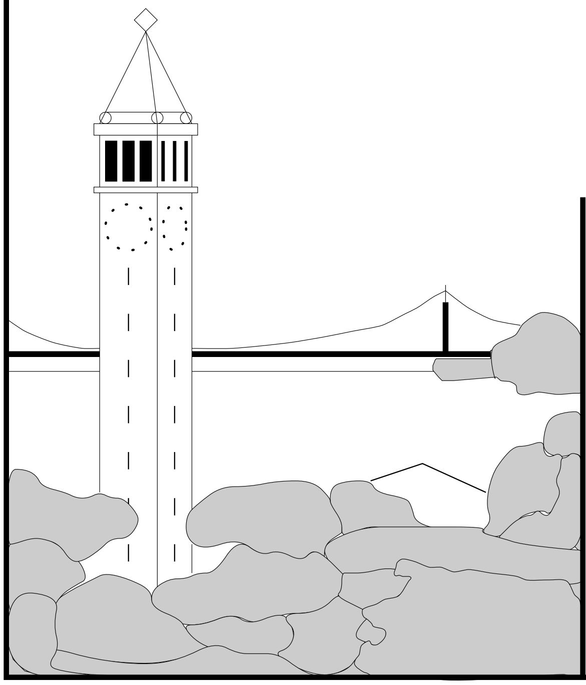
Motion Segmentation and Tracking Using Normalized Cuts

Jianbo Shi and Jitendra Malik



Report No. UCB/CSD-97-962

June 1997

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Motion Segmentation and Tracking Using Normalized Cuts

Jianbo Shi*† and Jitendra Malik*
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720
{jshi,malik}@cs.berkeley.edu

Abstract

We propose a motion segmentation algorithm that aims to break a scene into its most prominent moving groups. Instead of identifying point correspondences between the image frames, the idea to find what groups of pixels are transformed from one image frame to another. To do this, we treat the image sequence as a three dimensional spatiotemporal data set and construct a weighted graph by taking each pixel as a node, and connecting pixels that are in the spatiotemporal neighborhood of each other. We define a motion profile vector at each image pixel which captures the probability distribution of the image velocity at that point. By defining a distance between motion profile at two pixels, we can assign a weight on the graph edge connecting them. Using *normalized cuts* we find the most salient partitions of the spatiotemporal volume formed by the image sequence. Each partition, which is in the form of a spatiotemporal volume, corresponds to a group of pixels moving coherently in space and time. Normalized cuts can be computed efficiently by solving a generalized eignevalue problem.

For segmenting long image sequences, we have developed a recursive update procedure that incorporates knowledge of segmentation in previous frames for efficiently finding the group correspondence in the new frame. It speeds up the segmentation significantly when there is no major scene change, while in the worst case, when there is major scene change, the algorithm performs correctly as if no prior knowledge is used. Experimental results on various synthetic and real image sequences are presented.

Keywords: motion segmentation, grouping, graph partition, tracking

*Supported by (ARO) DAAH04-96-1-0341
†NSF Graduate Fellowship

1 Introduction

Grouping based on common motion, or what the Gestaltists[Wer38] called the factor of "Common Fate", is one of the strongest cues for segmenting an image sequence into separate objects. However, implementing this perceptual capability has proved to be very challenging for computer vision systems. Early approaches were based on trying to estimate optical flow first, and then looking for discontinuities. This proved difficult because of a number of reasons

1. Optical flow measurement is difficult in areas of little texture or primarily one-dimensional texture. Any real image is bound to have large regions with these properties.

2. To deal with difficulty (1) enforcing smoothness constraints to interpolate in the flow field were proposed. However this raises the requirement that one must first know the segmentation so as to avoid smoothing across motion discontinuities!

To cope with these problems, over the last few years a new framework has appeared based on the idea of simultaneous estimation of multiple global motion models and their spatial supports (so-called "layers"). This idea has evolved through a number of papers [AS95, WA96, DP91, WA94, HAP94, JB93]. Perhaps the cleanest current formulations are based on using the Expectation-Maximization (EM) algorithm[DLR77]. Typically the motion models are 2D parametric models, translational, affine or projective, the E-step is used to solve for the layers given the motions and their variances, and the M-step for solving for the motions and variances given the layers.

EM approaches offer a number of advantages over the previous approaches based on initial local measurement of optical flow. By combining information over large regions of the images, the motion estimates found are considerably more robust. Video data can be quite noisy because of camera jitter and repeated occlusion and discocclusion events, and the global analysis provides a way to overcome these difficulties. The layers that are extracted provide the desired scene segmentation. On the other hand, the assumption that image sequence have to follow a global rigid planar motion is clear too restrictive, and recently Weiss[Wei97] has developed a variation of EM approach that is based on a non-parametric mixture model using a probability distribution over flow fields that favors smooth flow fields. When only sparse point correspondences are sought, Torr and Murray[TM94] have developed an alternative approach based on characterizing rigid motions using Fundamental matrices.

In our opinion, the principal weakness of the EM approach to layered motion segmentation is in the initialization phase. How many models should one initialize and where and what should they be, and how can one ensure a global optimal solution have been reached? A representative approach due to Ayer and Sawhney[AS95] uses the Minimum Description length principle for selecting the number of models. Initialization is done by dividing up the image into a fixed number of tiles; estimating the initial motion parameters in these tiles and then using these as the initial conditions for the EM algorithm. Our experience however have shown finding a good initialization remains a nagging problem. Undoubtedly, further research in this area will provide improvements in this area; however in this paper we have chosen to develop an alternative approach that incorporates motion information across spatial and temporal neighborhood and finds a globally optimal segmentation solution without the difficulty of the initialization that EM approaches need to overcome.

Our approach is to look at the motion segmentation problem as a special instance of a more general grouping problem. We treat each pixel in the image sequence as a point living in a large feature space represented by its spatiotemporal position, and other feature information such as its color and motion information. We then ask the question: given these points in this large feature space, what is the best way of partitioning them? Our answer to this question is very similar to a philosophical perspective that could be traced back to the Gestalt school: image partitioning should be done at the 'big picture' level, rather like a painter first marking out the major areas and then filling in the details.

To that end, we have developed a novel grouping algorithm that takes in points in an arbitrary feature space, and produce partitions of them in this 'big picture' downward way. To be more specific, we construct a weighted graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, with each point in the feature space as a node, and the weight associated with each edge connecting two nodes being a function of their distance in the feature space. To partition this graph into two disjoint sets $\mathbf{V}_1, \mathbf{V}_2$, $\mathbf{V}_1 \cup \mathbf{V}_2 = \mathbf{V}$, we propose to use the *normalized cut* criterion which measures both the total dissimilarity between the two sets as well as the total similarity within the sets. Minimizing the *normalized cut* criterion leads to finding the most significant partition in the feature space, and recursive repartitioning can be used to forming successively less significant segments in the feature space. We will show that an efficient computational technique based on a generalized eigenvalue problem can be used to optimize this criterion. We have previously demonstrated static image segmentation based on brightness, color or texture using *normalized cuts*[SM97].

In this paper, we will show the development of the *normalized cuts* approach to motion segmentation (and by restricting to 2 frames, stereopsis based segmentation). Our approach provides a fresh outlook of the motion segmentation problem. In the segmentation algorithm, a motion sequence is regarded as a spatiotemporal data set and identified with a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where the nodes of the graph are image triplets, (x, y, t) (a particular pixel in a particular time frame). The weight on each edge $w(\mathbf{i}, \mathbf{j})$ is a function of the similarity between pair of nodes $\mathbf{i}$ and $\mathbf{j}$. Similarity can be estimated on the basis of any of a number of features, color, brightness, texture, motion, disparity etc. The results in this paper are based on using only motion features so as to permit a fair comparison with alternative motion segmentation schemes. By connecting each node to other nodes in its spatiotemporal neighborhood, we provide an effective way for the motion information to be integrated over space and time, which increase the robustness of the segmentation, without imposing any explicit global motion constraint. Successful partition of G gives us spatiotemporal volumes corresponding to different moving objects. By taking time slices of such a volume we can indicate image groups in each frame as well as identify what their corresponding groups are across time.

This notion of *group correspondence* is very useful because it leads to a measure of motion which applies to the group as a whole, not individual pixels as in the case of optical flow. Such a measure is considerably more robust and could be used for estimating gross measures such as divergence, deformation, rotation which have been shown to be useful variables for visual guidance of locomotion and manipulation [Koe86, CB92]. Snakes have been used in the computer vision literature[CB92] previously for this purpose. They are computationally efficient but difficult to initialize.

The paper is organized as follows. Section 2 explains the details of our graph partition algorithm based on the *normalized cut* criterion. This section follows our previous work for static image segmentation described in[SM97]. Section 3 describes the development of motion segmentation using *normalized cuts*. Section 4 shows how we can efficiently handle segmentation of long image sequences. We conclude in section 5.

2 Grouping as graph partitioning

A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ can be partitioned into two disjoint sets, A, B, $A \cup B = \mathbf{V}$, $A \cap B = \emptyset$, by simply removing edges connecting the two parts. The degree of dissimilarity between these two pieces can be computed as total weight of the edges that have been removed. In graph theoretic language, it is called the *cut*:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v). \tag{1}$$

The optimal bi-partitioning of a graph is the one that minimizes this *cut* value. Although there are exponential number of such partitions, finding the *minimum cut* of a graph is a well studied problem, and there exist efficient algorithms for solving it.

Wu and Leahy[WL93] proposed a clustering method based on this minimum cut criterion. In particular, they seek to partition a graph into k-subgraphs, such that the maximum cut across the subgroups is minimized. This problem can be efficiently solved by recursively finding the minimum cuts that bisect the existing segments. As shown in Wu & Leahy's work, this globally optimal criterion can be used to produce good segmentation on some of the images.

However, as Wu and Leahy also noticed in their work, the minimum cut criteria favors cutting small sets of isolated nodes in the graph. This is not surprising since the *cut* defined in (1) increases with the number of edges going across the two partitioned parts. Figure (1) illustrates one such case. Assuming the edge weights are inversely proportional to the distance between the two nodes, we see the cut that partitions out node n_1 or n_2 will have a very small value. In fact, any cut that partitions out individual nodes on the right half will have smaller cut value than the cut that partitions the nodes into the left and right halves.

To avoid this unnatural bias for partitioning out small sets of points, we propose a new measure of disassociation between two groups. Instead of looking at the value of total edge

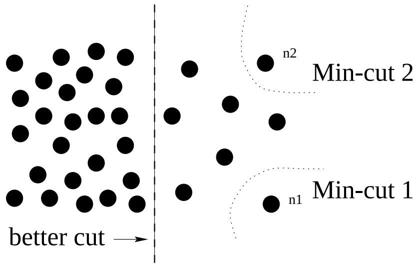


Figure 1: A case where minimum cut gives a bad partition.

weight connecting the two partitions, our measure computes the cut cost as a fraction of the total edge connections to all the nodes in the graph. We call this disassociation measure the *normalized cut* (*Ncut*):

$$Ncut(A,B) = \frac{cut(A,B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)} \tag{2}$$

where $asso(A,V) = \sum_{u \in A, t \in V} w(u,t)$ is the total connection from nodes in A to all nodes in the graph, and $asso(B,V)$ is similarly defined. With this definition of the disassociation between the groups, the cut that partitions out small isolated points will no longer have small $Ncut$ value, since the cut value will almost certainly be a large percentage of the total connection from that small set to all other nodes. In the case illustrated in figure 1, we see that the cut_1 value across node n_1 will be 100% of the total connection from that node.

In the same spirit, we can define a measure for total normalized association within groups for a given partition:

$$Nasso(A,B) = \frac{asso(A,A)}{asso(A,V)} + \frac{asso(B,B)}{asso(B,V)} \tag{3}$$

where $asso(A,A)$ and $asso(B,B)$ are total weights of edges connecting nodes within A and B respectively. We see again this is an unbiased measure, which reflects how tightly on average nodes within the group are connected to each other. Another important property of this definition of association and disassociation of a partition is that they are naturally related:

$$Ncut(A,B) \quad = \quad \frac{cut(A,B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)}$$

$$
\begin{aligned}
&= \frac{asso(A,V) - asso(A,A)}{asso(A,V)} \\
&\quad + \frac{asso(B,V) - asso(B,B)}{asso(B,V)} \\
&= 2 - \left(\frac{asso(A,A)}{asso(A,V)} + \frac{asso(B,B)}{asso(B,V)}\right) \\
&= 2 - Nasso(A,B)
\end{aligned}
$$

Hence the two partition criteria that we seek in our grouping algorithm, minimizing the disassociation between the groups and maximizing the association within the group, are in fact identical, and can be satisfied simultaneously. In our algorithm, we will use this *normalized cut* as the partition criterion.

Having defined the graph partition criterion that we want to optimize, we will show how such an optimal partition can be computed efficiently.

2.1 Computing the optimal partition

Given a partition of nodes of a graph, V, into two sets A and B, let $\boldsymbol{x}$ be an $N = |\boldsymbol{V}|$ dimensional indicator vector, $x_i = 1$ if node i is in A, and -1 otherwise. Let $\boldsymbol{d}(\mathrm{i}) = \sum_j w(i,j)$, be the total connection from node $\boldsymbol{i}$ to all other nodes. With the definitions $\boldsymbol{x}$ and $\boldsymbol{d}$ we can rewrite $Ncut(A,B)$ as:

$$
\begin{aligned}
Ncut(A,B) &= \frac{cut(A,B)}{asso(A,V)} + \frac{cut(B,A)}{asso(B,V)} \\
&= \frac{\sum_{(\boldsymbol{x}_i>0,\boldsymbol{x}_j<0)} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_i>0}\boldsymbol{d}_i} \\
&\quad + \frac{\sum_{(\boldsymbol{x}_i<0,\boldsymbol{x}_j>0)} -w_{ij}\boldsymbol{x}_i\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_i<0}\boldsymbol{d}_i}
\end{aligned}
$$

Let $\mathbf{D}$ be an $N \times N$ diagonal matrix with $\boldsymbol{d}$ on its diagonal, $\mathbf{W}$ be an $N \times N$ symmetrical matrix with W(i,j) $= w_{ij}$, $k = \frac{\sum_{x_i>0}\boldsymbol{d}_i}{\sum_i \boldsymbol{d}_i}$, and $\mathbf{1}$ be an $N \times 1$ vector of all ones. Using the fact $\frac{\mathbf{1}+\boldsymbol{x}}{2}$ and $\frac{\mathbf{1}-\boldsymbol{x}}{2}$ are indicator vectors for $x_i > 0$ and $x_i < 0$ respectively, we can rewrite $4[Ncut(\boldsymbol{x})]$ as:

$$
= \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{k\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\boldsymbol{x})}{(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}}
$$

$$= \frac{(\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}+\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{2(1-2k)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}}$$

Let $\alpha(\boldsymbol{x}) = \boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}$, $\beta(\boldsymbol{x}) = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}$, $\gamma = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}$, and $M = \mathbf{1}^T\mathbf{D}\mathbf{1}$, we can then further expand the above equation as:

$$\begin{aligned}
&= \frac{(\alpha(\boldsymbol{x})+\gamma)+2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} \\
&= \frac{(\alpha(\boldsymbol{x})+\gamma)+2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} - \frac{2(\alpha(\boldsymbol{x})+\gamma)}{M} \\
&\quad + \frac{2\alpha(\boldsymbol{x})}{M} + \frac{2\gamma}{M}
\end{aligned}$$

dropping the last constant term, which in this case equals 0, we get

$$\begin{aligned}
&= \frac{(1-2k+2k^2)(\alpha(\boldsymbol{x})+\gamma)+2(1-2k)\beta(\boldsymbol{x})}{k(1-k)M} + \frac{2\alpha(\boldsymbol{x})}{M} \\
&= \frac{\frac{(1-2k+2k^2)}{(1-k)^2}(\alpha(\boldsymbol{x})+\gamma)+\frac{2(1-2k)}{(1-k)^2}\beta(\boldsymbol{x})}{\frac{k}{1-k}M} + \frac{2\alpha(\boldsymbol{x})}{M}
\end{aligned}$$

Letting $b = \frac{k}{1-k}$, and since $\gamma = 0$, it becomes,

$$\begin{aligned}
&= \frac{(1+b^2)(\alpha(\boldsymbol{x})+\gamma)+2(1-b^2)\beta(\boldsymbol{x})}{bM} + \frac{2b\alpha(\boldsymbol{x})}{bM} \\
&= \frac{(1+b^2)(\alpha(\boldsymbol{x})+\gamma)}{bM} + \frac{2(1-b^2)\beta(\boldsymbol{x})}{bM} + \frac{2b\alpha(\boldsymbol{x})}{bM} - \frac{2b\gamma}{bM} \\
&= \frac{(1+b^2)(\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}+\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad + \frac{2(1-b^2)\mathbf{1}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad + \frac{2b\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})\boldsymbol{x}}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} - \frac{2b\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&= \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}} \\
&\quad + \frac{b^2(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-\boldsymbol{x})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}}
\end{aligned}$$

$$-\frac{2b(\mathbf{1}-\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}}$$
$$= \frac{[(\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})]^T(\mathbf{D}-\mathbf{W})[(\mathbf{1}+\boldsymbol{x})-b(\mathbf{1}-\boldsymbol{x})]}{b\mathbf{1}^T\boldsymbol{D}\mathbf{1}}$$

Setting $\boldsymbol{y} = (\mathbf{1}+\boldsymbol{x}) - b(\mathbf{1}-\boldsymbol{x})$, it is easy to see that

$$\boldsymbol{y}^T\boldsymbol{D}\mathbf{1} = \sum_{x_i>0}\boldsymbol{d}_i - b\sum_{x_i<0}\boldsymbol{d}_i = 0 \tag{4}$$

since $b = \frac{k}{1-k} = \frac{\sum_{x_i>0}\boldsymbol{d}_i}{\sum_{x_i<0}\boldsymbol{d}_i}$, and

$$\begin{aligned}\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y} = & \textstyle\sum_{x_i>0}\boldsymbol{d}_i + b^2\sum_{x_i<0}\boldsymbol{d}_i \\ = & \textstyle b\sum_{x_i<0}\boldsymbol{d}_i + b^2\sum_{x_i<0}\boldsymbol{d}_i \\ = & \textstyle b(\sum_{x_i<0}\boldsymbol{d}_i + b\sum_{x_i<0}\boldsymbol{d}_i) \\ = & b\mathbf{1}^T\boldsymbol{D}\mathbf{1}.\end{aligned}$$

Putting everything together we have,

$$min_{\boldsymbol{x}}Ncut(\boldsymbol{x}) = min_{\boldsymbol{y}}\frac{\boldsymbol{y}^T(\boldsymbol{D}-\boldsymbol{W})\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y}}, \tag{5}$$

with the condition $\boldsymbol{y}_i \in \{1, -b\}$ and $\boldsymbol{y}^T\boldsymbol{D}\mathbf{1} = 0$.

Note that the above expression is the Rayleigh quotient[GL89]. If $\boldsymbol{y}$ is relaxed to take on real values, we can minimize equation (5) by solving the generalized eigenvalue system,

$$(\mathbf{D}-\mathbf{W})\boldsymbol{y} = \lambda\mathbf{D}\boldsymbol{y}. \tag{6}$$

However, we have two constraints on $\boldsymbol{y}$, which come from the condition on the corresponding indicator vector $\boldsymbol{x}$. First consider the constraint $\boldsymbol{y}^T\boldsymbol{D}\mathbf{1} = 0$. We can show this constraint on $\boldsymbol{y}$ is automatically satisfied by the solution of the generalized eigensystem. We will do so by first transforming equation (6) into a standard eigensystem, and show the corresponding condition is satisfied there. Rewrite equation (6) as

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z} = \lambda\boldsymbol{z}, \tag{7}$$

where $\boldsymbol{z} = \mathbf{D}^{\frac{1}{2}}\boldsymbol{y}$. One can easily verify that $\boldsymbol{z}_0 = \mathbf{D}^{\frac{1}{2}}\mathbf{1}$ is an eigenvector of equation (7) with eigenvalue of 0. Furthermore, $\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}$ is symmetric semi-positive definite, since

$(\mathbf{D}-\mathbf{W})$, also called the *Laplacian* matrix, is known to be semi-positive definite[PSL90]. Hence $\boldsymbol{z}_0$ is in fact the smallest eigenvector of equation (7), and all eigenvectors of equation (7) are perpendicular to each other. In particular, $\boldsymbol{z}_1$ the second smallest eigenvector is perpendicular to $\boldsymbol{z}_0$. Translating this statement back into the general eigensystem (6), we have 1) $\boldsymbol{y}_0 = (0, \mathbf{1})$ is the smallest eigenvector, and 2) $0 = \boldsymbol{z}_1^T \ \boldsymbol{z}_0 = \boldsymbol{y}_1^T \ \mathbf{D} \ \mathbf{1}$, where $\boldsymbol{y}_1$ is the second smallest eigenvector of (6).

Now recall a simple fact about the *Rayleigh quotient*[GL89]:

Let $\mathbf{A}$ *be a real symmetric matrix. Under the constraint that* $\boldsymbol{x}$ *is orthogonal to the j-1 smallest eigenvectors* $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{j-1}$*, the quotient* $\frac{\boldsymbol{x}^T\mathbf{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ *is minimized by the next smallest eigenvector* $\boldsymbol{x}_j$*, and its minimum value is the corresponding eigenvalue* $\boldsymbol{\lambda}_j$*.*

As a result, we obtain:

$$\boldsymbol{z}_1 = arg.min_{\boldsymbol{z}^T\boldsymbol{z}_0=0} \frac{\boldsymbol{z}^T\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\boldsymbol{z}}{\boldsymbol{z}^T\boldsymbol{z}}, \tag{8}$$

and consequently,

$$\boldsymbol{y}_1 = arg.min_{\boldsymbol{y}^T\mathbf{D}\mathbf{1}=0} \frac{\boldsymbol{y}^T(\mathbf{D}-\mathbf{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathbf{D}\boldsymbol{y}}, \tag{9}$$

Thus the second smallest eigenvector of the generalized eigensystem (6) is the real valued solution to our *normalized cut* problem. The only reason that it is not necessarily the solution to our original problem is that the second constraint on $\boldsymbol{y}$ that $\boldsymbol{y}_i$ takes on two discrete values is not automatically satisfied. In fact relaxing this constraint is what makes this optimization problem tractable in the first place. We will show in section (2.2) how this real valued solution can be transformed into a discrete form.

A similar argument can also be made to show that the eigenvector with the third smallest eigenvalue is the real valued solution that optimally sub-partitions the first two parts. In fact this line of argument can be extended to show that one can sub-divide the existing graphs, each time using the eigenvector with the next smallest eigenvalue. However, in practice because the approximation error from the real valued solution to the discrete valued solution accumulates with every eigenvector taken, and all eigenvectors have to satisfy a global mutual orthogonality constraint, solutions based on higher eigenvectors become unreliable. It is best to restart solving the partitioning problem on each subgraph individually.

In summary, we propose using the *normalized cut* criteria for graph partitioning, and we have shown how this criteria can be computed efficiently by solving a generalized eigenvalue problem.

2.2 The grouping algorithm

As we saw above, the generalized eigensystem in (6) can be transformed into a standard eigenvalue problem. Solving a standard eigenvalue problem for all eigenvectors takes $O(n^3)$ operations, where n is the number of nodes in the graph. This becomes impractical for image segmentation applications where n related to the number of pixels in an image. Fortunately, our graph partitioning has the following properties: 1) the graphs often are only locally connected and the resulting eigensystems are very sparse, 2) only the top few eigenvectors are needed for graph partitioning, and 3) the precision requirement for the eigenvectors is low, often only the right sign bit is required. These special properties of our problem can be fully exploited by an eigensolver called the Lanczos method. The running time of a Lanczos algorithm is $O(mn) + O(mM(n))$[GL89], where m is the maximum number of matrix-vector computations allowed, and $M(n)$ is the cost of a matrix-vector computation. In the case where $(\mathbf{D} - \mathbf{W})$ is sparse, the matrix-vector product can be computed efficiently. As we should see in section 3, in the case of motion segmentation it costs only $O(n)$ time. The number m depends on many factors[GL89]. In our particular problem, m is typically less than $O(n^{\frac{1}{2}})$.

Once the eigenvectors are computed, we can partition the graph into two pieces using the second smallest eigenvector. In the ideal case, the eigenvector should only take on two discrete values, and the signs of the values can tell us exactly how to partition the graph. However, our eigenvectors can take on continuous values, and we need to choose a splitting point to partition it into two parts. There are many different ways of choosing such splitting point. One can take 0 or the median value as the splitting point, or one can search for the splitting point such that the resulting partition has the best $Ncut(A, B)$ value. We take the latter approach in our work. To find the best $Ncut$ value partition, one can perform linear search by checking l evenly spaced possible splitting points, and computing the best $Ncut$

among them. [1]

After the graph is broken into two pieces, we can recursively run our algorithm on the two partitioned parts. Or equivalently, we could take advantage of the special properties of the other top eigenvectors as explained in previous section to subdivide the graph based on those eigenvectors. The recursion stops once the *Ncut* value exceeds certain limit.

2.3 Related graph partition algorithms

The idea of using eigenvalue problems for finding partitions of graphs originated in the work of Donath & Hoffman[DH73], and Fiedler[Fie75]. Fiedler suggested that the eigenvector with the second smallest eigenvalue of the system $(\mathbf{D}-\mathbf{W})\boldsymbol{x} = \lambda\boldsymbol{x}$ could be used to split a graph. In fact the second smallest eigenvalue is called the Fiedler value, and corresponding eigenvector the Fiedler vector. This spectral partitioning idea has been revived and further developed by several other researchers, and recently popularized by the work of [PSL90], particularly in the area of parallel scientific computing.

In applications to several different areas, many authors have noted that the spectral partition method indeed provides good partitions of graphs [PSL90]. Most of the theoretical work done in this area has been focused on the connection between the *ratio of cut* and the Fiedler value. A *ratio of cut* of a partition of V, $P = (A, V-A)$ is defined as $\frac{cut(A,V-A)}{min(|A|,|V-A|)}$. It was shown that if the Fiedler value is small, partitioning graph based on the Fiedler vector will lead to good *ratio of cut*[ST96]. Our derivation in section 2.1 can be adapted (by replacing the matrix $\mathbf{D}$ in the denominators by the identity matrix $\mathbf{I}$) to show that the Fiedler vector is a real valued solution to the problem of $min_{A\subset V}\frac{cut(A,V-A)}{|A|} + \frac{cut(V-A,A)}{|V-A|}$, which we can call the *average cut*.

Although *average cut* looks similar to the *normalized cut*, *average cut* does not have the important property of having a simple relationship to the *average association*, which can be

[1] Or alternatively, one can look at the histogram of the eigenvector values, and choose the bins around the major peaks in the histogram as possible splitting points. The reason to choose those values as splitting points is that they roughly correspond to the end points for the gaps in the eigenvector values, and therefore most likely to produce good *Ncut* values. This is mainly for speeding up the computation, and in the worst case we can just do the search linearly for all the l possible values, with l usually set to a value less than 20.

analogously defined as $\frac{asso(A,A)}{|A|} + \frac{asso(V-A,V-A)}{|V-A|}$. Consequently, one can not simultaneously minimize the disassociation across the partitions, while maximizing the association within the groups. When we applied both techniques to the image segmentation problem, we found that the *normalized cut* produces better results in practice.

The generalized eigenvalue approach was first applied to graph partitioning by [DR95] for dynamically balancing computational load in a parallel computer. Their algorithm is motivated by [KYSK84]'s paper on representing a hypergraph in a Euclidean Space.

3 Motion Segmentation

To apply the normalized cut approach to motion segmentation, we treat the image sequence as a spatiotemporal data set. An associated graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is constructed by taking each image pixel (x, y, t) as a node and connecting it to nodes in its spatiotemporal neighborhood. The weight on each such arc is a measure of the similarity of feature vectors at the two nodes. In general, feature similarity can be estimated based on brightness, color, texture, motion, disparity or any combination thereof. In this paper we will use motion information exclusively.

The next question is to formulate a way to characterize the motion information locally. One way might be to measure the motion, or optical flow, at each pixel in an image sequence. Some of these algorithms are based on using the outputs of spatiotemporal filters [AB85, Hee87, FJ90] to estimate image velocity, while others use the brightness constancy assumption to derive differential techniques for computing the optical flow[LK81]. Although they differ in the details of their formulation, fundamentally they are equivalent[Sim93]. The basic limitations of those techniques are also quite similar. One can not determine the image velocity reliably at image patches where the intensity profile is flat, such as the image of a featureless wall, or image regions with a one dimensional intensity profile, such as an extended edge. Figure (2) illustrates these difficulties in one typical image sequence.

There have been various attempts to fix these problems in optical flow computation. Some restrict their algorithms to be run only at the places where velocity can be computed reliably[ST94], while others impose a smoothness constraint and apply regularization to ob-

(a) (b) (c)

Figure 2: Subimage (a) and (b) shows two frames of an image sequence, and optical flow computed by Lucas-Kanade type of algorithm is shown in (c). Notice that the optical flow estimates are reasonable in the textured regions, while in constant brightness regions such as the shirt and the area below the bookshelf, or in regions of repetitive structure, the algorithm performs poorly.

tain a smooth looking output[HS81, Ana89, NE86, BA91]. Alternatively, one could combine the process of motion measurement with image segmentation as has been done successfully in recent layer based approaches to motion analysis in the EM framework.

In our framework, the segmentation is going to emerge as a *result* of finding partitions that minimize the normalized cut, so evidently the feature similarity should be based on local measures of motion that can be computed *before* the segmentation is known. Instead of deciding locally and prematurely on the optical flow vector, we use the *motion profile*, a measure of the probability distribution of the image velocity at each pixel as our motion feature vector. Let $\boldsymbol{I}^t(\mathbf{X})$ denote a window centered at the pixel at location $\mathbf{X} \in R^2$ at time t. We denote by $P_i(\mathbf{dx})$ the probability of an image patch at node i, $\boldsymbol{I}^t(\mathbf{X}_i)$, at time t corresponding to another image patch $\boldsymbol{I}^{t+1}(\mathbf{X}_i + \mathbf{dx})$ at time $t+1$. $P_i(\mathbf{dx})$ can be computed by first estimating the similarity $S_i(\mathbf{dx})$ between $\boldsymbol{I}^t(\mathbf{X}_i)$ and $\boldsymbol{I}^{t+1}(\mathbf{X}_i + \mathbf{dx})$, and normalizing it to get a probability distribution:

$$P_i(\mathbf{dx}) = \frac{S_i(\mathbf{dx})}{\sum_{\mathbf{dx}} S_i(\mathbf{dx})}. \tag{10}$$

There are many ways one can compute similarity between two image patches; we will use

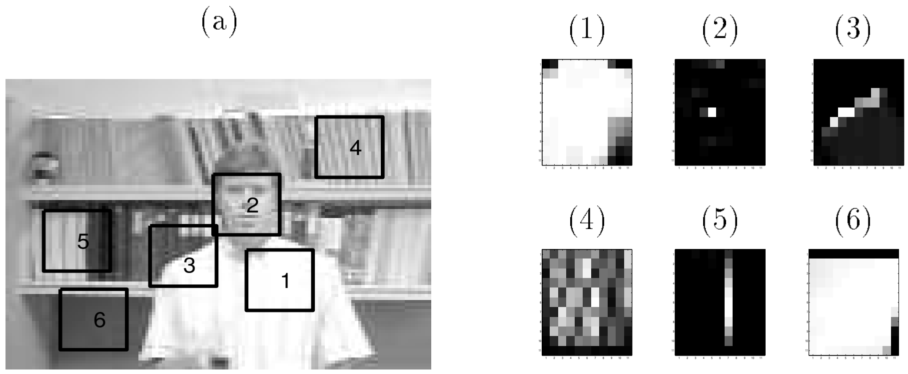


Figure 3: (a) outlines 6 image regions with various intensity profiles. Subimages (1)-(6) shows the corresponding motion profiles at pixels centered in the regions shown in (a). Note these motion vectors have captured the image velocities at those points as well as their associated uncertainties.

a measure that is based on the SSD difference:

$$S_i(\mathbf{dx}) = exp(-\sum_{\mathbf{w}}(\boldsymbol{I}^t(\mathbf{X}_i + \mathbf{w}) - \boldsymbol{I}^{t+1}(\mathbf{X}_i + \mathbf{dx} + \mathbf{w}))^2/\sigma_{ssd}^2), \tag{11}$$

where $\mathbf{w} \in R^2$ is within a local neighborhood of image patch $\boldsymbol{I}^t(\mathbf{X}_i)$. Figure (3) shows the motion profile computed according to the above definition on various image patches on an image shown in figure (2).

Now we are ready to assign a weight to the graph edge connecting two image pixels in the space-time domain based on their motion profile vectors. We found cross-correlation of the two motion profiles to be a simple, yet effective, measure of motion similarity. Define the distance between two image patches i and j as

$$\mathbf{d}(i,j) = 1 - \sum_{\mathbf{dx}} P_i(\mathbf{dx})P_j(\mathbf{dx}), \tag{12}$$

where $\mathbf{dx}$ range over possible displacements. The weight on graph edge (i,j) is then given by $w_{ij} = exp(-\mathbf{d}(i,j)/\sigma_m^2)$.

It should be noted that this measure of motion similarity will distinguish between two pixels which have exactly the same true motion, but where the brightness profiles are such that the associated motion uncertainties are very different. If one of the pixels is in a region of constant brightness and another in a region of rich texture this will happen. We believe that this is entirely appropriate given local information; the consequence can be over segmentation of a single rigidly moving object. This however is trivially handled in a postprocessing step.

For computational complexity reasons, we limit the number of nodes in the graph by subsampling the image (factor of 3 from the full spatial resolution in our examples), and limit the number of edges in the graph by having nonzero $w(i,j)$ only for nodes in a spatiotemporal window of ± 3 frames, and ± 5 in x and y. The image patch size used for computing patch–patch comparisons are 5×5 pixels.

To partition the graph, we construct the association matrix $\boldsymbol{W}$ with entry $\boldsymbol{W}(i,j) = w_{ij}$, and solve for the first few eigenvectors of the generalized eigensystem $(\boldsymbol{D}-\boldsymbol{W})\mathbf{x} = \lambda\boldsymbol{D}\mathbf{x}$. As we saw earlier, this generalized eigensystem can be transformed into a standard eigenvector system: $\boldsymbol{A}\mathbf{z} = \lambda\mathbf{z}$, where $\boldsymbol{A} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{D}-\boldsymbol{W})\boldsymbol{D}^{-\frac{1}{2}}$, and $\mathbf{z} = \boldsymbol{D}^{\frac{1}{2}}\mathbf{y}$. The Lanczos method can be used to obtain the first few eigenvectors efficiently if $\boldsymbol{A}$ is sparse. The running time of the Lanczos algorithm is dominated by the $O(mM)$ term where m is the maximum allowed iterations for solving the eigenvectors, and $O(M)$ the cost of computing matrix vector multiplication $\boldsymbol{A}\mathbf{x} = \mathbf{y}$. To see why this matrix vector multiplication costs only $O(n)$, where n is the number of nodes in the graph, we will look at the cost of inner product of one row of $\boldsymbol{A}$ with a vector $\mathbf{x}$. Let $\mathbf{y}_i = \boldsymbol{A}_i \cdot \mathbf{x} = \sum_j \boldsymbol{A}_{ij}\mathbf{x}_j$. For a fixed i, $\boldsymbol{A}_{ij}$ is only nonzero if node j is in a fixed space-time neighborhood of i. Hence there are only a fixed number of operations required for each $\boldsymbol{A}_i \cdot \mathbf{x}$, and the total cost of computing $\boldsymbol{A}\mathbf{x}$ is $O(n)$. Furthermore, each of those $\boldsymbol{A}_i \cdot \mathbf{x}$ inner product corresponds to a local space-time convolution type of operation, and therefore can be implemented efficiently on parallel processors.

To see what those eigenvectors look like, we will take an example of a synthetically generated image sequence. Figures (4) shows the images and the computed generalized eigenvectors. As we saw from section 2.1, the eigenvectors are real valued solutions to our *normalized cut* problem. Ideally they should take on discrete values, and as we can see from figure (4) this is indeed the case for the first two smallest eigenvectors. One can see that using these two eigenvectors we can segment out the two moving patches. However, for the third smallest eigenvector, the values in that eigenvector are smoothly varying, and are not close at all to any discrete values. There are many ways of interpreting this phenomena. In the view of segmentation, the third eigenvector is attempting to subdivide the background

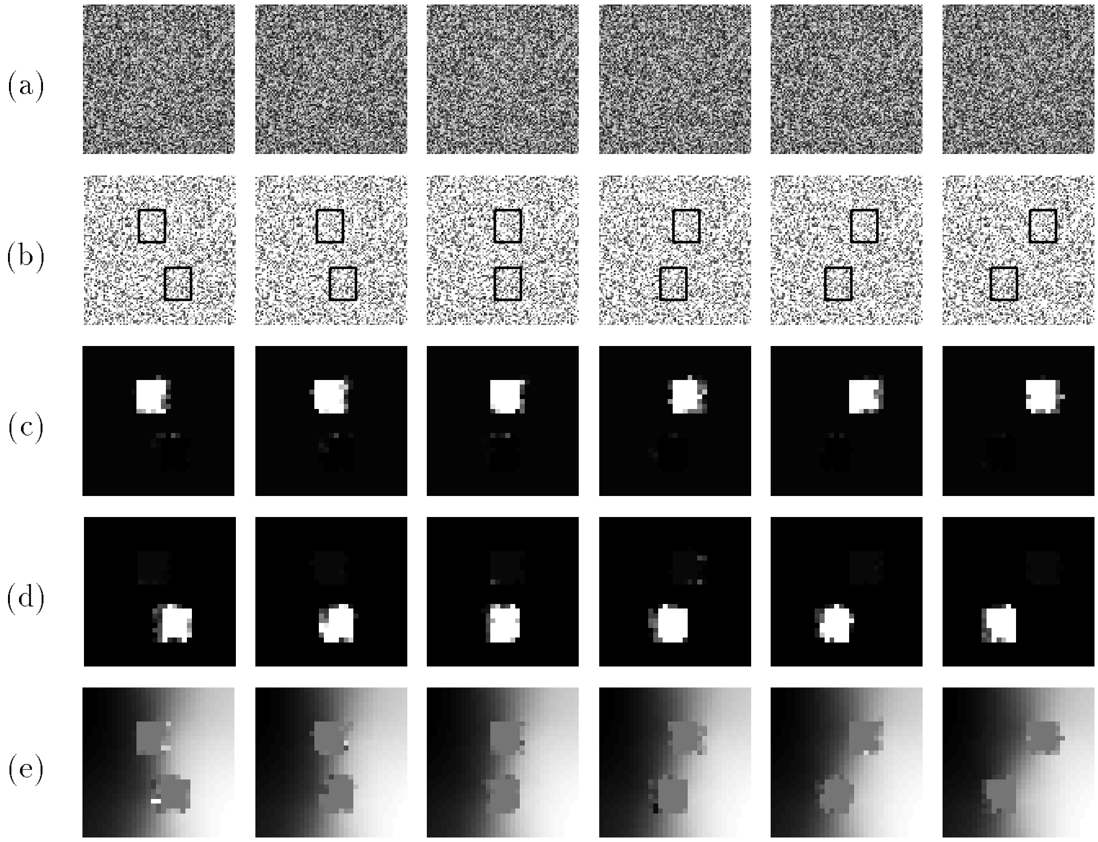


Figure 4: Row (a) of this plot shows the six frames of a synthetic random dot image sequence. Row (b) shows the outlines of the two moving patches in this image sequence. The outlines shown here are for illustration purposes only. Row (c) to (e) show the generalized eigenvectors computed broken into their components in each of the time frame. Only the smallest eigenvectors with eigenvalue less than 0.05 are shown here.

in the "random dot" image sequence. However, there are no real breaks in the background, hence no sure way of breaking it. The smoothly varying nature of the eigenvector values indicates precisely this problem. In fact, if we are forced to partition the image based on this eigenvector, we will see there are many different splitting points which have similar $Ncut$ values. Hence the partition will be highly uncertain and unstable. In our current segmentation scheme, we simply choose to ignore all those eigenvectors which have smoothly varying eigenvector values. We achieve this by imposing a stability criterion which measure the degree of smoothness in the eigenvector values. The simplest measure is based on first computing the histogram of the eigenvector values, and then computing the ratio between the minimum and maximum values in the bins. When the eigenvector values are continuously varying, the values in the histogram bin will stay relatively the same, and the ratio will be relatively high. In our experiments, we find that the eigenvector values either have a distribution with a small number of peaks (4(c), 4(d)) or a continuous one (4(e)). Therefore a simple thresholding on the ratio described above can be used to exclude unstable eigenvectors. We have set that value to be 0.06 in all our experiments.

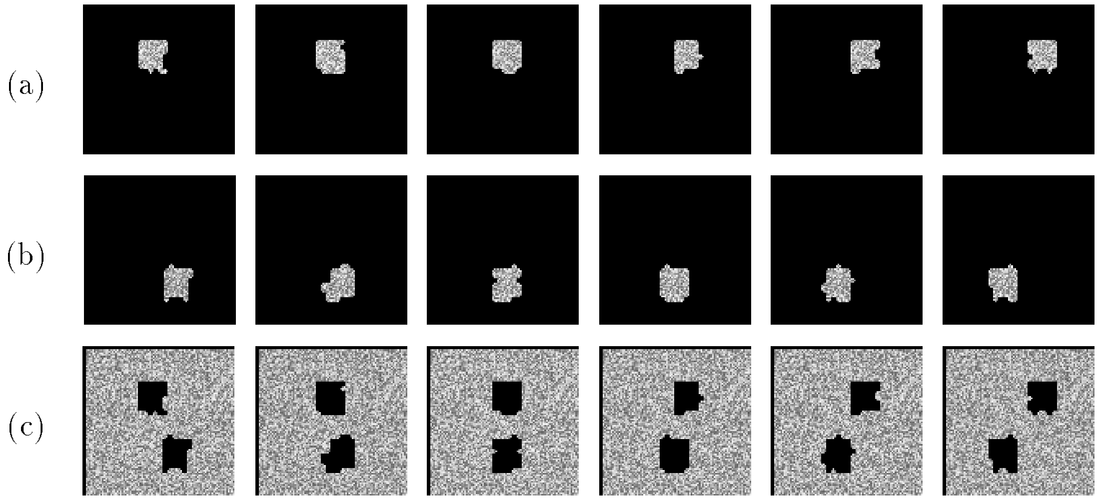


Figure 5: Row (a)-(c) shows the top three partitions of the "random dot" image sequence that have *Ncut* values less than 0.05. The same threshold is used in all our image sequences. The segmentation algorithm produces 3D space-time partitions of the image sequence. Cross-sections of those partitions are shown here. The original image size is 100×100, and the segmentation is computed using image patches(superpixels) of size 3×3. Each image patch is connected to other image patches that are less than 5 superpixels away in spatial distance, and 3 frames away in temporal distance.

In summary, our grouping algorithm can be described as:

1. Given an image sequence, set up a weighted graph $G = (V, E)$ by taking a subsample of the image pixels as the node of the graph(A factor of 3 from the full resolution in our examples). Connect nodes that are less than r_s superpixels apart in space, and r_t frames apart in time. For each pair of nodes compute their motion profiles, and define the weight of the graph edge connecting them as in equation (12). Summarize the information into $\mathbf{W}$, and $\mathbf{D}$. In our experiments, $r_s = 5$ and $r_t = 3$.

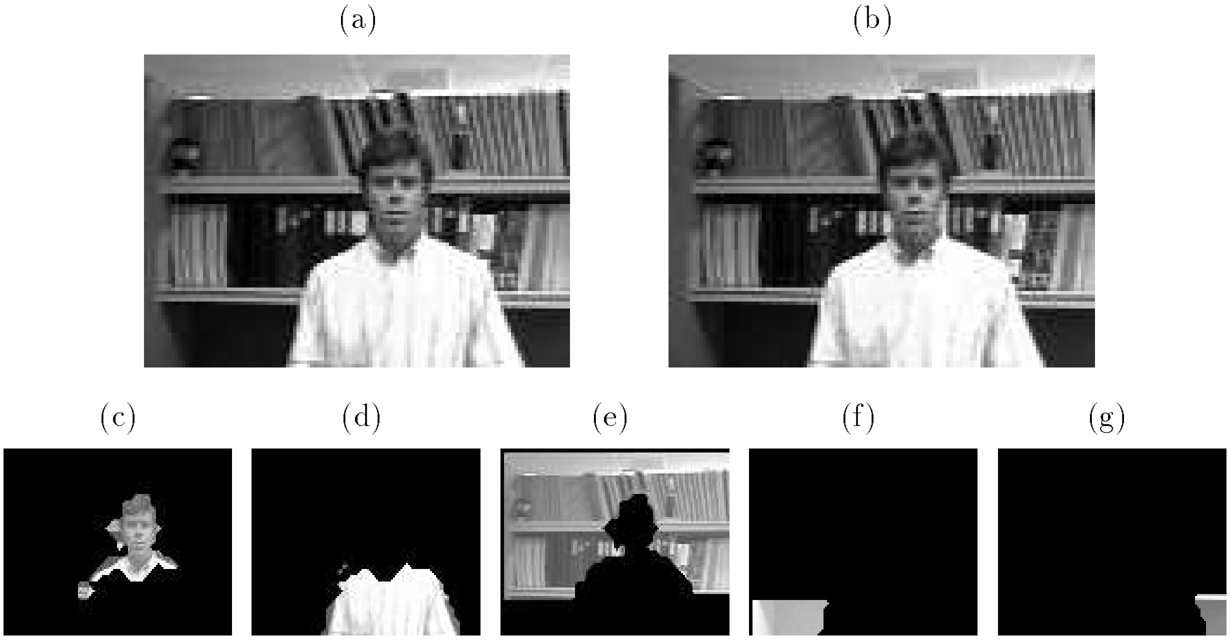


Figure 6: Subimage (a) and (b) show the left and right image of a stereo image pair, and segmentation is shown in subimages (c) to (g). Motion profiles are computed for the image patches(3×3) in left image only, and hence the graph consists of just those nodes from that image. Each of the image patch is connected to other image patches that are less than 5 superpixel away. Segments in (c) and (d) corresponds to the person in the foreground, and segments in (e) to (g) corresponds to the background. The reason that the head of the person is segmented away from the body is that although they have similar motion, their motion profiles are different. As we saw in figure (3) the head region contains 2D textures and the motion profile are more peaked, while in the body region the motion profiles are more spread out. Segment (e) is broken away from (f) and (g) for the same reason.

2. Solve $(\mathbf{D} - \mathbf{W})\boldsymbol{x} = \lambda \mathbf{D}\boldsymbol{x}$ for eigenvectors with the smallest eigenvalues.

3. Use the eigenvector with second smallest eigenvalue to bipartition the graph by finding the splitting point such that $Ncut$ is minimized,

4. Decide if the current partition should be sub-divided by checking the stability of the cut, and make sure $Ncut$ is below pre-specified value,

5. Recursively repartition the segments using the next smallest eigenvectors.

The number of groups segmented by this method is controlled directly by the maximum allowed $Ncut$, which is set to 0.05 in all our experiments.

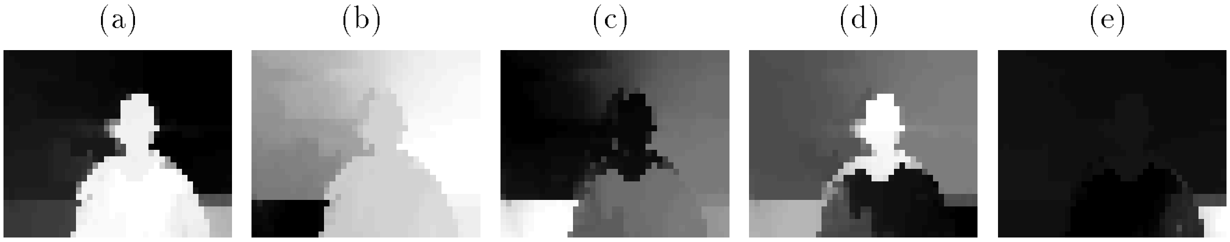


Figure 7: Subplot (a) to (e) shows the 5 smallest generalized eigenvectors with eigenvalue less than 0.05 computed for segmenting the image squence in figure (6). Noticing from the first eigenvector in (a), the foreground person is segmented from the background, while later on using the eigenvector in (d), the head of the person is segmented from the body.

Now we return to our synthetic "random dot" image sequence shown in figure (4). The segmentations are shown in figure (5). As we can see our algorithm produced the correct segmentations, and along with it the right group correspondences throughout the image sequence. The slightly jagged object boundaries are mainly due to low image resolution at which we compute the segmentation.

Figure (6) shows the segmentation result on the two frame image sequence shown in figure(2), and the associated generalized eigenvectors are shown in figure (7).

Figure (8) shows the result of motion segmentation on a seven frame Carl Lewis running image sequence. The image sequence is used because 1) it has very poor image quality: the image noise is very high, and image contrast is relatively low, and 2) it contains an articulated body with different motions on each of the limbs. Note that the camera is panning to keep Carl Lewis in the center of the frame, this results in a moving background which would make background subtraction fail.

We want to see if the algorithm is able to find the most dominate motion blocks in the image sequence. As we can see from the figure (8), we indeed achieve this goal even under poor image conditions.

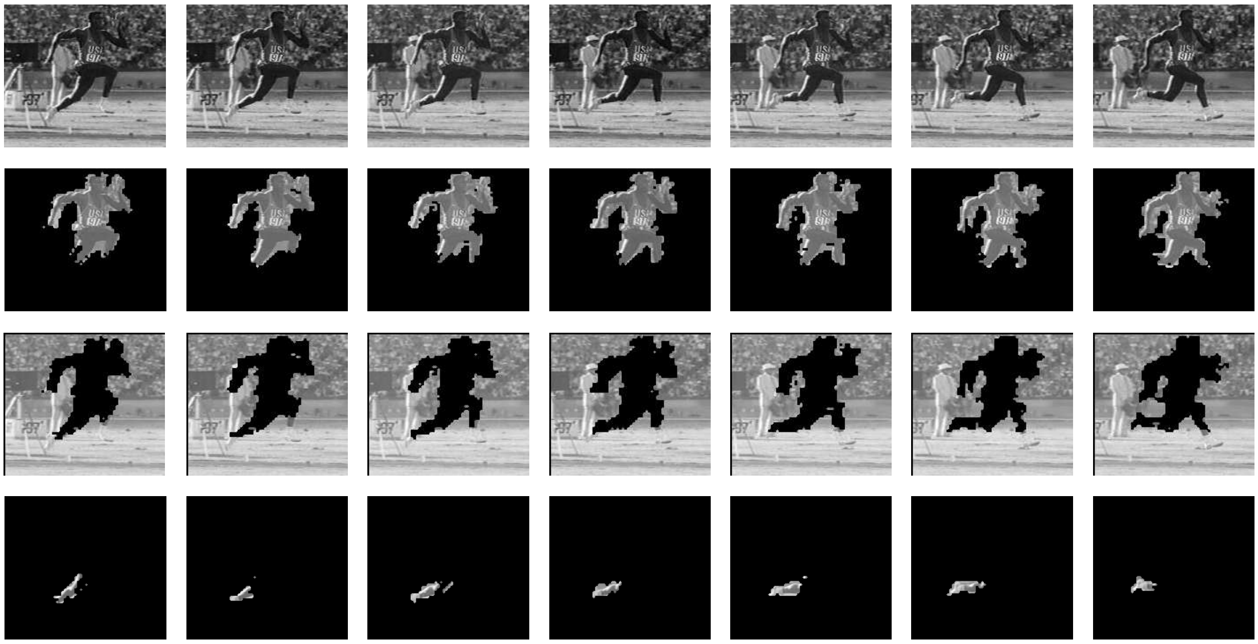


Figure 8: Row (a) shows an image sequence of Carl Lewis running. Notice that the background is moving to the left as the camera is panning to keep the runner in the center of the image, and therefore background subtraction would not work as an image segmentation technique. The original image size is 200×190, and image patches of size 3×3 is used to construct the partition graph. Each of the image patches are connected to others that are less than 5 superpixels and 3 image frames away. Row (b) to (d) show the motion segmentation produced by our algorithm. Note these regions found corresponds the runner in (b), moving background in (c), and the left lower leg in (d). The left lower leg is segmented from the runner because it undergoes significant upward rotation in these seven image frames. By recursive cuts and by lowering the maximum allowed *Ncut* value, the other moving limbs can be found.

4 Segmenting and Tracking Long Image Sequences

The method described above takes in a fixed number of image frames, and produce a segmentation in a *batch* type of operation. The advantage of computing segmentation based on multiple image frames is that one can incorporate information across several frames to produce the best partition. However, this becomes computationally impractical and inefficient if we have to segment a very long image sequence this way.

To solve this problem, we will use only a fixed number of image frames centered around each incoming image frame in the time domain to compute the segmentation. Figure (9) illustrates this idea.

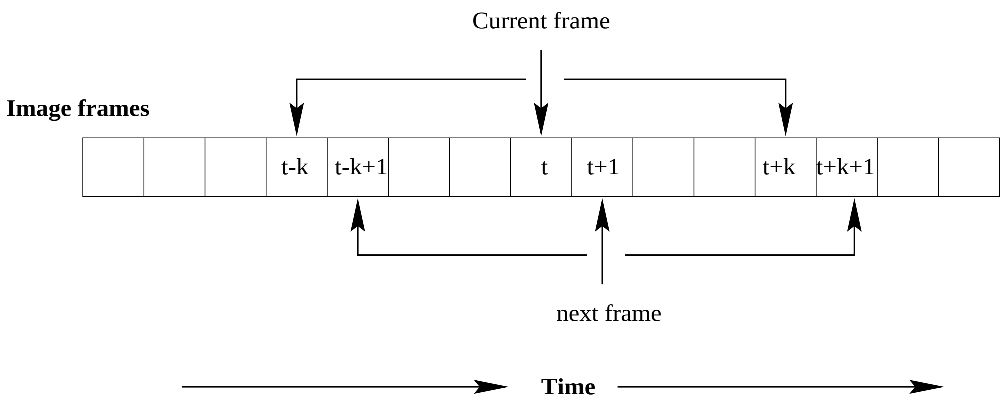


Figure 9: At any given time step t, only +/- k image frames center around t is used to construct the partition graph, for computing the segmentation and group correspondence. At the next time step *t+1*, image frame *t-k* is dropped and *t+k+1* is incorporated in our grouping algorithm.

Because there is a significant overlap of the image frames used to compute the segmentation from one time step to another, we can use it to our advantage to speedup our computation. The place where we can gain most of the speed up is in the step of solving the generalized eigensystem $(\boldsymbol{D}-\boldsymbol{W})\mathbf{x} = \lambda\boldsymbol{D}\mathbf{x}$, or its equivalent form $\boldsymbol{A}\mathbf{z} = \lambda\mathbf{z}$, where $\boldsymbol{A} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{D}-\boldsymbol{W})\boldsymbol{D}^{-\frac{1}{2}}$, and $\mathbf{z} = \boldsymbol{D}^{\frac{1}{2}}\mathbf{y}$. Without going into too much detail, the Lanczos method of computing eigenvector for a sparse matrix is closely related to the problem of computing orthonormal bases for the *Krylov subspace* associated with matrix $\boldsymbol{A}$. If we have a good guess of the vectors that spans that subspace, we can arrive at the solution very quickly. We shall see that this is indeed achieved by our algorithm.

Let $\boldsymbol{A}^t$ denote the matrix constructed at time t from image frame $t-k$ to $t+k$. We can break $\boldsymbol{A}^t$ into submatrixes corresponding the connections between each of the image frames:

$$\boldsymbol{A}^t = \begin{bmatrix} \boldsymbol{A}_{(t-k)(t-k)} & \boldsymbol{A}_{(t-k)(t-k+1)} & \cdots & 0 & \cdots & \\ \boldsymbol{A}_{(t-k+1)(t-k)} & \boldsymbol{A}_{(t-k+1)(t-k+1)} & \boldsymbol{A}_{(t-k+1)(t-k+2)} & \cdots & 0 & \cdots \\ & \vdots & \vdots & \vdots & & \\ \cdots & 0 & \cdots & \boldsymbol{A}_{(t+k-1)(t+k-2)} & \boldsymbol{A}_{(t+k-1)(t+k-1)} & \boldsymbol{A}_{(t+k-1)(t+k)} \\ \cdots & \cdots & 0 & \cdots & \boldsymbol{A}_{(t+k)(t+k-1)} & \boldsymbol{A}_{(t+k)(t+k)} \end{bmatrix}, \tag{13}$$

where each $\boldsymbol{A}_{ij}$ is related to the connections from nodes in frame i to nodes in frame j. Let $\mathbf{z}^t$ be an eigenvector for $\boldsymbol{A}^t$. $\mathbf{z}^t$ can also be broken into sub-vectors corresponding to each of the frames:

$$\mathbf{z}^t = \begin{bmatrix} \mathbf{z}^t_{t-k} \\ \mathbf{z}^t_{t-k+1} \\ \vdots \\ \mathbf{z}^t_{t+k} \end{bmatrix} \tag{14}$$

When the time step is advanced by one frame, the new $\boldsymbol{A}^{t+1}$ is related to the previous $\boldsymbol{A}^t$ by,

$$\boldsymbol{A}^{t+1} = \begin{bmatrix} & & \boldsymbol{A}^t(2:2k+1, 2:2k+1) & 0 \\ & & & \vdots \\ & & & \boldsymbol{A}_{(t+k)(t+k+1)} \\ 0 & \cdots & \boldsymbol{A}_{(t+k+1)(t+k)} & \boldsymbol{A}_{t+k+1}t+k+1 \end{bmatrix}. \tag{15}$$

Let $\mathbf{z}^{t+1}$ be the eigenvector for $\boldsymbol{A}^{t+1}$,

$$\mathbf{z}^{t+1} = \begin{bmatrix} \mathbf{z}^{t+1}_{t-k+1} \\ \vdots \\ \mathbf{z}^{t+1}_{t+k} \\ \mathbf{z}^{t+1}_{t+k+1} \end{bmatrix}. \tag{16}$$

Unless there is a major scene change at time $t+1$, we expect the first $2k$ frames of $\mathbf{z}^{t+1}$ to stay relatively the same. To initialize the component corresponding to the new frame, we can compute $\mathbf{z}^{t+1}_* = A^{t+1}\mathbf{z}$, where $\mathbf{z} = [\mathbf{z}^t_{t-k+1}, \cdots, \mathbf{z}^t_{t+k}, 0]^t$. Intuitively this amounts to interpolating from the eigenvectors from the previous frames to obtain a guess at the eigenvector for the frame $t+k+1$. $\mathbf{z}^{t+1}_*$ is then input to the Lanczos eigenvector solver to speed up the computation.

To test our method, we used the standard "flower garden" sequences as used by[AS95]. In this experiment, we set the half-width of the time window, k, to 2 frames. Figure (10) shows the first five frames of the image sequence along with the motion segmentation. The generalized eigenvectors are shown in figure (11). With the sliding time window updating mode, we see the cost of computing the generalized eigenvectors at the each of the new time step amounts to only 20% of cost we would incur if we had done it in the batch mode. Figure (12) shows the 15th to the 18th image frame along with the motion segmentation. Their generalized eigenvectors computed by methods described above are shown in figure (13).

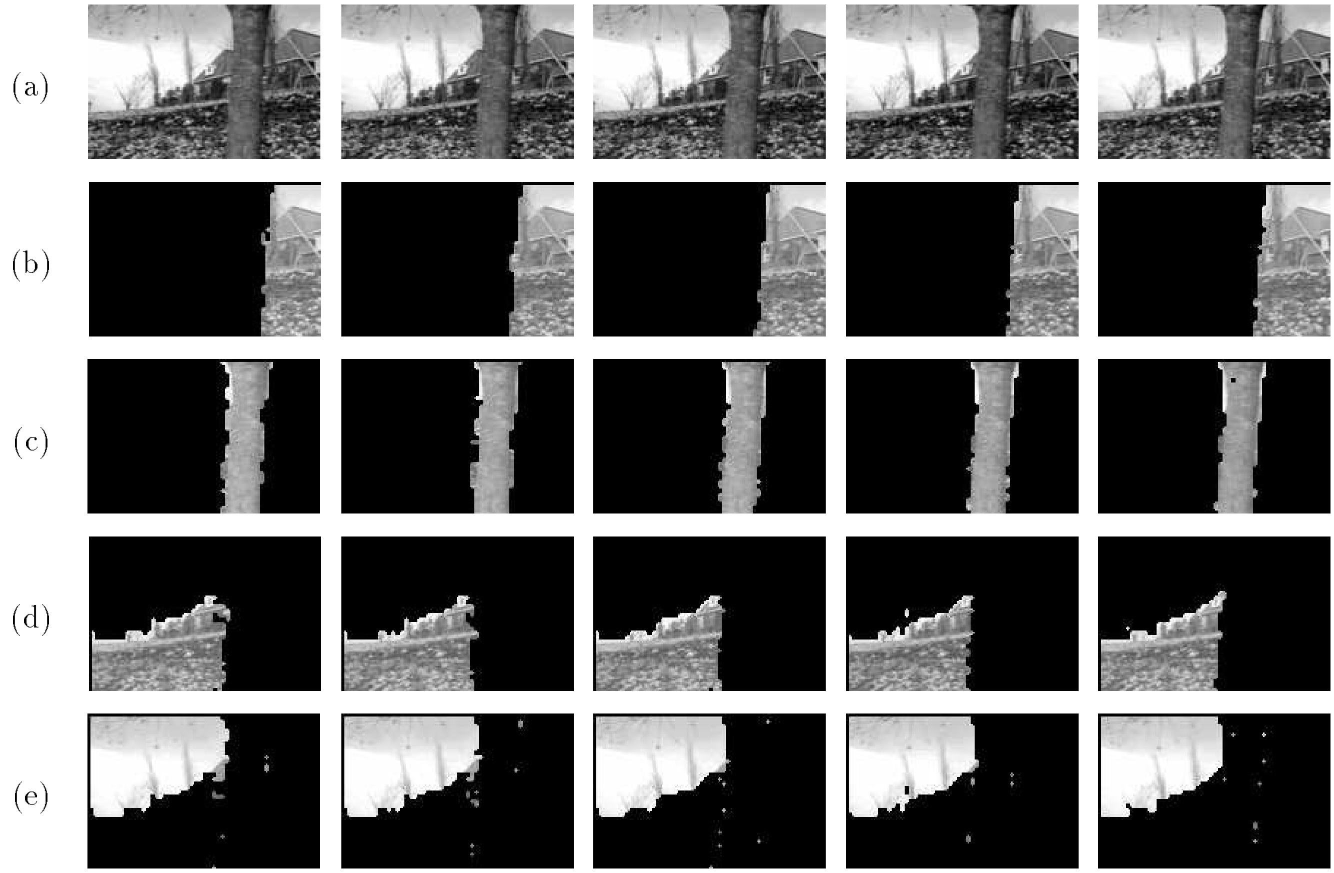


Figure 10: Row (a) shows six frames of the "flower garden" sequence. The original image size is 120×175, and image patch of size 3×3 is used to construct the partition graph. Each of the image patches are connected to others that are less than 5 superpixels and 3 image frames away. Row (b) to (e) shows the segmentation produced by our algorithm. Parts (b) and (e) are consistent with the same global motion model and can be merged in a postprocessing step if so desired.

5 Conclusion

In this paper, we have developed a motion segmentation algorithm based on the *normalized cuts* graph partitioning method. We treat the image sequence as a three dimensional spa-

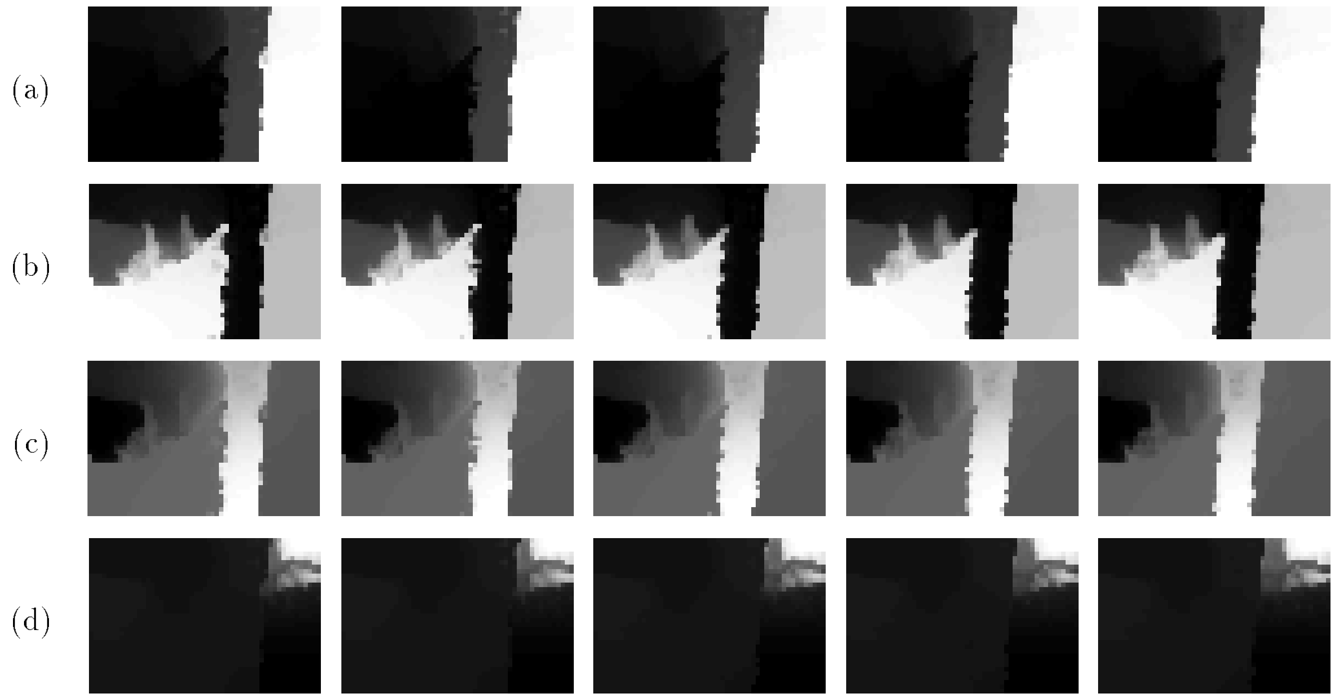


Figure 11: Row (a) to (d) shows first four generalized eigenvectors computed for segmenting the "flower garden" image sequence. Those eigenvectors are broken into components corresponding to each of the image frames seen in figure (10).

tiotemporal data set and construct a weighted graph by taking each pixel as a node, and connecting pixels that are in the spatiotemporal neighborhood of each other. We define a motion profile vector at each image pixel which captures the probability distribution of the image velocity at that point. By defining a distance between motion profile at two pixels, we can assign a weight on the graph edge connecting them. Using *normalized cuts* we find the most salient partitions of the spatiotemporal volume formed by the image sequence. Each partition, which is in the form of a spatiotemporal volume, corresponds to a group of pixels moving coherently in space and time. We have also developed a recursive technique for segmenting and tracking long image sequences. Experimental results on various synthetic and real image sequences are presented.

There are also various extensions to this work which we are currently working on:

1. Refining segmentation to a finer resolution. In our experiments, we have subsampled the image sequence spatially to reduce the computational cost. By projecting our

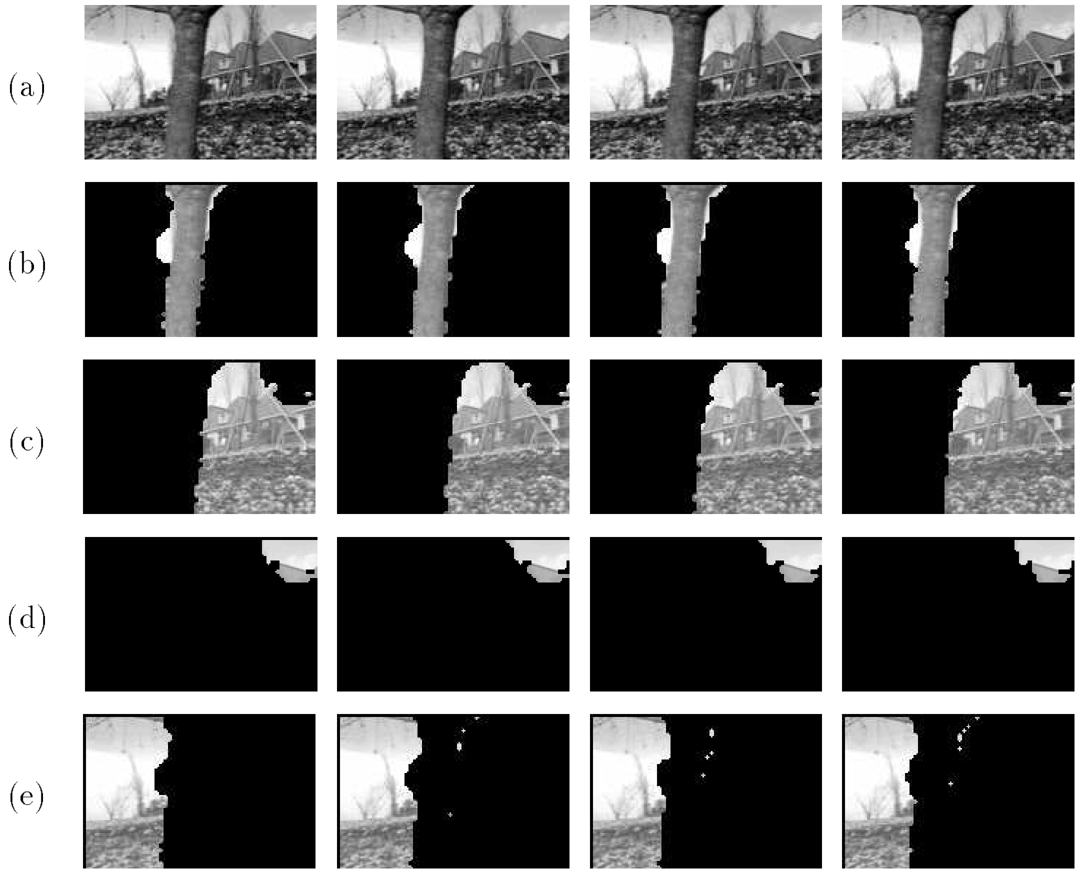


Figure 12: Row (a) shows the 15th to the 18th frame of the "flower garden" sequence. Row (b) to (e) show the motion segmentation computed by our algorithm with the sliding time window method. The eigenvectors from which this segmentation is produced are shown in figure (13).

segmentation back into the full resolution image space, we can use simple local search algorithms from the graph partitioning community such as the Kernighan and Lin algorithm[KL70] to refine the boundaries in our segmentation.

2. Incorporating multiple cues. Our grouping framework permits us to define the weight of a graph edge connecting two pixels in the image sequence using color or texture similarity[SM97] in addition to motion profile similarity. We have demonstrated the effectiveness of segmentation algorithms based on each of those cues in isolation. Combining them is a natural next step.

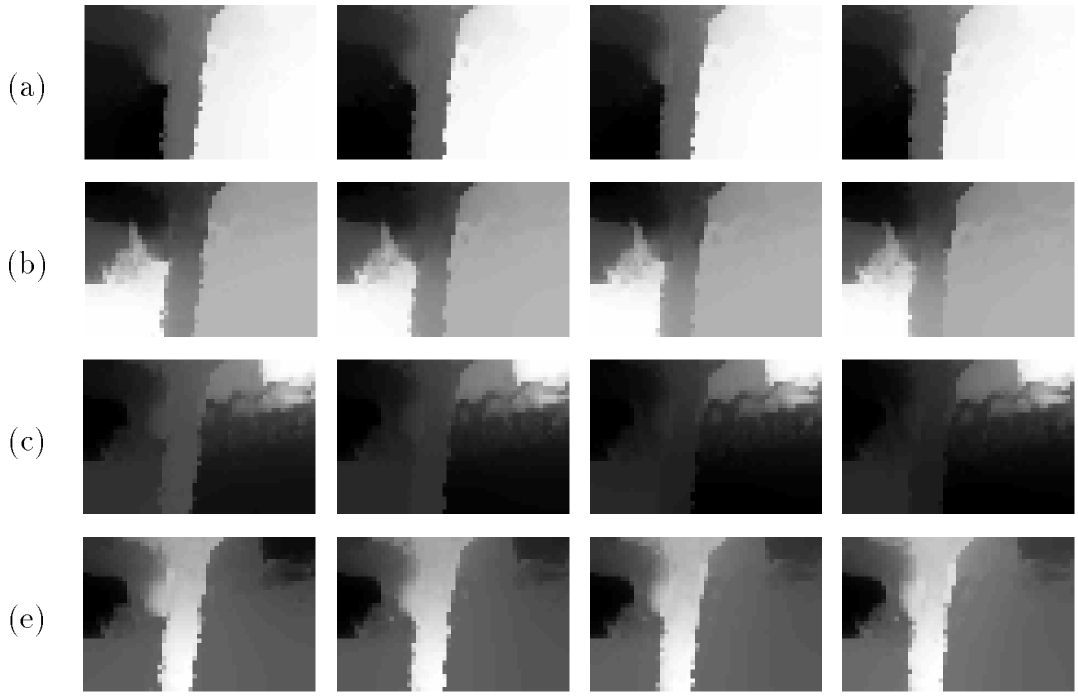


Figure 13: Row (a) to (d) shows first four generalized eigenvectors computed for frame 15 to 18 of the image sequence using the sliding window updating method. Those eigenvectors are broken into components corresponding to each of the image frames. Notice that there are changes in the eigenvectors as well as their ordering due the scene change, and our algorithm correctly handles this situation.

3. Measuring group motion explicitly. Given the group correspondence, we can estimate the image velocity more reliably for image patches within each the groups, using the knowledge of where the motion boundaries are located. Those measurements can then be used to reason about occlusion and disocclusion. Furthermore, our way of defining motion segments from the "big picture" downwards can provide a natural framework for describing articulated body motions in which the main motion of the whole body will be first extracted, and then the relative motions of the limbs could be computed by recursively subpartitioning the body. This should prove to be a great asset in human activity recognition.

References

[AB85] E. Adelson and J. Bergen. Spatiotemporal energy models for the perception of motion. *J. Opt. Soc. Am.*, 2(2):284–299, 1985.

[Ana89] P. Anandan. A computational framework and an algorithm for the measurement of vision motion. *Int. J. of Computer Vision*, 2:283–310, 1989.

[AS95] Serge Ayer and Harpreet S. Sawhney. Layered representation of motion video using robust maximum-likelihood estimation of mixture models and mdl encoding. In *Int. Conf. Computer Vision*, pages 777–784, Cambridge, MA., 1995.

[BA91] M. Black and P. Anandan. Robust dynamic motion estimation over time. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 296–302, 1991.

[CB92] R. Cipolla and A. Blake. Surface orientation and time to contact from image divergence and deformation. In *Second European Conference on Computer Vision*, pages 187–203, 1992.

[DH73] W.E. Donath and A.J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, pages 420–425, 1973.

[DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(B), 1977.

[DP91] T. Darrell and A. Pentland. Robust estimation of a multi-layered motion representation. In *IEEE Workshop on Visual Motion*, pages 173–178, 1991.

[DR95] R. Van Driessche and D. Roose. An improved spectral bisection algorithm and its application to dynamic load balancing. *Parallel Computing*, 21:29–48, 1995.

[Fie75] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. *Czech. Mathematical Journal*, 25(100):619–633, 1975.

[FJ90] D. Fleet and A. Jepson. Computation of component image velocity from local phase information. *Int. J. of Computer Vision*, 5:77–104, 1990.

[GL89] G. Golub and C. Van Loan. *Matrix computations.* John Hopkins Press, 1989.

[HAP94] S. Hsu, P. Anandan, and S. Peleg. Accurate computation of optical flow by using layered motion representation. In *12th International Conference on Pattern Regcognition*, 1994.

[Hee87] D. Heeger. Optical flow from spatiotemporal filters. In *Proceedings of the First International Conference on Computer Vision*, pages 181–190, 1987.

[HS81] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[JB93] A. Jepson and M. J. Black. Mixture models for optical flow computation. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 760–761, 1993.

[KL70] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 1970.

[Koe86] J.J. Koenderink. Optical flow. *Vision Research*, 26(1):161–179, 1986.

[KYSK84] K.Fukunaga, S. Yamada, H.S. Stone, and T. Kasai. A representation of hypergraphs in the euclidean space. *IEEE Trans. Comput.*, C-33:364–367, April 1984.

[LK81] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proc. 7th Int. Joinnt Conf. on Art. Intell.*, 1981.

[NE86] H. Nagel and W. Enkelmann. An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8:565–593, September 1986.

[PSL90] A. Pothen, H.D. Simon, and K.P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Anal. Appl.*, 11:430–452, 1990.

[Sim93] E. P. Simoncelli. *Distributed Representation and Analysis of Vision Motion*. PhD thesis, MIT Media Laboratory, 1993.

[SM97] J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[ST94] J. Shi and C. Tomasi. Good features to track. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, 1994.

[ST96] D.A. Spielman and S.H. Teng. Disk packings and planar separators. In *Proceedings of 12th ACM Symposium on Computational Geometry*, May 1996.

[TM94] P. H. S. Torr and D. W. Murray. Stochastic motion clustering. In *Proc. 3rd European Conf. on Computer Vision, Stockholm*, 1994.

[WA94] J.Y.A. Wang and E.H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing Special Issue: Image Sequence Compression*, pages 625–638, 1994.

[WA96] Y. Weiss and E.H. Adelson. A unified mixture framework for motion segmentation: Incorporating spatial coherence and estimating the number of models. In *CVPR*, 1996.

[Wei97] Y. Weiss. Smoothness in layers: Motion segmentation using nonparametric mixture estimation. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, 1997.

[Wer38] M. Wertheimer. *A Sourcebook of Gestalt Psycychology(Partial translation)*, chapter Laws of Organization in Perceptual Forms(1923), pages 71–88. Harcourt, Brace and Company, 1938.

[WL93] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *PAMI*, 11:1101–1113, November 1993.