2006 10th International Workshop on Cellular Neural Networks and Their Applications, Istanbul, Turkey, 28-30 August 2006

CNNA 2006
Istanbul

A Cellular Active Contours Algorithm Based on Region Evolution

Piotr Dudek[*], David Lopez Vilariño[+]
[*]School of Electrical and Electronic Engineering, University of Manchester, United Kingdom
e-mail: p.dudek@ manchester.ac.uk
[+]Department of Electronics and Computer Science, University of Santiago de Compostela, Spain
e-mail: dlv@dec.usc.es

***Abstract*—This paper introduces a new algorithm for implementing cellular active contour technique based on pixel-level snakes (PLS). The main motivation is the optimization of the computational performance, especially when PLS are implemented on pixel-parallel single instruction multiple data (SIMD) processor arrays. The algorithm is based on the evolution of an active region. This allows the implementation of the entire algorithm using very simple local rules. Additionally, nested contours and propagation into narrow cavities are supported. The algorithm and experimental results from real-time implementation on vision chips are presented.**

I. Introduction

Active contours [1] are elastic curves which deform controlled by image features and shape constraints to adapt themselves to the boundaries of the regions of interest. The deformable contour techniques are usually classified as either energy-based [2] or level-set based [3]. Both types differ in the contour representation, implementation and capabilities from the image processing point of view. However, they share high computational requirements which make them unsuitable for applications requiring short response time.

Cellular active contours (CAC) were introduced to reduce the computational effort inherent in the conventional active contour techniques. They are based on a pixel-level discretization of the contours and a massively parallel computation. Processing every contour cell concurrently leads to a higher processing speed without penalizing the efficiency of the contour evolution. Currently there are two different CAC approaches. First, a PDE-based approach proposed in [4] which implements active contours via a non-iterative region propagation technique, where the contours are defined as the fronts of the propagating trigger-waves. Like the classic implicit active contour techniques, this approach gives a very simple solution to the possible topologic transformations, but, in contract to those techniques, the speed of computation is very high. The main weakness of this approach comes from the difficulty in the control of the contour evolution. It does not allow the simultaneous expansion and contraction of different parts of the active regions. Furthermore, usually it requires sophisticated stop criteria to control the active wave propagation which can lead to a significant increase of the computational complexity in real applications.

On the other hand, the so called pixel level snakes (PLS) [5] are an iterative CAC technique where the contours are explicitly represented and evolve guided by local information and regularizing terms dependent on the contour curvature. The iterative computation of PLS allows an efficient control of the contour evolution. However, as a consequence of the explicit representation of the active contours, the algorithm complexity of PLS is clearly higher than the active wave based approaches [6].

In this paper, a CAC technique situated mid-way between the above described techniques is proposed. In a way similar to the active wave computing technique, the active contours are represented as the boundaries of active regions, thus reducing complexity of the contour evolution. On the other hand, keeping the iterative computation of PLS a versatile and simple control of the contour deformation is achieved

II. Pixel Level Snakes

The proposed algorithm builds upon the overall method of PLS introduced in [5]. In a PLS algorithm the contour is represented explicitly by pixels in an array corresponding topographically to the input image and the contour evolution is implemented by one-pixel shifts of the contour in this array. The overall flow of a PLS algorithm is presented in Figure 1. It is an iterative algorithm, in which the contour pixels evolve according to external and internal forces. The potential field is a topographic map, calculated as a weighted sum of External Potential, Internal Potential and Balloon Potential. This potential field produces forces that guide the contour evolution towards the minimum energy level.

The External Potential is calculated from the input image, and determines the overall target for the contour evolution, to fulfill the required segmentation task determined by the particular application. For example, the contour may track the boundaries of all objects, the edges of selected objects in the image (for example moving objects, or one marked object only), etc. Standard low-level image processing is performed on the input image to determine the External Potential (the operations may include noise filtering, edge-detectors, distance transforms, etc.). In some applications, involving static images, External Potential is calculated once. In other applications, involving moving images (e.g. real-time computer vision) the External Potential is calculated for each image frame. The External Potential can be updated on each iteration of the PLS evolution, or several iterations of PLS evolution can be performed for each External Potential.

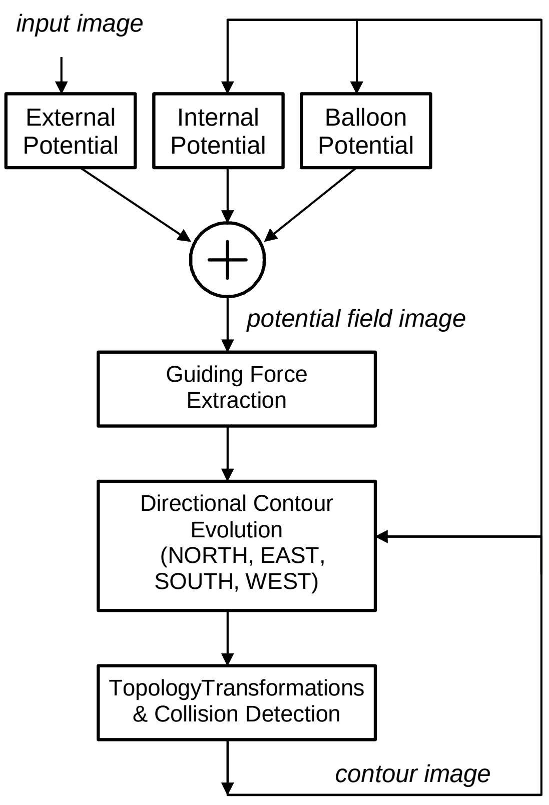

Fig.1. Overview of a pixel-level snakes (PLS) algorithm

The Internal Potential is a topographic map obtained by diffusing the contour image. The resulting potential field helps to keep a smooth shape of the contour.

The Balloon Potential produces additional inflating/deflating forces that are sometimes needed to guide the contour evolution towards the target, especially if the contours are initialised far from the desired minima of the External Potential field and could settle at some local minima locations, or if the gradients of the External Potential field are zero in some image areas.

The relative weights of External, Internal and Balloon Potential are adjusted, depending on the application. (In some situations, some of the weights can be zero, which obviously means that the corresponding potential does not have to be calculated, thus simplifying the overall algorithm)

Contour evolution is the main component of a PLS algorithm. It operates along the four cardinal directions in sequence. On each iteration, contour evolution is performed as a pixel-by-pixel shift of the contour image, towards a lower potential location. (Internal Potential and Balloon Potential can be updated on each iteration, or once for every four cardinal directions, to reduce computations). The requirement of maintaining one-pixel wide contour is the main difficulty when implementing a PLS algorithm, and the methods proposed earlier were based on a number of local rules to handle Directional Contour Expansion and Directional Contour Thinning. Furthermore, merging and splitting of contours had to be handled by a separate Topologic Transformations module, which could contain Collision Point Detection module (used in applications that require avoiding collisions between contours) and algorithms to handle splitting and merging of contours.

III. Proposed Algorithm

The algorithm proposed in this paper follows the general flow of the PLS algorithm, but is based on the evolution of an active region. The contour is defined implicitly as the boundary of the region. This greatly simplifies topographic transformations (merging and splitting of contours), as well as implementation of inflating/deflating forces. The main advantage of this approach is the elimination of the hole-filling step in the original PLS algorithm [5], which is time-consuming when performed iteratively on a processor array. It should be noted, that in theory a CNN-type array processor could handle hole-filling in a single operation, using the 'hole_fill' template. However, the difficulty in implementing an 'ideal' CNN processor in silicon [7,8] leads to the situation that hole-filling still has to be implemented iteratively, even on the most recent CNN chips [6]. Simple, robust solutions for hole-filling exist, based on asynchronous cellular logic [9]. Nevertheless, an approach which eliminates hole-filling leads to an algorithm suitable for many present-day processor arrays, such as [10-13], as well as implementation of pixel-level snakes in software, on conventional microprocessors.

Furthermore, the proposed algorithm implements contour evolution using very simple local rules, which results in a fast implementation. In contrast to previously reported implementations, the proposed algorithm also enables nested contours (closed contours within other contours), and enables contour propagation inside 1-pixel wide cavities.

A. Contour Definition

Instead of evolving 1-pixel wide contours, the method presented in this paper relies on the evolution of active regions. The overall topographic map is initially segmented into two sets of pixels, one set are pixels belonging to the 'active regions', the remaining pixels are the 'background regions'. We can represent the regions using a binary array **a**, where the active regions are '1', and the background regions are '0'.

The contour could be defined implicitly, as the boundary between two regions, as illustrated in Figure 2a. To maintain the one-pixel wide contour (useful for display purposes, and producing output compatible with previous PLS algorithms, and also necessary to calculate Internal Potential) we assume that the 1-pixel wide contour is defined as a boundary of the 'active region', and lies within the active region, as illustrated in Figure 2b. In general, there may be multiple disjoint active regions in an image, resulting in multiple closed contours, as shown in Figure 3a. This definition supports also a situation, where a contour in placed inside another contour, as shown in Figure 3b.

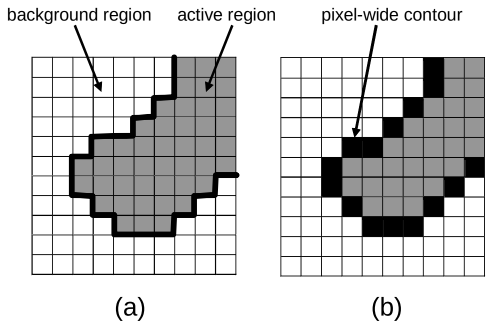

Fig.2. Definitions of a contour: (a) as a boundary between regions, (b) as a 1-pixel wide edge inside the active region

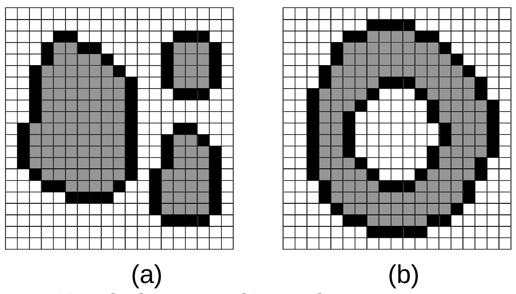

Fig.3. (a) Multiple contours (b) Nested contours

The contour evolution is implemented through conditional expansion of active regions in four cardinal directions (North, South, East, West). This is followed by an inversion of the active regions (i.e. the active regions become background and the background becomes active regions) and another conditional expansion of the new active regions. In this way, the contours can move in every direction, and follow the guiding forces to settle into the minimum potential location. The overall flow of the algorithm is shown in Figure 4.

B. Potential Field

The calculation of the External Potential **ep** is performed according to the application requirements. Since it is application-dependent, the detailed discussion of various techniques is beyond the scope of this paper.

The Internal Potential is calculated from the contour image. The binary contour image **c** can be obtained from the active region image **a** calculating for each pixel (x,y)

$$c_{x,y} = a_{x,y} \text{ and not } (a_{x,y-1} \text{ and } a_{x,y+1} \text{ and } a_{x-1,y} \text{ and } a_{x+1,y}) \quad (1)$$

where $c_{x,y}$ represents the pixels in the contour image, $a_{x,y}$ represents pixels in the active region image, and $a_{x,y-1}$, $a_{x,y+1}$, $a_{x-1,y}$, $a_{x+1,y}$ are active region image pixels in the North, South, West and East direction from the pixel (x,y), respectively.

We will use indices N,S,E,W to represent arrays shifted by one pixel in the respective direction. The operation (1) can be thus represented in the array notation as

$$\mathbf{c} \leftarrow \mathbf{a} \text{ and not } (\mathbf{a}_N \text{ and } \mathbf{a}_S \text{ and } \mathbf{a}_W \text{ and } \mathbf{a}_E) \quad (2)$$

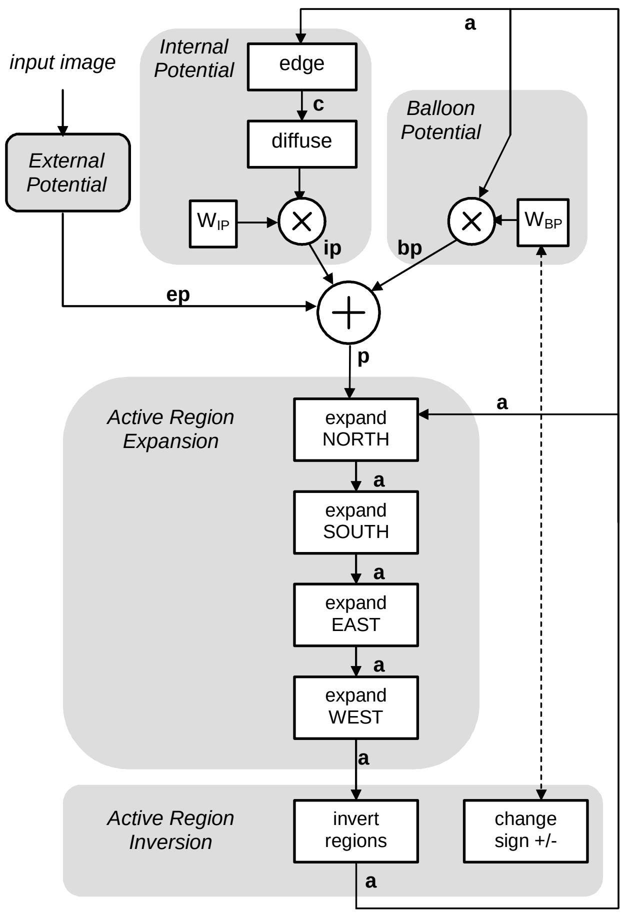

Fig.4. The proposed algorithm

The Internal Potential is calculated by gray-level diffusion of the contour image **c**, multiplied by appropriate weight.

$$\mathbf{ip} \leftarrow \text{diffuse } (\mathbf{c}) \times W_{IP} \quad (3)$$

Implicit definition of the contour as the boundary of the active region makes the calculation of the Balloon Potential a very simple operation. The Balloon Potential **bp** is obtained by simply multiplying the binary active region image **a** with the Balloon Potential weight.

$$\mathbf{bp} \leftarrow \mathbf{a} \times W_{BP} \quad (4)$$

Clearly, the definition of the “inside” and the “outside” of the balloon is determined by the definition of the active region. Thus the closed contour inflation can be achieved with positive W_{BP} or negative W_{BP}, e.g. in situation shown in Fig.3b the outer contour expands and the inner contour shrinks with positive W_{BP}.

The total potential field is the sum of the potentials:

$$\mathbf{p} \leftarrow \mathbf{ep} + \mathbf{bp} + \mathbf{ip} \quad (5)$$

C. Active Region Expansion

The expansion of the active region in one direction is achieved simply by checking the sign of the potential difference at the relevant region boundary. In other words, we can execute: "Set the active region map to '1' for pixel locations that have a neighbour belonging to the active region and the potential lower than the potential of that neighbour"

For example, expansion in the North direction is implemented by executing (in parallel, for all pixels in the array) the following:

$$\text{if } (a_{x,y+1} = \text{'1'}) \text{ and } (p_{x,y} < p_{x,y+1}) \text{ then } a_{x,y} = \text{'1'} \qquad (6)$$

where $a_{x,y}$ represents the pixel in the active region map, $a_{x,y+1}$ is the 'South' neighbour value of the active region map, $p_{x,y}$ is the pixel in the potential map and $p_{x,y+1}$ is the value of the potential in the 'South' direction.

Using array notation, the above operation can be denoted as:

$$\text{if } (\mathbf{a}_S) \text{ and } (\mathbf{p} < \mathbf{p}_S) \text{ then } \mathbf{a} \leftarrow \text{'1'} \qquad (7)$$

It has to be noted, that the operations after 'then' are executed only in those elements of the array for which the logic expression after 'if' evaluates to TRUE (i.e. logic '1'). Such conditional instruction is easily implemented in SIMD processor arrays which support a local activity flag register. The logic operation is evaluated in all processors in the array, and the activity flag register is set/reset accordingly. The following operations are executed only in the active processors in the array.

In a similar way, expansion in the other directions is executed. It should be noted, that since the contour is defined implicitly as the 1-pixel wide boundary of the active region, as the active region expands the contour shifts by one pixel, without need for Directional Contour Thinning operation.

D. Topologic Transformations and Contour Collisions

As the active region expands, it is possible that it merges with another active region. This results in corresponding merging/splitting of contours, as illustrated in Figure 5. Again, this topologic transformations are achieved through implicit definition of the contour, and do not require any additional computations.

In some applications, however, it is important to prevent the contours from colliding with each other. To achieve this, the active region expansion has to be modified. A simple local rule is used in this case. For the expansion in the North direction: if the North neighbour of the pixel under consideration belongs to the active region then there is a danger of collision; furthermore if the North-West neighbour belongs to the active region and the West neighbour belongs to the background, or the North-East neighbour belongs to the active region and the East neighbour belongs to the background, there is a danger of collision. Some of these situations are illustrated in Figure 6. In summary, looking at three neighbours of a pixel in the North direction (i.e. $a_{x-1,y-1}$, $a_{x-1,y-1}$, $a_{x+1,y-1}$) if any of them belongs to the active region and

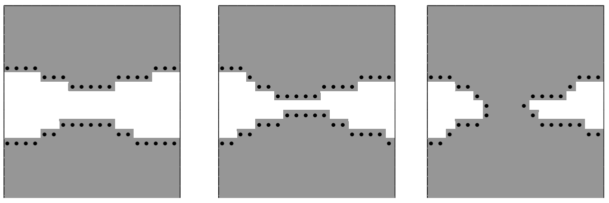
Fig.5. Merging and splitting of contours as a result of active region evolution

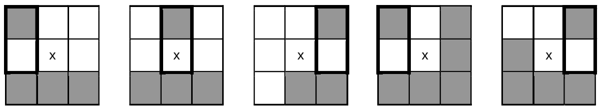

Fig.6. Example cases of contour collision detection

its South neighbour does not, then there is a "danger" of collision. The rule can be thus efficiently implemented on a pixel-parallel processor array first determining the active/background pixel pairs in the vertical direction, **r**, and then determining the danger of collision, **d**, by logic 'or' of the three neighbour pairs, as follows:

$$\mathbf{r} \leftarrow (\text{not } \mathbf{a}) \text{ and } \mathbf{a}_N \qquad (8)$$
$$\mathbf{d} \leftarrow \mathbf{r} \text{ or } \mathbf{r}_E \text{ or } \mathbf{r}_W \qquad (9)$$

The expression for directional expansion (4) should be then modified to include the collision point detection.

$$\text{if } (\text{not } \mathbf{d}) \text{ and } \mathbf{a}_S \text{ and } (\mathbf{p} < \mathbf{p}_S) \text{ then } \mathbf{a} \leftarrow \text{'1'} \qquad (10)$$

Similar calculations are performed for expansion in the other directions.

E. Active Region Inversion

The above operations lead to the expansion of the active region. In some algorithms (e.g. when a large inflating force is used, or when the active region is initialised always inside the target location) this is sufficient to ensure the contour evolution towards the minimum potential. In general, however, it is important that the contours can move in every direction. To achieve this, the next step in the proposed algorithm is to invert the active region, before next iteration of the algorithm loop.

$$\mathbf{a} \leftarrow \text{not } \mathbf{a} \qquad (11)$$

It should be noted, that inversion of the active region results in the new contour (defined implicitly as lying inside the active region) shifted by one pixel, as shown in Figure 7a-b. Two strategies can be employed at this stage. First, is to simply ignore this shift, acknowledging the fact that the 1-pixel wide contour image will be different on even and odd iterations. Second is to determine the contour from the active region first, then invert the active region, and then add the contour to the active region (this is equivalent to a dilation of the inverted active contour with a 4-neighbour structuring

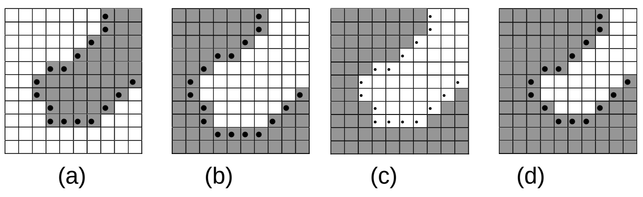

Fig.7. Active region inversion (a) contour before inversion, (b) contour after active region inversion, (c) dilation of the active region, (d) contour after inversion and dilation

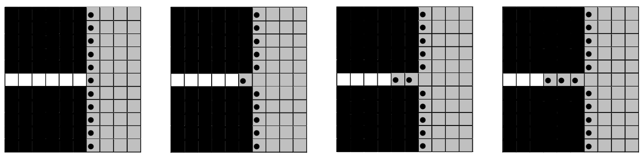

Fig.8. Contour evolution illustrating ability to propagate into 1-pixel wide cavities (black pixels denote potential barrier, inflating balloon potential is applied)

element), as illustrated in Figure 7c-d. In this way, the new contour is coincident with the previous contour. The active region inversion is therefore performed as

$$\mathbf{c} \leftarrow \mathbf{a} \text{ and not } (\mathbf{a}_N \text{ and } \mathbf{a}_S \text{ and } \mathbf{a}_W \text{ and } \mathbf{a}_E) \quad (12)$$
$$\mathbf{a} \leftarrow (\text{not } \mathbf{a}) \text{ or } \mathbf{c} \quad (13)$$

Due to the contour definition, as 1-pixel wide boundary inside of the new active region, the contour after such inversion may still be not exactly the same as before the inversion, the possible single pixel exceptions exist at the sharp corners which get smoothed (this can be seen at the bottom left corner of the shape shown in Fig.7), but this is, generally, a desirable effect.

After the active region inversion, the contour evolution is performed again. In this way, the algorithm cycles through expansions and contractions of active/background regions. Since the active region is inverted on each iteration, to maintain the consistent direction of the inflating/deflating forces the Balloon Potential needs to be inverted as well. This can be simply implemented by inverting the weight W_{BP} on each iteration.

$$W_{BP} \leftarrow -W_{BP} \quad (14)$$

It should be noted, that while the second version of region inversion, i.e. equations (12) and (13), maintains the location of the contour on even and odd iterations (which may be useful for display purposes, and may speed up the algorithm since the Internal Potential that approximates the snake internal energy could be calculated only once, every two iterations), it limits the ability of the contours to propagate into narrow cavities. On the other hand, it is possible, with the first version, to achieve contours filling 1-pixel wide cavities, as illustrated in Figure 8. Collision point detection needs to be enabled in this case, to prevent the 1-pixel narrow region to disappear on the odd iteration. Alternatively, if a large inflating balloon force is used for region growing constrained by hard potential barriers, which is typical in application such as expansion step in a path finding algorithm, it is possible to continue unidirectional evolution of the active region, i.e. without any inversion. In this case, collision point detection can be enabled.

It also can be noted, that the contour calculated by (12) can be used on the subsequent iteration to calculate the Internal Potential, instead of equation (1), thus further improving performance of the algorithm.

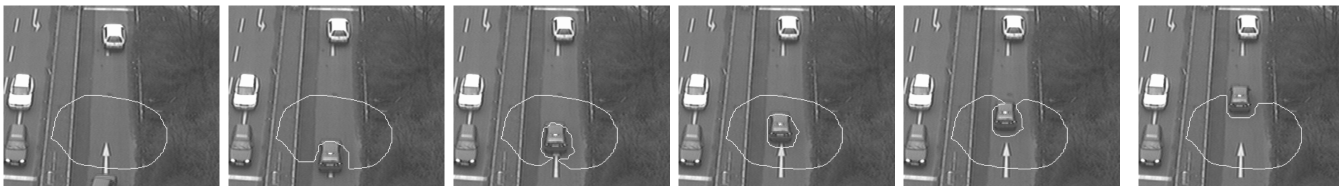

Fig.9. Several frames illustrating processing of a traffic sequence.

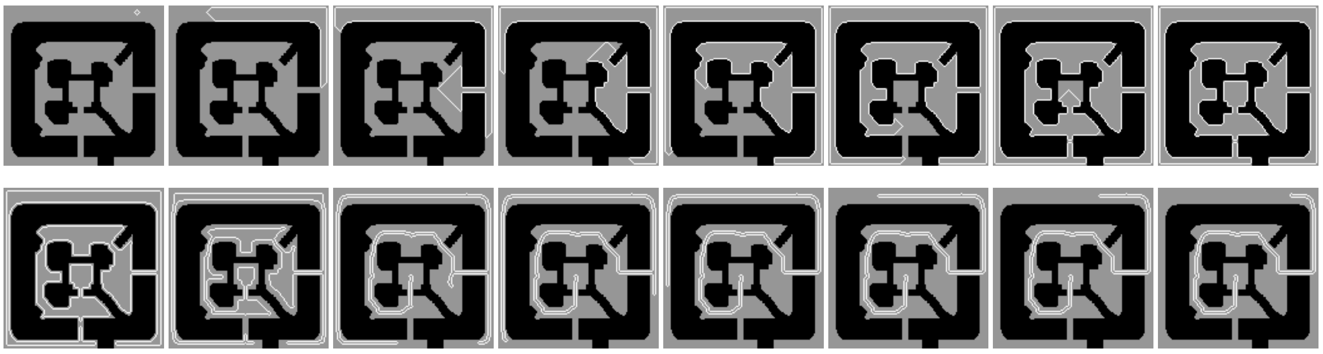

Fig.10. Path finding (target is the centre of the labyrinth): Top row sequence shows expansion of a contour due to positive inflating force (using unidirectional expansion of the active region), bottom row sequence shows deflating of the contour, anchored at START/STOP points.

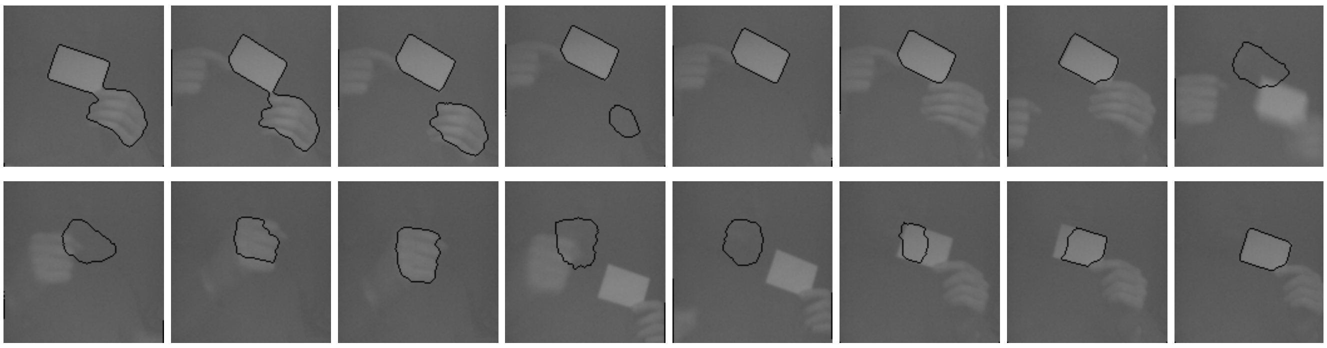
Fig.11. The algorithm executing on a 128×128 pixel-parallel processor array (SCAMP-3 vision chip).

Table 1. Execution times for the algorithm (one iteration, evolution in four cardinal directions, including) implemented on the SCAMP-3 chip

External Potential (edge map)	30 µs
Internal Potential	131 µs
Balloon Potential	4 µs
Collision Detection	64 µs
Active Region Expansion	52 µs
Active Region Inversion	22 µs
TOTAL	**303 µs**

IV. RESULTS

We have implemented the above algorithm in software on a desktop PC, using MATLAB. Figure 9 depicts several frames from a contour evolution sequence applied to a traffic monitoring application. Merging and splitting of contours, as well as nested contours, are illustrated in this sequence. Another sequence, shown in Figure 10, illustrates application of the algorithm technique to calculate the shortest-path in a labyrinth.

We have also implemented the algorithm on an SIMD processor array vision chip, SCAMP-3 [10]. The chip executes a sequence of simple array instructions (addition, inversion, one-neighbour access), which operate in a pixel-parallel fashion on 128x128 arrays, at a rate of 1.25 MOPS per pixel. Figure 11 illustrates the contour following objects in a video sequence (the images are acquired and processed in real-time on the vision chip). The time required to implement various stages of the algorithm is given in Table 1. The overall contour evolution (including potential calculations) takes 303µs per iteration, which means that many iterations of contour evolution can be executed on each frame of real-time video stream.

V. CONCLUSIONS

A new algorithm implementing cellular active contours has been proposed in this paper. The algorithm is based on the evolution of active regions, while the contours are defined implicitly as the boundary of these regions. The main advantage of the proposed algorithm is the ease of implementing contour evolution, through very simple local rules. Additionally, the calculation of closed contours (to determine inflating/deflating forces) and the implementation of topologic transformations are handled implicitly, without any additional operations, in particular eliminating the need for time-consuming ‘hole-filling’ operation. Furthermore, the algorithm enables nested contours and allows filling 1-pixel wide cavities. Overall, it is well suited for implementation on pixel-parallel processor arrays, while also offering advantages in terms of computational complexity when implemented on conventional, sequential computers.

REFERENCES

[1] A. Blake and M. Isard, Active contours, (Springer-Verlag, 1998).

[2] M. Kass, A. Witkin and D. Terzopoulos, Int. J. Comput. Vision 1, pp. 321-331, 1988.

[3] R. Malladi, J.A. Sethian and B. C. Vemuri, Shape Modeling with Front Propagation: A Level Set Approach, IEEE Trans. Patt. Anal. Mach. Intell. 17, pp. 158-174, 1995.

[4] Cs. Rekeczky and L.O. Chua, Computing with Front Propagation: Active Contour and Skeleton Models in Continuous-Time CNN. Journal of VLSI Signal Processing Systems, 23(2/3), pp. 373-402, 1999.

[5] D.L. Vilariño and Cs. Rekeczky. Pixel Level Snakes on the CNNUM: Algorithm Design, On-chip Implementation and Applications . Int. Journal of Circuit Theory and Applications, 33, pp. 17–51, 2005.

[6] D. Hillier, V. Binzberger, D.L. Vilariño and Cs. Rekeczky. Topographic Cellular Active Contour Techniques: Theory, Implementations and Comparisons, Int. J. of Circ. Theory and Appl., 34, pp. 183-216, 2006.

[7] P.Dudek, "Accuracy and Efficiency of Grey-level Image Filtering on VLSI Cellular Processor Arrays", CNNA 2004, pp.123-128, , July 2004

[8] P.Dudek, "Fast and Efficient Implementation of Trigger-Wave Propagation on VLSI Cellular Processor Arrays", CNNA 2004, pp.117-122, July 2004

[9] P.Dudek, "An Asynchronous Cellular Logic Network for Trigger-Wave Image Processing on Fine-Grain Massively Parallel Arrays", IEEE Trans. on Circuits and Systems, vol. 53, no.5, pp. 354-358, May 2006

[10] P.Dudek and S.J.Carey, "A General-Purpose 128x128 SIMD Processor Array with Integrated Image Sensor", Electronics Letters, vol.42, no.12, pp.678-679, June 2006

[11] A.Lopich and P.Dudek, "Architecture of a VLSI cellular processor array for synchronous/asynchronous image processing" IEEE International Symposium on Circuits and Systems, ISCAS 2006, pp.3618-3621

[12] M.Ishikawa et. al. “A CMOS Vision Chip with SIMD Processing Element Array for 1ms Image Processing”, Proc. ISSCC’99, TP 12.2, 1999.

[13] G. Liñán et. al. “Architectural and Basic Circuit Considerations for a Flexible 128x128 Mixed-Signal SIMD Vision Chip”, Analog Integr. Circuits and Sig. Proc. , vol.33, pp.179–190, 2002