Spatial Temporal Active Contour Interpolation for Semi-automatic Video Object Generation *

Huitao Luo and Alexandros Eleftheriadis
Dept. of Electrical Engineering, Columbia University
{luoht, eleft}@ee.columbia.edu

Abstract

An active-contour-interpolation approach is proposed for semi-automatic video object generation. In this approach, user can define the contour of the same video object on multiple frames and the computer interpolates the missing contours automatically. Typical active contour model is adapted and the contour interpolation problem is decomposed into two directional contour tracking problems and a merging problem. In addition, a new concept of parametric neighborhood template is introduced in order to improve the robustness of contour tracking.

1 Introduction

Tracking semantically meaningful video objects (VOs) through a video sequence is currently an interesting topic. Typical approaches in literature, for example tracking [4, 5], generally have some user interactions to define a VO in the first frame, and then let the computer to track the VO temporally. A problem in these approaches is that during the tracking course, no quality criteria have been proposed for the computer to detect the loss of tracking and thus ask for additional user input. Instead, the user has to observe the tracking course from time to time and offer new input when he or she finds necessary. In this paper, we propose a new interpolation model for semi-automatic segmentation. In our system, the user can define a video object by specifying its contour on multiple frames rather than only on the first frame. The computer then uses input information from multiple frames to "interpolate" the VO contour on every frame. Compared with the pure tracking approach, the interpolation approach is more predictable in that the user can define the VO on frames where large occlusion or motion occurs and most tracking algorithms are likely to fail. In addition, the interpolation approach makes better use of user input because the user input on one frame contributes to the VO definition on the frames before as well as after it, while in the tracking case, the user input on one frame only influences the frames after it.

More specifically, our work can be expressed as follows. Given two input contours C_b and C_e of a video object on frame b and frame e, try to find the object contours C_i on frame i, $i = b+1, b+2, \cdots, e-1$. We call it a "contour interpolation" problem. As a comparison, the "contour tracking" problem is expressed as: given one input contour C_b, try to find the object contour C_i on frame i, $i = b+1, b+2, \cdots, e$. It is natural to consider an interpolation problem as two tracking problems, i.e., to maximize the use of input information on two frames, we can track the input contour from C_b to C_e also well as from C_e to C_b. In this sense, all the available VO tracking algorithms can be used. However, how to merge the results from these two directional tracking and produce a final best result is obviously an open problem.

To maximize the use of user input on two frames, an active contour (snake) model [1] is used in our interpolation algorithm. In our work, a traditional snake is extended in the following ways. First, we use nodes to represent snakes and we design a "contour matching" algorithm to match the node-representation of two user input contours. Based on contour matching, a contour temporal smoothness criterion is defined. Later discussion will show that this criterion is essential for fusing the multiple tracking results. Second, we extend the 2-dimensional snake model to 3-dimensional model in which new energy terms that reflect spatial temporal constraints are included. Third, a "parametric neighborhood template" is designed to improve the robustness against background boundary noises during the active-contour tracking.

The organization of this paper is as follows. First Section 2 introduces the contour representation and contour matching algorithm. Section 3 discusses in detail the contour interpolation algorithm based on

*For better view of color pictures, ps file downloadable at http://www.ee.columbia.edu/~luoht/research/active_contour

contour matching. Finally in Section 4, experiments are presented and the paper is concluded.

2 Contour Matching Algorithm

2.1 Contour Representation

In this work, a contour can be represented by a vector array $\{\mathbf{v}_{s,k}\}$, $(s = 0, 1, \cdots, N)$, where k is the temporal location and s is the spatial index of its contour pixels. The spatial location of each contour pixel is denoted by a vector $\mathbf{v}_s = (x_s, y_s)$. In addition, for concise representation and easy matching, a contour can also be represented with subsampled nodes as $\{\mathbf{v}_{S_k(i),k}\}$, where $S_k : i \mapsto j$, $i \in [0, 1, \cdots, N_s]$, $j \in [0, 1, \cdots, N]$ is the subsampling function related to temporal position k. For example, uniform subsampling function is defined as $S_k(i) = i \cdot unit$, where *unit* is the subsampling unit. In this paper, we name $\{\mathbf{v}_{s,k}\}$ as *pixel representation* and $\{\mathbf{v}_{S_k(i),k}\}$ as *node representation*. Note that the node representation is actually a first order polynomial approximation of the pixel representation.

2.2 Contour Matching

The purpose of contour matching is to find the correspondence between two contours C_b, C_e, which are the contours of the same video object at the different temporal locations (b and e). Because in pixel representation, the length of C_b and C_e is not necessarily the same, we use the subsampled node representation for contour interpolation study and a contour matching algorithm is used to create correspondence between two subsampled contours. Mathematically, the matching process can be expressed as follows. Given two input contours: $C_b = \{\mathbf{v}_{s,b}\}$, $C_e = \{\mathbf{v}_{s,e}\}$ (in pixel representation), and a subsampled node representation for the first contour $\{\mathbf{v}_{S_b(i),b}\}$, find the corresponding node representation for the second contour $\{\mathbf{v}_{S_e(i),e}\}$. Here the matching of the nodes of the two contours can be expressed with the mapping function $f_m : \mathbf{v}_{S_b(i),b} \mapsto \mathbf{v}_{S_e(i),e}$, $i \in [0, 1, \cdots, N_s]$. Generally, we fix the node representation of the first contour C_b by uniform subsampling, and the matching process is reduced to finding the corresponding subsampling function $S_e(i)$ for the second contour.

In order to find the mapping function f_m, we define a local energy term for each matched node pair. The final matching result is determined by the global minimization of the matching energy of the two contours. This is similar to the matching approach in [2]. In this work, we assume the motion of the considered video object is nonrigid globally, but rigid locally, and the shape of its two contours is similar locally everywhere. Two rigid motion models, i.e., translation and affine, are possible for local matching energy definition.

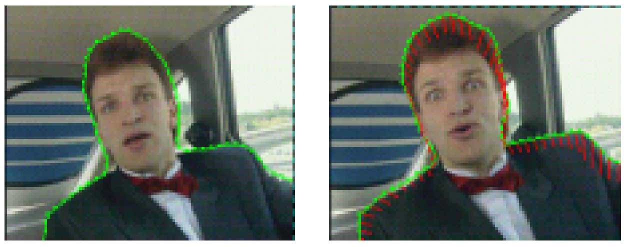
(a) (b)

Figure 1: Results of contour matching for carphone sequence.

Translation Motion Model: If we denote the motion vector between the two matched nodes as $MV_i = \mathbf{v}_{S_b(i),b} - \mathbf{v}_{S_e(i),e}$, then the matching energy term is defined as

$$E_i = \mu||MV_i - MV_{i-1}|| + \eta(S_e(i) - S_e(i-1))^2,$$

where the first term is the smoothness evaluation of the motion vectors of two neighboring nodes, the second term is an elastic constraint on the distance of two neighboring nodes, and μ, η are two weighting factors.

Affine Motion Model: In contrast to the translation model, the affine model needs three motion vectors to define. Here we denote the affine function as $\mathcal{A}_i = \text{Affine}_i(MV_{i-1}, MV_i, MV_{i+1})$. Under this function, local contour segment $\{\mathbf{v}_{s,b}\}$, $s \in [S_b(i-1), S_b(i+1))$ is projected to pixel set $\{\mathbf{v}'_e(s)\}$, while its corresponding contour segment in frame e is denoted as pixel set $\{\mathbf{v}_{S_e(s),e}\}$, $s \in [S_e(i-1), S_e(i+1))$. Then the matching energy term is defined as

$$E_i = \mu D\big(\{\mathbf{v}'_e(s)\}, \{\mathbf{v}_{S_e(s),e}\}\big) + \eta(S_e(i) - S_e(i-1))^2,$$

where operator $D(\cdot)$ is the distance measure of two sets, which we define as

$$D(A, B) = \Big[\sum_{\mathbf{v}_i \in B} \min_{\mathbf{v}_j \in A} ||\mathbf{v}_i, \mathbf{v}_j|| + \sum_{\mathbf{v}_j \in A} \min_{\mathbf{v}_i \in B} ||\mathbf{v}_j, \mathbf{v}_i||\Big]/2.$$

With the definition of local matching energy terms, the matching problem is converted to an energy minimization problem. This can be easily solved by DP algorithm [3], which we do not discuss in detail here. In Fig. (1), we show a result of contour matching under translation model. Fig. (1a) is the 50th, (1b) is the 70th frame of carphone sequence. Green lines are their contours and red lines link the matched node pairs.

3 Contour Interpolation Based on Bidirectional Contour Tracking

3.1 Localized Energy Model

The essence of Kass' snake model [1] is to define a local energy term for each node and the shape of the contour is determined by minimizing the total snake energy globally. When we go from a 2-dimensional spatial snake model to a 3-dimensional spatial/temporal model, it is possible to extend the local energy terms from 2D to 3D as well. In this work, we study the extended local energy terms as intraframe energy terms and interframe energy terms respectively. From now on, because no subsampling issue will be involved, snakes are assumed to be in node representation. The node representation $\{\mathbf{v}_{S_k(i),k}\}$ is simplified to $\{\mathbf{v}_{i,k}\}$ whenever possible.

Intraframe Energy: As usual, we define the intraframe energy of a snake node as

$$E_{intra,i} = \eta_{edge} E_{edge,i} + \eta_{smooth} E_{smooth,i}, \quad (1)$$

where $E_{edge,i} = \int_{s=S(i-1)}^{S(i)} 255/(10 + || \bigtriangledown (\mathbf{c}(\mathbf{v}_s))||)ds$, ($\mathbf{c}(\mathbf{v})$ is the color vector of node $\mathbf{v}$), and

$$E_{smooth,i} = \Big(\frac{2||\mathbf{v}_{S(i-1)} + \mathbf{v}_{S(i+1)} - 2\mathbf{v}_{S(i)}||}{||\mathbf{v}_{S(i-1)} - \mathbf{v}_{S(i+1)}|| + ||\mathbf{v}_{S(i)} + \mathbf{v}_{S(i+1)}||} \Big)^2.$$

Interframe Energy: In available papers on temporal active contour tracking [4, 5], generally employed interframe energy terms include optical flow, motion smoothness, interframe color, etc. A basic problem in their approaches is that most of their node energy definitions are based on the image feature only at the node's position rather than in its neighborhood. This makes the color and especially motion information generally not accurate. Ideally, contour nodes' neighborhood should be observed in order to track them frame by frame. However, a problem here is, different from typical point tracking problem, contour nodes are on the boundary of a moving object, so their neighborhood is not constant. In order to capture the consistency through the tracking course, we introduce the concept of *parametric neighborhood template*. A parametric template T is defined as a data structure including two arrays: a vector array $\{\mathbf{dv}_0, \mathbf{dv}_1, \cdots, \mathbf{dv}_n\}$ and a weighting array $\{w_0, w_1, \cdots, w_n\}$. For each contour node $\mathbf{v}_{i,k}$, a parametric template $T_{i,k}$ is defined and kept updated frame by frame through the tracking course.

With parametric template $T_{i,k}$, the color of two temporally neighboring contour nodes $\mathbf{v}_{i,k}$, $\mathbf{v}_{i,k-1}$ can be compared as $\text{Diff}_c(\mathbf{v}_{i,k}, \mathbf{v}_{i,k-1}, T_{i,k}) =$

$$\sum_{\mathbf{dv}_j \in T_{i,k}} w_j ||\mathbf{c}(\mathbf{v}_{i,k} + \mathbf{dv}_j) - \mathbf{c}(\mathbf{v}_{i,k-1} + \mathbf{dv}_j)||, \quad (2)$$

where $\mathbf{c}(\mathbf{v})$ is the color vector of node $\mathbf{v}$. In addition,

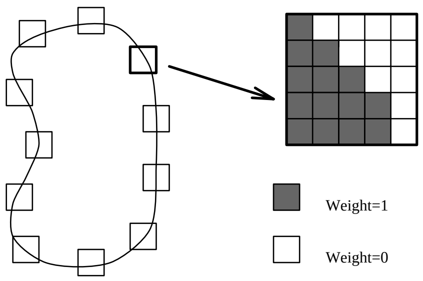

Figure 2: Parametric template illustration.

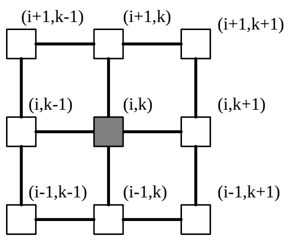

Figure 3: Spatial temporal neighborhood of a contour node for local energy definition. 'k' is the temporal and 'i' is the spatial index.

we define the motion vector at the node $\mathbf{v}_{s,k}$ as

$$MV(\mathbf{v}_{i,k}, T_{i,k}) = \sum_{\mathbf{dv}_j \in T_{i,k}} w_j \cdot MV^{(p)}(\mathbf{v}_{i,k} + \mathbf{dv}_j) / \sum_{\mathbf{dv}_j \in T_{i,k}} w_j, \quad (3)$$

where $MV^{(p)}(\mathbf{v}_{i,k})$ is the estimated motion vector at $\mathbf{v}_{i,k}$. If we do not consider occlusion, a simple way to determine the weights of template T is to set w_j for those neighboring pixels inside the contour as 1 and for those outside the contour as 0. This can be illustrated in Fig. 2. For the occlusion case as was discussed in [5], we can easily switch the weights by setting inside weights to 0 and outside weights to 1. In the general cases, both the size and the weights of the parametric template can be adjusted flexibly to get the best results of the contour nodes tracking.

Based on parametric template, we sum up the interframe energy for each contour node as follows.

1. Color similarity:
   $E_{color,i,k} = \text{Diff}_c(\mathbf{v}_{i,k}, \mathbf{v}_{i-1,k}, T_{i,k})$. Here function $\text{Diff}_c()$ is defined in Eq. 2.

2. Optical flow: $E_{optical,i,k} = MV(\mathbf{v}_{i,k-1}, T_{i,k-1}) - MV_{i,k-1}$. Here the first term $MV(\mathbf{v}_{i,k-1}, T_{i,k-1})$ is defined in Eq. 3 and we assume forward motion estimation is used. The second term $MV_{i,k-1} = (\mathbf{v}_{i,k} - \mathbf{v}_{i,k-1})$.

3. Motion smoothness: $E_{motion,i,k} = ||MV_{i-1,k} + MV_{i+1,k} - 2MV_{i,k}||$.

4. Shape stiffness:
$E_{shape,i,k} = ||\text{angle}(\mathbf{v}_{i-1,k-1}, \mathbf{v}_{i,k-1}, \mathbf{v}_{i+1,k-1}) - \text{angle}(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k})||$.

5. Temporal smoothness: $E_{temporal,i,k} = ||\mathbf{v}_{i,k-1} + \mathbf{v}_{i,k+1} - 2\mathbf{v}_{i,k}||$.

The interframe energy E_{inter} is then the weighted sum of above terms.

Search Algorithm for Minimization: With the definition of local energy terms, the contour interpolation problem can be expressed as an energy minimization problem as follows. Given two contours $\{\mathbf{v}_{i,b}\}, \{\mathbf{v}_{i,e}\}, (b < e, i = 0, 1, \cdots, N_s)$, find the contour nodes $\{\mathbf{v}_{i,k}\}, (k = b+1, \cdots, e-1, i = 0, 1, \cdots, N_s)$, that minimize the global energy $\sum_{i,k}(E_{inter,i,k} + E_{intra,i,k})$.

Though it is natural to extend local energy terms from 2D to 3D, the increasing in computation complexity is an important problem. In the 2D case, each node $\mathbf{v}_i$'s local energy is defined in relation to two neighbors, i.e., $E_{intra} = f(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$, while in the 3D case, the local energy terms for each node $\mathbf{v}_{i,k}$ is defined in relation to eight neighbors! This change is illustrated in Fig. 3. If each node has a search region of n, the local searching complexity for each node increases from n^3 to n^9, which makes global minimization algorithm difficult to design. Though powerful algorithms such as *simulated annealing* should still be able to solve this minimization problem, the slow speed of convergence makes it inappropriate for an interactive segmentation tool.

In this work, we try to find a sub-optimal solution by converting the contour interpolation problem into two tracking problems (a forward tracking from C_b to C_e and a backward tracking from C_e to C_b. They are also referred to as bi-directional tracking in this paper). When converted to a tracking problem, the interframe energy $E_{inter,s,k}$ is reduced to depending only on two neighboring nodes in current frame k, which is the same as intraframe energy $E_{intra,s}$. Therefore, it is easy to design a searching algorithm with DP. After the bi-directional tracking, another searching process is used to find the optimal contours out of the previous tracking results.

3.2 Bi-directional Tracking and Merging of Multiple Results

Due to space limit, we do not discuss the contour tracking algorithm in detail. Generally we employ a DP algorithm similar to [3]. The contour in the current frame is first projected with dense motion vectors to the next frame, and its shape is then decided by energy minimization. However, because the parametric template is used in our algorithm, experiments show that our tracking algorithm is more robust than the available contour tracking algorithms such as [4].

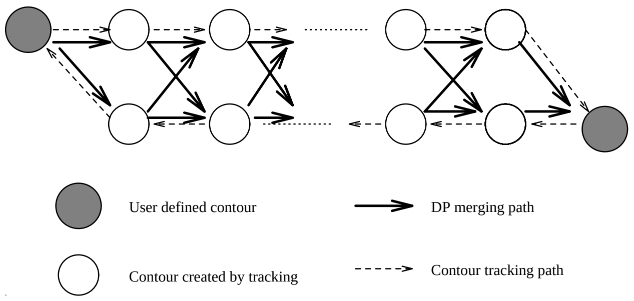

Figure 4: Illustration of DP approach for merging the bi-directional contour tracking.

Though the bi-directional tracking approach reduces the searching complexity, its limitation is that it only makes use of user input on one frame (either C_b or C_e). Due to error accumulation, the tracked contour C_k always degrades when k approaching e, if tracked from C_b to C_e, and vice versa. In this work, we design an efficient DP algorithm to merge the contours created by the two tracking processes. The problem may be expressed as follows. Given two set of contours $\{C_k^{(1)}\}$, $\{C_k^{(2)}\}$, $(k = b, b+1, \cdots, e)$, that are created by two contour tracking processes (one from C_b to C_e and the other from C_e to C_b), find a contour set $\{C_k\}$, $(k = b, b+1, \cdots, e, C_k = C_k^{(1)}$ or $C_k = C_k^{(2)})$ that meets certain *merit criterion* as the final output of contour interpolation. In this work, the *merit criterion* is a localized energy terms: temporal smoothness energy E_T, which is defined as: $E_T(C_k) = \sum_{i=0}^{i=N_s} ||\mathbf{v}_{i,k-1} + \mathbf{v}_{i,k+1} - 2\mathbf{v}_{i,k}||$, where N_s is the number of nodes in each contour. Note here $E_T(C_k)$ is different from the previously defined $E_{temporal}$ in that $E_T(C_k)$ is defined for each contour while $E_{temporal}$ is defined for each contour node. Because $E_T(C_k)$ is defined in relation to two neighbors C_{k-1} and C_{k+1}, it is easy to solve the minimization problem with DP. Here Fig. 4 is used to illustrate the DP based merging algorithm. In Fig. 4, each circle represents a contour $C_k^{(d)}$, the DP algorithm is used to find an optimal path in the temporal direction that has the best temporal smoothness quality.

4 Experiments and Conclusion

The proposed algorithm was tested on several MPEG-4 video sequences. First, we compared the effect of the parametric template for the active contour

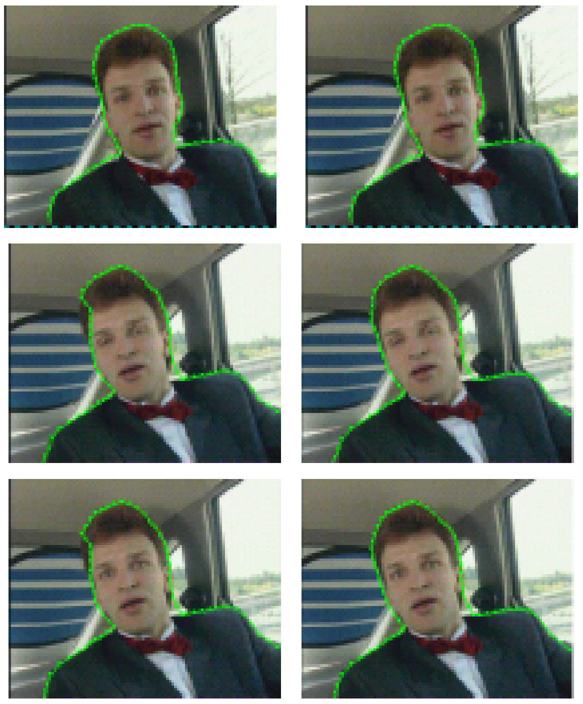

Figure 5: Comparison of the effect of parametric template on active contour tracking.

tracking on carphone sequence. In Fig. 5, the left column is the tracking result without and the right column is the result with the parametric template. The first row is the initial contour user defined on frame 0, the following two rows are the tracked contours on frames 43 and 47. Obviously the parametric template improves the tracking robustness on complex backgrounds. Fig. 6 depicts the merging of tracking results on mother-daughter sequence. User defined VO contours are on frame 75 and 180. The left column is the results of the backward tracking from frame 180 to frame 75, and the right column is the interpolation results that merged bi-directional trackings. The frame numbers, from top to bottom are 170, 160, 150, 130 and 90. We can see that from frame 180 to 160, the merged results are taken from backward tracking, while from 160 to 75, the merged results are taken from forward tracking (which is not shown here due to space limit). That is, the merged interpolation results are better than tracking results on either direction.

References

[1] M. Kass et al., “Snakes: Active contour models,” *Int. J. Comput. Vision*, vol. 1, no. 4, 1988.

[2] D. Geiger et al., “Dynamic programming for detecting, tracking and matching deformable contours,” *IEEE Trans. PAMI*, vol. 17, no. 3, 1995.

[3] A.A. Amini et al., “Using dynamic programming for solving variational problems in vision,” *IEEE Trans. PAMI*, vol. 12, no. 9, 1990.

[4] Y.-T. Lin et al., “Tracking deformable objects with the active contour model,” in *IEEE ICMCS*, Ottawa, 1997.

[5] Y. Fu, A.T. Erdem, and A.M. Tekalp, “Occlusion adaptive motion snake,” in *ICIP’98*, Chicago.

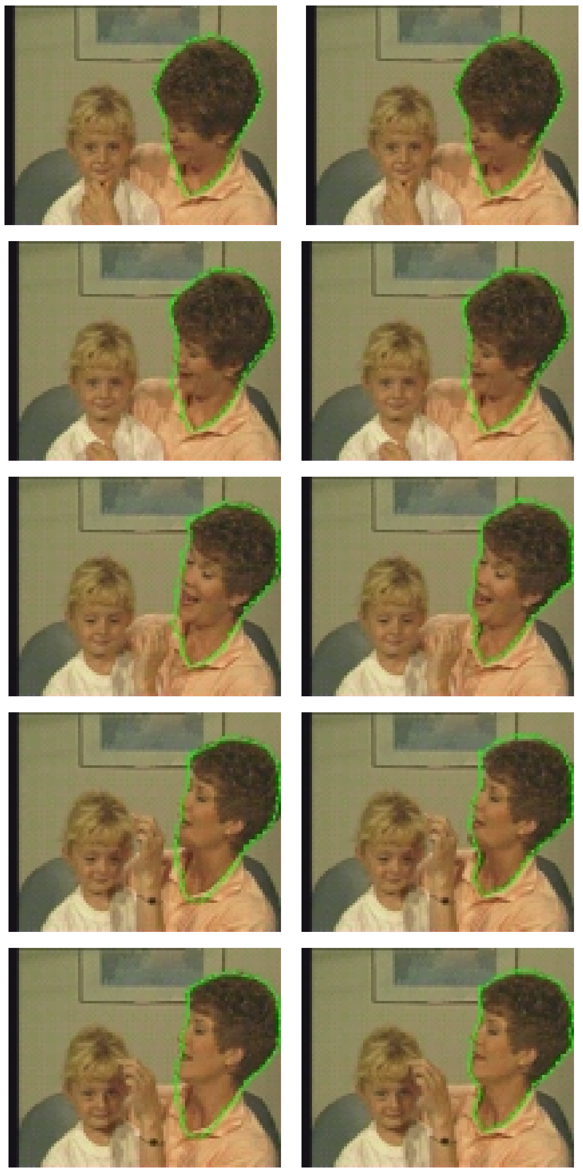

Figure 6: Illustration of tracking results merging.