TRACING ON HETEROGENEOUS GRIDS TO IMPROVE THE CONCAVITY PERFORMANCE OF SNAKE ALGORITHMS

András Hajdu, Ioannis Pitas

Artificial Intelligence and Information Analysis Laboratory,
Department of Informatics, Aristotle University of Thessaloniki, Greece
pitas@aiia.csd.auth.gr

ABSTRACT

Existing contour tracing algorithms operate on binary images at pixel level to traverse the boundary of an object. However, if the original grayscale image has obscure edge transitions, it can be very challenging to extract binary objects properly for pixelwise tracing. A more reliable approach can be to consider a rough approximation of the objects and perform tracing on that, followed by a final adjustment. We can use e.g. the quadtree representation which belongs to the family of heterogeneous grid representations. In this paper, we show how a contour tracing algorithm operating at pixel level can be extended to a heterogeneous grid representation. The robustness of our method is demonstrated in snake algorithms to improve their concavity performance.

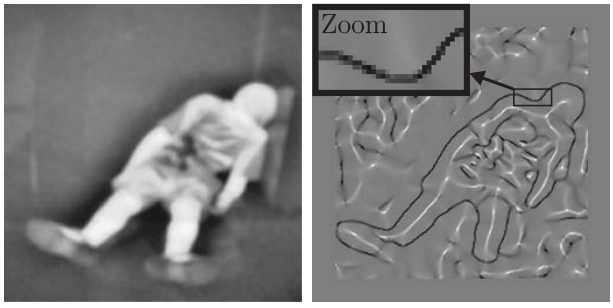
1. INTRODUCTION

Boundary extraction is an important topic in digital image processing. Thus, several approaches have been developed in the past to this end. One of them is the snake (active contour) model introduced in [1]. The basic idea here is to evolve a curve iteratively so that it approaches the object boundary.

Considering its traditional formulation, the snake is a parametric contour that deforms over a series of iterations influenced by internal and external forces. Internal forces control the snake stretching and bending, while external forces push the snake toward image edges. The problem with the traditional snake model is that it provides poor convergence to object concavities. Therefore, initial snake should be close to the desired boundary. Recently, improved snake methods were proposed [2, 3] to overcome these difficulties. However, these approaches are computationally expensive and many iteration steps might be needed to occupy concavities. Thus, if the snake contains many points (like in [4], where the final snake is used as an input for object recognition), it is highly recommended to save iteration steps. A fast and simple approach for that is presented in [5]. In our experiments, we used the Gradient Vector Flow (GVF) snake [3], which is known to have large capture range and good concavity performance.

Research was supported by the project SHARE: Mobile Support for Rescue Forces, Integrating Multiple Modes of Interaction, EU FP6 Information Society Technologies, Contract Number FP6-004218.

The novel approach we present here starts with some snake iterations. In this way, the snake usually reaches the object boundary except at the concave regions. We call a snake point (snaxel) *good*, if it has already reached the boundary of the object, and *bad*, if it has not. We find the bad segments of the snake, which usually indicate the entrances to concave object regions. We locate the good end points of the bad segments and perform a direct tracing to connect them to force the snake into the concavity. It is a natural idea to make any decisions on snaxels based on the forces directing the snake. For this reason, we consider the divergence image of the GVF external force field to classify snaxels as good or bad, and also to find a way to trace the boundary at the missing (bad) parts. A naive idea for tracing could be to try to trace the boundary directly on the divergence field. However, a closer look to Figure 1 suggests that a pixel level tracer would be really risky, as the object to trace here is rather obscure and "thick-arc" like; see [6] for the proper definition of thick-arcs.



(a) (b)

Fig. 1. Thick-arc like object boundary in the GVF divergence image; (a) thermal image for object detection, (b) its divergence image with a zoomed window to show the thick-arc like behavior.

Thus we need a more reliable representation of the boundary on the divergence image, which can be achieved e.g. by its quadtree representation. For this reason, we use those quadtree cells which have low mean divergence values, as they are expected to cover the object boundary. The cells belonging to the quadtree decomposition

generate a heterogeneous grid [7], which let the usage of squares of any size and generalizes the classic 4- and 8-neighboring relations, accordingly.

The tracing can be performed on the quadtree cells by extending known contour tracing algorithms operating at pixel level. Note that, roughly speaking, the heterogeneous grid representation covers the boundary of the object, and thus the tracing performed inside the concavity will direct the new snaxels only very close but not exactly onto the boundary. However, this small inaccuracy can be nicely and easily compansated by applying again some snake iteration steps to complete the procedure.

The paper is organized as follows. Section 2 describes the basic concepts about heterogeneous grids and shows how a tracer can be generalized to this representation. In section 3 we present how the bad snake segments are located. Section 4 contains our experimental tracer results for a quadtree-based object representation. Finally, some conclusions are drawn in section 5.

2. BOUNDARY TRACING ON HETEROGENEOUS GRIDS

2.1. Heterogeneous grids

The idea of the 2D heterogeneous grid [7] generalizes the classic rectangular grid $\mathbb{Z}^2$ equipped by the well-known 4- and 8-neighboring relations. In this representation the plane is tiled with squares of any side length. Some examples are shown in Figure 2a. Note that one of the examples is just the quadtree structure that we will use later on. Two basic adjacency relations can be defined [7] for touching squares, see Figure 2b,c.

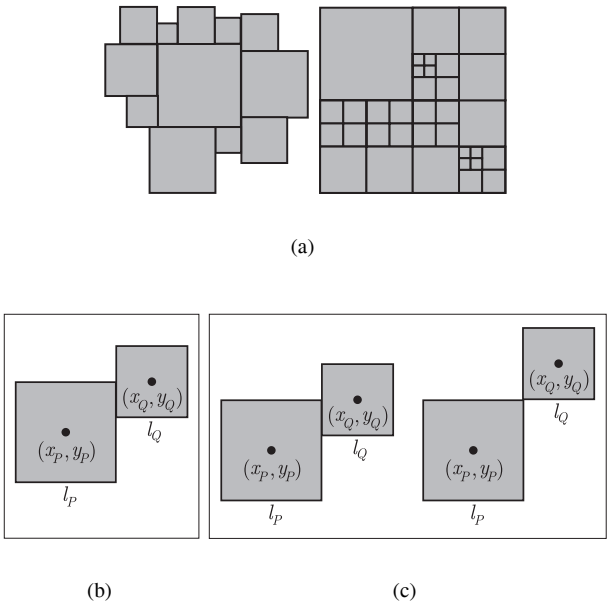


Fig. 2. (a) Heterogeneous grids; (b) e-adjacency, (c) ve-adjacency.

More precisely, a 2D heterogeneous pixel P (square) is a triplet $(x_P, y_P, l_P) \in \mathbb{R}^3$, where (x_P, y_P) is the center of the square, while l_P denotes its side length. The heterogeneous pixels P and Q are:

- *ve*-adjacent, if $\max(|x_P - x_Q| - |y_P - y_Q|) = \frac{l_P + l_Q}{2}$,
- e-adjacent, if $\max(|x_P - x_Q| - |y_P - y_Q|) = \frac{l_P + l_Q}{2}$, and $|x_P - x_Q| \neq \frac{l_P + l_Q}{2}$ or $|y_P - y_Q| \neq \frac{l_P + l_Q}{2}$.

As we will focus on finite subsets of heterogeneous grids, we will call any finite subset of such a grid a heterogeneous object.

2.2. Generalizing a tracer for heterogeneous grids

Many boundary tracers are known from the literature [8]. Any of them could be generalized to heterogeneous grids. However, we will use a less-known, but rather simple, elegant and effective tracer instead. This tracer (which was built in the Recognita® OCR software [9]) considers the vectors between object and object's complement boundary points instead of object points. This tracer is based on a very pure idea. Namely, if we make the tracing clockwise (CW), we keep the object always at our right hand size. According to the relative actual direction at a tracing step (see Figure 3a), we turn "left", if a is an object point, else go "straight", if b is an object point, or else turn "right". For a counter-clockwise (CCW) tracing, we have to invert the directions.

Our intention is to generalize this tracer for heterogeneous objects. We extract these objects through the quadtree representation by keeping those cells which meet a special requirement. It is obvious that a simple approach could be to use the original tracer idea to traverse the boundary of the heterogeneous object at pixel level. However, we can achieve an improvement here by focusing only on the essential information provided by the heterogeneous cell descriptors.

Namely, from the heterogeneous object representation we extract maximal line segments which are between a boundary heterogeneous cell and the complement of the object, as can be seen in Figure 3b,c.

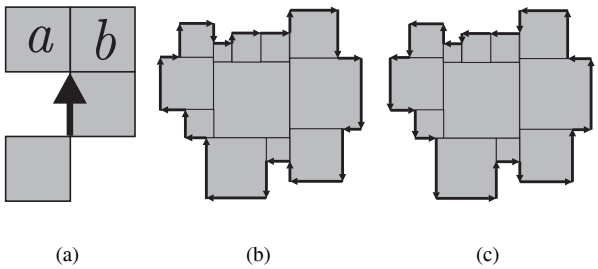


Fig. 3. Extending a pixelwise tracer to heterogeneous grids, (a) a pixelwise algorithm tracing in CW direction (see text); (b) its extension to heterogeneous grids in CW direction, (c) its extension to heterogeneous grids in CCW direction.

The line segments can be extracted easily from the quadtree, by checking the e- and ve-neighbors of each

cell. These segments are represented by (sp, ep) pairs, where sp, ep, denote the start and end pixel of the line segment, respectively. Note that we have directed line segments, since $(sp, ep) \neq (ep, sp)$. Then we organize them into two groups, based on whether they can be used in CW or CCW tracing. Note that if (sp, ep) belongs to the CW group, then (ep, sp) belongs to the CCW one. In both these groups, we can apply a simple indexing to achieve fast access to the next segment during tracing. Thus, we assign the line segment (sp, ep) to the index entry sp. Then, to find the next segment, we take simply the segment corresponding to the current endpoint entry of the index table. It can be easily checked that this heterogeneous object tracer (just like its pixelwise ancestor) always returns to the starting vector (sp_1, ep_1). This property can be used as a stopping criterion to extract closed object boundaries.

3. LOCATING BAD SNAKE SEGMENTS

The main idea behind GVF snakes [3] is to calculate a GVF force field based on the diffusion of the gradient values. The final snake is achieved by an iterative process, which can be tuned by weight parameters for the shape and the external field. The GVF snake is known about its concavity resistance, though it may need many computationally expensive iteration steps to occupy them. See Figure 4a,b for the performance of the GVF snake after some iteration steps.

(a) (b) (c)

Fig. 4. The performance of the GVF snake; (a) input thermal image with initial snake points, (b) the GVF snake after some iteration steps, (c) locating bad snake parts based on the divergence image of the GVF external force field.

As we use the GVF field for deforming the snake, we calculate the divergence of the GVF field to decide whether snaxels reached the desired boundaries. Let

$$F(x, y) = F_x(x, y)\mathbf{i} + F_y(x, y)\mathbf{j} \tag{1}$$

be the GVF field of the image, where F_x and F_y are the coordinate functions of F (that is the horizontal and vertical component of the GVF field), and $\mathbf{i}$, $\mathbf{j}$ the corresponding basis unit vectors. The divergence of F, denoted by $divF$, is the scalar valued function [10]:

$$divF = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} \tag{2}$$

that can be also considered as the divergence image of the GVF field. The physical significance of the divergence is the rate at which GVF field "density" exits a given region of space. In the divergence image, low values correspond to the object boundaries, while large values to those areas which are far from the boundaries. We use divergence values to check whether a snaxel has already converged to the boundary or not.

The use of the divergence instead of some simple considerations on the local vector behavior (like in [11]) provides a more concrete basis for this decision. Namely, we say that a snaxel reached the boundary, if its divergence value satisfies:

$$divF < \theta, \tag{3}$$

where θ is an appropriately chosen threshold. The magnitude of θ should be adjusted according to the edge strength of the image. Snaxels that do not fulfill (3) are considered belonging to bad segments, and as it can be seen in Figure 4c are removed from the snake. As snaxels are stored in an ordered sequence, we can easily locate the remaining good end-points of the bad segments. Based on the distance of the remaining consecutive snaxels, we can decide whether to initiate a tracer between them. Accordingly, we select a start ssp and end sep snaxel to trace between.

4. TRACING QUADTREE BASED HETEROGENEOUS OBJECTS

In our current approach, we perform a quadtree decomposition of the GVF divergence field, as shown in Figure 5a. To approximately cover the boundary of the object by larger cells, we select only those cells from the quadtree, which contain low divergence values. It is well-known that the parameters of the quadtree procedures provide wide flexibility regarding the desired size and intensity distribution of the cells. In our case, as a natural threshold, we keep only those blocks, which average intensity is smaller than $\theta + \varepsilon$, with some $\varepsilon > 0$, where θ is the same threshold as in (3). The role of θ is to include all the boundary points within the blocks preserved. Larger ε makes the preserved blocks larger, by allowing points a bit more away from edges. A too large ε might lead to cover narrow concavities completely, which makes the main goal meaningless. The result is shown in Figure 5b. We shall denote by H the heterogeneous object defined as the union of the selected quadtree cells. H can be even disconnected, but has a subset which can be considered as a rough estimation of the object contour. See Figure 5c for a closer look of a countersection, where the snake breaks.

As the next step, we extract the necessary indexed data to trace the boundary of H as it was described in Section 2. To start the tracer from ssp we have to determine a traceable edge (sp_1, ep_1) of H. To set up a successful stopping rule, we have to determine a similar edge for sep. The tracer stops when reaching the end point of the stopping edge.

We also have to determine whether tracing should be performed in CW or CCW. We follow the process presented in [5] here, on how to estimate the direction of the

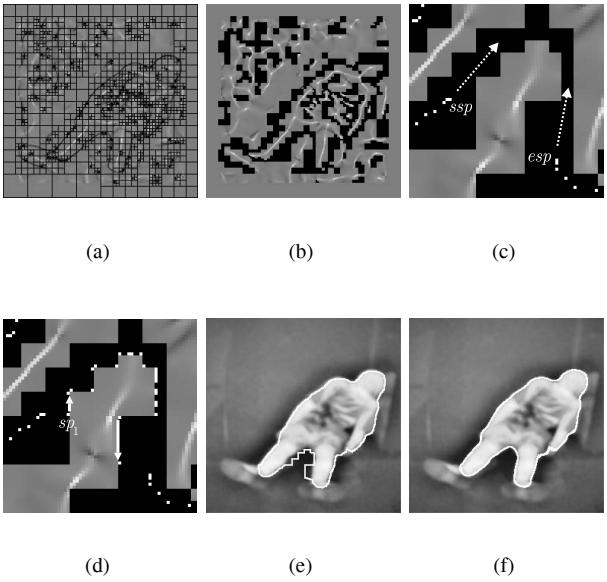


Fig. 5. Steps of tracing on quadtree based heterogeneous grid, (a) quadtree decomposition of the divergence field, (b) heterogeneous object H (black) to cover the object boundary, (c) missing snake part, and the estimation of boundary direction, (d) tracing the boundary of H, (e) input snake for the closing GVF iterations, (f) final result.

boundary. (In short, we perform a check to detect pixels with low divergence around ssp not occupied by the snake. The estimated directions are shown with dotted line in Figure 5c.) When we have an estimation for the behavior of the boundary from ssp on, we start searching for a boundary edge of H from ssp in the direction of esp also considering the estimated direction of the boundary. This edge will be considered as the starting edge of the tracer. According to the CW and CCW selection, the tracer should start in that direction, which is closer to the estimated boundary direction. The stopping edge of H can be found similarly, by changing the role of ssp end esp. See Figure 5d, where the starting and ending edges are shown by solid arrows, together with the tracing result.

We insert the new part found by tracing between ssp and esp into the snake. As the tracing also has sequential behavior, the ordering of the snaxels can be preserved without any difficulties. To complete, we perform some additional GVF iterations to match this rough estimation accurately to the object boundary. However, few number of iterations are needed now, as the snake is already close to its desired final position. See Figure 5e, for the snake after the tracing step, and the final result in Figure 5f after some more iteration steps.

5. CONCLUSION

The quadtree representation offers many advantages. On the one hand, we do not have to mind noise and boundary linkage problems, as these local inadequacies can be compensated by larger blocks. On the other hand, we can save computations with working on a sparser grid.

The basic model presented in the paper can be improved and extended in many ways. For example the presented tracer could operate in any other representations where the boundary edges can be easily extracted (e.g. grid of rectangles). Moreover, the method can be used both for (thick-) arc processing, and object boundary detection. Nevertheless, one point should be kept in mind, namely that the resulted approximate is just a rough estimation of the pixelwise boundary. For instance, the snake approach considered here lends a perfect tool to perform the final adjustment. Our approach can have even higher importance for snake approaches that are less tolerant to concavities (e.g. [12]).

6. REFERENCES

[1] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," Int. J. Comp. Vis., vol. 1, pp. 321-331, 1987.

[2] D. Gil and P. Radeva, "Curvature vector flow to assure convergent deformable models for shape modelling," LNCS, vol. 2683, pp. 357-372, 2003.

[3] C. Xu, and J.L. Prince, "Snakes, shapes, and gradient vector flow," IEEE Trans. IP, vol. 7, pp. 359-369, 1998.

[4] A. Hajdu, A. Roubies and I. Pitas, "Improving the performance of the GVF snake algorithm," ISCCSP, 2006.

[5] A. Roubies, A. Hajdu and I. Pitas, "Optimized chamfer matching for snake-based image contour representations," ICME, 2006.

[6] F. Alhalabi and L. Tougne, "Toward polygonalisation of thick discrete arcs," LNCS, vol. 3691, pp. 197-204, 2005.

[7] D. Coeurjolly and L. Tougne, "Digital straight line recognition on heterogeneous grids," Vision Geometry XII, pp. 283-294, 2004.

[8] T. Pavlidis, "Algorithms for graphics and image processing," *Computer Science Press*, 1982.

[9] http://www.nuance.com/omnipage/

[10] E.C. Young, "Vector and tensor analysis," *Marcel Dekker*, 1993.

[11] C.H. Chuang and W.N. Lie, "A downstream algorithm based on extended gradient vector flow field for object segmentation," IEEE Trans. IP, vol. 13, pp. 1379-1392, 2004.

[12] L.D. Cohen and I. Cohen, "Finite-element methods for active contour models and balloons for 2-D and 3-D images," IEEE Trans. PAMI, vol. 15, pp. 1131-1147, 1993.