Title: Designing an Interactive Tool for Video Object Segmentation and Annotation

Note: applied for a candidate for the student paper competition (the primary author is currently a third year Ph.D. student at Columbia Unversity).

Corresponding Author:

Huitao Luo
Dept. of EE, Columbia Unversity
1312 Mudd, 500 West 120th Str., MC4712
New York, NY 10027
luoht@ctr.columbia.edu

Technical Area:
multimedia tools, end-systems and applications

Designing an Interactive Tool for Video Object Segmentation and Annotation[*]

Huitao Luo and Alexandros Eleftheriadis
Advent Group, Columbia University
{luoht, eleft}@ctr.columbia.edu

ABSTRACT

An interactive authoring system is proposed for semi-automatic video object segmentation and annotation. This system features a new contour interpolation algorithm, which enables the user to define the contour of a video object on multiple frames while the computer interpolates the missing contours of this object on every frame automatically. Typical active contour (snake) model is adapted and the contour interpolation problem is decomposed into two directional contour tracking problems and a merging problem. In addition, new user interaction models are created for the user to interact with the computer. Experiments indicate that this system offers a good balance between algorithm complexity and user interaction efficiency.

Key words: snake, DP, graph search, Interactive Rubber, authoring tool, MPEG-4.

1. INTRODUCTION

To segment and track semantically meaningful video objects (VOs) through a video sequence is currently an interesting topic. The need of video objects arises from two major multimedia applications. One of them is extended video annotation on traditional video. At the editing stage, video objects are specified and annotated with hyperlinks. When the user clicks on the object in any frame that it appears, the annotated data pops up. This is similar to the hyperlink mechanism in HTML. Because video is becoming rapidly a popular media form, video object annotation is also becoming a very important topic in multimedia authoring research. Actually, several authoring systems are already available that support video object annotation, such as IBM's HotVideo [1] and HyperVideo from Veon [2]. In addition, video object annotation is also an important technique for interactive TV, in which hot links are embedded along with video information. The other application associated with video objects is the new object oriented video representation technique, which is specified within the framework of MPEG-4 and upcoming MPEG-7 standards. In the terminology of MPEG-4, video objects are segmented from traditional video. Each video object is compressed and decompressed individually. The composition of multiple video objects is coordinated by MPEG-4 Systems. Though these two applications are different, they have the common problem at the authoring stage, that is, how to specify video objects from large amount of video data in both spatial and temporal domains.

Segmenting objects from image and/or video data has been under intensive research. Numerous algorithms are available in literature. However, it is becoming widely accepted recently that fully automatic segmentation is difficult. Instead, a semi-automatic approach is a more feasible solution. Several papers have been published according to this idea, for example [3], [4], [5], [6], etc. These papers all sidestepped full automation and designed tracking algorithms to work along with user interaction. However, though their contribution to the "tracking" (algorithm) part of the problem is different, their user interaction models are all very simple, i.e., the user defines a VO in the first frame, and then lets the computer track the VO temporally. This model cannot meet the requirements of interactive authoring tools. For example, a common problem is that no quality criteria are 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 it necessary.

In this paper, we propose a new "interpolation" approach for semi-automatic video object segmentation, and discuss an authoring tool prototype design based on this approach. Our focus is to design the algorithm and the user interaction models at the same time, which helps to solve the efficiency problem in interactive authoring systems. In our system, the user defines a video object by specifying its contour on multiple anchor

[*]For better view of color pictures, ps file downloadable at http://www.ctr.columbia.edu/~luoht/research/paper/acm99.ps.gz

frames rather than only on the first frame. The computer then uses input information from multiple anchor frames to "interpolate" the VO contours on every frame. Compared with the pure tracking approach, the interpolation approach offers more "interaction points" between the algorithm and the user. From the algorithm's point of view, the algorithm makes use of user input more efficiently, because each user input contour contributes to VO definitions on those frames before as well as after it, while in the tracking case, a user input only influences the frames after it. From the user's point of view, the new approach makes the system performance more predictable because the user can define a VO on frames where large occlusion or motion occurs and most tracking algorithms are likely to fail. In addition, it is more controllable in that with two or more input VO contours, the possible interpolated contours can be effectively limited to certain searching region.

More specifically, the problem can be defined 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 frames i, $i = b+1, b+2, \cdots, e-1$. We call it a "contour interpolation" problem. As a comparison, we express the "contour tracking" problem as: given an 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 usage 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, we use active contour models (snakes) [7] in our interpolation algorithm. In a snake model, a planar curve is parameterized with nodes and local energy functions are defined for each node. The final shape and position of the curve is determined by the global minimization of the snake's energy. In our work, a traditional snake model is extended in the following ways. First, we use nodes to represent snakes and we design a "contour matching" algorithm to match the node-representations of two user input contours. Based on contour matching, contour temporal smoothness and shape similarity criteria are defined. Later discussion will show that these criteria are essential for merging 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 edge noises during the active contour tracking course.

The algorithm design in this work is directly influenced by the work [8], in which the idea of contour matching was first proposed. In our work, we further develop their contour matching algorithm by introducing different local motion models. Based on the contour matching, the interpolation problem is then modeled as a bi-directional snake tracking stage and a merging stage. In addition, similar work on spatial temporal active snake model can be found in [9], [10], [11] etc. In [9], efforts were made to combine the active contour model with motion estimation. Quality criteria were also suggested to evaluate the quality of contour tracking. However, their experimental results were not good because no node neighborhood information was used. [11] used similar snake technique to track video object contours for annotation. [10] tried to solve the occlusion problem in active contour tracking by segmenting the contours into multiple segments. They used neighborhood information for motion estimation and got better results than [9]. We will later show in this paper that their work can be included in ours as a specific case of our parametric template.

The user interface model in this work is related to the work in [12]. In their system, the user selects key points and the computer grows the corresponding contour segment that links the key points. The current key point is moved by the user until the contour segment grown by the computer is desirable. This process involves both the computer's searching as well as the user's decision. It is an efficient model for user/machine interaction. In our system, an improved active searching mechanism (*interactive rubber*) is designed for the user to define the initial VO contours on anchor frames. In addition, an *iterative interpolation* mechanism is designed for the user to offer the error feedback to the interpolation algorithm.

This paper is organized as follows. In Section 2, we introduce the contour representation and contour matching algorithm. In Section 3, we discuss in detail our contour interpolation algorithm based on contour matching. Then in Section 4 we summarize the user interaction model in several stages of the algorithm. Experimental results are presented in Section 5, and in Section 6, we present some concluding remarks.

2. CONTOUR REPRESENTATION AND MATCHING

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 contour pixels. The spatial location of each contour pixel is denoted by the 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, a uniform subsampling function is defined as $S_k(i) = i \cdot unit$, where *unit* is the subsampling unit. In this paper, we name $\{\mathbf{v}_{s,k}\}$ the *pixel representation* and $\{\mathbf{v}_{S_k(i),k}\}$ the *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). This is essential for interpolating the object contours between them. In the pixel representation, the length of C_b and C_e is not necessarily the same and pixel by pixel correspondence can not be created. Therefore, we use subsampled node representation for the contour interpolation study and contour matching algorithms are used to find the correspondence between the two subsampled contours.

Mathematically, the matching process can be defined as follows. Given two input contours: $C_b = \{\mathbf{v}_{s,b}\}$, $C_e = \{\mathbf{v}_{s,e}\}$ in the 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 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 (Other subsampling algorithms are also possible, see [13] for a detailed discussion), 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 [8]. 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. This assumption is true in general if the motion of the video object is not too fast in relation to the temporal distance between the two frames on which the contours C_b and C_e are defined. Two rigid motion models, i.e., translation and affine, are possible for local matching energy definition.

Translation Motion Model: If we denote the motion vector between 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 mapping function as $\mathcal{A}_i = \text{Affine}_i(MV_{i-1}, MV_i, MV_{i+1})$. Under this function, the local contour segment $\{\mathbf{v}_{s,b}\}$, $s \in [S_b(i-1), S_b(i+1))$ is projected to a 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 the operator $D(\cdot)$ is the distance measure of two sets, which we define as

$$D(A, B) = \Big[\sum_{\mathbf{v}_i \in B} \min_{\mathbf{v}_j \in A} ||\mathbf{v}_i - \mathbf{v}_j|| + \sum_{\mathbf{v}_j \in A} \min_{\mathbf{v}_i \in B} ||\mathbf{v}_j - \mathbf{v}_i||\Big]/2. \quad (1)$$

In practice, Eq. (1) can be implemented in an iterative manner for each pixel. Here we define $d(\mathbf{v}_i, A) = \min_{\mathbf{v}_j \in A} ||\mathbf{v}_i - \mathbf{v}_j||$, and denote $d_n(\mathbf{v}_i, A)$ as the n-th value for $d(\mathbf{v}_i, A)$ in the iteration. At the initialization stage,

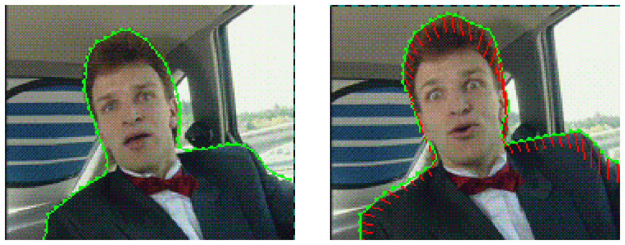
$$d_0(\mathbf{v}_i, A) = \begin{cases} 0 & \text{if } \mathbf{v}_i \in A \\ +\infty & \text{otherwise} \end{cases}.$$

The iteration is then defined as

$$d_{n+1}(\mathbf{v}_i, A) = \min_{\mathbf{v}_j \in N(\mathbf{v}_i)} \Big(d_n(\mathbf{v}_j, A) + ||\mathbf{v}_i - \mathbf{v}_j||\Big),$$

where $N(\mathbf{v}_i)$ is the 8-neighborhood set of pixel $\mathbf{v}_i$.

With the definition of local matching energy terms, the matching problem is converted to an energy minimization problem. This can be easily solved by the DP algorithm [14], which we do not discuss in detail here. In practice, the affine model is more complex than the translation model, but the quality is better, especially when the subsampling distance between neighboring nodes is big. In Figure 1, we show a result of contour matching under the translation model. Figure 1(a) is the 50th, 1(b) is the 70th frame of the Carphone sequence. Green lines are the contours and red lines link the matched node pairs.



(a) (b)

Figure 1: Results of contour matching for the Carphone sequence. (a) is the 50th and (b) is the 70th frame of the Carphone sequence. Green lines are the user specified contours and the red lines link the matched node pairs.

3. CONTOUR INTERPOLATION ALGORITHM

3.1. Localized Energy Minimization Model

The essence of Kass' snake model [7] 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 2-dimensional spatial snake to 3-dimensional spatial/temporal snake, it is possible to extend the local energy term from 2D to 3D as well. In this work, we study the extended local energy term as the intraframe energy and the interframe energy respectively. From now on, because no subsampling issue will be involved, snakes are assumed to be in the node representation. The node representation notation $\{\mathbf{v}_{S_k(i),k}\}$ is simplified to $\{\mathbf{v}_{i,k}\}$ whenever possible.

Intraframe Energy: As usual, the intraframe energy includes two terms, a gradient term and a smoothness term:

$$E_{intra,i} = \eta_{edge} E_{gradient,i} + \eta_{smooth} E_{smooth,i}. \quad (2)$$

In this equation, the first gradient term is defined as

$$E_{gradient,i} = \int_{s=S(i-1)}^{S(i)} \frac{255}{(10 + || \bigtriangledown (\mathbf{c}(\mathbf{v}_s))||)} ds,$$

in which 255 and 10 are two empirical values. $\mathbf{c}(\mathbf{v})$ is the color vector of node $\mathbf{v}$. The second smoothness term is defined as

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

which is designed to eliminate the influence of the node distance on the contour smoothness measure.

Interframe Energy: In available papers on temporal active contour tracking [9, 10], generally employed interframe energy terms include optical flow, motion smoothness, interframe color, etc. A basic problem in these approaches is that most of their node energy definitions are based only on the image feature at the node's position rather than on its neighborhood. This makes the color and especially motion information generally not accurate. Ideally, the contour nodes' neighborhood should be observed in order to track them from frame to frame. However, a problem here is, different from typical point tracking problem, contour nodes are most likely 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 the definition of 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

$$\text{Diff}_c(\mathbf{v}_{i,k}, \mathbf{v}_{i,k-1}, T_{i,k}) = \sum_{\mathbf{dv}_j \in T_{i,k}} w_j ||\mathbf{c}(\mathbf{v}_{i,k} + \mathbf{dv}_j) - \mathbf{c}(\mathbf{v}_{i,k-1} + \mathbf{dv}_j)||, \quad (3)$$

where $\mathbf{c}(\mathbf{v})$ is the color vector of node $\mathbf{v}$. In addition, we define the motion vector at the node $\mathbf{v}_{s,k}$ as

$$MV(\mathbf{v}_{i,k}, T_{i,k}) = \sum_{\mathbf{dv}_j \in T_{i,k}} w_j \cdot MV^{(p)}(\mathbf{v}_{i,k} + \mathbf{dv}_j) / \sum_{\mathbf{dv}_j \in T_{i,k}} w_j, \quad (4)$$

where $MV^{(p)}(\mathbf{v}_{i,k})$ is the estimated motion vector at the $\mathbf{v}_{i,k}$. 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 Figure 2. For the occlusion case as was discussed in [10], we can easily switch the weights by setting inside weights to 0 and outside weights to 1. In general cases, both the size and the weights of the parametric template can be adjusted flexibly to get the best results of the contour node tracking.

Based on the parametric template, the interframe energy terms for each contour node are enumerated 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. (3).

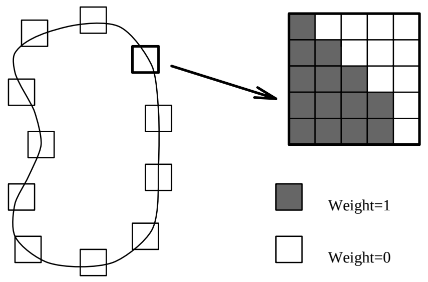


Figure 2: Illustration of parametric template concept. During the tracking course, both the size and weights of the template can be adapted.

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. (4) and we assume forward motion estimation is used. The second term is $MV_{i,k-1} = (\mathbf{v}_{i,k} - \mathbf{v}_{i,k-1})$.

3. Motion smoothness: $E_{motion,i,k} = ||MV_{i-1,k} + MV_{i+1,k} - 2MV_{i,k}||$. This is a smoothness measure for motion vectors on spatially neighboring nodes. It is accurate for local motion in the translation form.

4. Shape stiffness: $E_{shape,i,k} = ||\text{angle}(\mathbf{v}_{i-1,k-1}, \mathbf{v}_{i,k-1}, \mathbf{v}_{i+1,k-1}) - \text{angle}(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k})||$, where $\text{angle}(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$ is the angle based on the three spatially neighboring nodes. This term measures the local shape similarity. It is accurate if the local motion is in the rotation form.

5. Temporal smoothness: $E_{temporal,i,k} = ||\mathbf{v}_{i,k-1} + \mathbf{v}_{i,k+1} - 2\mathbf{v}_{i,k}||$. This is the smoothness measure for the three temporally neighboring nodes.

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 $\{\mathbf{v}_{s,k}\}$, $(k = b+1, \cdots, e-1, s = 0, 1, \cdots, N_s)$, that minimize the global energy $\sum_{s,k}(E_{inter,s,k} + E_{intra,s,k})$.

Though it is natural to extend the local energy terms from 2D to 3D, the increase in computational 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 $\mathbf{v}_{s,k}$ is defined in relation to eight neighbors! This change is illustrated in Figure 3. If each node has a search region of n, the local searching complexity then 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.

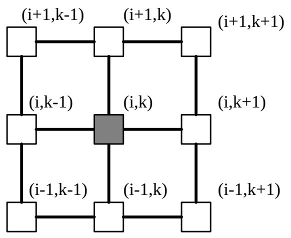


Figure 3: Spatial temporal neighborhood of a contour node for the local energy definition. In the figure, i is the spatial index and k is the temporal index.

In this work, a sub-optimal solution is obtained 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 neighborhood definition of the current pixel (i, k) cannot include nodes in a neighboring frame that has not been processed. Therefore, we shift the neighborhood definition in Figure 3 by one frame. For example in the forward tracking model, if the current node is (i, k) in Figure 3, then its eight nodes are $(i+1, p), (i, p), (i-1, p)$, $p = k-2, k-1, k$ (not including (i, k)). Among them, six are already fixed and only two variable nodes $(i+1, k+1)$ and $(i-1, k+1)$ will influence its local energy. This is the same as the case with the intraframe energy $E_{intra,s}$. Therefore, it is easy to design a search algorithm with DP. After the bi-directional tracking, another search process is used to find the optimal contours out of the previous tracking results.

3.2. Bi-directional Tracking

In the tracking model, each node $\mathbf{v}_{i,k}$'s total node energy can be written as

$$E_{total,i,k} = f_E(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k}) \tag{5}$$

because the other six neighboring nodes $\mathbf{v}_{i+p,k-2}$, $\mathbf{v}_{i+p,k-1}$, $p = (-1, 0, 1)$ in the previous frames are all fixed. This energy expression is similar to those used for 2D active contour models, except that the detailed expression of f_E is different. Therefore, we use the DP algorithm similar to that used in [14].

Limited Search Region v.s. Searching Complexity According to [14], the local energy expression in Eq. (5) is a second order expression in the sense that it includes two neighboring nodes as variables. In this case, a two-element vector $(\mathbf{v}_{i+1}, \mathbf{v}_i)$ is used as the status index for DP. The DP search is then carried out as follows

$$S_i(\mathbf{v}_{i+1}, \mathbf{v}_i) = \min_{\mathbf{v}_{i-1}} \Big[S_{i-1}(\mathbf{v}_i, \mathbf{v}_{i-1}) + f_E(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1}) \Big].$$

Note that the frame index k is omitted in the above expression because no temporal information is involved. If the search size for each node is n and the total number of nodes in each contour is m, the complexity of DP is $O(mn^3)$. Obviously, the search size n is the important factor in the overall complexity and should be limited as much as possible.

Two clues are used to limit the search size in our work. First at the global level, a search stripe can be created for each matched node pair. This is illustrated in Figure 4. For ease of discussion, the temporal orbit of the matched node is mapped into one frame. On this frame, a search stripe is defined. Please note that the definition of search stripe is different according to different spatial location of the matched node pair. In Figure 4(a), and 4(b), two basic types of search stripe definitions are depicted. The orientation of width and height are differently defined as well based on the different orientation of the matched node pairs. In practice, both the width and height of the global search stripe can be determined by the user in an interactive way, according to the motion of video object. That is, if the motion is more like a pure translation, the height of the stripe $S^{(G)}_{height}$ may be reduced, otherwise it is increased. The bottom line is that the global search region should be at least big enough to include the temporal orbit of the node's motion. In addition, in the case of large search stripe that exceeds a certain threshold, the global search stripe is further re-sampled to reduce the overall search size. The two axes "x" and "y" used for resampling are marked in Figure 4(a) and 4(b). Please note that the parallelogram search region definition is more efficient than a strictly rectangular one in the computational sense, while without much performance degradation.

Once the global stripes are defined for node pairs, the tracking of nodes is constrained within their stripes on every frame. In addition, at the local level, the local search region is further determined frame by frame by the forward motion vector at the current frame. In our work, the determination of $S^{(L)}_{height}$ and $S^{(L)}_{width}$ is based on $S^{(G)}_{height}$ and $S^{(G)}_{width}$, and the local search is carried out on top of the re-sampled grids created for the global search stripe, i.e. the re-sampled grids along the "x" and "y" axes.

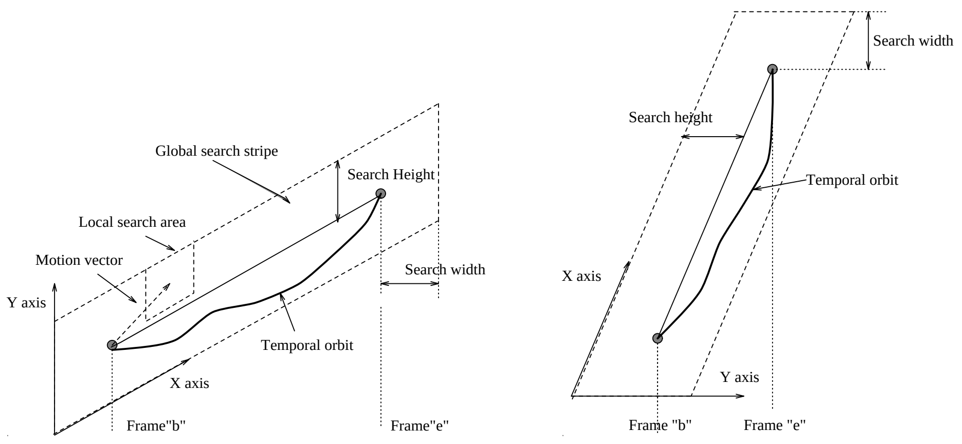
In above two clues, the global stripe limits the possible location of the local search region. Note that in the pure "interpolation" case, both the width and height of the global search stripe is zero. Therefore, the global search stripe is actually a generalization of interpolation.

Closed Contour Problem Until now, the discussed contours C_i are all by default open. In practice when contours are constrained to be closed, the minimization search used for tracking purpose can be approximated with two-pass open contour searching. That is, for a closed contour $C_i, i = 0, 1, 2, \cdots, n$, node $C_0 = C_n$, its minimization status can be found as follows. First break the closed contour at node C_0, use previous algorithm to process open contour $C_i, i = 0, 1, 2, \cdots, n-1$, which produces a temporary contour C'_i. Second, close C'_i by setting $C'_n = C'_0$ and then break it half way at node $C'_{(n/2)}$, this produces an open contour C''_i. C''_i is further processed with the open contour algorithm and the final result is obtained.

3.3. Merging of Multiple Results

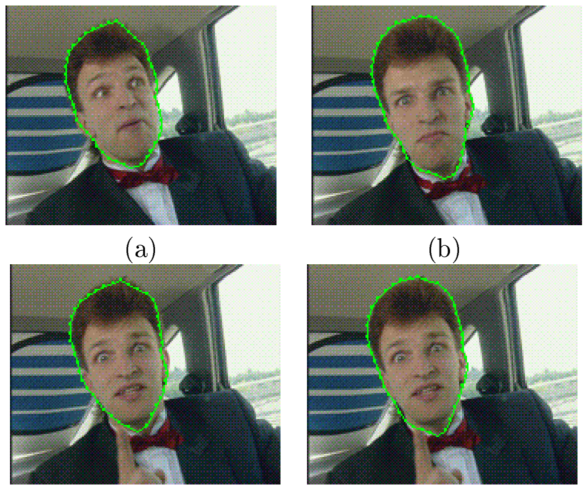
Though the bi-directional tracking approach reduces the searching complexity, its limitation is that it only makes use of user input in one frame (either C_b or C_e) at a time. Due to the error accumulation, the tracked contour C_k always degrades when k approaches e, if tracked from C_b to C_e, and vice versa. Actually we have observed that if the object's motion involves self-occlusion and/or uncovering, sometimes it is very difficult for the active contour model to track its contour in one direction, but easy to do it in the other direction. Figure 5 is such an example. In Figure 5, (a), (b) are the user defined VO on the 118th and the 132nd frame of the Carphone sequence. From frame 118 to frame 132, the man's left ear was uncovered because of the rotation of his head. If we track the object contour forward, i.e. from the 118th frame to the 132nd frame, the uncovered ear was not included as part of the video object. This is depicted in Figure 5(c). On the other hand, if we track the contour backward, i.e. from the 132nd frame to 118th frame, the motion is reversed and the uncovering motion of left ear changes to occlusion, which is easy for the active contour tracking algorithm to handle. The result of backward tracking for frame 125 is shown in Figure 5(d). Note that the left ear is correctly included as part of the video object.

Therefore, a good algorithm to merge the results from the two tracking processes is important. In this



(a) (b)

Figure 4: Global and local search region limitation. The two grayed circles represent the matched node pair. The size of the global search region is determined by "search height" and "search width", while the local search region is determined by motion estimation.



(a) (b) (c) (d)

Figure 5: Illustration of the necessity of bi-directional tracking and result merging. (a), (b) are the user defined VO on the 118th and the 132nd frame of the Carphone sequence. (c) is the tracked object contour on the 125th frame by forward tracking. (d) is the tracked object contour on the 125th frame by backward tracking.

work, an efficient DP algorithm is designed for the merging job. The problem may be defined 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 criteria* as the final output of contour interpolation. In the terminology of DP, we can say that the target is to find an optimal path from C_b to C_e that maximize the merit criteria.

In this work, the *merit criteria* for each candidate contour include two terms: a temporal smoothness energy term E_T, and a shape merit energy term E_S, i.e.

$$E_{merge}(C_k) = \eta_T \cdot E_T(C_k) + \eta_S \cdot E_S(C_k). \qquad (6)$$

The first term is defined in a localized form:

$$E_T(C_k) = \sum_{i=0}^{i=N_s} ||\mathbf{v}_{i,k-1} + \mathbf{v}_{i,k+1} - 2\mathbf{v}_{i,k}||, \qquad (7)$$

where N_s is the number of nodes in each contour. Note that $E_T(C_k)$ is different from previously defined $E_{temporal}$ in that $E_T(C_k)$ is defined for each contour while $E_{temporal}$ is defined for each contour node.

The second term of Eq. (6) is based on the shape similarity comparison between two contour pairs: (C_b, C_k) and (C_k, C_e). If we denote the shape similarity measure of two contours C_k and C_l as shape(C_k, C_l), then the E_S term can be written as

$$E_S(C_k) = \Big[w_1(k)\cdot\text{shape}(C_b, C_k) + w_2(k)\cdot\text{shape}(C_e, C_k)\Big], \qquad (8)$$

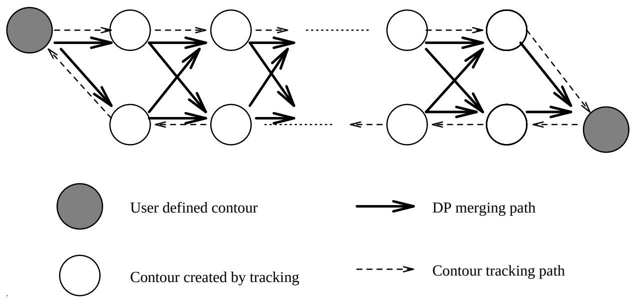


Figure 6: Illustration of DP approach for merging the bi-directional contour tracking. Each circle represents a contour $C_k^{(d)}$, DP searching is used to find an optimal path in the temporal direction that has the best merit quality.

where $w_1(\cdot)$ and $w_2(\cdot)$ are two weighting functions. They are designed as a linear function of frame indices k, b, and e. In addition, the shape similarity measure used here can be defined based on two interframe node energy terms E_{shape} and E_{motion}, as discussed in Section 3.1. At the contour level, this expression is defined as

$$\text{shape}(C_k, C_l) = \sum_{i=0}^{i=N_s} \eta_1[(\mathbf{v}_{i,k} - \mathbf{v}_{i,l}) - (\mathbf{v}_{i-1,k} - \mathbf{v}_{i-1,l})]$$

$$+\eta_2[\text{angle}(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k}) - \text{angle}(\mathbf{v}_{i-1,l}, \mathbf{v}_{i,l}, \mathbf{v}_{i+1,l})].$$

In the above Equations (6)-(8), $E_{merge}(C_k)$ is actually defined in relation to three contours: C_{k-1}, C_k and C_{k+1}. Therefore, it is easy to solve the minimization problem with DP. Here Figure 6 is used to illustrate the DP based merging algorithm. In Figure 6, each circle represents a contour $C_k^{(d)}$, DP searching is used to find an optimal path in the temporal direction that has the best merit quality, i.e., in the sense of temporal smoothness and shape similarity.

4. USER INTERACTION MODEL

In a typical semi-automatic system, the user's role includes two important functions. One is to give the initial data for the computer to begin the computation, the other is to correct the computer's errors. In our system, these two functions are handled by *Interactive Rubber* and *Iterative Interpolation* respectively.

4.1. Interactive Rubber

In available image authoring systems, two types of solutions are common for a user to specify an object contour in an image (or a video frame). In one of them,

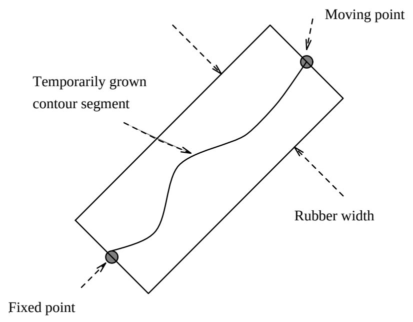


Figure 7: Illustration of active rubber. The containing rectangle is determined by two points: a fixed point and a moving point, and the rubber width. The width of the rubber is adjustable by the user.

the user specifies every point of the contour, while the computer does nothing but record the positions of each mouse click and links the positions with line segments. Typical examples include the polyline drawing in XFIG and free style drawing in PHOTOSHOP. This type of solution gives the user full control of the shape and position of the contour, but ignores the computational power of the computer. Obviously, it is tedious to input an accurate contour point by point. On the other hand, in the other type of solutions, the image is modeled as grids and the contour as paths linking the grids. The user has only to select a starting point and an ending point of a contour segment, the computer finds the whole segment by searching the minimal cost path that links the two points. This can be done with either dynamic programming or graph searching algorithms such as Dijkstra's algorithm [15]. Publications belonging to this type of solution include [12], [16], [17], etc. Compared with the first type of solutions, this type of approach relieves the user's labor by introducing computer searching during the interaction. However, its problem is that the user has less control on the contour. Sometime when the gradient information within the image is complex, an intended contour segment may be attracted to an erroneous strong neighboring edge, which is totally undesirable. In addition, the "Active-Scissors" approach defined in [12] requires the computer to calculate the optimal path to every pixel within the image every time a new contour point position is chosen by the user. This is not efficient if the size of image is big and real time performance is hard to achieve.

To overcome the problems while retaining the benefits of above two groups of solutions, we design an *Interactive Rubber* tool, which is a hybrid of the two of them.

An Interactive Rubber is in principle a dynamic graph searching edge detection algorithm that is similar to the second of the above mentioned types of solutions. The difference is that it comes with an adjustable containing rectangle that limits the range of graph searching. This is illustrated in Figure 7. The user moves the current moving point, which, together with the fixed point, determines a containing rectangle. Graph searching is then carried out within the rectangle and a contour segment is grown to link these two points. Compared with the Active-Scissors approach in [12], Interactive Rubber is more efficient because it limits the graph searching range to the neighborhood of the desirable contour segment. Moreover, the user can control the result of graph searching by adjusting the width of the rectangle, e.g. if there is strong noisy edges in the neighborhood, the user may get rid of them by narrowing the containing rectangle. In the extreme case, the width can be set to zero, then the Interactive Rubber reduces to above type one solution. In this sense, the Interactive Rubber is a generalization of the previous type one and type two solutions.

4.2. Iterative Interpolation

Though in experiments the discussed interpolation algorithm showed good performance, error is inevitable in practice, especially when the two anchor frames on which the user specifies object contours C_b and C_e are far away in the temporal direction. Iterative interpolation is found to be a good solution to this problem.

In general, the reasons for errors in interpolation are complex. However, in the bi-directional tracking based interpolation algorithm, a video object contour can always be reliably tracked from C_b to a certain $C_{(b+t_1)}$, and from C_e to a C_{e-t_2}, where $t_1 > 0$ and $t_2 > 0$. If $b + t_1$ turns out equal to $e - t_2$, the problem is solved. Otherwise, we can move the C_b to $C_{(b+t_1)}$, and C_e to C_{e-t_2}, and begin the interpolation again. In this way, the interpolation is done iteratively until the two contours C_b and C_e converge. This process can be better illustrated in Figure 8. In Figure 8, points $P(1)$ and $P(2)$ are a matched node pair on contours C_b and C_e. After the first round of interpolation, point $P(1)$ is correctly tracked to $P(3)$ and $P(2)$ to $P(4)$. At this stage, the user changes the C_b to the temporal position at $P(3)$ and C_e to the position at $P(4)$, sets the global search parameters accordingly, and begins the interpolation again. As depicted in the figure, the global search area in the second round for point pair $P(3) - P(4)$ is much smaller than that of $P(1) - P(2)$. This is an important factor that helps the iterative interpolation process converge.

In Figure 9, a practical example is given on the Foreman sequence to show how the iterative interpolation works. In the first interpolation round, the user specifies C_b on frame 50 and C_e on frame 100, the global searching parameters are $height = 15$(pixels), $width = 4$(pixels). Subfigures (a) to (e) are results on frames 80, 84, 87, 90, 94 in this round. Obviously, the tracking result is poor from frame 80 to 94, mainly because a strong neighboring edge has attracted the snake erroneously (due to space limits, other frames are not included in the figure). To overcome the error, the user moves C_b to frame 80 and C_e to frame 94, and change the global searching parameters to $height = 4$, $width = 4$. Based on the contours on frame 80 and 94, the results after the new interpolation round on frames 84, 87, 90 are shown in subfigures (f), (g), (h), respectively. Obviously, the second interpolation round improves the accuracy of tracked contours if we compare the contours in subfigures (b), (c), (d) with those in (f), (g), (h). It is worth noticing that in the second round of interpolation, the user does not have to tell the computer laboriously what exactly a correct contour is. Instead, the user just chooses new anchor frames on which the tracked contours are correct, and changes the global searching parameters accordingly (limits the global searching area as much as possible). It is the computer's work to do the interpolation again based on new user input information!

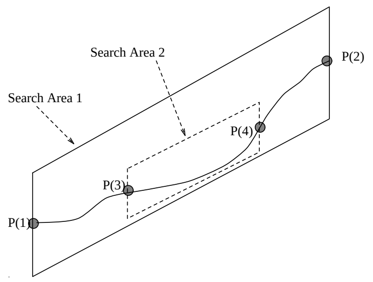


Figure 8: Illustration of iterative interpolation. Point $P(1)$ and $P(2)$ are a matched node pair on initial contours C_b and C_e. After the first round of interpolation, point $P(1)$ is correctly tracked to $P(3)$ and $P(2)$ to $P(4)$. Their corresponding contours are used as new C_b and C_e and the interpolation is done iteratively until converges.

5. EXPERIMENTS

The discussed video object annotation system was implemented on PCs running Windows 95. Experiments were carried out over MPEG-4 testing video sequences as well as sequences from a library used by Columbia's VideoQ[1] system.

[1]http://www.ctr.columbia.edu/VideoQ

First, we compared the effect of parametric template for active contour tracking on the Carphone sequence. In the experiment, single direction forward tracking was used. In Figure 10, the left column is the tracking result without and the right column is the result with the parametric template. The first row is the beginning user input contour in frame 0, the following two rows are the tracked results for the 43rd and 47th frames. Obviously, the parametric template improves the tracking robustness in complex boundary conditions.

Figure 11 shows the results of tracking result merging on the Mother-Daughter sequence. The user defined VO contours on frame 75 and 180. The first row is the result of backward tracking from frame 180 to frame 75, and the second row is the interpolation results that merged bi-directional trackings. The frame numbers, from left to right, 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 the space limit). That is, the merged interpolation results are better than tracking results on either direction.

Figure 12 is a fully finished segmentation result on the first 100 frames of the Foreman sequence. Subfigures (a)-(h) are the produced contours on frame 10, 20, 30, 40, 60, 70, 80, 90. To finish the segmentation, the user specified three initial contours on frame 0, 50 and 100. One additional iteration was involved in the interpolation between frame 0 and frame 50, and frame 50 and frame 100, respectively.

In practice, the computational complexity of the algorithm depends on the size of the searching area and the complexity of the contours. In our experiment, a 200-MHz Pentium was used, the average speed of the interpolation algorithm was about 0.01 second/node and/or 0.6 second/frame (tested on several MPEG-4 sequences in QCIF size).

6. CONCLUDING REMARKS

In this paper, an interactive authoring system is designed for video object segmentation and annotation. This system features a new contour interpolation algorithm, which makes better use of user inputs and has more stable performance than traditional single directional tracking algorithms. In addition, efficient user interaction models are built for both initial data input and machine error feedback. Our experiments show that this prototype system works efficiently both from the machine's and the user's point of view, in that it elegantly balances the use's decision making capability

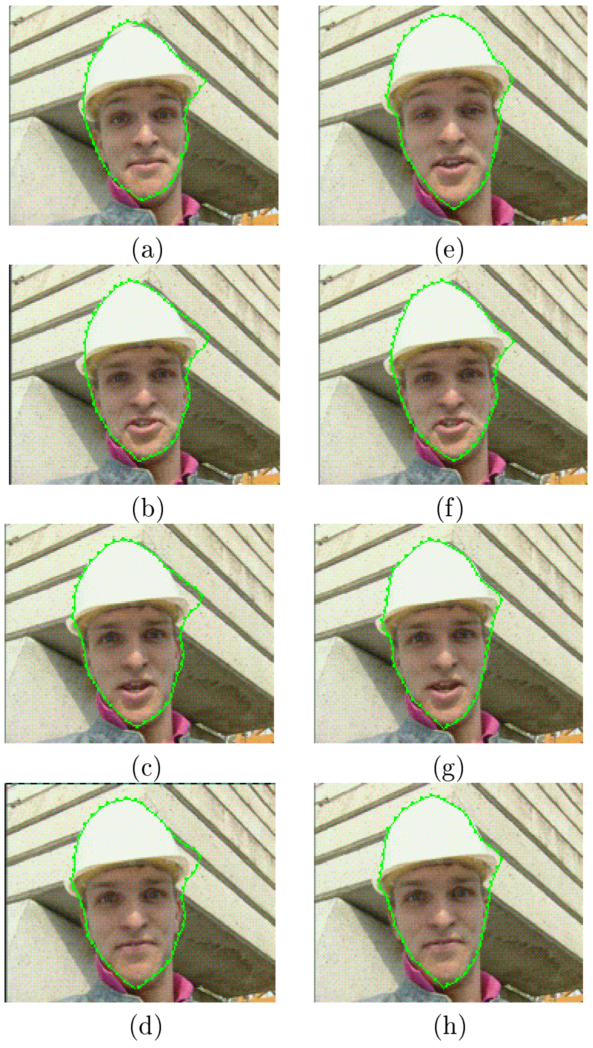


Figure 9: An example of iterative interpolation on the Foreman sequence. In the first round, the user specifies C_b at frame 50 and C_e at frame 100, the global search parameters are $height = 15$(pixels), $width = 4$(pixels). (a) to (e) are results on frames 80, 84, 87, 90, 94 at this round. In the second round, previous contour result on frame 80 is used as C_b and contour result on frame 94 is used as C_e, the global search parameters are reduced to be $height = 4$, $width = 4$. The new interpolation results on frame 84, 87, 90 are showed in (f), (g), (h), respectively.

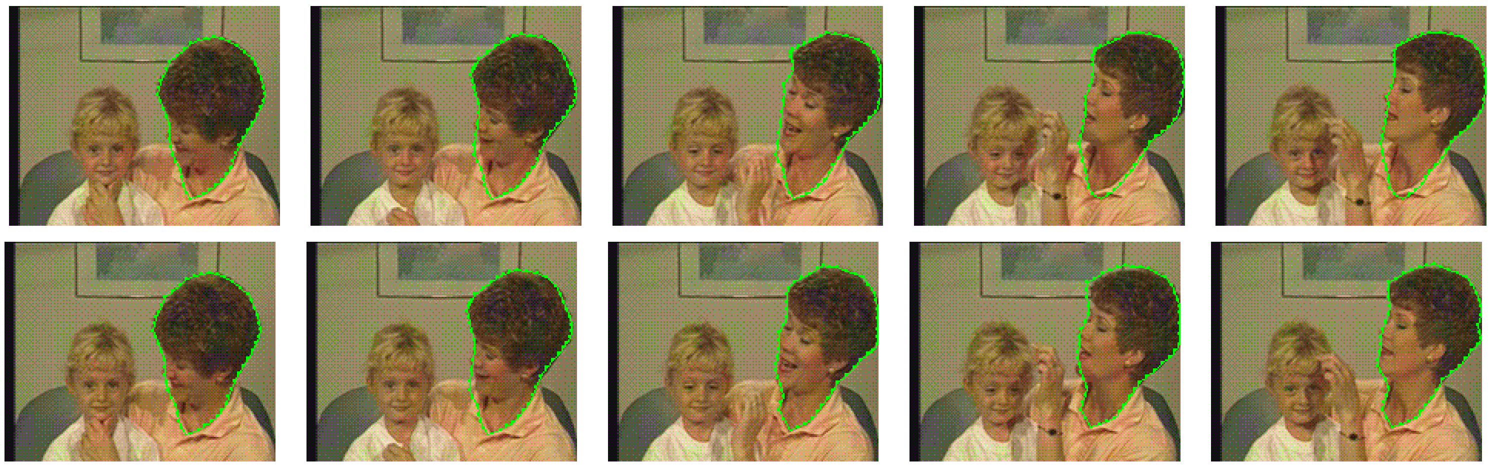


Figure 11: Illustration of tracking results merging. The user defined VO contours on frame 75 and 180. The first row is the result of backward tracking from frame 180 to frame 75, and the second row is the interpolation results that merged bi-directional trackings. The frame numbers, from left to right, are 170, 160, 150, 130 and 90.

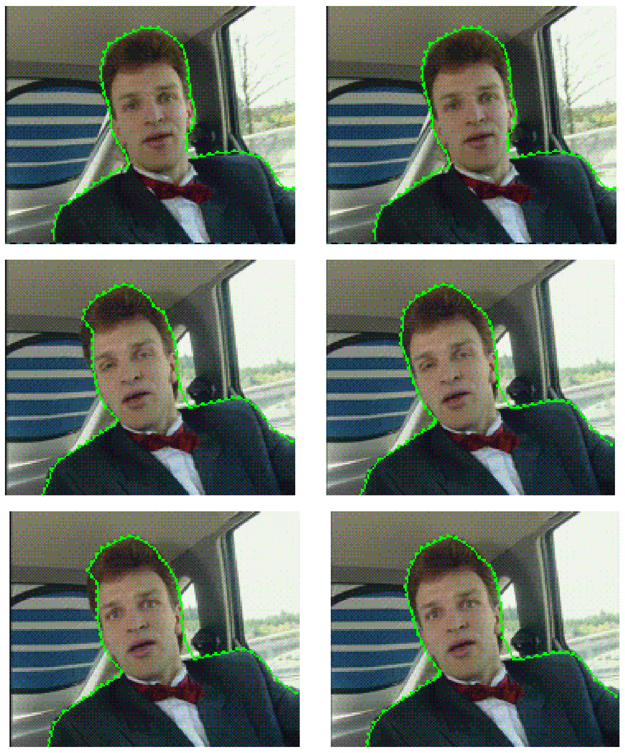


Figure 10: Comparison of the effect of parametric template on active contour tracking. The left column is the tracking result without and the right column is the result with parametric template. First row is the beginning user input contour in frame 0, the following two rows are the tracked results for the 43rd and 47th frames.

with the computer's processing and searching power.

7. REFERENCES

[1] IBM HotVideo website, "http://www.software.ibm.com/net.media/hotvideo/index.html".

[2] Veon website, "http://www.veon.com/".

[3] E. Chalom and V.M. Bove Jr., "Segmentation of an image sequence using multi-dimensional image attributes," in *Proc. IEEE Int. Conf. Image Processing*, Lausanne, Sept. 1996.

[4] P. Correia and F. Pereira, "The role of analysis in content-based video coding and indexing," *Signal Processing*, vol. 66, pp. 125–142, 1998.

[5] D. Zhong and S.F. Chang, "AMOS: an active system for MPEG-4 video object segmentation," in *Proc. IEEE Int. Conf. Image Processing*, Chicago, Oct. 1998.

[6] C. Gu and M.C. Lee, "Semantic video object tracking using region-based classification," in *Proc. IEEE Int. Conf. Image Processing*, Chicago, Oct. 1998.

[7] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Comput. Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[8] D. Geiger, A. Gupta, L.A. Costa, and J. Vlontzos, "Dynamic programming for detecting, tracking and matching deformable contours," *IEEE Trans. PAMI*, vol. 17, no. 3, pp. 294–302, 1995.

[9] Y.T. Lin and Y.L. Chang, “Tracking deformable objects with the active contour model,” in *IEEE Int. Conf. On Multimedia Computing and Systems*, Ottawa, Canada, June 1997.

[10] Y. Fu, A.T. Erdem, and A.M. Tekalp, “Occlusion adaptive motion snake,” in *Proc. IEEE Int. Conf. Image Processing*, Chicago, Oct. 1998.

[11] T.C Chiueh, T. Mitra, and C.K. Yang, “Zodiac: a history-based interactive video authoring system,” in *ACM Int. Multimedia Conf.*, Bristol, England, Sept. 1998.

[12] E.N. Mortensen and W.A. Barrett, “Interactive segmentation with intelligent scissors,” *Graphical Models and Image Processing*, vol. 60, pp. 349–384, 1998.

[13] K. Sobottka and I. Pitas, “Segmentation and tracking of faces in color images,” in *Proceedings of 2nd Intl. Conf. Automatic Face and Gesture Recog.*, Killington, Vermont, 1996, pp. 236–241.

[14] A.A. Amini, T.E. Weymouth, and R.C. Jain, “Using dynamic programming for solving variational problems in vision,” *IEEE Trans. PAMI*, vol. 12, no. 9, pp. 885–867, 1990.

[15] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, chapter 25.2, MIT Press, 1990.

[16] A. Martelli, “A heuristic search methods to edge and contour detection,” *Commun. ACM*, vol. 19, no. 2, pp. 73–83, 1976.

[17] E. Mortensen and W. Barrett, “Intelligent scissors for image composition,” in *Proceedings of ACM SIGGRAPH 95*, Los Angeles, CA, 1995, pp. 191–198.

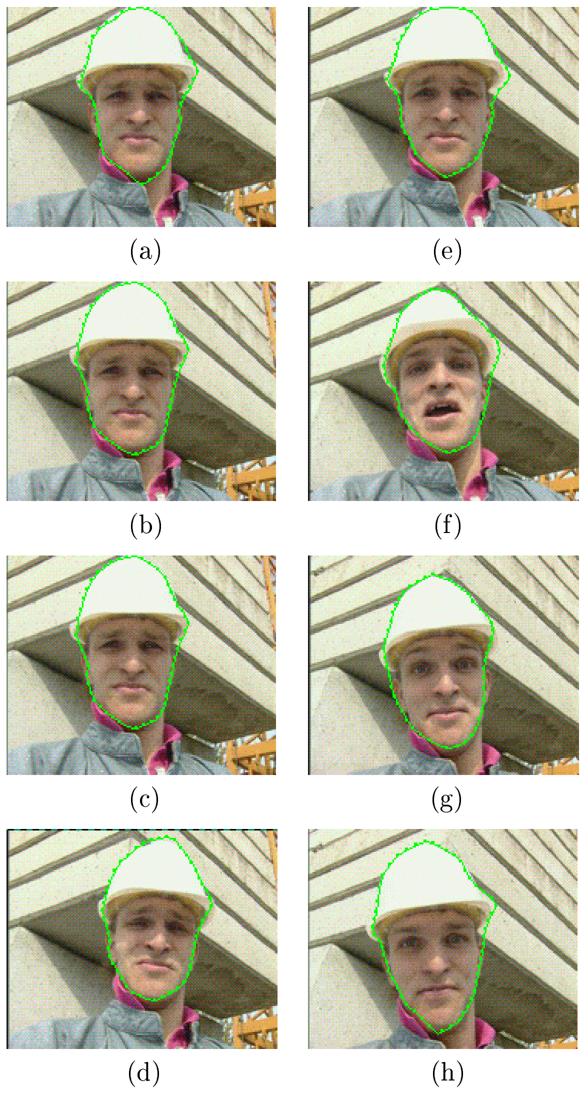


Figure 12: Illustration of a full segmentation result on the Foreman sequence. The frame numbers are 10, 20, 30, 40, 60, 70, 80, 90 for subfigures (a)-(h).