Long term video segmentation through pixel level spectral clustering on GPUs*

Narayanan Sundaram and Kurt Keutzer
University of California at Berkeley
narayans,keutzer@eecs.berkeley.edu

Abstract

We introduce a new technique for performing video segmentation combining the state-of-the-art image segmentation and optical flow algorithms on GPUs. We avoid pre-clustering into superpixels and probabilistic reasoning, and instead view the problem as a generalization of image segmentation techniques. Utilizing spectral clustering techniques at the pixel level (as opposed to 2D/3D superpixels), we demonstrate video segmentation over hundreds of frames - far beyond what has been achieved through pixel level spectral segmentation techniques before. Our algorithm achieves comparable accuracy as other sparse motion clustering techniques while still maintaining 100% density in segmentation over long time periods. We achieve better accuracy with lower oversegmentation compared to dense video segmentation techniques. We exploit increased computational power made available through parallelism in GPUs and efficient numerical algorithms to achieve these results. We show our results on the motion segmentation dataset [4]. Our technique can also be used to provide good quality 3D superpixels and extended to tasks where the ability to track 3D volumes over time is useful.

1. Introduction

The amount of video data being generated has increased tremendously in recent years and is expected to grow even faster in coming years. For instance, Youtube adds 48 hours of video every minute [23]. There is still a huge gap between video generation and video analysis capabilities that has restricted the use of more advanced computer vision techniques for performing high level understanding in videos. In order to analyze videos, it is necessary to segment the objects in videos into multiple semantically meaningful 3D (2D image + time) volumes. While image segmentation has had a significant amount of success thanks to the techniques like normalized cuts and gPb [12, 15], video segmentation is still a work in progress.

A sea change has occurred in computer science in the recent past with the use of highly parallel processors such as multicore CPUs and GPUs. Only scalable, parallel algorithms can take advantage of this development. Parallel algorithms enable applications to run much faster, which in turn has widened the applicability of many algorithms which were earlier considered too slow to be practical. Some of the key application areas that have benefited from this trend towards using GPUs are optical flow[18, 21], 3D reconstruction [8], feature tracking [16] etc. We believe that video segmentation is an application area that can benefit tremendously from parallelism. The increase in computational power can translate to increased performance and improved accuracy.

We present our technique for performing video object segmentation through

- a novel method that combines state of the art image contour detection (gPb) [12] and large displacement optical flow [3] in a single spectral segmentation framework (at the pixel level) that scales to 100's of frames
- efficient and scalable numerical algorithms derived from existing literature [5, 18]
- efficient implementation on a cluster of GPUs

We demonstrate that video object segmentation can utilize techniques similar to image segmentation techniques that are known to work like gPb [12]. In particular, this obviates the need for extensive probabilistic reasoning and/or hypergraph creation that other video segmentation algorithms utilize when dealing with 2D super pixels. In contrast to such techniques, we perform spectral segmentation at the pixel level. The increased complexity in graph creation (e.g. multi scale superpixels), connectivity (e.g. hypergraphs) and processing (e.g. probabilistic inference) has not solved the problem of video segmentation satisfactorily.

Although our technique seems a natural extension of image segmentation, there are several key challenges to be overcome for it to succeed. We explain some of these problems (Section 2) and present our solutions to them (Section 3.4).

*The authors acknowledge the support of the Gigascale Systems Research Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.
Motion based segmentation of point trajectories is an active area of research [4, 13, 20, 22]. However there is a lack of quantitative comparison between these “sparse” classifiers and other “dense” video segmentation algorithms. We use the motion segmentation dataset [4] to provide labeled images in space and time, so that we can compare the accuracy of our technique against other existing approaches.

Using the motion segmentation benchmark [4] to evaluate our technique, we demonstrate a per pixel classification error of 7 – 15% in labeling 100% of the pixels in the video compared to sparse track segmentation techniques which can only label about 3% of the pixels. Our contribution shows that with sufficient computational power, video segmentation is just a natural extension of image segmentation. It is possible with today’s hardware to do a pixel-wise spectral segmentation on video sequences that are hundreds of frames long. High quality video segmentation will enable new directions to applications including intelligent video editing, object detection & recognition, 3D reconstruction and others.

2. Related Work

Dense video segmentation is being actively studied by the computer vision community. We will discuss some of the more recent approaches to this problem. Some of the recent techniques for doing video segmentation include Multiple Hypothesis Video Segmentation (MHVS) [19], hypergraph segmentation[11], hierarchical graph based segmentation [9], Circular Dynamic Time Warping (CDTW) [2] and [6]. MHVS [19] works by performing 2D superpixel segmentation of the frames at multiple scales and tries to find the best hypothesis that connects the superpixels from one frame to another. In order to do this, MHVS uses higher order potentials on a Conditional Random Field (CRF) and performs inference on the resulting graphical model. Hierarchical graph based segmentation [9] (which builds on [7]) proposes the creation of a hierarchical 3D superpixel (super voxels) representation for videos. They achieve this by creating a region graph and combining adjacent nodes if the “internal variation” of both the nodes is larger than the edge weight between the nodes. Hypergraph segmentation [11] works by extending the classical normalized cuts approach [15] to hypergraphs created from 2D superpixels produced from the oversegmentation of individual frames. DeMenthon [6] uses a Hough transform-like approach with 3D volume to identify objects in video. Brendel and Todorovic [2] use CDTW to temporally coalesce 2D superpixels obtained from image segmentation into 3D volumes. All of these approaches are either based on 2D superpixels (in which case temporal coherence has been difficult to maintain) or are not very robust (segmentation is noisy).

On the other hand, multi-body factorization and related methods are being used for motion based classification in videos. These methods usually rely on point trajectories extracted from the video. Some of the recent techniques that have been proposed for classifying point trajectories based on motion include [4, 13, 20, 22]. The technique in [4] performs spectral clustering on point trajectories. Other methods for performing clustering of point trajectories are factorization based approaches like Generalized Principal Component Analysis (GPCA) [20], Local Subspace Analysis(LSA) [22] and Agglomerative Subspace Clustering (ALC) [13]. A major drawback of all these approaches is the difficulty in getting dense labels for videos. At best, these techniques label about 3 – 4% of the pixels in the video. This might be sufficient for identifying large moving blobs like people or vehicles, but is not adequate for doing tasks like editing videos by cutting/pasting objects or segmenting objects that are distinct in appearance but are not moving independently.

Using Spectral clustering for segmentation was popularized by normalized cuts [15]. Shi and Malik [14] used a normalized cuts approach to do video segmentation. However, the technique was constrained by the lack of computational power, limiting video segmentation to only about 10 frames. Also, image contour detection has since improved through the addition of multiscale local information and an improved integration of spectral and multi scale local information (gPb) [12]. These additions along with the improved reliability in extracting optical flow from videos and tremendous improvements in computational power from GPUs have brought back the viability of organizing video segmentation as a generalization of image segmentation.

It should be noted that serious challenges remain for such spectral techniques to be successful. Spectral segmentation techniques work by constructing a graph of pixels (or superpixels) as nodes. Edges represent the affinities of pixels to other pixels. Pixels belonging to the same object should have higher affinities. Eigenvectors of the graph Laplacian matrix correspond to segmentations of the original image (or video). At the core of spectral segmentation is a sparse symmetric eigensolver. The amount of computation required for running the eigensolver on all pixels in a video simultaneously can be prohibitive. Increased computational power through a cluster of GPUs resolves the computational bottleneck for our experiments. Also, GPU implementations of the gPb image contour detector [5] and Large displacement Optical Flow [18] have enabled these algorithms to run within a few seconds compared to several minutes earlier. Smooth transitions in eigenvectors can create artificial boundaries for segmentation, but the edge weights across these boundaries are usually very low. This problem of oversegmentation due to eigenvector blending is avoided through the use of ultrametric contour maps [1] on 3-D voxels. This enables us to perform mergers of artificially broken regions while still maintaining distinctions.
between different objects.
Our contributions are two-fold -
1. a novel technique to combine image segmentation and optical flow in a single framework running efficiently on a cluster of GPUs.
2. Quantitative results on an existing data set comparing the accuracy of our video segmentation technique against existing point trajectory-based classifiers.

3. Proposed Method
Our proposed method combines image contour detection and optical flow in a single spectral segmentation framework.

3.1. Overview
Our technique generalizes image contour detection algorithms to perform video segmentation. This work is inspired by the image contour detection algorithm called gPb [12]. We briefly describe how we modify and extend it to video segmentation.

Our technique works as follows: We compute boundaries at the frame level independently using intensity, color, texture and motion cues. We use these cues to create an affinity matrix for the entire video. The generalized eigenvectors of the normalized affinity matrix corresponding to the smallest eigenvalues are calculated. A clustering algorithm is run on the eigenvectors and 3D superpixels are obtained. This initial boundary is refined using ultrametric contour maps and a stable segmentation is produced.

3.2. Frame level computations
For images, gPb requires the computation of multiscalePb - Local gradients of intensity, color and texture are captured at multiple scales and combined into a boundary detector that is local in scope. In order to extend it to video, we add motion information in the form of optical flow vectors to the local gradient computations. We perform local histogram calculations around every pixel for horizontal and vertical flow in different orientations (after normalizing to the maximum flow in the neighborhood). The motion gradient at any pixel is the chi-squared distance between the histograms of the two half-disks centered at that pixel. More details on the implementation of the multi scale boundary detector can be found in [12, 5]. We linearly combine this to existing intensity, color and texture gradients. After computing the multiscalePb signal, inter-pixel affinities are calculated using the intervening contour approach as shown in [12].

3.3. Optical Flow
We use Large displacement optical flow (LDOF) [3] for computing the motion vectors between adjacent frames. LDOF has been shown to be accurate in tracking fast moving objects compared to other trackers. We use [18] for running on GPUs. LDOF works well in natural videos where fast motion is common (e.g. fast moving hands, legs etc). Fast and accurate optical flow computation is a crucial component for the success of video segmentation.

In addition to computing the optical flow, we also calculate the confidence that the computed flow is correct. If the forward (from frame $i$ to $i+1$) and backward (from frame $i+1$ to $i$) flow vectors do not match at any point, then it implies an error in the optical flow calculation or occlusion, both of which are valid reasons to reduce our confidence. We use the following function derived from [18] for forward-backward reliability check and scale the affinities accordingly. The confidence map is given by

$$C(w, \hat{w}) = 1 - \tanh \left( \frac{|w + \hat{w}|^2}{\alpha (|w|^2 + |\hat{w}|^2) + \beta} \right)$$

where $w := (u, v)$ denotes the flow from frame $i$ to $i+1$ and $\hat{w} := (\hat{u}, \hat{v})$ denotes the flow from frame $i+1$ back to frame $i$. We use $\alpha = 0.01$ and $\beta = 0.5$.

3.4. Spectral clustering
At the core of our algorithm is spectral clustering. The video is represented as a graph with pixels as nodes and inter-pixel affinities as edge weights. Affinities are computed over a small radius ($r = 5$) around every pixel and a pixel affinity matrix is computed. Affinities are calculated between pixels in the same frame and between frames. This is essential for successful clustering of moving regions in different frames. In the absence of inter-frame pixel affinities, the problem just decomposes into computing the eigenvectors of different frames independently and no information is shared. The affinity values computed between pixels in different frames provide the necessary linkage.

Figure 1. Calculating inter frame pixel affinities. Frames 45 and 46 of the Tennis sequence [4] are shown. In order to calculate the affinity between pixels $P$ and $Q$, we calculate the forward projection of $P$ ($Q'$) and the backward projection of $Q$ ($P'$). Affinity between $P$ & $Q$ is calculated using equation (2).
the following reason: While defining affinities based on the intervening contour is possible in images in spite of the (relatively) large neighborhood sizes, it becomes impractical to do in 3D (2D space + time) because of the explosion in matrix size to impractical limits. We limit this by only calculating affinities between pixels in adjacent frames.

After limiting the pixel affinities to just 2 frames at a time, we define the neighborhood around which affinities are calculated based on optical flow. Figure 1 shows how this is done. For a pixel $P$ in frame $i$, we calculate affinities between $P$ and pixels centered around the point $Q'$ in frame $i + 1$. We calculate the affinity between pixel $P$ in frame $i$ and pixel $Q$ in frame $i + 1$ as follows:

$$f(P, Q) = \min(f_i(P, P'), f_{i+1}(Q', Q))$$  \hspace{0.5cm} (2)

where $Q'$ is the projection of pixel $P$ in frame $i + 1$ using forward flow, $P'$ is the back projection of pixel $Q$ in frame $i$ using backward flow and $f_i(a, b)$ refers to the affinity between pixels $a$ and $b$ in frame $i$. $f_i(a, b)$ is calculated using the intervening contour technique that gPb uses. Intervening contour assigns affinities between pixels based on the presence of edges along the line joining the two pixels (strong edge implies low affinity). Details on the intervening contour technique can be found in [12]. This affinity is scaled according to the confidence we have on the optical flow vectors (equation 1).

The overall affinity matrix $W$ is a symmetric matrix of size = number of pixels in the video. It has the following structure -

$$W = \begin{pmatrix}
W_{1,1} & W_{1,2} & 0 & 0 & \cdots & 0 \\
W_{1,2} & W_{2,2} & W_{2,3} & 0 & \cdots & 0 \\
0 & W_{2,3} & W_{3,3} & W_{3,4} & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \cdots & W_{n,n}
\end{pmatrix}$$

where $W_{i,i}$ are intra frame pixel affinity matrices and $W_{i,j}$ are inter frame affinity matrices.

We compute the normalized graph Laplacian of $W$ i.e. $A = D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}}$ is calculated from the pixel affinity matrix $W$. $D$ is a diagonal matrix = $\text{diag}(W \cdot 1)$ [15].

This matrix is then run through the eigensolver to get the eigenvectors corresponding to the $n$ smallest eigenvalues. By construction, the smallest eigenvalue is 0 and the eigenvector corresponding to it is ignored. As noted by [5], the pixel affinity matrices derived from natural images have properties that can be exploited to produce very efficient eigensolvers. In particular, it was noticed that it is very unlikely for these matrices to have multiple eigenvalues converging to the same number and that the eigenvalues are well separated. These properties were exploited to avoid a costly reorthogonalization step that resulted in a 20× speedup over a general eigensolver. We take this algorithm and extend it to apply to our matrix.

There are some problems with using the raw eigenvectors for video segmentation. The number of eigenvectors needed for effective segmentation may increase with the length of the video (Objects entering/exiting, occlusions etc). We found that calculating 21 eigenvectors (including the information-less eigenvector with eigenvalue = 0) was sufficient for all the sequences shown in this paper. Increasing or decreasing this number slightly does not alter the results significantly. Another option is to threshold the number of eigenvectors calculated based on the eigenvalue e.g. calculating all eigenvectors with eigenvalues $\leq 10^{-3}$.

There is also the known problem of leakage i.e. eigenvectors can have smooth transitions leading to the break up of a single object into multiple objects at different time scales. This is clearly illustrated in Figure 2. The figure shows how the identity of the person shifts through the sequence as seen from an eigen-perspective. The values of the eigenvector on the person varies between 0.07 and 0.9 (the eigenvector is normalized to be in $[0, 1]$), effectively taking the full range of possible values. In order to avoid this problem of oversegmentation (and maintain coherent object boundaries), we use ultrametric contour maps for segmentation as a post processing step. This step provides the benefit of combining segment boundaries that are created artificially due to the smooth transitions. Details of this step are provided in the next section.

3.5. Post processing

The eigenvectors are clustered into 500 clusters using k-means, giving us an over-segmentation of the video in the form of 3D superpixels. It is essential to perform a oversegmentation at this stage so that we do not miss any significant boundaries. In order to avoid the k-means algorithm getting stuck in a local optimum, we run k-means clustering 3 times and pick the best clustering as one that has the maximum edge weight along the segment boundaries (computed using the gPb signals of the individual frames). Ultrametric Contour Maps [1] is applied to the resulting supervoxels to cluster them into a small number of regions. This step removes unwanted boundaries created due to the k-means clustering and the smooth transitions in the eigenvectors.

It is important to note that our segmentation algorithm does not rely only upon motion information, but rather motion is just one of the many features used (along with intensity, color and texture). In such a scenario, it is not expected that we will segment only those regions that have different motions. Objects with distinct appearances are segmented even if they do not have motion that is independent of the background or other objects. Depending on the weights assigned to the multiple cues, different features may be emphasized. [12] had performed training using the Berkeley Segmentation Data Set (BSDS) in order to get optimal weights for finding boundaries in images. We use the
same weights as [12], and assign the motion features the same weights as the color features. We have seen that this assignment produces good results while still being robust to errors in optical flow (especially along motion boundaries).

It is possible to run a second level cluster processing algorithm along the lines of [4] or [10] to identify affine motion parameters for the different clusters and combine them based on the information. This will reduce the number of final segments obtained while emphasizing differences in motion and de-emphasizing other differences.

3.6. Parallelization & Implementation details

The video segmentation algorithm was implemented using CUDA for GPU programming and Message Passing Interface (MPI) for cluster programming. The affinity matrix for the video is very large - about 20 GB for 200 frames of the marple1 sequence of resolution $352 \times 288$ (>20 million rows in single precision). All of the optical flow, local gradient and eigensolver computations happen in the GPUs. It is to be noted that the individual GPUs have limited memory (only 3 GB in our case). Hence, we divide the video into smaller sets of frames and assign it to different GPUs. Each GPU is allocated 2-6 frames depending on the size of the frame ranging from $640 \times 480$ to $352 \times 288$. The affinity matrix is partitioned row-wise among the different GPUs. Each GPU stores only a small portion of the matrix and the eigenvectors (corresponding to a few frames each). Every iteration of the Lanczos algorithm involves two forms of communication between the GPUs - (1) all-to-all communication for computing vector dot products and (2) adjacent-node communication for computing Sparse Matrix Vector Multiply ($y = Ax$). Storing the final eigenvectors alone takes 1.7 GB of memory (not including the temporary data structures needed for the eigenvector computation).

We use a modified version of Sparse Matrix Vector Multiply (SpMV) routine developed by [5] as part of our routine. Because of the way the inter frame pixel affinities are calculated, the off-diagonal matrices $W_{i,i+1}$ are not symmetric. Maximum computational efficiency can be obtained only if we represent $W_{i,i+1}$ and $W_{i,i+1}^T$ in different data structures. However, in order to save storage space at the expense of computational efficiency, we store only $W_{i,i+1}$ and use a slower SpMV for $W_{i,i+1}^T$ computations. The eigensolver for the marple1 sequence takes 3 minutes on 34 GPUs.

We run the application on a GPU cluster. Each node has a Nvidia Tesla C2050 having 3 GB of onboard memory and 14 processors (448 Floating point units). We used as many GPUs as needed according to the length of the video processed (up to 34 nodes). It should be noted that it is not necessary to run the program on the cluster and the code could also be run on a single GPU (taking proportionally more time). On average, the runtime for the segmentation was about 5 minutes for 200 frames of size $352 \times 288$.

4. Evaluation & Results

As mentioned earlier, to the best of our knowledge, there is currently no available literature that does a quantitative comparison between segmentation algorithms based on point trajectories and dense video segmentation algorithms on the same data set. The motion segmentation dataset [4] provides ground truth data for studying both sparse and dense labeling in videos. In particular, the data set has frames labeled densely in space and sparsely in time. Although point trajectory segmentation ostensibly does not use visual information but only motion information, it should be noted that optical flow and tracking algorithms themselves utilize visual information. With increasing density of coverage and ability to use incomplete tracks, motion segmentation is getting closer to full video segmentation. We provide quantitative comparison of sparse and dense video segmentation on a single data set. We also...
compare against other recent video segmentation works like [11, 9].

4.1. Comparison to point trajectory based segmentation

We compare our technique against the motion segmentation results from [4] and [13] on the data set from [4]. Comparisons to other sparse motion trajectory classifiers on this data set can be found in [4]. Table 1 shows that our technique compares favorably to the sparse trajectory classifier in [4]. The overall pixel error is the number of bad labels over the total number of labeled pixels. The overall accuracy is the total number of correctly labeled pixels over the total number of pixels in the video. The average error is computed similar to the overall error, but over regions (not pixels) after computing the error for each region separately. Since the evaluation tool automatically combines segments to avoid high penalties for oversegmentation, the average number of clusters combined is also reported. The total number of objects extracted with ≤ 10% error is also reported.

We achieve comparable errors while labeling 100% of the pixels in the video as compared to only about 3% by sparse segmentation algorithms such as [4, 13]. In absolute terms, we label far more pixels in the video accurately compared to the sparse techniques. Figure 3 shows the results of our approach on one of the sequences in the data set. The motion segmentation results from [4] shifts the identity of the background during the sequence. It is to be noted that the ground truth in this data set is limited to labeling based on motion only. We believe that video segmentation is not limited to just segmentation based on motion, but rather segmentation based on both motion and appearance. Hence, the segmentation we produce almost always produces a certain degree of oversegmentation intentionally.

4.2. Comparison to video segmentation algorithms

Table 1 also shows that our technique compares favorably to hierarchical graph segmentation [9] on sequences of 200 frames. Both techniques can reduce the per pixel error through oversegmentation; however, from the table it is clear that our technique produces far less oversegmentation (about a factor of 2) with slightly better accuracy. Oversegmentation can be quite problematic for many algorithms that use video segmentation and post processing cannot fix it in many cases. Hence, a technique that produces the same accuracy with less oversegmentation is preferable.

From Figure 3, we can also see that hierarchical graph segmentation produces segmentation edges that are not smooth. There is also a tendency to produce segments that are unconnected spatially (even though they are connected in 3-D). These artifacts make the segmentation results hard to use for algorithms that require them for editing video or unsupervised learning. Our results conform to the natural image contours and provides mostly spatially connected components. The final segmentations shown are thresholded at 65% of peak for our technique and 90% for hierarchical graph segmentation. Our technique is also able to retrieve 3 objects with less than 10% error compared to 0 objects for hierarchical graph segmentation.

We compare our technique to video segmentation techniques in [9, 2] on the flower garden sequence in Figure 4. Note that we avoid the oversegmentation in the flower bed that [9] and [2] produce due to its highly textured nature. Due to smooth illumination changes on the tree, the segmentation edges from [9] are not smooth. Also notice the presence of multiple “holes” in the segmentation. The result from [2] uses mean shift clustering, and is unstable in noisy and textured regions. The identity shifts of the house and background are other shortcomings of this method. Our result demonstrates the effect of combining image contour detection with optical flow. The segmentation edges are smooth and the noise in the flower bed is removed. Also note that we identify each house as a separate segment.

Figure 5 compares our segmentation with hypergraph segmentation [11] on the rocking horse sequence from the occlusion data set [17]. This data set contains video sequences of objects with little to no motion. Camera motion is predominant and the objective is to identify occlusion boundaries. Hypergraph segmentation requires knowledge of the number of segments in the video. Our technique does not have this problem. Our method is more robust as we automatically choose our desired segmentation granularity after the processing is complete. Another advantage of our technique is the ability to tailor our result to the choice of segmentation. Since this dataset considers only occlusion boundaries, we doubled the weights of the motion features for the local gradient computation in our method. This emphasizes motion and de-emphasizes other features. The results from the new weights are also shown in the figure. We see that some of the appearance-based edges are removed and occlusion edges are identified.

5. Conclusion

We have demonstrated that video segmentation can be seen as a generalization of image segmentation. We have developed a scalable algorithm that overcomes the problems associated with spectral segmentation in long videos using the computational power of GPUs. We have compared our results quantitatively to motion trajectory based sparse classifiers and have shown that it is possible to achieve comparable accuracy while still labeling 100% of the pixels (motion segmentation techniques such as [4, 13] label ∼ 3% of the pixels). We also produce less oversegmentation and better accuracy compared to dense video segmentation algorithms like [9]. We believe that high quality video segmen-
<table>
<thead>
<tr>
<th>Name</th>
<th>Label density</th>
<th>Overall (per pixel)</th>
<th>Overall (per pixel)</th>
<th>Average (per region)</th>
<th>Over segmentation error</th>
<th>Extracted objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our technique</td>
<td>100%</td>
<td>6.97%</td>
<td>93.03%</td>
<td>19.72%</td>
<td>7.15%</td>
<td>20</td>
</tr>
<tr>
<td>Brox and Malik</td>
<td>3.34%</td>
<td>7.75%</td>
<td>3.08%</td>
<td>25.01%</td>
<td>0.54%</td>
<td>24</td>
</tr>
<tr>
<td>ALC incomplete</td>
<td>3.34%</td>
<td>11.20%</td>
<td>2.97%</td>
<td>26.73%</td>
<td>0.54%</td>
<td>19</td>
</tr>
<tr>
<td>Original video</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 frames (26 sequences)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our technique</td>
<td>100%</td>
<td>8.42%</td>
<td>91.58%</td>
<td>28.76%</td>
<td>13.0%</td>
<td>5</td>
</tr>
<tr>
<td>Brox and Malik</td>
<td>3.27%</td>
<td>7.13%</td>
<td>3.04%</td>
<td>34.76%</td>
<td>0.53%</td>
<td>9</td>
</tr>
<tr>
<td>ALC incomplete</td>
<td>3.27%</td>
<td>16.42%</td>
<td>2.73%</td>
<td>49.05%</td>
<td>6.07%</td>
<td>2</td>
</tr>
<tr>
<td>50 frames (15 sequences)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our technique</td>
<td>100%</td>
<td>15.46%</td>
<td>84.54%</td>
<td>45.05%</td>
<td>5.86%</td>
<td>3</td>
</tr>
<tr>
<td>Brox and Malik</td>
<td>3.43%</td>
<td>7.64%</td>
<td>3.17%</td>
<td>31.14%</td>
<td>3.14%</td>
<td>7</td>
</tr>
<tr>
<td>ALC incomplete</td>
<td>3.43%</td>
<td>19.33%</td>
<td>2.77%</td>
<td>50.98%</td>
<td>54.57%</td>
<td>0</td>
</tr>
<tr>
<td>Hierarchical Graph Segmentation</td>
<td>100%</td>
<td>20.77%</td>
<td>79.23%</td>
<td>40.37%</td>
<td>10.42%</td>
<td>0</td>
</tr>
<tr>
<td>200 frames (7 sequences)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison of our video segmentation with other approaches.

Figure 3. Comparison of our technique vs motion segmentation by [4] and [9] on one of the sequences from [4]. From top to bottom Video frames (1, 50, 140, 200), Segmentation results from [4], Hierarchical graph segmentation [9], Our final segmentation result. Each tracked point in [4] is shown as 9 pixels large for illustration purposes. Notice that background shifts identity in [4]. Hierarchical graph segmentation produces oversegmentation and non-smooth boundaries. Also note the confusion between the person and the background in the later frames. (Figure best viewed in color)

Our approach Final segmentation

Motion segmentation Brox & Malik

Hierarchical graph Segmentation

Hierarchical graph Segmentation

References

Video Processing Workshop, 2002.


