (Figure 9).

The fast snake's energy algorithm is termed 'greedy' because it calculates the energy values for each location in the control point's neighbourhood, whereas the original active contour model only calculated values for the current (x,y) location of the control point. Whilst

this may sound a longer process (and indeed iterations take more CPU time in fast snakes), it leads to faster convergence (fewer iterations) of the snake onto a local minima.

User-defined parameters still exist to allow the user to alter the precedence of the continuity (α), curvature (β) and image (γ) energies, allowing any one of a wide range of behaviours for the snake.

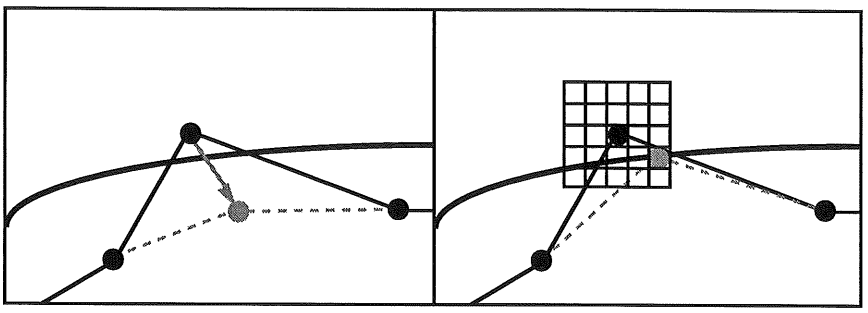

Figure 9: The original active contour model can sometimes fail to detect edges as a result of its internal energy functions moving a control point too far too soon (left). With the fast snake model however (right), the control point's entire neighbourhood is searched for a location which minimises its overall energy. During this process, the image energy is able to locate all the edges in the locality. The control point can only be moved to another location within that neighbourhood in one move, so there is a maximum distance that the control point can travel per iteration (the radius of the neighbourhood); the distance is not proportional to the magnitude of the control point's energy value. As such, a control point cannot fail to detect edges in its locality.

4.1 Energy algorithm used in fast snakes

The technique for moving a control point in a fast snake involves imagining the control point has been moved to each location in turn in its neighbourhood, and to then calculate its total energy value. Collecting the total energy values for each of the neighbourhood locations allows the snake to choose which location offers the lowest energy, and to move the control point there (thus minimising the control point's energy).

Unlike their predecessor, fast snakes update their control point locations asynchronously (one by one during the iteration); one control point's movement may affect other control points' energy values (and therefore movements) during that same iteration.

In fast snakes, a control point's overall energy can be defined as [2]:

$$\text{ControlPointEnergy}_i = (\text{IntEnergy}_i + \text{ImageEnergy}_i) \qquad \text{[Equation 10]}$$

where $\text{ControlPointEnergy}_i$ represents the overall energy for control point i, IntEnergy_i represents that control point's internal energy (as defined in Equation 11), and ImageEnergy_i represents that control point's image energy (as defined in Equation 14). Notice that, unlike the original implementation, there is no external energy provided by user interaction; the fast snake model aims to perform automatic object detection, with the user only providing the initial snake position and the energy parameters α, β and γ.

4.2 Internal energy functions

The fast snake algorithm employs two internal energy terms which, similar to the original implementation, measure the continuity (elasticity) and curvature (bending) properties of the snake. However the two functions used in fast snakes claim to be more accurate and lead to faster convergence of the snake [2].

The internal energy value for a control point can be defined as [2]:

$$IntEnergy_i = ContEnergy_i + CurvEnergy_i \quad \text{[Equation 11]}$$

$$ContEnergy_i = \alpha(\bar{d} - |v_i - v_{i-1}|) \quad \text{[Equation 12]}$$

$$CurvEnergy_i = \beta(|v_{i-1} - 2v_i + v_{i+1}|^2) \quad \text{[Equation 13]}$$

where in Equation 11, $IntEnergy_i$ represents the internal energy for control point i, $ContEnergy_i$ represents that control point's continuity energy value, and $CurvEnergy_i$ represents that control point's curvature energy.

In Equation 12, $ContEnergy_i$ represents the continuity energy value for control point i, α represents the user defined continuity weighting parameter, $\bar{d}$ represents the average distance between all adjacent control points along the snake (which of course changes during the iteration as control points are updated asynchronously), v_i represents the control point's (x,y) coordinates, and v_{i-1} represents the previous control point's (x,y) coordinates (which have just been updated).

In Equation 13, $CurvEnergy_i$ represents the curvature energy value for control point i, β represents the user defined curvature weighting parameter, v_{i-1} represents the (x,y) coordinates of the previous control point along the snake, v_i represents the (x,y) coordinates of control point i, and v_{i+1} represents the (x,y) coordinates of the next control point along the snake.

Spline continuity term ($ContEnergy_i$)

The continuity term in fast snakes replaces what was the elasticity term in the original implementation. Its precedence in a control point's total energy value is still controlled by the user-defined constant α; setting α to 0.0 still has the effect of removing the continuity term's influence over the snake's future movement. It too aims to make the control points equidistant along the snake. By keeping the control points along a snake equidistant, the snake is better controlled as the points do not become bunched up. Unlike the original model, the Fast Snake's continuity energy spaces the control points evenly without explicitly contracting or expanding the snake. This lack of contraction or expansion prevents the snake from being able to 'shrink wrap' onto or expand out to fit the target object's outline, which may at first appear to be a disadvantage of using the Fast Snake algorithm. It can be seen however that the original contraction or expansion in many ways offered a false sense of security. It is documented in the original snake method [1] that the snake should be initialised close enough to the target object to allow it to lock onto the correct outline, rather than locking onto misleading strong edges which lie between the snake's initial position and the target object. As such, 'shrink wrapping' would not overcome this flaw and, in fact, would only be advantageous when there is a trouble-free path (i.e. no other edges) between the snake's initial position and the target object's outline. Furthermore 'shrink-wrapping' can continue to pull the snake in even when the snake is already situated on the target contour.

The continuity energy for a control point on a fast snake is defined in Equation 12, above. Once the continuity energy has been calculated for each location in the neighbourhood, each value is normalised. The rationale for this procedure is discussed in section 4.4.

By comparing Equation 12 to Equation 4, one major difference can be seen. In the original implementation the control point's elastic energy value is determined by the distance between the control point and its neighbours. In this way, the further away the control point is from its neighbours, the larger its elastic energy will be (or the smaller it will be if α is set to being negative). Consequently the distance itself is being minimised, and the snake can be seen to shrink. In Fast Snakes, the actual distance between the control point and its adjoining neighbours does not affect the size of the elastic energy value. Instead the elastic energy value is determined by comparing a control point's distance from its previous neighbour to the average distance every other control point is away from its respective adjoining neighbour. This

results in a minimisation of the imbalance in adjacent distances (making the control points equidistant), rather than minimising the distances themselves. With fast snakes it does not matter how far apart a control point is from its neighbours, so long as that distance is the average for all other adjacent pairs of control points along the snake too. The fast snake's continuity energy can therefore both contract or expand the snake, depending upon which direction will make the control points equidistant.

Spline curvature term ($CurvEnergy_i$)

The second of the fast snake's two internal energy components is that of curvature. The control points' curvature term is defined in Equation 13, above. As with the original implementation, the curvature energy's influence over snake movement is governed by the user defined parameter β. Setting β to 0.0 removes the curvature term's influence over the snake's future movement. Like the continuity energy, once the curvature terms for a neighbourhood have been calculated, they are normalised (section 4.4).

The model also allows corners to form at certain control points on the snake. The curvature is measured at each point on the new snake (following an iteration) and, if it exceeds a curvature threshold value and is on a strong edge, β_i is set to 0.0, thus disabling that control point's curvature energy from taking part in future movements. If, following future iterations, the curvature at point i is no longer above the threshold (because the adjacent control points have been moved to other locations), or if point i is no longer on a strong edge (because the image has changed, for example the snake has been moved onto the next frame of a movie) then the value for β_i is reset to whatever it was before it was disabled. The control point's curvature energy is then re-enabled for future iterations of the snake. By automatically enabling (or disabling) a control point's curvature, the control point is able to lock onto corners in the target object's outline. In the original model, the user-defined parameters are identical for all control points on the snake. Consequently the original snakes have a tendency to cut the corners off the target contour, as they are aiming to achieve global smoothness.

4.3 Image energy functions

The image energy function in fast snakes consists of just one component: the gradient magnitude value for each location in the control point's neighbourhood.

Image gradient magnitude term ($ImageEnergy_i$)

To calculate the gradient magnitude value for a location in the control point's neighbourhood, two image pre-processing techniques are used. Firstly the patch of image which constitutes the control point's neighbourhood is passed through a Sobel edge detecting mask [9]. This enables all edges in the area to be identified, and changes each pixel's greyscale value. It changes pixels on strong edges into light grey or white (white representing the strongest edge possible in the image's resolution), it changes pixels in areas of flat colour (i.e. where there are no edges) dark grey or black (black representing uniform colour), and it changes pixels housing intermediate edges into a shade of grey. The results of applying a 3x3 Sobel mask over an image can be seen in Figure 10.

Once the Sobel mask has been applied, the resulting pixel values for that neighbourhood are then normalised using the following formula [2]:

$$ImageEnergy_i = \gamma \frac{min - S_{(x,y)}}{max - min} \qquad \text{[Equation 14]}$$

where $ImageEnergy_i$ represents the image energy for location (x,y) in image S, γ represents the user-defined image weighting term, S represents the original image after being passed through a Sobel mask, $S_{(x,y)}$ represents the pixel value at location (x,y) in the masked image, min represents the minimum pixel value in the control point's neighbourhood within the

masked image S (i.e. the weakest (darkest) edge in the neighbourhood), and max represents the maximum pixel value within the control point's neighbourhood in the masked image S (i.e. the strongest (brightest) edge in the neighbourhood). $ImageEnergy_i$ is smallest when $S_{(x,y)}$ is one of the larger neighbourhood values. As the image energy $ImageEnergy_i$ is being minimised, the control point will be attracted towards pixels with large values for $S_{(x,y)}$; that is, towards the white edges in the masked image. The effects of the normalisation are discussed in the next section.

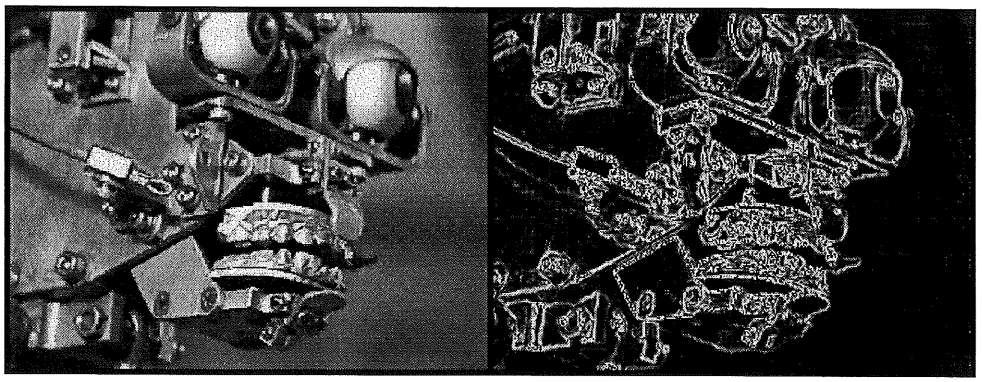

Figure 10: The effects of a Sobel mask (right) on an image (left). Edges are depicted in lighter greys; areas of flat colour are depicted in darker greys. A 3x3 mask was used in to generate the image on the right. Alternative mask sizes can be used to abstract different strength edges from the image.

4.4 Normalisation of the continuity, curvature and image energy values

Once the continuity values have been calculated for every location in a control point's neighbourhood, each value is divided by the largest within that neighbourhood, normalising them. The same is done to the curvature values for that neighbourhood.

The image energy values for a neighbourhood are normalised using a different mechanism (Equation 14). For the image energy values, the gradient magnitude values of each location in the neighbourhood are normalised, making the biggest edge in that neighbourhood more pronounced than the others. The values are normalised by the neighbourhood's maximum gradient minus its minimum gradient. This procedure is used in place of simply normalising all values by 255 (the maximum gradient possible) as this would not emphasise the difference between points with similar gradient magnitudes, for example 200 and 225 [2]. The internal energy could, in such a situation, have a greater influence over which locations are 'better' and which are 'worse'. Normalising the gradients by (max-min) results in any slight variations in gradient being magnified, allowing the image energy to keep its influence even when the strongest edge in a neighbourhood is only marginally stronger than its competitors.

Since all three energy values ($ContEnergy_i$, $CurvEnergy_i$, $ImageEnergy_i$) are normalised they can each be added together for each location in the neighbourhood, giving a total energy value for a location in the neighbourhood. If the energies weren't normalised, then a large continuity value could sabotage the effect of the other two energy values, largely irrespective of the values of the energy weighting parameters α, β, and γ.

4.5 Strengths & limitations of the Fast Snake model

The fast snake implementation could overcome many of the original model's problems were it to be used in an unaided vision system. Fast snakes relax onto a minimum in fewer iterations than the original snakes, making them faster than their predecessor and consequently

of more use in a real-time vision environment [2].

Control points are moved within a neighbourhood, having first considered every other location in that neighbourhood in terms of the different types of energy (continuity, curvature and gradient magnitude). This guarded movement is an improvement upon the original model, in which control points were able to jump across the image, missing important edges situated between the previous location and the current location. In this sense, the model is more admissive; fast snakes will find nearby edges, but whether or not they lock onto them depends upon the other energy values for those locations, and subsequently on the precedence of α, β and γ.

The user-defined α, β and γ parameters in fast snakes only affect the precedence of the three energies, and consequently the criteria of what defines a 'better' location for the control point to be in. The parameters do not affect how far the control point will be moved; in the original implementation, setting α=1, β=2 and γ=4 would move a given control point to a different location than if the parameters had been set α=2, β=4 and γ=8. In this example, the control point would be moved further (and not necessarily in the same direction, due to the image energy) with the second set of parameter values. In fast snakes, the two sets of parameters would generate the same movement of a given control point, as the ratio of α:β:γ is the same in both cases; it is only the ratio and not the magnitude of these parameters which matters, as each energy value is normalised [2]. This in turn means that the user does not have to be so accurate in setting the parameter values for fast snakes as they do with the original snakes; with fast snakes they only have to worry about the ratio of α:β:γ. Williams and Shah (the developers of fast snakes) provide guidelines for setting the α, β and γ parameters, whereas in the original model this is left as an exercise for the user. As such it was difficult for a developer or end user to know whether they were using the most appropriate settings for the snake behaviour they required.

The fast snake model allows for 'personalised' energy functions that correspond to particular control points on the snake, rather than being identical for all control points on the snake (as is the case with the original model). This allows corners to form at certain points in the snake, and not at others [2]. With the original model, the parameters are identical for every control point on the snake. Even if the original snake does attach to the target object, it will attempt to 'cut the corners' of the target contour, as its bending energy tries to smooth out the snake's bumps. In this way, fast snakes provide a much closer mapping to the actual 2D shape of the target object.

Fast snakes require no user guidance once the snake has been given an initial starting position and values for the parameters α, β and γ [2]. They are more suitable for use in an unaided vision system than the original implementation, which required the user to push and pull the snakes around the image.

The fast snake energy function remains modular, so if particular characteristics are needed by the snakes, such as latching onto red edges, then a module can be constructed which measures this energy, and the module can then be 'plugged in' to replace or augment the other components in the energy function. As with the original model, the success of the snake relies upon the developer's ability to extract the particular type of energy they wish to measure.

The continuity energy function does not allow the fast snake to shrink-wrap or expand onto the target object, whereas the previous model could, meaning that it does not attempt to contract/expand the snake further once it has locked onto an outline. The method it uses to keep control points equidistant is more mathematically robust, as it is checking each control point's distance to its neighbour against the average distance between every other pair of adjacent control points.

Finally, like its predecessor, fast snakes cannot predict where the target object will be or what position it will be in, as the snake itself does not contain any information pertaining to the target object's patterns of motion or articulation. All information is gathered either from the current image, or from the snake's current position. Similarly fast snakes cannot track an object during occlusion, as they rely upon the visibility of the target object's edges.

5 Discussion & Developments

Both of the models of snakes detailed in this paper have much to offer a real-world vision system. Both can locate and track nearby objects. Changing the components or parameters of the snake's energy function will target the snake to specific types of object. The spline properties of the original Active Contour model, and the neighbourhood-searching properties of the Fast Snake, mean that either model can 'close in' on the target object, allowing some flexibility in the accuracy of the snake's initial coordinates. Neither of the techniques involve complex model scaling or rotation operations, so both are computationally fast.

One problem facing all active contour systems based on the models discussed is that of snake initialisation. No guidelines are provided concerning where the snakes should be initialised, yet a snake's initial positioning can be critical to its success in locking onto the target object's contour. To make matters even more complicated, the ideal starting location of a snake is dependent upon the snake's energy function. For example, in the original active contour model, the variable α determines whether a given snake will attempt to shrink or expand itself onto the target object's contour. As such, if the snake is initialised inside the target object's outline but α is set to shrink the snake, then the snake is unlikely to converge on the target object's outline.

One solution to the problem of snake initialisation which could be implemented in systems involving motion (i.e. those tracking moving objects rather than static objects in static images) but which use fixed position cameras would be to find the difference between the frames of the movie prior to any other image pre-processing or snake processing [8]. This would render the first frame of the movie unusable, but all subsequent frames would provide a record of movement (movement being defined as pixels whose values have changed significantly between frames) in that movie. After differencing has been performed, each frame of the movie would contain black and white pixels only (in the case of 1-bit differencing, where a pixel's colour has either changed significantly or not) or shades of grey (in the case of >1-bit differencing, where the shade of grey represents the amount that the pixel's colour has changed). Snakes' control points could then be initially positioned around the perimeters of areas of movement in the image, giving the snake a good starting location for locking onto the moving object's outline (Figure 11). In systems where total accuracy is needed and where user interaction is not a problem, frame differencing could be used to place the snake initially, prior to minor alterations made by the user repositioning specific control points.

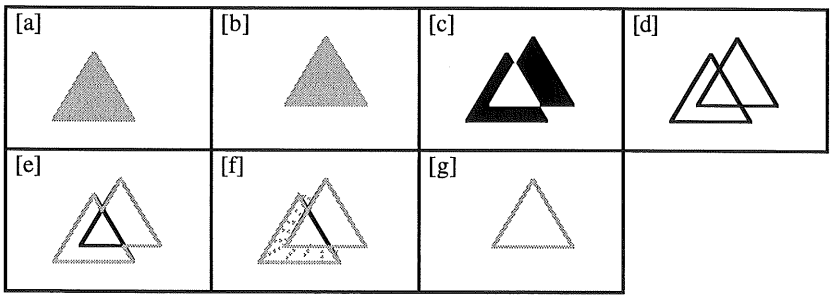

Figure 11: Using pixel differencing to aid snake initialisation. [a] and [b]: A triangular target object moves across the visual field. [c]: The original images ([a] and [b]) have been differenced using 1-bit differencing, producing an image containing black pixels (denoting colour change between images [a] and [b]) and white pixels (denoting no colour change between the two images). [d]: The result of applying a Sobel mask to image [c]; black pixels denote strong edges. [e]: A snake (grey) has been automatically initialised on the outer edges of the intersecting shapes in [d]. [f]: The snake's spline (original active contour model) or neighbourhood-searching (Fast Snake model) properties, coupled with the lack of misleading edges, enable the snake to locate the second triangle (from image [b]). [g]: The snake has successfully detected the target object in image [b]. Note that for each sequence of frames, the first frame must be sacrificed in order to perform the differencing.

As they stand, neither the original nor Fast Snake models could be used in a real-world

vision environment without the addition of other techniques working alongside them. In a real-world visual field, the presence of multiple objects complicates the tracking situation, as does the ability of deformable objects to change shape and position in a sequence of images. The situation is further complicated when objects within an image become occluded.

Problems arising from the deformability (articulation) and mobility of the target object can be minimised to some degree by using video footage with a high frame rate. Recording video at 30fps (frames per second) results in less change between adjacent frames than video recorded at 15fps. Changes between frames in a 15fps video could be (in the worst instance) twice as extreme as in a 30fps video of the same scene. Smaller differences between frames means movement of the target object appears less jerky, and so its location and posture become easier to track. If the snakes are to track rapidly changing objects (irrespective of whether that change is in posture, speed of movement or both) and accuracy is an issue, it makes sense to stream the video at a high frame rate.

Whilst high frame rates make the task of tracking somewhat easier for the snake, the associated disadvantage is that it requires twice as many (in the 30fps versus 15fps example) images to be processed per second. This can have a crippling effect on the speed of processing of the movie, especially if real-time performance is required. Decisions made regarding this speed/accuracy trade-off will ultimately depend upon the needs of the particular system being built.

Increasing the frame rate does not in itself guarantee successful tracking. If the movie is noisy (due to low resolution recording, for example), the noise may be classed a salient contour in the image, and the snake may inevitably be attracted towards it. In cases where noisy data is unavoidable (if visibility is not clear, for instance), amplification of the information that is present by preprocessing with image filters [4, 9] is recommended.

Other improvements can be derived from changes made to the snake's energy function. If the snake cannot detect the target contour then it is possible that the snake's energy function is pulling it away from the target object. The snake's energy function should be configured such that it assists the attraction of the snake towards contours that resemble those of the target object. For instance in colour vision, if the target object is always red in colour, then the snake's energy function should attract the snake towards red edges. It is usually more difficult than this to make such clear cut rules about the target object however. This difficulty is magnified if the target object is deformable; what might be a successful energy function at one stage of the object's deformation might be totally unsuccessful at another stage in that same object's deformation.

Techniques such as Smart Snakes [10] and the Point Distribution Model [11] have been developed for abstracting information about a particular object's pattern of deformation. Smart snakes take into account a target class' pattern of deformation, and from that information are able to prune the search space involved in relaxing the snake, thus rendering the snake more efficient. The Point Distribution Model achieves a similar goal by deriving a target class' principal components, thus obtaining it's directions of deformation. Both techniques provide a means of abstracting a priori information about the target class' deformation characteristics, which can then be used to forge appropriate energy function components for the snake. These components can be 'plugged in' alongside any other components to enable accurate tracking of the target object during deformation.

If such rules could be constructed for the energy function, two pieces of information would be predicted: the target object's next shape (its posture) following deformation; and its next location in the image, based upon its trajectory.

This type of a priori solution assumes that data can be obtained regarding the target object's (or furthermore it's class') pattern of motion, and subsequently that suitable energy function components can be constructed. Both Smart Snakes and the Point Distribution Model require a wide range of specimens from the target class to be available. In most cases where deformable objects are tracked it is not possible to obtain a set of data that covers an entire class of objects; instead only particular specimens are available. In such cases there is no guarantee that any energy functions obtained would successfully guide a snake onto a previously unseen member of that object class. The samples used in constructing the energy function may have been too narrow a slice of the class population, resulting in rules that are too specific to cover

the entire class of object. What is needed to overcome this problem is a technique which is able to generalise from a set of specific data so that previously unseen examples of the target object can be tracked. Neural networks have the ability to abstract general rules from data sets, and are able to cope with noisy or atypical data [12]. If used as part of an active contour system, neural nets may help bridge the gap between a vision system's ability to track a set of previously experienced specimens with a pre-determined motion pattern, and unseen specimens of the same type.

The problem of handling occlusion is beyond the capability of either of the active contour models discussed in this paper. Both systems rely heavily upon the target object's edges being visible in the image; if the target object is occluded, then its edges are not available to be attached to by the snake.

One potential method of minimising the risk of 'losing sight' of a fully occluded target would be to use model-based techniques to predict where it will reappear in the image. The technique would use knowledge of the target class' recent motion pattern, and the object's actual rate of deformation and velocity in previous video frames, to produce a 'blind' snake. This knowledge could be obtained from a Smart Snake or Point Distribution Model. The blind snake would be introduced to the image as a prediction of the actual object's position. When a sufficiently large percentage of the blind contour corresponded to actual edges in the image, the active contour system could then regain control of it and continue.

6 Conclusion

This paper has detailed two designs of active contour technology. The underlying mechanisms of the original Active Contour Model and the Fast Snake Model have been detailed, along with a discussion of the strengths and limitations of both models for the purpose of tracking moving, deformable objects in a visual field. Issues presented by real-world tracking tasks have been examined, and suggestions for future developments made. Neural Networks are proposed as potential solutions to some of the problems encountered, such as deformable and / or occluded objects, especially when used in collaboration with other techniques for modelling motion, such as the Point Distribution Model or Smart Snakes.

Appendix A

Algorithm for moving control points on an Active Contour (original model)

A control point consists of an x and y coordinate. To move that control point, it is necessary to find out where to move it to, based upon its total energy values. If its original coordinates are defined as (CPX_i,CPY_i), where i represents the control point's index in the list of points, then its position after one iteration will be (CPX_i', CPY_i'). These new coordinates are obtained as per Equation 15:

$$(CPX_i', CPY_i') = ((CPX_i + EnergyX_i),(CPY_i + EnergyY_i)) \quad \text{[Equation 15]}$$

where $EnergyX_i$ represents the total energy along the X axis for control point i, and $EnergyY_i$ represents the total energy along the Y axis for the same control point. Finding out the values of $EnergyX_i$ and $EnergyY_i$ involve totalling up the component energies along those axes (Equation 16). For the sake of brevity, only the equations for determining $EnergyX_i$ are detailed here. The same calculations are performed in order to obtain $EnergyY_i$. Many of the below equations can be found elsewhere in this paper; they have been duplicated here for clarity.

$$EnergyX_i = IntX_i + ImageX_i + ExtX_i \quad \text{[Equation 16]}$$

where $IntX_i$, $ImageX_i$ and $ExtX_i$ represent control point i's internal, image and external energies in the X axis.

$$IntX_i = \frac{ElasticX_i + BendingX_i}{2} \quad \text{[Equation 17]}$$

where $ElasticX_i$ and $BendingX_i$ represent control point i's elastic and bending energies in the X axis.

$$ElasticX_i = \alpha ((x_{i-1} - x_i) + (x_{i+1} - x_i)) \quad \text{[Equation 18]}$$

where α represents the user-defined elastic energy weighting term, x_i represents control point i's coordinate in the X axis, x_{i-1} represents the x coordinate of the control point preceding control point i along the snake, and x_{i+1} represents the x coordinate of the control point following control point i along the snake.

$$BendingX_i = \beta (4(x_{i-1} + x_{i+1}) - 6x_i - (x_{i-2} + x_{i+2})) \quad \text{[Equation 19]}$$

where β represents the user-defined bending energy weighting term, x_i represents control point i's coordinate in the X axis, x_{i-1} represents the x coordinate of the control point preceding control point i along the snake, x_{i-2} represents the x coordinate of the control point preceding that control point, x_{i+1} represents the x coordinate of the control point following control point i along the snake, and x_{i+2} represents the x coordinate of the control point following that control point.

$$ImageX_i = \gamma ((\lambda\ Line_i) + (\mu\ Edge_i) + (\nu\ Term_i)) \quad \text{[Equation 20]}$$

where γ represents the user-defined image energy weighting term, λ represents the user-defined line energy weighting term, $Line_i$ represents control point i's line energy, μ represents the user-defined edge energy weighting term, $Edge_i$ represents control point i's edge

energy, v represents the user-defined termination energy weighting term, and $Term_i$ represents control point i's termination energy.

$$Line_i = I_{(x,y)} \qquad \text{[Equation 21]}$$

where x and y represent control point i's X and Y coordinates, and $I_{(x,y)}$ represents the greyscale pixel value at location (x,y) in the image, i.e. at the pixel the control point is located.

$$Edge_i = -\sqrt{(I_{(x,y)} - I_{(x+1,y)})^2 + (I_{(x,y)} - I_{(x,y+1)})^2} \qquad \text{[Equation 22]}$$

where $I_{(x,y)}$ represents the greyscale pixel value at location (x,y) in the image, i.e. at the pixel the control point is located, and $I_{(x+1,y)}$ and $I_{(x,y+1)}$ represent the greyscale values of the pixels to the right and below that pixel, respectively.

$$Term_i = \frac{\delta\theta}{\delta n_\perp} \qquad \text{[Equation 23]}$$

$$= \frac{\delta^2 C / \delta n_\perp^2}{\delta C / \delta n} \qquad \text{[Equation 24]}$$

$$= \frac{C_{yy}C^2_x - 2C_{xy}C_xC_y + C_{xx}C^2_y}{(C^2_xC^2_y)^{3/2}} \qquad \text{[Equation 25]}$$

where C is the original image following a standard Gaussian smoothing of the greyscale pixel values, θ is the gradient angle (equal to $\tan^{-1}(C_y/C_x)$), n is the unit vector running along the gradient direction, equivalent to ($\cos\theta$, $\sin\theta$), and $n_\perp$ represents the unit vector running perpendicular to the gradient direction, equivalent to ($-\sin\theta$, $\cos\theta$).

The original algorithm for implementing user interaction with snakes ($ExtX_i$) is not documented.

References:

[1] Kass M., Witkin A. & Terzopoulos D. (1988) *Snakes: active contour models.* In International Journal of Computer Vision (1988), pp.321-331.

[2] Williams D.J. & Shah M. (1992) *A fast algorithm for active contours and curvature estimation.* In CVGIP - Image Understanding 55, pp.14-26.

[3] Marr D. (1982) Vision. W.H. Freeman & Company.

[4] Sonka M., Hlavac V. & Boyle R. (1994) Image Processing, Analysis and Machine Vision. Chapman & Hall.

[5] Fok Y.L., Chan J.C.K. & Chin R.T. (1996) *Automated Analysis of Nerve-Cell Images using Active Contour Models.* In IEEE Transactions On Medical Imaging, 1996, Vol.15, No.3, pp.353-368.

[6] Caselles V. & Coll B. (1996) *Snakes in Movement.* In Siam Journal On Numerical Analysis, 1996, Vol.33, No.6, pp.2445-2456.

[7] Leymarie F. & Levine M.D. (1990) *New Method for Shape Description based on an Active Contour Model.* In Intelligent Robots and Computer Vision VIII : Algorithms and Techniques, Pts 1 & 2, 1990, Ch. 82, Pp.536-547.

[8] Young D. (1995) Sussex Computer Vision: Teach Vision 7. World Wide Web URL: http://www.cogs.susx.ac.uk/users/davidy/teachvision/vision7.html. URL valid on 6/3/97.

[9] Dean T., Allen J. & Aloimonos Y. (1995) Artificial Intelligence: Theory and Practice. Benjamin/Cummings Publishing Company Inc.

[10] Cootes T.F. & Taylor C.J. (1992) *Active shape models - 'smart snakes'.* In British Machine Vision Conference Sept 1992, pp.276-285.

[11] Baumberg A.M. & Hogg D.C. (1993) *Learning Flexible Models from Image Sequences*. University of Leeds School of Computer Studies Research Report Series, Report 93.36.

[12] Anderson J.A. (1995) An Introduction to Neural Networks. MIT Press, Cambridge.