Long-Range Video Motion Estimation using Point Trajectories

by

Peter Sand

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

July 2006

Author . . . .
Department of Electrical Engineering and Computer Science
July 31, 2006

Certified by . . . .
Seth Teller
Associate Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Long-Range Video Motion Estimation using Point Trajectories

by

Peter Sand

Submitted to the Department of Electrical Engineering and Computer Science
on July 31, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis describes a new approach to video motion estimation, in which motion is represented using a set of particles. Each particle is an image point sample with a long-duration trajectory and other properties. To optimize these particles, we measure point-based matching along the particle trajectories and distortion between the particles. The resulting motion representation is useful for a variety of applications and differs from optical flow, feature tracking, and parametric or layer-based models. We demonstrate the algorithm on challenging real-world videos that include complex scene geometry, multiple types of occlusion, regions with low texture, and non-rigid deformation.

Thesis Supervisor: Seth Teller
Title: Associate Professor

Acknowledgments

Thank you to Seth Teller for numerous suggestions and discussions. Many thanks to the other members of the thesis committee, Bill Freeman and Berthold K.P. Horn, for their time and feedback. Thank you to Matt Brand for discussions and support of early experiments in long-range motion estimation. Thanks to Frédo Durand, Jovan Popović, Bryt Bradley, Tom Buehler, and other members of the MIT Computer Graphics Group. Thank you also to the faculty and students in Computer Vision and other areas of Computer Science. Thank you to the staff of EECS and MIT. Thank you also to National Science Foundation for a fellowship supporting this research.

I especially wish to acknowledge my father Paul Sand, mother Peggy Booth, and the memory of Carl Sand.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Video is typically represented as a sequence of images. When these images are obtained from a camera observing the real world, they have substantial consistency and redundancy. However, the relationships between the images are not captured explicitly. A video representation consisting of a large number of pixel values does not adequately describe the underlying processes that generate the pixels: geometry, motion, light, and reflectance.

Our goal is to take a step towards a content-aware video representation by finding correspondences between video frames. This problem is typically addressed using feature tracking [71] or optical flow [9]. Feature tracking follows a set of salient image points over many frames, whereas optical flow estimates a dense vector field mapping one frame to the next. Our goal is to combine these two approaches: to produce motion estimates that are both spatially dense and temporally long-range (Figures 1-1 and 1-2). For any image point, we would like to know where the corresponding scene point appears in all other video frames (until the point leaves the field of view or becomes permanently occluded).

Our approach represents video motion using a set of particles that move through time. Each particle denotes an interpolated image point sample, in contrast to a feature patch that represents a neighborhood of pixels [71]. Particle density is adaptive, so that the algorithm can model detailed motion with substantially fewer particles than pixels.

The algorithm optimizes particle trajectories using an objective function that combines point-based image matching and inter-particle distortion. The algorithm extends and truncates particle trajectories to model motion near occlusion boundaries.

Our contributions include posing the particle video problem, defining the particle video representation, and presenting an algorithm for building particle videos. We provide a new motion optimization scheme that combines variational techniques with an adaptive motion representation. The algorithm uses weighted links between particles to implicitly represent grouping, providing an alternative to discrete layer-based representations. We present a complete application algorithm that demonstrates the use of particles for interactive video manipulation.

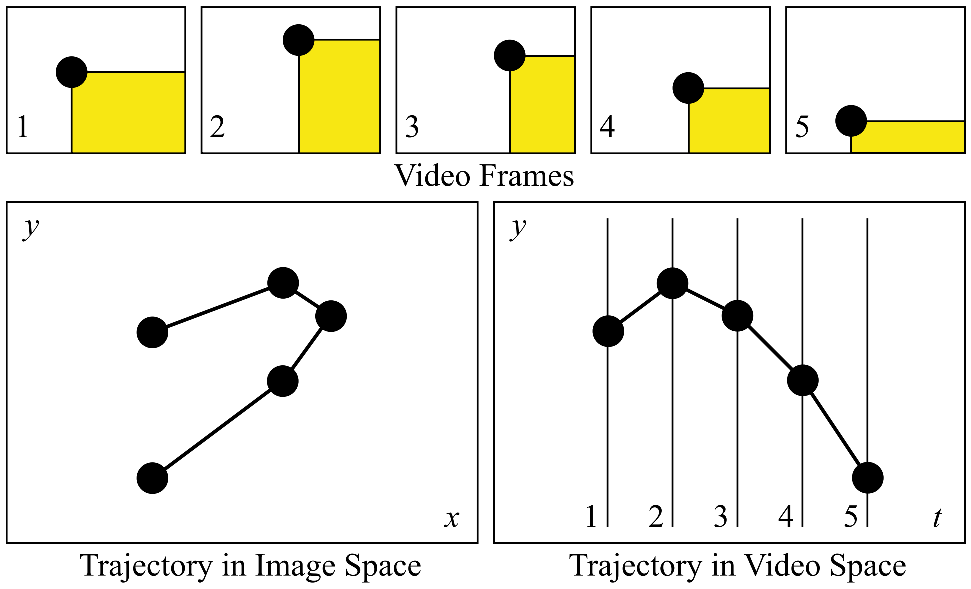


Figure 1-1: The top images show a world-space feature that appears in the frames of a video. This feature defines a trajectory through a video cube (a sequence of video frames stacked such that time forms a third dimension). We can view this trajectory in image coordinates (lower left) or spatio-temporal coordinates (lower right).

1.1 Motivation

Finding pixel correspondences is a step toward a much larger goal of decomposing video into physical components (geometry, motion, light, and reflectance). This kind of decomposition has many applications in robotics, surveillance, and human-computer interaction. As memory, bandwidth, and imaging electronics become cheaper, we have begun to find cameras in more locations (cars, robots, offices, streets, cell phones, etc.). These cameras

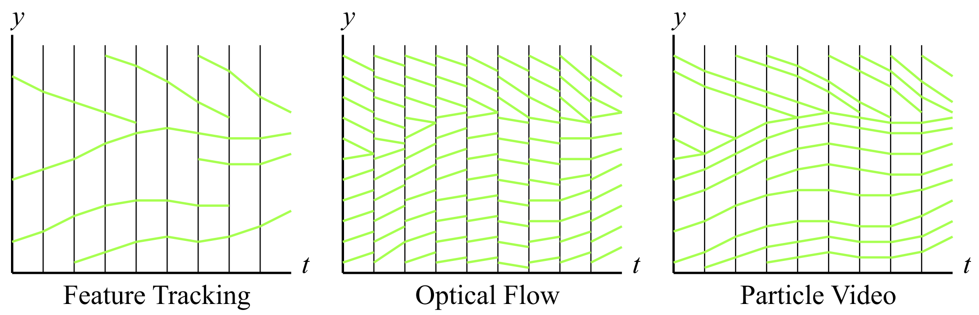


Figure 1-2: Each diagram represents point correspondences between frames of a hypothetical sequence. Feature tracking is temporally long-range and spatially sparse. Optical flow is temporally short-range and spatially dense. Our particle video representation is both temporally long-range and spatially dense.

provide a flood of information, but for this information to be useful and manageable, it should be properly decomposed.

Of the potential applications for this work, this thesis will focus on one class of applications: the manipulation of video for synthesizing new video. We would like to explore how video correspondences facilitate manipulation of video content.

Examples of video manipulation include copying objects from one video to another, removing objects, changing the timing of some video elements with respect to others, modifying reflectance, and stabilizing the camera viewpoints. Multiple observations of each scene point can be combined for super-resolution [5, 32, 59, 95], noise-removal [83, 50, 10], segmentation [23], and increased effective dynamic range [25, 46, 10]. The correspondences can also improve the temporal coherence of image filters that operate independently on each frame. Additionally, long-range motion estimation can simplify interactive video manipulation, including matting [21, 52, 85], rotoscoping [1, 88], and object removal [92, 77, 24].

With existing software, many of these tasks require substantial user interaction, due to the redundant nature of standard video representations. When the user wants to make a change, many corresponding changes must also be made. We seek to minimize this manual intervention.

Nonetheless, we recognize that some manual intervention is required for creative decisions. Video manipulation by definition requires some specification of how the video is to be manipulated (how it is transformed from an input video to an output video). This involves human interaction that could be as simple as picking a new viewpoint or as complex as manually modifying the appearance, arrangement, and timing of numerous components of the video.

We focus on creative video applications for a variety of reasons. The initial motivation is a personal interest in artistic filmmaking. Applications in filmmaking provide challenging research problems that require solutions with a high standard of quality. People from non-film backgrounds are attracted to these applications because of widespread exposure to visual media such as films, music videos, and television commercials. A visual demonstration of an algorithm with visual inputs and visual outputs could also motivate people to explore non-visual aspects of computer science. The proper manipulation of video can create new imagery that is compelling to the human mind and visual system, yet could not otherwise be observed.

1.2 Design Goals and Assumptions

The particle video problem can be described as dense feature tracking or long-range optical flow. We want to track the trajectory of each pixel through a given video. Ideally each trajectory would correspond to the motion of a physical real-world point (or surface patch).

Our primary goal is the ability to model complex motion and occlusion. We want the algorithm to handle general video, which may include close-ups of people talking, handheld camera motion, multiple independently moving objects, textureless regions, narrow fields of view, and complicated geometry (e.g. trees or office clutter).

A particle approach provides this kind of flexibility. Particles can represent complicated geometry and motion because they are small; a particle's appearance will not change as rapidly as the appearance of a large feature patch, and it is less likely to straddle an occlusion boundary. Particles represent motion in a non-parametric manner; they do not assume that the scene consists of planar or rigid components.

To avoid being ill-posed, flexible systems often need to be augmented with constraints, which motivates another design decision: consistency is more important than correctness.

If the scene includes arbitrary deforming objects with inadequate texture, finding the true motion may be hopeless. Typically, this problem is addressed with geometric assumptions about scene rigidity and camera motion. Instead, we simply strive for consistency; for example, that red pixels from one frame are matched with red pixels in another frame. For many applications, this kind of consistency is sufficient (and certainly more useful than outright failure due to non-uniqueness).

This flexibility in modelling complex motion can also be achieved by optical flow, but the optical flow representation is best suited to successive pairs of frames, not to long sequences (Figure 1-3). Frame-to-frame flow fields can be concatenated to obtain longer-range correspondences, but the resulting multi-frame flow must be refined at each step to avoid drift.

In contrast, the particle representation allows a form of random-access motion evaluation: given a set of particles, we can easily find correspondences between any pair of frames (assuming the frames have a sufficient number of particles in common). Furthermore, unlike a sequence of motion fields, the particle representation provides discrete motion primitives, which are valuable for subsequent use of the motion information, as shown in Chapter 5.

1.3 Overview

Chapter 2 describes related work in video motion estimation. We combine several of these previous methods to create an optical flow algorithm described in Chapter 3. This algorithm provides a starting point for our particle-based motion estimation.

The particle video algorithm is explained in Chapter 4, which describes how particles are added, propagated, linked, optimized, and pruned. These steps are performed as the algorithm sweeps back and forth across a video, constructing a complete particle representation of the video's motion.

Chapter 5 applies the particle video algorithm to a real-world problem: interactive selection of time-varying video regions (for video matting, filtering, and other manipulations). This application demonstrates the practical value of the particle approach in terms

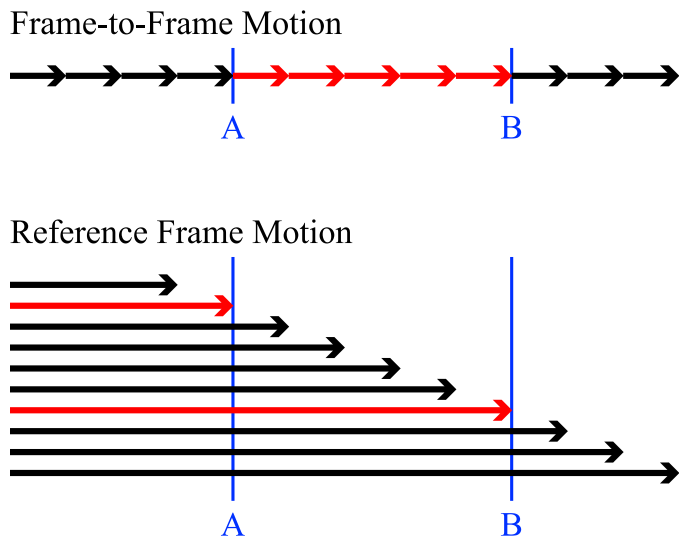


Figure 1-3: Each diagram shows optical flow fields between different frames (increasing left-to-right). To find correspondences between frames A and B, the frame-to-frame fields (red, top) can be concatenated, but this requires an expensive refinement process to avoid drift. Alternately, correspondences can be obtained from reference-frame flow fields (red, bottom), but these do not provide information about regions that do not appear in the reference frame. In contrast, the particle video representation provides efficient and complete long-range correspondences; we simply select the particles that occur in common between the two frames.

of computational efficiency, algorithm design, and user interaction.

Chapter 6 provides an evaluation of the particle video algorithm on a variety of real-world videos. We quantify the performance of the algorithm and possible alternatives. We provide several mechanisms for visualizing the algorithm's results and measuring its performance.

Chapter 2

Related Work

Finding correspondences between two or more images is one of the most studied subjects in computer vision. The approach most closely related to our work is optical flow, as detailed in Section 2.1. In Section 2.2 we describe other motion estimation problems, such as stereo reconstruction, ego-motion estimation, feature tracking, and 3D scene reconstruction.

2.1 Optical Flow

Optical flow is the apparent motion between a pair of images [38, 9, 8]. The estimation of optical flow is usually performed with coarse-to-fine optimization based on local image gradients, using some form of regularization. The problem is difficult because of a lack of constraints (i.e. the aperture problem) and insufficient spatiotemporal sampling, especially near occlusion boundaries.

2.1.1 Parametric and Semi-Parametric Optical Flow

One way to overcome optical flow uncertainty is to assume that the motion can be captured by a set of simple parametric models. Many of these approaches perform simultaneous estimation and segmentation of image motion [70, 57, 84]. Some methods allow components to spatially overlap, in order to model transparency and fragmented occlusion [44, 15, 41]. In other cases, correspondences are refined by finding deviations (e.g. due to depth) from the simple motion models [43, 34]. These methods can successfully identify multiple moving objects, but typically produce poor object boundaries, because disjoint parametric models do not adequately describe real-world motion.

Wills and Belongie [93] present a somewhat more sophisticated approach, in which a layer-based flow algorithm is initialized using feature correspondences. This allows the algorithm to find dense, long-range correspondences that model large deformations. The algorithm addresses the case of two widely separated views, but does not provide a mechanism for finding correspondences over many views. It also focuses on scenes with simple geometry (that can be described by a few coherent layers), whereas we hope to address more complicated scenes.

2.1.2 Variational Optical Flow

Variational optical flow methods estimate dense motion fields using continuous differential objective functions. These functionals are equivalent to many of the energy functions used by other flow algorithms (in fact, the original optical flow algorithm by Horn and Schunck [38] is variational).

As computational power increases, more complex variational methods have become feasible. Recent work includes higher-order data constancy constraints and robust data penalty functions [89]. These methods use a variety of regularization terms, including discontinuity-preserving smoothness (driven by image edges and/or flow edges) with an assortment of robust penalty functions [90]. The algorithms also vary in the methods of optimizing the functional, including different linearizations, multi-grid accelerations, and linear system solvers [19, 89].

Brox *et al.* [18, 63] present a variational approach that provides a robust data term and robust spatio-temporal regularization. They focus on mathematical methods for easing the optimization without the overuse of approximation. Because the algorithm makes few simplifications of its functional, the algorithm is highly successful, though computationally expensive. As discussed in Section 2.1.4, the algorithm does not provide a good model of occlusion boundaries.

2.1.3 Multi-Frame Optical Flow

Some methods make use of multiple frames, aiming to disambiguate motion boundaries through additional data. When estimating motion over many frames, these methods may be more computationally efficient than computing flow independently for each frame.

Most multi-frame optical flow methods rely on some form of temporal coherence assumption [8]. Black and Anandan [13] use a basic temporal smoothness constraint as part of a method that provides robustness in the data terms and spatial smoothness terms. Black [14] subsequently presents a method that adapts to temporal disruptions. Chin *et al.* [20] use an approximate Kalman filter to model temporal variations within a differential flow estimation algorithm. Elad and Feuer [27] present a differential estimation technique with decaying temporal constraints. Farnebäck [28] uses oriented structures in a spatiotemporal video volume as the basis for locally parametric flow estimation. Shi and Malik [70] use multiple frames to aid the segmentation and estimation of distinct motions.

For real-world video sequences, the temporal smoothness assumption is often violated. Some sharp motion changes (e.g. due to hand-held camera operation) can be reduced by whole-frame stabilization algorithms. However, other fast motions (such as someone walking or talking) cannot be stabilized. These motions violate temporal smoothness assumptions because of the limited time-domain sampling found in most videos.

Flow rank methods provide a substantially different approach, with the advantage of not relying on assumptions of spatial or temporal smoothness. Irani [40] shows that matrices of flow components are geometrically restricted to lie in low-dimensional subspaces. Using these constraints, she presents an algorithm to simultaneously estimate flow over multiple frames. Brand [16] applies a similar approach to non-rigid scenes by describing deformable objects as linear combinations of basis shapes. Unfortunately, these constraints are only valid for weak perspective or short windows in time. Nonetheless rank constraints could be incorporated into particle video estimation.

2.1.4 Occlusion Detection for Optical Flow

Occlusion modelling is the most difficult part of estimating optical flow. All optical flow algorithms rely on spatial agglomeration of information, but this information may be misinterpreted when combined from both sides of an occlusion boundary. Furthermore, the core assumption of most flow algorithms is that each pixel goes somewhere, when in fact some pixels may disappear due to occlusions.

A common way of handling occlusion boundaries is robustness in the data and smoothness terms [15, 18]. This robustness allows an algorithm to cope with assumption violations that occur near flow discontinuities. In the data term, a robust distance function allows occluded pixels to mismatch. In the smoothness term, a robust distance function allows discontinuities in the flow field. Because of this robustness, these algorithms fail gracefully near occlusion boundaries, but they still fail. Methods that use anisotropic regularization (whether robust or not) [89, 8], similarly fail to model the process of occlusion.

Amiaz and Kiryati [3] use level sets (rather than standard regularization) to refine the localization of the Brox *et al.* [18] occlusion boundaries. By defining an explicitly piecewise smooth objective, optimized as a post-process to the Brox *et al.* algorithm, the error near the boundaries is reduced. However, the algorithm still does not account for pixels that disappear.

Thompson [81] explores occlusion boundaries in more depth. He describes several of the difficulties with traditional boundary handling. He argues that, even though flow estimates are regularized, the underlying point estimates can be seriously corrupted near occlusion boundaries, because they usually have some spatial extent. (Computing a derivative always requires more than one pixel.) Also, he explains, if the boundary itself has good motion estimates, the maximal flow gradient will systematically mislocate the boundary to be over the occluded surface. Thompson proceeds by presenting an algorithm that addresses some of these problems. His algorithm explicitly identifies the direction of occlusion at each boundary. The algorithm also uses flow and boundary projection based on assumptions of temporal continuity. The main limitation of Thompson's method is that it only estimates motion at image edges, ignoring valuable but subtle image textures.

Zitnick *et al.* [96] estimate optical flow using correspondences between segmented image regions. Like particles, these segments provide small, simple, discrete motion entities. The algorithm estimates blending between segments in order to model mixed pixels at occlusion boundaries. The segments provide well-defined occlusion boundaries between objects of different colors, but the algorithm fails when motion boundaries do not coincide with segment boundaries. Also, the algorithm does not account for segments that become fully occluded.

Because occluded pixels violate a basic assumption of optical flow (that each pixel goes somewhere), several methods attempt to identify occluded pixels explicitly. Silva and Victor [73] use a pixel dissimilarity measure to detect brightness values that appear or disappear over time. Alvarez *et al.* [2] present an algorithm that simultaneously computes forward and reverse flow fields, labelling pixels as occluded where the two disagree. Strecha *et al.* [75] treat occlusion labels as hidden variables in an EM optimization. In this case, pixel value mismatches (rather than flow mismatches) are used to identify occlusions. The occluded pixels modulate anisotropic regularization, such that flow values to do not diffuse across occluded regions.

Xiao *et al.* [94] also use pixel value mismatches to detect occluded regions across which flow diffusion is restricted. They regularize flow estimates using a bilateral filter that incorporates flow from neighboring pixels that are similar in motion and appearance and that lie outside occluded regions. The resulting algorithm is relatively successful at identifying occlusion boundaries and computing accurate flow on both sides of such boundaries. We incorporate some elements of this bilateral filter into the flow algorithm described in Chapter 3.

2.2 Other Motion Estimation Techniques

Other motion estimation algorithms also have elements in common with the algorithm described in this thesis, including stereo reconstruction, feature tracking, estimation of camera motion, and generalized 3D scene reconstruction.

2.2.1 Stereo Reconstruction

Stereo reconstruction is a special case of optical flow in which known camera poses limit the pixel matching to a single direction along single dimension. Because the stereo problem is easier than the optical flow problem, researchers place a greater emphasis on the accuracy of the results, especially near occlusion boundaries. Recent stereo algorithms [45, 48, 69] reduce the search for occlusion boundaries to a graph cut problem [31]. To further improve occlusion handling, Kang *et al.* [45] use shifted matching windows and dynamically selected subsets of multiple input frames.

The graph cut formulation is a limited case of a Markov Random Field [69]; each pixel is given a probability of having each label and a conditional probability of having a label given the labels of its neighbors. Unlike most graph-cut stereo algorithms, we avoid representations based on discrete layers (instead using a weighted linking structure).

Both stereo and optical flow estimation present an issue of matching pixels. Pixels are difficult to match because they are merely samples of the true image (samples produced by integrating incoming light over small sensor regions). In the context of stereo, Birchfield and Tomasi [12] overcome translation-induced pixel sampling artifacts by comparing a pixel with a small neighborhood of pixels in another image. Szeliski and Scharstein [79] provide several extensions to Birchfield and Tomasi's sampling insensitive dissimilarity measure. Kutulakos [49] matches pixels within a given radius of an image location, in order to obtain a degree of spatial invariance. Several stereo algorithms [78, 97] attempt to model mixed foreground/background pixels at occlusion boundaries. In the future we would like to explore these kinds of techniques for particle/pixel matching.

2.2.2 Feature Tracking

Like our particle video approach, feature tracking involves finding trajectories of scene points as they move through a sequence of images. Unlike particles, these features are selected to be distinctive textured patches. Most feature point selectors [35, 60, 71] choose patches that have significant texture along more than one direction. Once the features are selected, they are tracked from one frame to the next, using iterative gradient-based registration [56, 6, 71].

Recent methods utilize more sophisticated feature detectors and descriptors that are invariant to certain changes in viewpoint and illumination [17, 47, 29]. Tracking these features may allow correspondences to be found over a wider range of viewpoints than our current particle approach.

Like optical flow and stereo, feature tracking has difficulty with occlusion boundaries. When a feature patch lies across two independently moving surfaces, the feature cannot correctly follow both. For example, an algorithm may track what appears to be a 'T' junction, but which is in fact a pair of overlapping lines, neither of which is tracked correctly. These kinds of errors can be detected using correlation error [71, 33] or geometric constraints such as the fundamental matrix [37]. Another alternative is to adjust the region of support for a feature to fall one one side of the occlusion [68, 55].

2.2.3 3D Scene Reconstruction

Both stereo and optical flow use a small number of images and work in image-centric coordinates. In contrast, scene reconstruction methods work in world coordinates and usually use a larger number of images. Given a set of images of a scene from a number of viewpoints, these reconstruction algorithms estimate the 3D geometry of the scene [65]. This process typically proceeds by finding 3D coordinates of a set of distinctive image points that are shared between the images, then interpolating these features using a 3D model that fits the remaining observations.

When the viewpoints are too far apart for feature tracking, correspondences are matched using geometric constraints. These methods start by finding salient points in each image using the same feature detectors as used for feature tracking. Matches between feature points are pruned using robust fitting methods (such as RANSAC [30]) with geometric constraints [37]. Once correspondences are found, they are converted to 3D points using methods such as bundle adjustment [37]. This process also produces a camera pose estimate for each input image.

The 3D coordinates of the feature points provide some information about the scene structure, but not a complete model. Many of the pixels still must be assigned 3D coordinates, which can be performed using the stereo methods described in Section 2.2.1. For simple architectural scenes, the model may be constructed from planar components [91]. These methods can produce good results in certain cases, but have difficulty with general video, because of issues such as non-Lambertian reflectance, pixel matching, and geometric ambiguities (both in camera motion and scene geometry).

2.2.4 World-Space Ambiguity and Uncertainty

Irani and Anandan [41] discuss the difficulty of camera motion estimation across transitions between sequences with significant depth effects and sequences without significant depth effects. The first case requires a 3D formulation, while the second is best suited to a 2D formulation, due to a lack of 3D constraints. Motion estimation algorithms typically handle the 2D case or the 3D case, but not both.

To cope with these issues, some methods explicitly model ambiguity and uncertainty in the estimation of motion and structure. In this context, uncertainty arises from errant inputs (e.g. due to sensor noise) while ambiguity is a fundamental geometric property that can occur even with noiseless inputs.

The ambiguity and uncertainty of estimating fundamental matrices and other structure-from-motion quantities can be characterized in terms of the uncertainty of the input correspondences [58, 37]. Modelling pose uncertainty is also central to many localization and mapping algorithms [51, 82].

Modelling pixel-level uncertainty arises in several methods for volumetric reconstruction. Bhotika *et al.* [11] describe a method for voxel reconstruction that uses a probabilistic formulation of global occlusion dependencies, explicitly distinguishing and modelling both uncertainty and abiguity. Kutulakos [49] handles different kinds of uncertainties: inaccurate intrinsic calibration, inaccurate camera poses, and small movements of the subject. Both of these methods are limited in their ability to reconstruct general scenes because they assume Lambertian reflectance and use voxel-based representations of geometry.

One way to handle motion ambiguity in 3D reconstruction is to utilize more of the provided information. “Direct methods” for camera motion estimation [39, 67, 72, 42, 61] use all pixels in the image, rather than discrete image features. These algorithms typically use the brightness constraint equation [38] to estimate camera motion directly, rather than computing optical flow as an intermediate step. These methods may provide better camera motion estimates, but will still have some trouble with narrow fields of view and a lack of apparent depth.

World-space uncertainty is one motivation for basing our algorithm in image space. Although world-space constraints are useful, the algorithm needs to be able to handle cases in which they do not apply. In the future, particle video algorithms could incorporate both geometric and photometric approaches to uncertainty. The algorithm could then represent motion both in 2D and 3D, as levels of certainty permit.

Chapter 3

Variational Optical Flow

Our particle video algorithm uses frame-to-frame optical flow to provide an initial guess of particle motion. The algorithm treats flow estimation as a black box that can be replaced with an alternate flow algorithm. Rather than assuming temporal smoothness, we estimate optical flow independently for each frame; this enables the algorithm to perform well on hand-held video with moving objects.

3.1 Overview

Our optical flow algorithm uses a combination of the variational approach of Brox *et al.* [18] and the bilateral filtering approach of Xiao *et al.* [94]. The algorithm optimizes a flow field over a sequence of increasing resolutions. At each resolution, the algorithm performs the following steps:

- optimize the flow field using a variational objective with robust data and smoothness terms (Section 3.2),
- identify the occluded image regions using flow field divergence and pixel projection difference (Section 3.3),
- and regularize the flow field using an occlusion-aware bilateral filter (Section 3.4).

The sequence of resolutions is obtained by recursively reducing the original resolution by a factor η. As shown in Figure 3-1, a standard image pyramid uses $\eta = 0.5$ whereas we (following Brox *et al.* [18]) use a larger factor ($\eta = 0.9$) to obtain better results at a cost of increased computation. We set a 0.05 lower bound on the scale factor, which results in 29 resolution levels from a standard video frame; the smallest level is 36 by 24 pixels. (We crop the video frame from 720x480 to 712x480 to remove boundary artifacts before estimating flow.) After scaling the image, we filter it using a $\sigma = 1$ Gaussian kernel.

To handle large motions, we add an initialization step consisting of estimating whole-frame translation. The algorithm uses the KLT [56, 6] gradient-based optimization to register the frames, in a course-to-fine sequence of resolutions (with a factor of 2 scale change between each resolution). At each step we perform 8 optimization iterations. The entire process takes a fraction of a second for a full-resolution frame pair. The resulting whole-frame translational offset is used to initialize the flow field at the lowest resolution level.

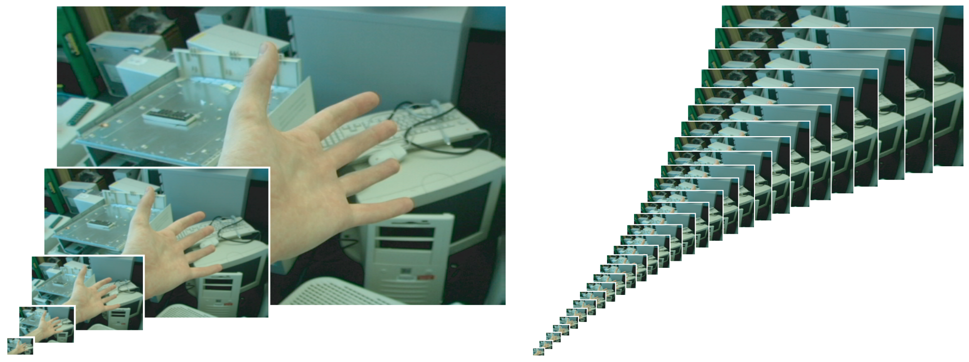


Figure 3-1: A standard image pyramid uses a scale factor of 0.5 between each resolution level, whereas we use a scale factor of 0.9, improving the results at a cost of increased computation.

3.2 Variational Flow Optimization

Our variational flow optimization is adapted from Brox *et al.* [18]. The approach has proved successful because it makes relatively few simplifications of the functional.

3.2.1 Objective Function

Let $u(x,y,t)$ and $v(x,y,t)$ denote the components of an optical flow field that maps image point $I(x,y,t)$ to an image point in the next frame:

$$I(x+u(x,y,t),y+v(x,y,t),t+1). \tag{3.1}$$

Like many optical flow methods, the Brox *et al.* [18] objective function combines a data term and smoothness term:

$$E_{Flow}(u,v,t) = E_{FlowData}(u,v,t) + E_{FlowSmooth}(u,v,t). \tag{3.2}$$

Although these terms are motivated as functionals, for clarity we give them in discrete form, in which u and v are estimated at integer indices.

Data Term

In our algorithm, we replace the scalar-valued image I with a multi-channel image $I^{[k]}$. We also modulate the data term by a visibility term $r(x,y,t)$ (described in Section 3.3):

$$E_{FlowData}(u,v,t) = \sum_{x,y,k} r(x,y,t)\Psi([I^{[k]}(x+u(x,y,t),y+v(x,y,t),t+1) - I^{[k]}(x,y,t)]^2). \tag{3.3}$$

Here k is summed over image channels. We use the same robust norm as Brox *et al.* [18]:

$$\Psi(s^2) = \sqrt{s^2+\varepsilon^2};\ \ \varepsilon = 0.001. \tag{3.4}$$

This function, a differentiable form of the absolute value function, does not respond as strongly to outliers as the standard L^2 norm.

The original Brox *et al.* [18] formulation analytically enforces constancy of the image gradient (and optionally other linear differential operators [63]), whereas we simply treat the gradient as another image channel. Specifically, we use image brightness I (range $[0,255]$), the green minus red color component, the green minus blue color component, and the x and y derivatives of brightness (I_x and I_y), as shown in Figure 3-2. We scale the color difference channels by 0.25 to reduce the impact of color sampling/compression artifacts common in video. These additional channels do not substantially increase the algorithm's running time because they do not increase the number of terms in the sparse linear system that consumes the majority of the computation.

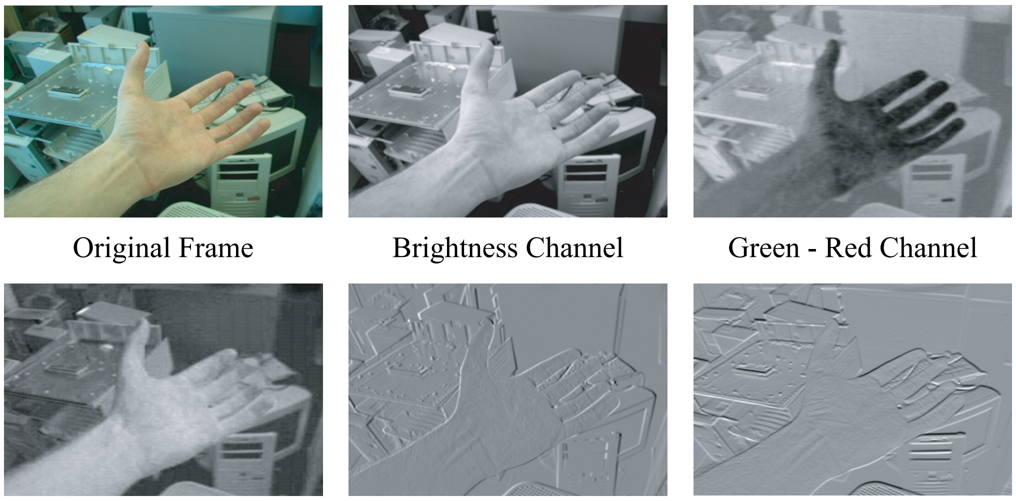


Figure 3-2: We use five image channels for flow estimation: the image brightness, green component minus red component, green component minus blue component, x gradient, and y gradient.

Smoothness Term

As in the Brox *et al.* [18] algorithm, the smoothness term measures the variation of the flow field using the robust norm Ψ. We modify the smoothness term to discourage flow discontinuities at locations with small image gradients:

$$E_{FlowSmooth}(u,v,t) = \sum_{x,y}(\alpha_g + \alpha_l \cdot b(x,y,t)) \cdot \Psi(u_x(x,y,t)^2 + u_y(x,y,t)^2 + v_x(x,y,t)^2 + v_y(x,y,t)^2). \quad (3.5)$$

Here α_g is a global smoothness factor (equivalent to the α parameter in the original Brox *et al.* [18] formulation) and α_l is a local smoothness factor, which is modulated by the local smoothness $b(x,y,t)$ (Figure 3-3).

We compute local smoothness using a Gaussian prior on the image gradient:

$$b(x,y,t) = N(\sqrt{I_x(x,y,t)^2 + I_y(x,y,t)^2}; \sigma_b). \tag{3.6}$$

Here N denotes a zero-mean non-normalized Gaussian. We set $\sigma_b = 2$, $\alpha_l = 15$, and $\alpha_g = 10$, based on a variety of flow experiments.

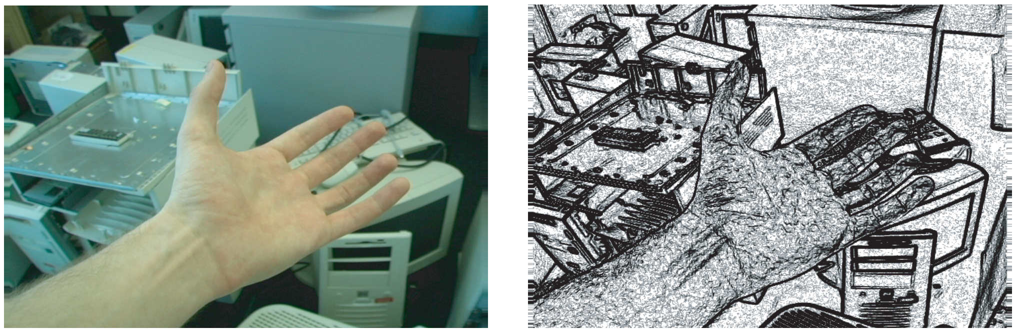


Figure 3-3: The local smoothness image modulates the smoothness term in the optical flow objective function. The objective discourages flow discontinuities in uniform image regions.

3.2.2 Objective Derivatives

To optimize the objective function (Equation 3.2), we construct a system of equations across the image domain R:

$$\left\{ \frac{\partial E_{Flow}}{\partial u(x,y,t)} = 0, \frac{\partial E_{Flow}}{\partial v(x,y,t)} = 0 \mid x,y \in R \right\}. \tag{3.7}$$

For the data term, we define the following shorthand notation:

$$\begin{aligned} I_x &= \frac{\partial}{\partial x} I^{[k]}(x+u(x,y,t), y+v(x,y,t), t+1), & (3.8)\\ I_y &= \frac{\partial}{\partial y} I^{[k]}(x+u(x,y,t), y+v(x,y,t), t+1), & (3.9)\\ I_z &= I^{[k]}(x+u(x,y,t), y+v(x,y,t), t+1) - I^{[k]}(x,y,t). & (3.10) \end{aligned}$$

For the remainder of this section, we omit the k and t indexing. Also, for the remainder of this section, we only compute the derivatives with respect to u, which are analogous to the derivatives with respect to v.

The derivative of the kth channel of the data term is:

$$\frac{\partial E_{FlowData}}{\partial u(x,y)} = 2\Psi'(I_z^2)I_z I_x. \tag{3.11}$$

Here Ψ' is the derivative of Ψ with respect to its argument s^2.

The derivative of the smoothness term is:

$$\begin{aligned}\frac{\partial E_{FlowSmooth}}{\partial u(x,y)} &= 2\alpha(x,y)\cdot\Psi'_{FlowSmooth}(x,y)\cdot[u_x(x,y)+u_y(x,y)]\\ &- 2\alpha(x+1,y)\cdot\Psi'_{FlowSmooth}(x+1,y)\cdot[u_x(x+1,y)]\\ &- 2\alpha(x,y+1)\cdot\Psi'_{FlowSmooth}(x,y+1)\cdot[u_y(x,y+1)].\end{aligned} \tag{3.12}$$

Here we define:

$$\Psi'_{FlowSmooth}(x,y) = \Psi'(u_x(x,y)^2+u_y(x,y)^2+v_x(x,y)^2+v_y(x,y)^2), \tag{3.13}$$

$$\alpha(x,y) = (\alpha_g+\alpha_l\cdot b(x,y)). \tag{3.14}$$

In these equations, we discretize the partial derivatives of u and v:

$$u_x(x,y) = u(x,y)-u(x-1,y), \tag{3.15}$$

$$u_y(x,y) = u(x,y)-u(x,y-1). \tag{3.16}$$

3.2.3 Fixed-Point Scheme

These derivatives do not immediately provide a linear system of equations. Rather than directly linearizing the equations, Brox *et al.* [18] use a fixed-point optimization. (This is actually a nested pair of fixed-point loops, where the outer loop consists of the coarse-to-fine iterations described in Section 3.1.)

At the current resolution level, the algorithm computes an update $(du(x,y),dv(x,y))$

that is added to the previous flow field to obtain a new flow field:

$$\begin{aligned} u_{new}(x,y) &= u(x,y)+du(x,y), \\ v_{new}(x,y) &= v(x,y)+dv(x,y). \end{aligned} \tag{3.17}$$

We use these offsets as the basis of a data-term linearization that replaces I_z:

$$I_z \rightarrow I_x du + I_y dv + I_z. \tag{3.18}$$

Iteration j of the fixed-point loop computes (du^{j+1}, dv^{j+1}) from (du^j, dv^j) (where du^0 and dv^0 are set to 0). This provides a new data derivative:

$$\frac{\partial E_{FlowData}}{\partial u(x,y)} \approx 2\Psi'([I_x du^j + I_y dv^j + I_z]^2) \cdot [I_x du^{j+1} + I_y dv^{j+1} + I_z] I_x. \tag{3.19}$$

The smoothness derivative is:

$$\begin{aligned} \frac{\partial E_{FlowSmooth}}{\partial u(x,y)} &\approx 2\alpha(x,y) \cdot \Psi'_{FlowSmooth}(x,y)^j \\ &\qquad \cdot [u_x(x,y) + u_y(x,y) + du_x(x,y)^{j+1} + du_y(x,y)^{j+1}] \\ &- 2\alpha(x+1,y) \cdot \Psi'_{FlowSmooth}(x+1,y)^j \\ &\qquad \cdot [u_x(x+1,y) + du_x(x+1,y)^{j+1}] \\ &- 2\alpha(x,y+1) \cdot \Psi'_{FlowSmooth}(x,y+1)^j \\ &\qquad \cdot [u_y(x,y+1) + du_y(x,y+1)^{j+1}]. \end{aligned} \tag{3.20}$$

Here we define:

$$\begin{aligned} \Psi'_{FlowSmooth}(x,y)^j &= \Psi'([u_x(x,y) + du_x(x,y)^j]^2 + [u_y(x,y) + du_y(x,y)^j]^2 \\ &\qquad + [v_x(x,y) + dv_x(x,y)^j]^2 + [v_y(x,y) + dv_y(x,y)^j]^2). \end{aligned} \tag{3.21}$$

The partial derivatives of du and dv are defined with finite differences in the same way as the derivatives of u and v (Equation 3.16).

Both the data and smoothness derivatives (Equations 3.19 and 3.20) are linear in du^{j+1} and dv^{j+1}, so we can construct a sparse linear system. The linear system has two equations for each pixel of the image (for the derivatives with respect to $u(x,y)$ and $v(x,y)$). Each equation has seven terms; the equation for flow component $u(x,y)$ has coefficients for $u(x,y)$, $v(x,y)$, $u(x+1,y)$, $u(x-1,y)$, $u(x,y+1)$, $u(x,y-1)$, and a constant term.

By default, we solve the system using the successive over-relaxation method (SOR) [7]. At a given resolution level, the algorithm makes 3 fixed-point steps, each consisting of 500 SOR iterations. We also provide the option of solving the system using the preconditioned conjugate gradient technique [7].

3.3 Occlusion Detection

Handling occlusions is the most challenging aspect of building a particle video. It is also the most challenging part of optical flow estimation, stereo reconstruction, feature tracking, and motion estimation in general.

Rather than solving the problem purely with particles, we use optical flow estimation to provide information about occlusions. Ideally, the flow estimates will be able to incorporate subtle details of the surface being occluded, which are not necessarily captured by the particles.

The Brox *et al.* [18] algorithm uses the robust distance function Ψ to handle occlusions. As discussed in Chapter 2, using robustness to account for occlusion boundaries is not ideal. Rather than properly modeling the physical behavior of the occlusion boundary, the algorithm is simply allowed to fail (with a small penalty due to the robust distance function). In practice, the Brox *et al.* [18] algorithm produces flow fields that incorrectly push the occluded pixels along with the occluding object, because this produces a lower objective value.

Like other approaches [73, 2, 75, 94], we model occlusion by explicitly labelling occluded pixels. Once the pixels are labelled, they can be excluded from the data term, rather than incorrectly matched with non-occluded pixels. A flow field augmented with an occlusion mask correctly models the fact that some pixels disappear.

Our algorithm uses a combination of flow divergence and pixel projection difference to identify occluded pixels. The divergence of an optical flow field distinguishes between different types of motion boundaries:

$$div(x,y,t) = \frac{\partial}{\partial x} u(x,y,t) + \frac{\partial}{\partial y} v(x,y,t). \tag{3.22}$$

The divergence is positive for disoccluding boundaries, negative for occluding boundaries, and near zero for shear boundaries (Figure 3-4). To select occluding boundaries, but not disoccluding boundaries, we define a one-sided divergence function d:

$$d(x,y,t) = \begin{cases} div(x,y,t) & div(x,y,t) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.23}$$

Pixel projection difference provides another occlusion cue:

$$e(x,y,t) = I(x,y,t) - I(x + u(x,y,t), y + v(x,y,t), t+1). \tag{3.24}$$

We combine the one-sided divergence and pixel projection using zero-mean non-normalized Gaussian priors:

$$r(x,y,t) = N(d(x,y,t); \sigma_d) \cdot N(e(x,y,t); \sigma_e). \tag{3.25}$$

The $r(x,y,t)$ values are near zero for occluded pixels and near one for non-occluded pixels. We set $\sigma_d = 0.3$ and $\sigma_e = 20$ based on experimental observation of occluded regions.

3.4 Bilateral Flow Filtering

Detecting occluded pixels is part of the occlusion modelling process, but we must still handle the mixing of pixel properties across boundaries. This mixing occurs for all types of motion boundaries: disocclusions, occlusions, and shear motions. To improve boundary sharpness, we use a bilateral filter based on the work of Xiao *et al.* [94].

Xiao and colleagues motivate the approach by pointing out an equivalence between

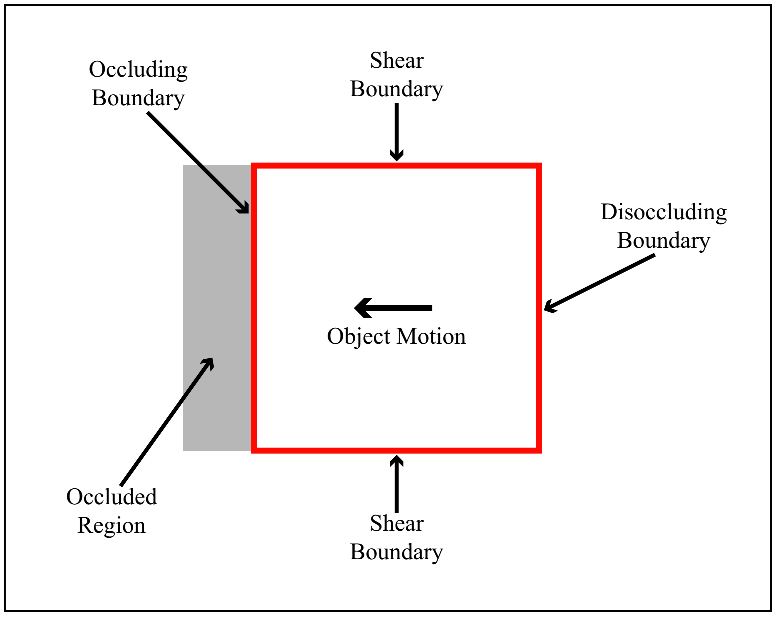


Figure 3-4: In this diagram, the motion discontinuities (red) include occluding boundaries, disoccluding boundaries, and shear boundaries. The occluded region is the set of pixels that are not visible in the subsequent frame.

variational smoothness optimization and Gaussian filtering of the flow fields. Using this observation, they replace traditional anisotropic regularization with a filter that better separates distinct motions.

The filter sets each flow vector to a weighted average of neighboring flow vectors:

$$u'(x,y,t) = \frac{\sum_{x_1,y_1} u(x_1,y_1,t)w(x,y,x_1,y_1,t)}{\sum_{x_1,y_1} w(x,y,x_1,y_1,t)}. \tag{3.26}$$

The update for v is analogous. The algorithm weights the neighbors according to spatial proximity, image similarity, motion similarity, and occlusion labelling:

$$\begin{aligned} w(x,y,x_1,y_1,t) \quad = \quad & N(\sqrt{(x-x_1)^2+(y-y_1)^2};\sigma_x) \\ & \cdot N(I(x,y,t)-I(x_1,y_1,t);\sigma_i) \\ & \cdot N(\sqrt{(u-u_1)^2+(v-v_1)^2};\sigma_m) \\ & \cdot r(x_1,y_1,t) \end{aligned} \tag{3.27}$$

Here u denotes $u(x,y,t)$ and u_1 denotes $u(x_1,y_1,t)$ (and v similarly). We set $\sigma_x = 4$, $\sigma_i = 7.5$,

Variable	Description	Value	Units	Section
η	multi-resolution scale factor	0.9	N/A	§3.1
α_g	global smoothness factor	10	N/A	§3.2.1
α_l	local smoothness factor	15	N/A	§3.2.1
σ_b	image gradient prior	2	pixel value gradient	§3.2.1
σ_d	flow divergence prior	0.3	flow gradient	§3.3
σ_e	pixel mismatch prior	20	pixel values	§3.3
σ_x	bilateral filter size	4	image space	§3.4
σ_i	filter image difference	7.5	pixel values	§3.4
σ_m	filter motion difference	0.5	flow values	§3.4
σ_g	flow gradient filter	3	image space	§3.4

Table 3.1: For our experiments, we use these optical flow parameter settings.

$\sigma_m = 0.5$, and restrict (x_1, y_1) to lie within 10 pixels of (x, y).

This filter computes weights for a neighborhood of pixels around each pixel, so it is quite computationally expensive. Thus, for efficiency, we apply the filter only near flow boundaries, which we localize using the flow gradient magnitude:

$$g(x,y,t) = \sqrt{u_x^2(x,y,t) + u_y^2(x,y,t) + v_x^2(x,y,t) + v_y^2(x,y,t)} \tag{3.28}$$

The algorithm filters $g(x,y,t)$ using a spatial Gaussian kernel ($\sigma_g = 3$), producing a smoothed gradient magnitude $\hat{g}(x,y,t)$. Note that, unlike the divergence, this gradient magnitude is large for all types of motion boundaries (occlusions, disocclusions, and shear boundaries). We apply the bilateral filter (Equation 3.26) to pixels with $\hat{g}(x,y,t) > 0.25$.

Table 3.1 summarizes the parameters for our complete optical flow algorithm. Figure 3-5 shows flow fields generated by the algorithm.

3.5 Multi-Frame Optical Flow

Typically video is temporally undersampled, but in some cases the motion between frames is too small to obtain clear information about occlusions (e.g. if the flow is on the scale of a single pixel). To better handle occlusion boundaries, we provide a mechanism for estimating flow over a range of different frame separations (not just between one frame and

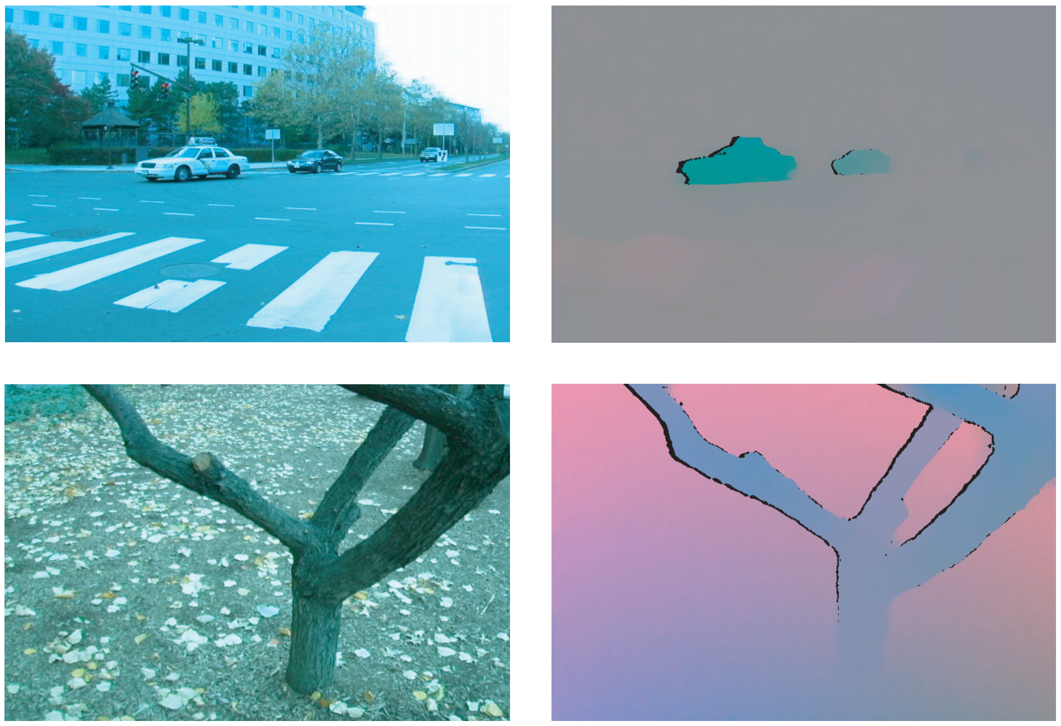


Figure 3-5: Each flow field is generated between a video frame (left) and the subsequent video frame. The flow field is visualized (right) using hue to denote flow direction and saturation to denote flow magnitude. The black regions are labelled as occluded.

the next).

This is not used in the current implementation of the particle video algorithm, but it could be incorporated in the future. For example, during the particle propagation step (Section 4.3), the algorithm could use occlusion masks from flow fields of multiple different frame separations.

Unlike some multi-frame flow algorithms, we do not assume temporal motion smoothness. Instead, we concatenate and refine a sequence of individually estimated frame-to-frame flow fields.

3.5.1 Flow Concatenation and Refinement

Given two flow fields (u_1, v_1) and (u_2, v_2) we concatenate them to produce a new flow field (u_3, v_3). The algorithm performs this concatenation using the first flow field to look up the appropriate flow extension in the second field:

$$u_3(x,y) = u_1(x,y) + u_2(x + u_1(x,y), y + v_1(x,y)), \tag{3.29}$$

$$v_3(x,y) = v_1(x,y) + v_2(x + u_1(x,y), y + v_1(x,y)). \tag{3.30}$$

The algorithm uses bilinear interpolation of the second flow field. We augment the first flow field's occlusion mask with any pixels that are projected into the second field's occlusion mask.

This concatenation may result in drift, which we reduce by refining the concatenated flow field, using its start and end frame, ignoring the intermediate frames. This refinement consists of running the optical flow algorithm (variational update, occlusion labelling, and bilateral filtering) at the full image resolution. We do not refine the flow at lower resolutions because this would discard any motion detail estimated in the original flow fields.

3.5.2 Maintenance of Optical Flow Sets

Let $F(t_1,t_2)$ denote a flow field from frame t_1 to frame t_2. For a given frame t, we construct flow fields from a sequence of previous frames: $\{F(t-d,t) \mid d \in [1,T]\}$.

For computational efficiency we reuse previous multi-frame flow estimates as we move forward through a video. Suppose we have previously computed $\{F(t-d-1,t-1) \mid d \in [1,T]\}$. We then compute $F(t-1,t)$ and concatenate it onto each of the previously computed flow fields, refining each one. This produces the desired set: $\{F(t-d,t) \mid d \in [1,T]\}$. The process is illustrated in Figure 3-6.

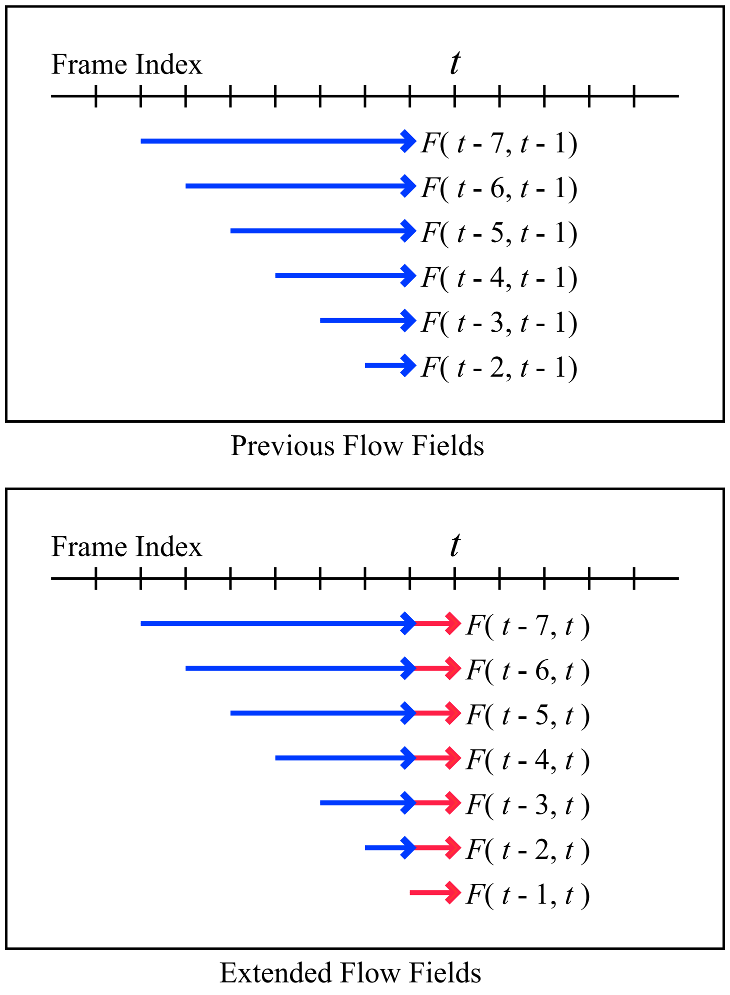


Figure 3-6: $F(t_1,t_2)$ denotes a flow field from frame t_1 to t_2. To move the multi-frame flow fields (top) forward by one frame, the algorithm computes a new flow field $F(t-1,t)$ that it appends (bottom) to each of the previously computed flow fields. After concatenating the new flow field, each flow field is refined (from its start frame to end frame) as described in Section 3.5.1.

Chapter 4

Particle Video Algorithm

A particle video is a video and corresponding set of particles. Particle i has a time-varying position $(x_i(t), y_i(t))$ that is defined between the particle's start and end frames. (Each particle has its own start time and end time.)

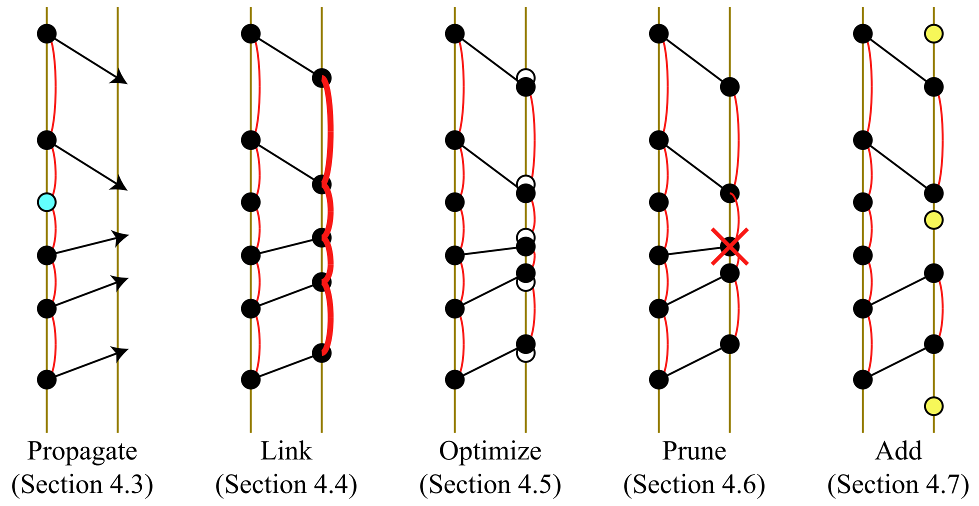


Figure 4-1: Each plot denotes a pair of consecutive frames. The algorithm propagates particles from one frame to the next according to the flow field, excluding particles (blue) that lie within the flow field's occluded region. The algorithm then adds links (red curves), optimizes all particle positions, and prunes particles with high error after optimization. Finally, the algorithm inserts new particles (yellow) in gaps between existing particles.

4.1 Top-Level Particle Video Algorithm

Our algorithm builds a particle video by moving forward and backward across the video, as illustrated in Figure 4-2. Moving backwards, occlusion boundaries become disocclusion boundaries, which are easier to interpret than occlusion boundaries. By moving through the video in both direction, new particles can be extended in both directions.

For each processed frame, the following steps are performed (Figure 4-1):

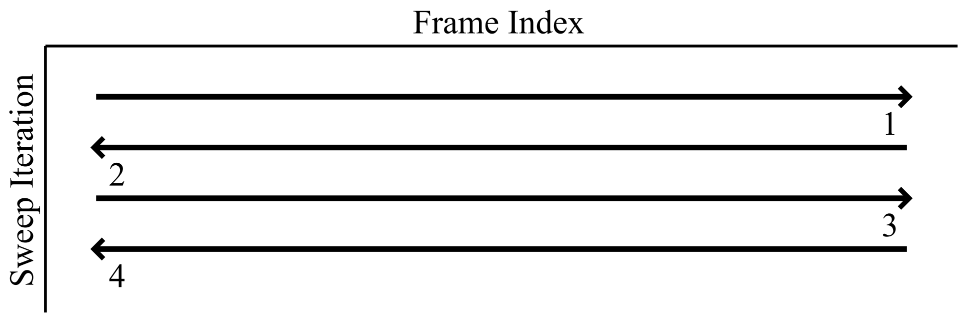
- **Propagation.** Particles terminating in an adjacent frame are extended into the current frame according to the forward and reverse flow fields (Section 4.3).
- **Linking.** Particle links are updated (Section 4.4).
- **Optimization.** Particle positions are optimized (Section 4.5).
- **Pruning.** Particles with high post-optimization error are pruned (Section 4.6).
- **Addition.** New particles are added in gaps between existing particles (Section 4.7).

To reduce computation, the algorithm maintains a cache of information for each video frame. This cache includes the frame itself, color and gradient channels (and gradients thereof), a scale map (Section 4.7), forward flow (and its gradient magnitude), and reverse flow.

4.2 Particle Channels

We use the same 5 channels used for flow estimation (Chapter 3): image brightness, green minus red channel, green minus blue channel, x gradient, and y gradient. As before, k denotes the channel index; at time t the kth image channel is $I^{[k]}(t)$.

The color and gradient channels are moderately insensitive to changes in lighting and reflectance, which facilitates matching a particle with a temporally distant frame. However, these channels depend on a wider spatial area of support, which may cause mismatches for particles near occlusion boundaries. (The gradient is computed using multiple pixels and the color channel has a low spatial resolution due to common video color compression.)



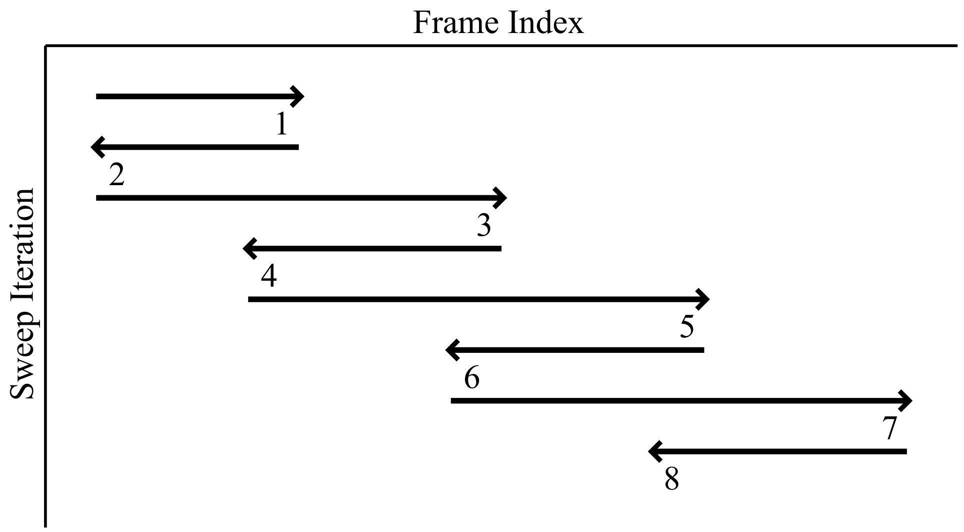


Figure 4-2: The algorithm sweeps back and forth across the video to better model occlusions. The top sweeping scheme involves four passes across the entire video. The bottom sweeping scheme makes shorter passes as it progresses forward. We compare these schemes in Chapter 6.

To address this, we disable the gradient and color channels near occlusion boundaries, as determined by the filtered flow gradient magnitude $\hat{g}(x,y,t)$ (Section 3.4). When $\hat{g}(x_i(t),y_i(t),t) > 0.01$, the particle is probably near a flow boundary, so we exclude all but brightness channel, because the other channels may be influenced by pixels on the other side of the boundary.

We scale the gradient and color channels by a factor of 0.1 to reduce the effects of noise in these channels. In our experiments, we find that these channels provide only a small benefit. For the sake of simplicity, others may choose to omit these channels.

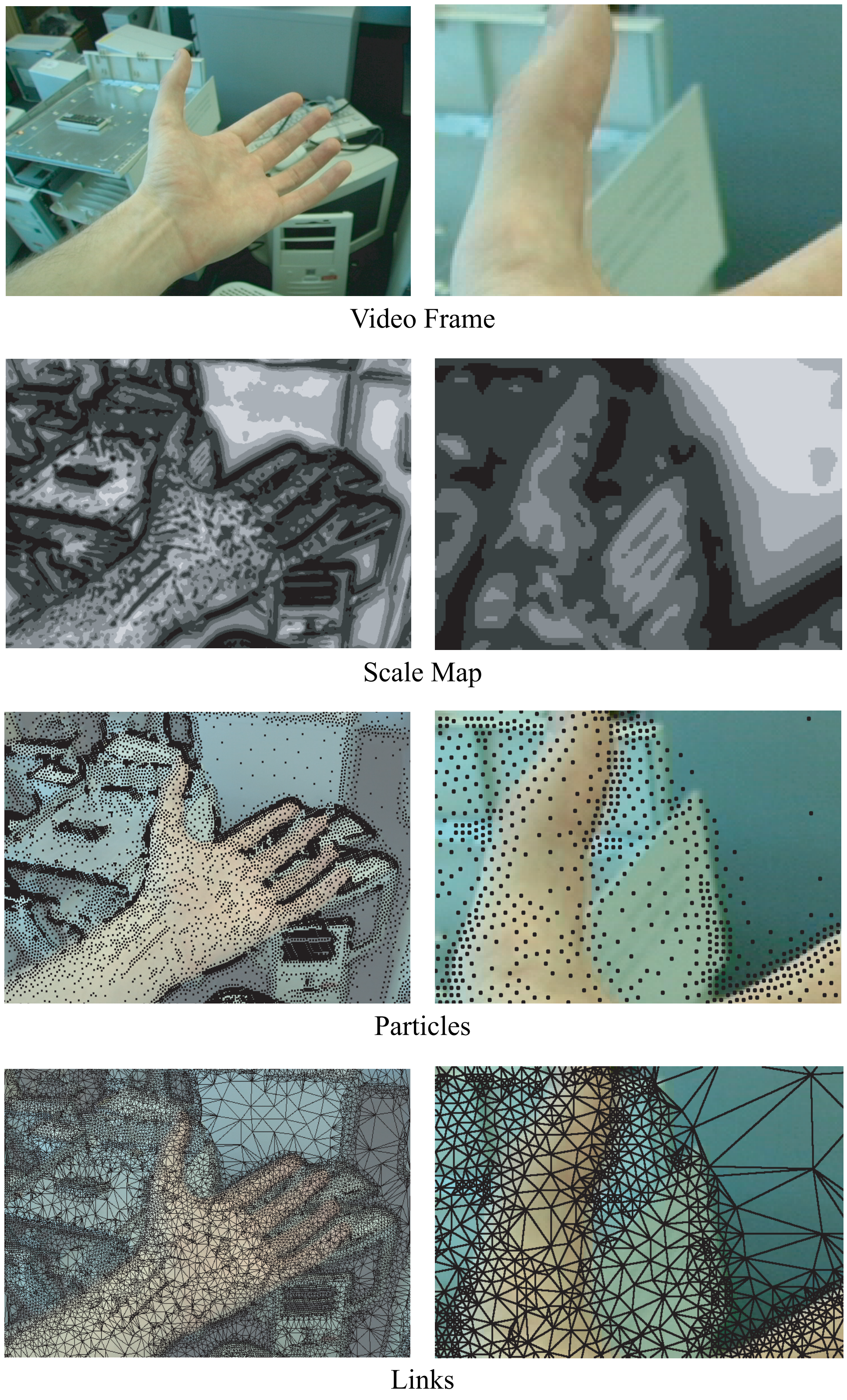


Figure 4-3: For each video frame, the algorithm computes a scale map that determines the placement of new particles (Section 4.7). Links are added using a particle triangulation (Section 4.4). The left side shows an entire frame. The right side shows a magnified portion of the frame.

4.3 Propagating Particles

When propagating particles to a given frame, all particles defined in adjacent frames, but not defined in the given frame, are placed in the frame according to the flow fields between the frames. To propagate particle i from frame $t-1$ to t, we use the flow field $u(x,y,t-1), v(x,y,t-1)$:

$$x_i(t) = x_i(t-1) + u(x_i(t-1), y_i(t-1), t-1), \tag{4.1}$$

$$y_i(t) = y_i(t-1) + v(x_i(t-1), y_i(t-1), t-1). \tag{4.2}$$

Backward propagation from frame $t+1$ to t is defined analogously. (When making the first forward pass through the video, there are no particles to propagate backward.) If the optical flow field indicates that a particle becomes occluded, the particle is not propagated.

4.4 Particle Links

To quantify relative particle motion, our algorithm creates links between particles using a constrained Delaunay triangulation [54] (Figure 4-4). The triangulation ensures a good directional distribution of links for each particle. This is preferable to simply linking each particle to its N nearest neighbors (which could all be in one direction from a given particle).

For any given frame, we create a particle link if the corresponding triangulation edge exists for the frame or an adjacent frame. Using links from adjacent frames reduces temporal linking variability, while still allowing links to appear and disappear as particles pass by one another.

The algorithm assigns a weight to each link based on the difference between the trajectories of the linked particles. If the particles have similar trajectories, they are probably part of the same surface, and thus should be strongly linked. If the particles are separated by an occlusion boundary, the weight should be near zero.

The algorithm computes the mean squared motion difference between particles i and j

over the set T of frames in which the link is defined: :

$$D(i,j) = \frac{1}{|T|} \sum_{t \in T} (u_i(t) - u_j(t))^2 + (v_i(t) - v_j(t))^2. \tag{4.3}$$

Here we let $u_i(t) = x_i(t) - x_i(t-1)$ and $v_i(t) = y_i(t) - y_i(t-1)$. The algorithm computes the link weight using a zero-mean Gaussian prior ($\sigma_l = 1.5$):

$$l_{ij} = N(\sqrt{D(i,j)}; \sigma_l). \tag{4.4}$$

Link weights are illustrated in Figure 4-4.

4.5 Particle Optimization

The core of the particle video algorithm is an optimization process that repositions particles. As described in Section 4.3, a flow field provides an initial location for each particle in a given frame; the optimization refines these positions.

For a given particle, this optimization can modify the particle's position in any frame except for the frame in which the particle was first added. This original frame defines the particle's reference position. (The original frame will be different from the particle's start frame if it was propagated backward from the original frame.)

4.5.1 Particle Objective Function

The algorithm repositions particles to locally minimize an objective function that includes two components for each particle: a data term and a distortion term. This objective function has some similarities to the variational flow functionals described in Chapter 3, but it operates just on the particles, not the full set of pixels.

The energy of particle i in frame t is:

$$E(i,t) = \sum_{k \in K_i(t)} E_{Data}^{[k]}(i,t) + \alpha \sum_{j \in L_i(t)} E_{Distort}(i,j,t). \tag{4.5}$$

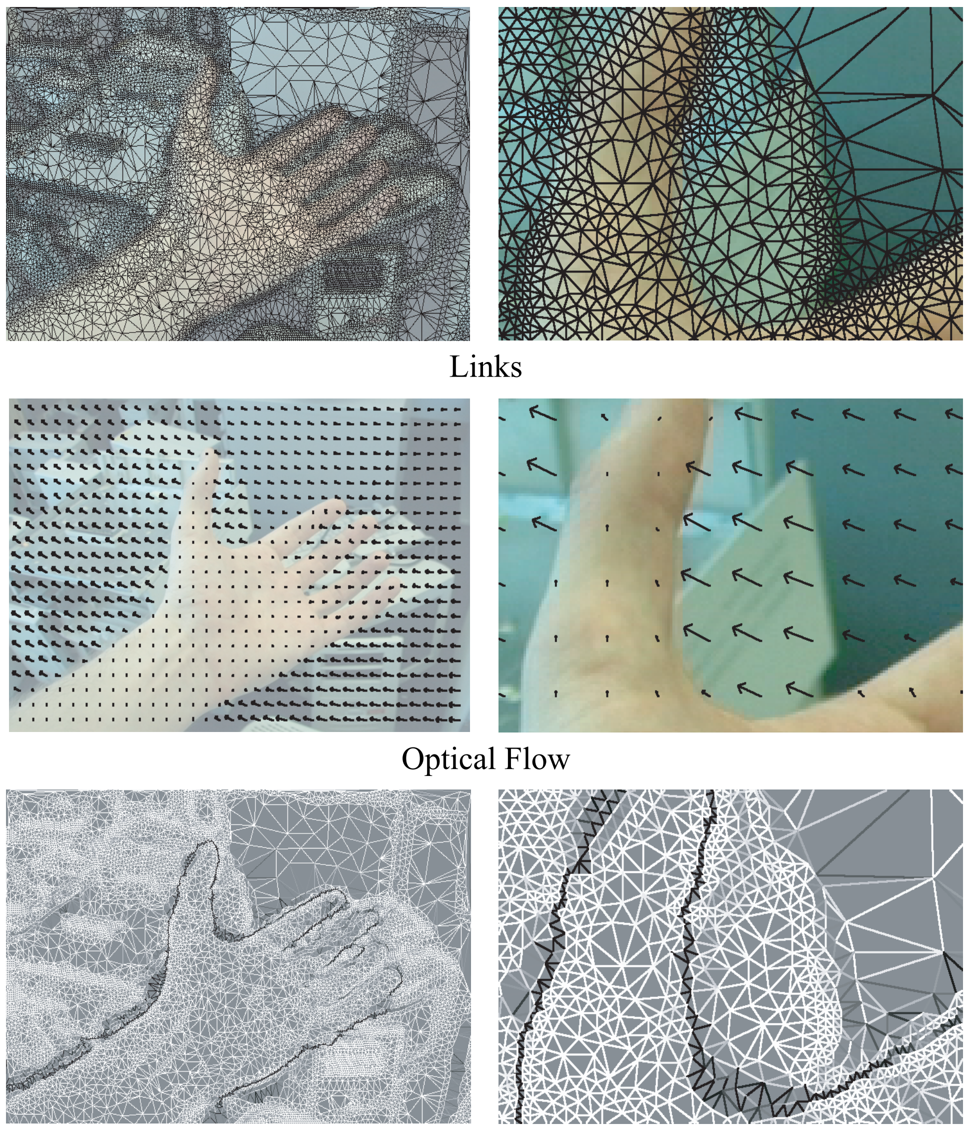


Figure 4-4: Particles with similar trajectories are connected by strong links (lighter) while particles with different trajectories are connected by weak links (darker). The left side shows an entire frame. The right side shows a magnified portion of the frame.

Here $K_i(t)$ denotes the set of active channels (Section 4.2), and $L_i(t)$ denotes the set of particles linked to particle i in frame t. We find that $\alpha = 1.5$ provides a reasonable trade-off between the two terms.

Given a set P of particle indices and a set F of frame indices, the complete objective function is:

$$E = \sum_{t \in F, i \in P} E(i,t). \tag{4.6}$$

Data Energy

The data term measures how well a particle's appearance (Section 4.2) matches the video frames. We allow particle appearance to change slowly over time, to cope with non-Lambertian reflectance and changes in scale. For particle i at time t the kth channel of the particle's appearance is:

$$c_i^{[k]}(t) = I^{[k]}(x_i(t), y_i(t), t). \tag{4.7}$$

Using a Gaussian kernel ($\sigma_c = 5$), we filter these appearance values along the time axis, producing a slowly-varying appearance denoted by $\hat{c}_i^{[k]}(t)$. For a given frame, the data term measures the difference between the observed appearance and filtered appearance:

$$E_{Data}^{[k]}(i,t) = \Psi(c_i^{[k]}(t) - \hat{c}_i^{[k]}(t)]^2). \tag{4.8}$$

Here Ψ is the robust norm described in Section 3.2.1. Although we assume temporal appearance smoothness, we do not assume temporal motion smoothness. The data term suggests that a particle's appearance changes slowly, but does not depend on the smoothness of the particle trajectory.

Distortion Energy

The distortion term measures the relative motion of linked particles. If two linked particles move in different directions, they will have a larger distortion term. If they move in the same direction, they will have a smaller distortion term (Figure 4-5).

The distortion term is defined between a pair of linked particles i and j. As before, we let $u_i(t) = x_i(t) - x_i(t-1)$ and $v_i(t) = y_i(t) - y_i(t-1)$. The larger the difference between these motion values, the larger the distortion term:

$$E_{Distort}(i,j,t) = l_{ij}\Psi([u_i(t) - u_j(t)]^2 + [v_i(t) - v_j(t)]^2). \tag{4.9}$$

Note that this is symmetric: $E_{Distort}(i,j,t) = E_{Distort}(j,i,t)$.

The distortion term is modulated by the link weight l_{ij} so that a link across an occlusion boundary (i.e. a low-weight link) is allowed greater distortion for an equivalent penalty. Both the link weights and distortion term measure the relative motion of particles, but the link weights take into account entire particle trajectories whereas the distortion term refers to a single frame. By modulating the distortion term using the link weights, the algorithm encourages particles that have moved together to continue moving together in the current frame, while particles that have moved differently are allowed to move differently in the current frame.

Note that the distortion term (like the data term) does not require or encourage temporal motion smoothness. It measures the relative motion of particles, so the global motion does not need to be smooth (the camera motion can be unstable).

The distortion term resists incorrect motions caused by the data term, especially near occlusion boundaries. In the case that a particle is being occluded (but is not pruned by an occlusion mask), the data term may push the particle into an unoccluded part of the background surface (unless the particle happens to better match the foreground surface). Also, the flow field may incorrectly push or pull background pixels along with the foreground surface. In both cases, a strong distortion term will improve the correctness of the particle motion.

However, the distortion term cannot be too strong, because this rigidity would prevent certain correct motions, such as those caused by changes in viewpoint or non-rigid object deformation. This tradeoff can be modulated by adjusting the distortion factor α in Equation 4.5.

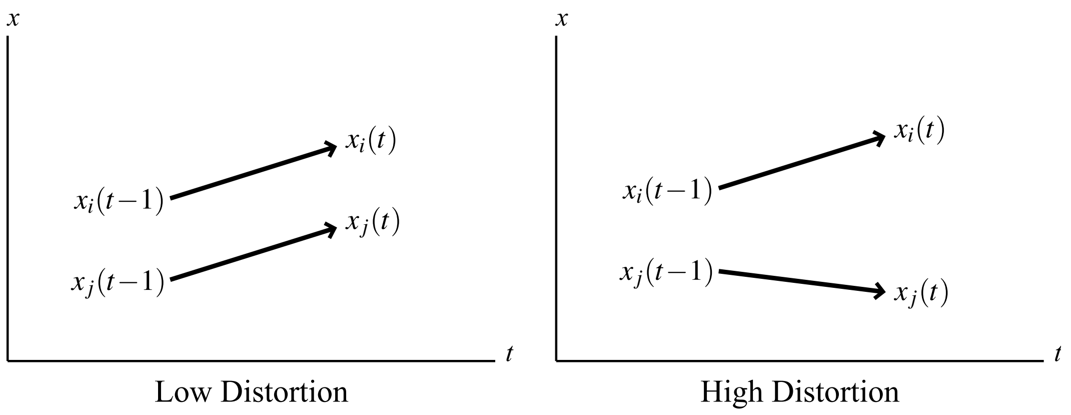


Figure 4-5: A pair of linked particles moving in the same direction (left) have a low distortion energy while linked particles moving different directions (right) have a high distortion energy.

4.5.2 Constructing a Sparse Linear System

The algorithm optimizes Equation 4.6 in a manner similar to the variational technique described in Chapter 3, using a fixed-point loop around a sparse linear system solver. In this section, we describe the construction of this sparse linear system. In Section 4.5.3 we provide the complete optimization algorithm.

Within the objective function E, we substitute $dx_i(t) + x_i(t)$ for $x_i(t)$ (and instances of y accordingly). Taking partial derivatives, we obtain a system of equations, which the algorithm solves for $dx_i(t)$ and $dy_i(t)$:

$$\left\{ \frac{\partial E}{\partial dx_i(t)} = 0, \frac{\partial E}{\partial dy_i(t)} = 0 \mid i \in P, t \in F \right\}. \tag{4.10}$$

The $dx_i(t)$ and $dy_i(t)$ values produced by solving this system are added to the current particle positions ($x_i(t)$ and $y_i(t)$).

Data Derivative

For the data term, we use the image linearization from Brox *et al.* [18]:

$$I_z^{[k]} = I_x^{[k]} dx_i(t) + I_y^{[k]} dy_i(t) + I^{[k]} - \hat{c}_i^{[k]}, \tag{4.11}$$

$$\frac{\partial E_{Data}^{[k]}(i,t)}{\partial dx_i(t)} \approx 2\Psi'([I_z^{[k]}]^2)(I_z^{[k]})I_x^{[k]}. \tag{4.12}$$

Here we omit the $(x_i(t), y_i(t), t)$ indexing of I, I_x, I_y, and I_z. (I_x and I_y are the spatial derivatives of I.) Ψ' is the derivative of Ψ with respect to its argument s^2. Note that this linearization occurs inside the fixed-point loop; the algorithm is still optimizing the original non-linearized objective function.

Distortion Derivative

For the distortion term, we use $du_i(t)$ as shorthand for $dx_i(t) - dx_i(t-1)$ and $dv_i(t)$ for $dy_i(t) - dy_i(t-1)$. This gives the following partial derivative:

$$\frac{\partial E_{Distort}(i,j,t)}{\partial dx_i(t)} 2l_{ij}(t)\Psi'_{Distort}(i,j,t)(u_i(t) + du_i(t) - u_j(t) - du_j(t)). \tag{4.13}$$

Here we define:

$$\Psi'_{Distort}(i,j,t) = \\ \Psi'([u_i(t) + du_i(t) - u_j(t) - du_j(t)]^2 + [v_i(t) + dv_i(t) - v_j(t) - dv_j(t)]^2). \tag{4.14}$$

The $dx_i(t)$ variable also appears in the term for link i, j at time $t+1$:

$$\frac{\partial E_{Distort}(i,j,t+1)}{\partial dx_i(t)} = \\ -2l_{ij}(t+1)\Psi'(i,j,t+1)(u_i(t+1) + du_i(t+1) - u_j(t+1) - du_j(t+1)). \tag{4.15}$$

The $dx_i(t)$ variable also appears in the terms for particle j at times t and $t+1$. These derivatives are identical (since the terms are identical via the i, j symmetry of the distortion energy), so we add an extra factor of two to the distortion derivatives.

4.5.3 Fixed-Point Scheme

Like the variational flow algorithm described in Chapter 3, the particle optimization iteratively solves for updates to the particle positions:

```
Loop until convergence
    $dx_i(t), dy_i(t) \leftarrow 0$
    Solve system for $dx_i(t)$, $dy_i(t)$
    $x_i(t) \leftarrow x_i(t) + dx_i(t)$
    $y_i(t) \leftarrow y_i(t) + dy_i(t)$
End Loop
```

In our implementation, the loop terminates when the mean change in position is less than 0.005 (with an upper bound of 10 iterations). The linear system solver performs 200 iterations inside each of the loop iterations. These numbers control the tradeoff between accuracy and running time. Fortunately, because the optimization is performed only on the particles (not on every pixel), the process is relatively fast.

The algorithm uses a pair of integer matrices to keep track of which sparse system variables correspond to which particles. One matrix maps variable indices to (i,t) pairs. The other maps (i,t) to variable indices.

The solver uses the SOR algorithm [7], with some conditioning and smoothing to make sure the optimization does not become unstable. We limit $|dx_i(t)|$ and $|dy_i(t)|$ to be less than 2 pixels for each step.

4.6 Pruning Particles

After optimizing the particles, we prune particles that continue to have high energy values. These particles have high distortion and/or a large appearance mismatch, indicating possible occlusion.

As defined in Section 4.5.1, $E(i,t)$ denotes the objective function value of particle i in frame t. To reduce the impact of a single bad frame, we filter each particle's energy values using a Gaussian ($\sigma_t = 1$ frames). (Note: this Gaussian is not strictly temporal; it filters the values for the given particle, which is moving through image space.) If in any frame the filtered energy value is greater than $\delta = 5$, the particle is deactivated in that frame.

4.7 Adding Particles using Scale Maps

After optimization and pruning, the algorithm adds new particles in gaps between existing particles. The algorithm arranges for higher particle density in regions of greater visual complexity, in order to model complex motions. (Motion complexity often implies visual complexity, though the reverse is not generally true.)

To add new particles to a given frame, the algorithm determines a scale value $s(x,y)$ for each pixel. The scale values are discrete, taken from the set $\{\sigma(j) = 1.9^j \mid 0 \leq j \leq 5\}$. To compute the scale map, we start by filtering the image using a Gaussian kernel for each scale $\sigma(j)$, producing a set of images $\{I_j\}$.

Then, for each pixel, we find the range of scales over which the blurred pixel value does not change substantially. If the pixel has the same color in a large scale image as in all smaller scale images, it is a large scale pixel (Figure 4-6). Specifically, the algorithm chooses the maximum scale index $k(x,y)$ such that $||I_j(x,y) - I_1(x,y)||_2 < \delta_s$ for all $j \leq k(x,y)$. (Here we use (r,g,b) vector distance when comparing pixel values.)

These scale indices are filtered with a spatial Gaussian ($\sigma_s = 2$), producing a blurred scale index map $\hat{k}(x,y)$ (which we round to integer values). We then set the scale values from the indices: $s(x,y) = \sigma(\hat{k}(x,y))$. Figure 4-3 provides an example scale map.

Given the scale map, we iterate over the image adding particles. For each pixel, if the distance to the nearest particle is greater than $s(x,y)$, we add a particle at that pixel. The algorithm does this efficiently in time (linear in the number of particles) by creating an occupancy map at each scale.

The same process is used to position all particles in the first video frame. For the first video frame, the algorithm adaptively sets the δ_s parameter that controls the creation of the scale map. The parameter is initially set to 10, then adjusted up and down until the number of created particles falls between 8000 and 12000. The same δ_s is used for the remainder of the video.

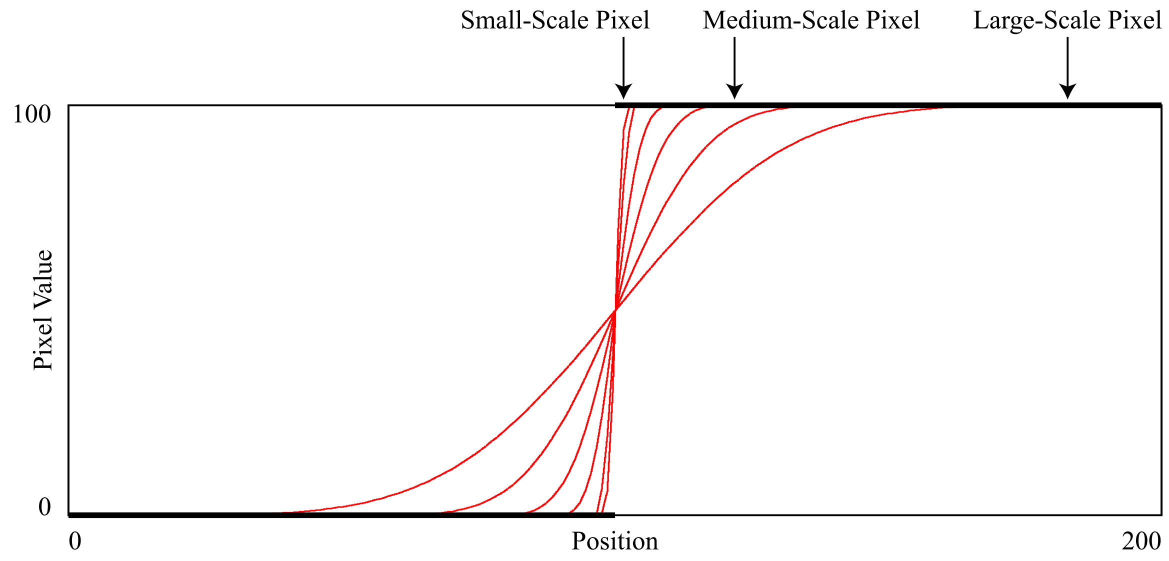


Figure 4-6: The algorithm computes a set of blurred images (red) for a given color channel (black). A pixel for which all of the images agree is considered a large-scale pixel. If the images disagree, it is a smaller-scale pixel.

Chapter 5

Interactive Selection of Video Regions

In this chapter, we focus on one application of the particle video algorithm: interactively selecting part of a video. The goal is to identify the pixels belonging to a particular object, person, or surface in the video, as specified by a human operator. The result is a time-varying spatial mask (or *matte*), identifying a subset of pixels in each video frame.

This single application has a wide variety of uses. The selected region can be removed from the video and composited into another video (as one could do via chroma-keying [74] using a blue or green background). Alternatively, the brightness, color balance, focus/sharpness, or other properties of the region can be modified (to improve the aesthetic impact of a video shot). Another application is painting onto the video; the selected region can guide the movement of paint strokes that the user applies to objects in the video.

Ideally, the user could manipulate video as easily as a photo in Photoshop [26]. Existing software takes steps toward this goal by using optical flow between key-frames. However, to be successful, this often requires a large number of key-frames. Our particle video representation reduces the user's workload via long-range tracking of video elements, allowing the user to achieve the same goal with substantially fewer key-frames. This goal, manipulating videos as easily as images, is a major target of visual effects research. The ultimate goal is not only to minimize the user's work, but also to increase the range of possible creative choices.

Our approach is similar to methods previously developed for image matting. Chuang *et al.* [22] introduce the Bayesian matting technique, which uses foreground and background color density models to obtain high-quality mattes of mixed pixels. Sun *et al.* [76] present poisson matting, a gradient-based approach to the same problem. The foreground and background color models have been combined with a graph-cut [31] segmentation to create more efficient and powerful image matting algorithms [53, 66, 86]. These methods reduce the required amount of user input.

These image matting techniques have also been extended to video. Apostoloff and Fitzgibbon [4] apply the Bayesian matting approach to video. Chuang *et al.* [21] present a similar approach using optical flow between labeled key-frames. Agarwala *et al.* [1] optimize key-framed curves to track video boundaries. Li *et al.* [52] use tracking to improve a graph-cut video matting algorithm. Wang *et al.* [85] present another graph-cut video matting algorithm that improves efficiency by using a spatiotemporal hierarchical mean-shift segmentation [23].

The remainder of this chapter presents our approach to the video matting and region selection problem. In Section 5.1 we describe the algorithm's user interface. Section 5.2 presents a method for particle clustering that we use as a pre-process for interactive labelling. Section 5.3 describes how interactive labelling uses the clustering to estimate foreground/background probabilities for each particle. These probabilities are spatially reconciled using graph cuts, as described in Section 5.4. After the interactive labelling is complete, a post process produces the final foreground/background assignment, as described in Section 5.5. Section 5.6 discusses future directions for this work.

5.1 User Interface

The algorithm's interface allows the user to provide information about what region of the video should be selected. For brevity, we refer to the object or region being selected as the *foreground* and the rest of the video as the *background*; despite this terminology, we do not assume any depth ordering (the selected object could be the furthest from the camera). We use the term *label* to refer to user-provided foreground/background designations and the term *assignment* for computed foreground/background designations (which are estimated with the help of the user's labels).

5.1.1 User-Painted Labels

The user interacts with the algorithm by painting foreground and background labels onto the video, as shown in Figure 5-1. The user can erase these labels, reverting the pixels to an unlabeled state. (Initially all pixels are unlabeled.) Standard painting mechanisms are provided, such as setting the brush size and performing flood fills.

The algorithm uses the labeled pixels as training data for a model that classifies the other pixels (as described in Sections 5.3 and 5.5). As the user draws the labels, the algorithm dynamically re-computes and re-displays the foreground/background assignment. This allows the user to quickly correct the algorithm's mistakes and to see where the algorithm is successful (and thus does not need more information). The user is free to paint labels scattered throughout the video; the algorithm does not require any completely labeled frames or any particular spatiotemporal arrangement of the labels.

The user can adjust the visualization so that it does not interfere with viewing the underlying video. The algorithm can display the computed assignment for every pixel, just for particles, or just along the boundary between the foreground and background. The user can also specify the level of transparency for the visualization.

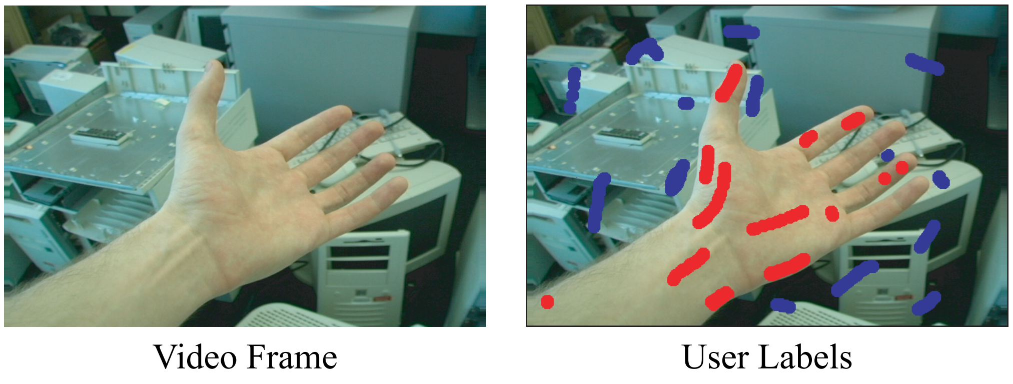


Figure 5-1: The user paints foreground (red) and background (blue) labels onto the video.

5.1.2 Compositing

The algorithm outputs a mask sequence specifying which pixels belong to the foreground for each frame. This mask sequence can be further manipulated, used to combine videos, or exported for use by other software.

The system developed for this thesis includes a variety of mask manipulation tools. The software can fill mask holes, smooth mask boundaries, expand/contract mask boundaries, and filter mask components by size. The user can also choose to combine a mask with other masks, using intersection, union, subtraction, and other operators.

The masks can be used to manipulate videos via hierarchical compositing specifications (Figure 5-2). These specifications include data sources (videos, mask sequences, images, etc.), data filters (blurring, color correction, contrast adjustment, etc.), and compositing operations (blend, add, etc.). This specification is an expression that evaluates to a video, consisting of values (data sources), unary operators (filters), and binary operators (composites). Each filter or composite operator can be modulated by a mask that specifies which part of the image is filtered or composited. Each source, filter, or compositor can also take a variety of parameters (such as the amount of blur or contrast adjustment), which can be key-framed along the video's time axis.

The user can interactively construct the hierarchical specification and change node parameters, then execute a single command to render a new video. This form of compositing specification is similar to what can be found in professional compositing programs, such as Shake [62].

5.2 Hierarchical Particle Clustering

The algorithm builds a hierarchical particle clustering that groups particles with similar properties, such as appearance, spatiotemporal position, and motion. This clustering allows the algorithm to model the distribution of particles over these properties and to find particles with a given set of properties.

Particle clustering demonstrates an advantage of discrete motion primitives such as particles: they can be grouped into higher-level motion objects that allow sophisticated yet efficient processing.

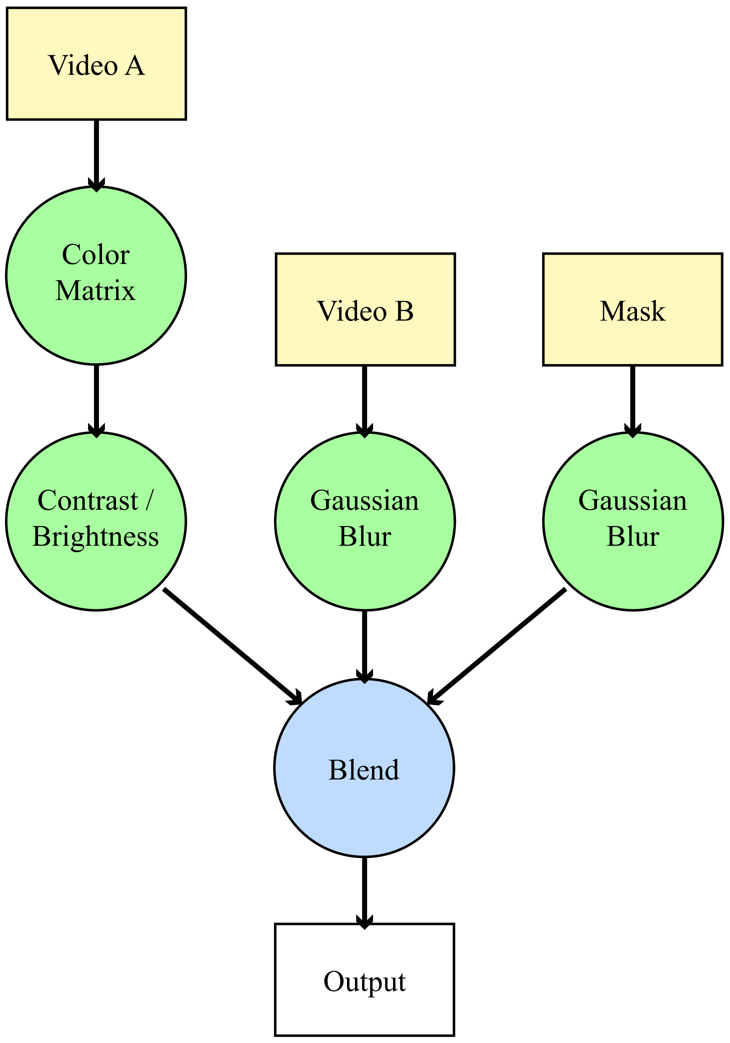


Figure 5-2: A compositing specification describes how to create a new video by combining and manipulating existing videos. The specification can include data sources (yellow), data filters (green), and compositing operators (blue), each with optional parameters.

5.2.1 Particle Feature Vectors

Particle clusters can be constructed using any distance function defined on pairs of particles. For simplicity, we assign a feature vector to each particle and use Euclidean distance within this feature space.

At any given frame index t along particle i's trajectory, we define a vector that combines the particle's first three appearance channels, position, and motion:

$$f_i(t) = \{c_i^{[0]}(t), \beta_c \cdot c_i^{[1]}(t), \beta_c \cdot c_i^{[2]}(t), \beta_s \cdot x_i(t), \beta_s \cdot y_i(t), \beta_t \cdot t, \beta_m \cdot u_i(t), \beta_m \cdot v_i(t)\}. \quad (5.1)$$

Here $c_i^{[1]}(t)$ and $c_i^{[2]}(t)$, denote the green minus red and green minus blue channels (scaled as described in Section 4.2). The β factors control the trade-off between different dimensions in the feature space (which have different units and different relative importances for the clustering). We use $\beta_c = 3$, $\beta_s = 0.02$, $\beta_t = 0.02$, and $\beta_m = 2$. For each particle, the mean of these vectors (averaged along t) provides the particles feature vector f_i.

5.2.2 Clustering the Particles

After assigning a feature vector to each particle, the algorithm clusters particles within the feature space. The algorithm starts with a single cluster containing all the particles, then recursively splits the cluster into sub-clusters.

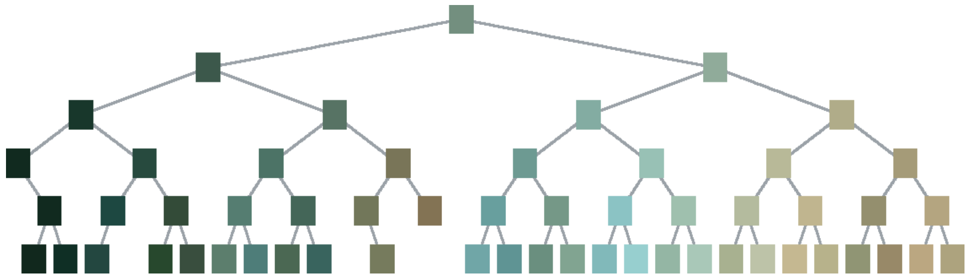
To split a cluster, the algorithm uses K-means clustering [36] with $K = 2$, producing a pair of children for each cluster. If a cluster has fewer than 20 particles, the algorithm does not split it further. We assign each cluster a feature vector that is the mean of its particle feature vectors.

The resulting cluster tree has a depth that is approximately logarithmic in the number of particles. (The K-means algorithm does not typically split each cluster evenly.)

5.3 Foreground/Background Modelling using Clusters

The clustering depends only on the particle video, not on the user's labels. After the algorithm has clustered the particles, the user can start the interactive labelling process.

During interactive labelling, the algorithm builds models of the distribution of the user-labelled foreground and background pixels. For example, given a set of labeled pixels, the algorithm could determine that the foreground is brown and blue while the background is green, orange, and white. Sometimes color may not be sufficient to distinguish the fore-


Particle Clustering

Video Frame

Figure 5-3: Each cluster is represented as a rectangle in this plot, colored according to the mean color of the particles within the cluster. For the purpose of visualization, the tree is truncated when a cluster has fewer than 100 particles or is below 6 levels deep.

ground and background; blue pixels could occur both in the foreground and background. In this case, the algorithm distinguishes the foreground and background according to time, space, and motion within the video.

Our algorithm represents these foreground and background properties using particle clusters. A standard matting algorithm may represent the foreground and background distributions using Gaussian mixture models in the color/space/time dimensions. Instead, we use clusters to provide the structure of the model.

5.3.1 Pixel Distributions

For each cluster j, the algorithm estimates a simple foreground/background probability using a foreground count $F(j)$ and background count $B(j)$, based on the user-specified labels. When the user draws a new labelled region, each pixel in the region is recursively inserted into the cluster tree.

To add a labelled pixel to a given cluster, the algorithm increments the foreground or background count, then recurses into whichever child cluster is closest to the pixel. To measure distance between a pixel and cluster, we define a pixel feature vector:

$$f(x,y,t) = \{c^{[0]}(x,y,t), \beta_c \cdot c^{[1]}(x,y,t), \beta_c \cdot c^{[2]}(x,y,t), \beta_s \cdot x, \beta_s \cdot y, \beta_t \cdot t\}. \qquad (5.2)$$

This is similar to the particle feature vector, but lacks the final two dimensions (particle motion). At each level, we only recurse into a single child, so the total computation is roughly logarithmic in the number of particles.

5.3.2 Particle Probabilities

To assign a probability to each particle, we use the cluster foreground/background counts. The algorithm recurses down the cluster tree, finding the lowest level (smallest cluster) with a sufficient number of labelled pixels. We choose a low-level cluster because it provides information that is more specific to the particle's properties than a high-level cluster. Specifically, we use the lowest cluster along each branch for which the labelled pixel count ($F(j)+B(j)$) is greater than 10. For each particle i within cluster j, we assign a foreground probability according to the cluster's counts:

$$P_{FG}(i) = \frac{F(j)}{F(j)+B(j)}. \qquad (5.3)$$

We compute the particle probabilities lazily, only updating the particles in the clusters for which the foreground/background counts have changed since the last update.

Note that this algorithm uses the same data structure (a particle cluster tree) to model the distribution of foreground and background properties and to efficiently find particles with those given properties. The algorithm incorporates new information (new labels) without any need for re-balancing; it automatically adjusts the level of detail within the cluster tree according to the currently available data. This allows an interactive algorithm in which the user immediately sees global changes as a result of providing new local information.

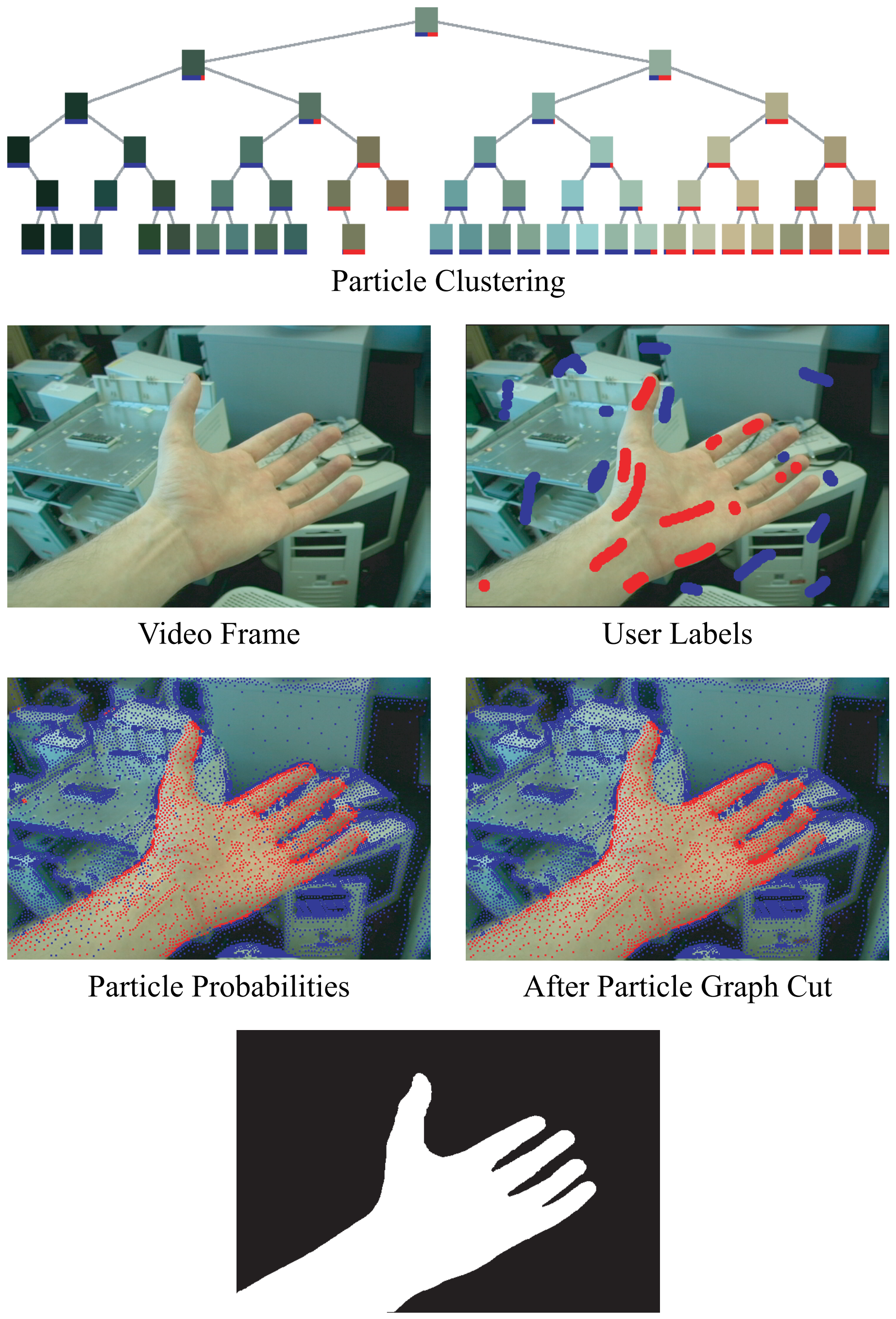


Figure 5-4: Each cluster (top) is augmented with a bar that shows the fraction of foreground pixels (red) vs. background pixels (blue). The clusters are used to assign a foreground probability to each particle (Section 5.3). These probabilities are spatially reconciled using a graph cut (Section 5.4). As a post-process, the algorithm uses a per-pixel graph cut along the particle assignment boundary to generate the final matte (Section 5.5).

5.4 Label Propagation using Particle Graph Cuts

The particle clusters provide a probability that each particle belongs to the foreground. We use these probabilities as local evidence in a belief propagation problem that spatially distributes and reconciles the local information in order to compute a binary assignment for each particle.

For the sake of computational efficiency, we use a graph-cut method to perform the belief propagation. The algorithm constructs a weighted undirected graph in which each particle is represented by a node. Graph edges represent particle links. Additionally, each node is linked to a node representing the foreground and a node representing the background (Figure 5-5). After the graph is cut, particles attached to the foreground node are given foreground assignments and particles attached to the background node are given background assignments.

The use of particles as nodes, rather than pixels, provides a substantial increase in efficiency. This is similar to approaches that segment pixels into groups to accelerate graph-cut matting in images [53] and video [52, 85]. In most cases, our particles exhibit more temporal longevity than these segmentations, allowing greater graph-cut efficiency. (Determining the assignment for a single particle provides information about many pixels over many frames.)

Using particle foreground probabilities, the algorithm sets the edge weights between particle nodes and foreground/background nodes. Let $w_{FG}(i)$ denote the edge weight between the foreground node and the node for particle i. Let $w_{BG}(i)$ denote the corresponding background edge weight. We set the weights so that the higher the foreground probability, the larger the cost of separating the particle node from the foreground node:

$$w_{FG}(i) = P_{FG}(i), \tag{5.4}$$

$$w_{BG}(i) = 1 - P_{FG}(i). \tag{5.5}$$

Any particle that intersects a foreground label is locked to the foreground by setting

$w_{FG}(i) = 1000$. Similarly, any particle that intersects a background label is locked to the background by setting $w_{BG}(i) = 1000$. (If, for some reason, a particle intersects conflicting labels, we do not lock the particle to either assignment.)

We set the inter-particle edge weights using link weights. Let $w(i, j)$ denote the weight of the edge corresponding to a link between particles i and j:

$$w(i,j) = s_{part} \cdot l_{ij}. \tag{5.6}$$

Here l_{ij} is the link weight defined in Section 4.4). The parameter s_{part} is a smoothness factor that influences the spatial contiguity of the resulting assignment; we find $s_{part} = 0.2$ provides a good results.

5.5 Post Processing

After computing particle assignments, the algorithm refines the boundary between foreground and background regions. To maximize the quality of the final matting, we do this as a non-interactive batch post-process.

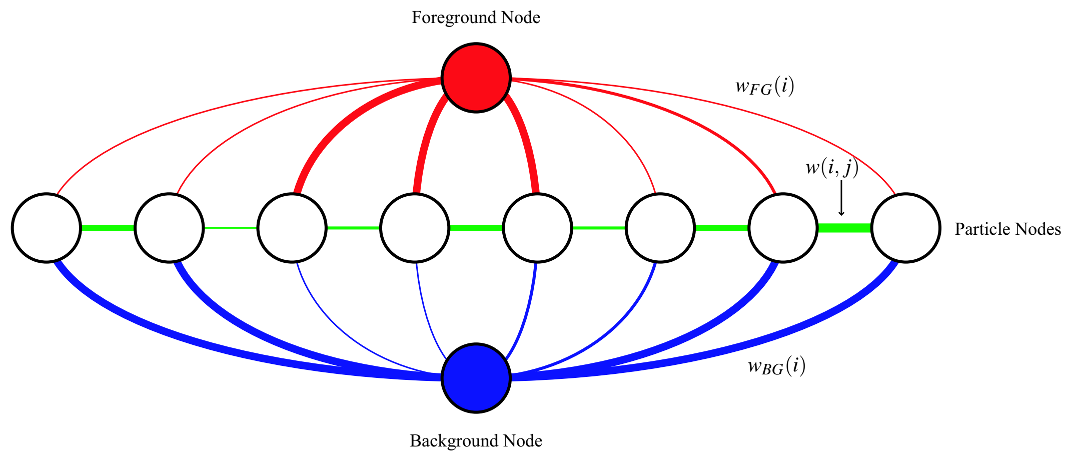
5.5.1 Pixel Models

To compute probabilities for individual pixels (rather than particles) the algorithm builds Gaussian mixture models of the foreground and background labelled pixels. For each labelled pixel, the algorithm computes a feature vector $f(x, y, t)$ as described in Section 5.3.1. The algorithm then builds separate Gaussian mixture models for the foreground and background labels. For each label, the algorithm positions Gaussians in the feature space using the K-means algorithm [36] to find $K = 70$ cluster centers.

Let μ_k denote the kth component's mean and w_k denote its mixture weight. We set $\sigma = 5$. The combined density for a pixel (x, y, t) is:

$$D(x,y,t) = \sum_{k=1}^{K} w_k N(f(x,y,t); \mu_k, \sigma^2). \tag{5.7}$$

For each unlabeled pixel, the algorithm computes a foreground density $D_{FG}(x, y, t)$ and



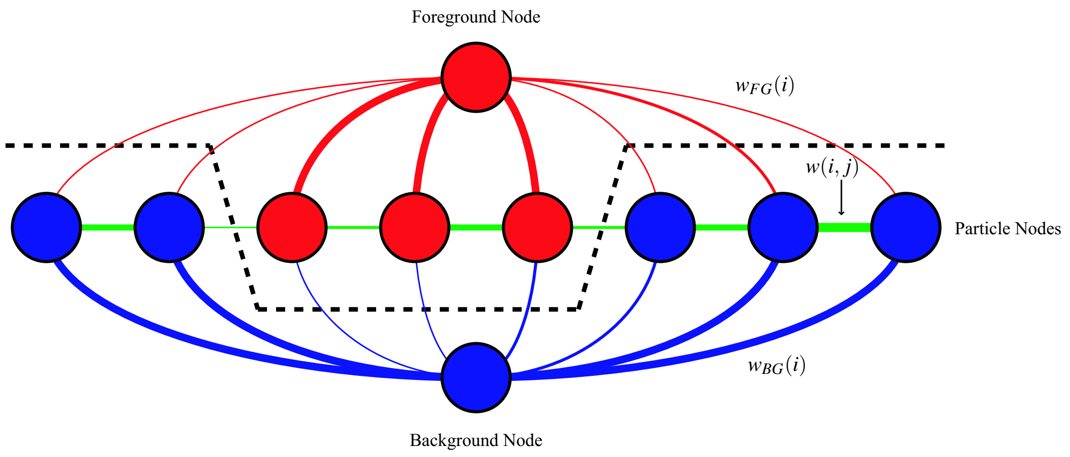


Figure 5-5: The algorithm constructs a graph (top) with a node for each particle (white circles) and special nodes denoting the foreground (red) and background (blue). The foreground and background weights (red and blue curves) are set according to the particle cluster models. Other edges (green) describe the connection strength between particles. (Thicker lines denote stronger weights in this hypothetical example.) The minimum cut through this graph (black dashed line, bottom) corresponds to assigning each particle to the foreground or background.

background density $D_{BG}(x,y,t)$ according to the Gaussian mixture models. If the pixel is sufficiently likely in either model (if $D_{FG}(x,y,t) > \varepsilon$ or $D_{BG}(x,y,t) > \varepsilon$ for $\varepsilon = 0.0001$), then the Gaussian mixture models determine the pixel's foreground probability:

$$P_{FG}(x,y,t) = \frac{D_{FG}(x,y,t)}{D_{FG}(x,y,t) + D_{BG}(x,y,t)}. \tag{5.8}$$

Otherwise, if the pixel does not fall into either model, the algorithm assigns the probability according to the distance to the nearest neighbor in each model.

5.5.2 Boundary Graph Cut

For each frame, the algorithm computes a pixel assignment for a band of pixels along the boundary between the foreground and background particles. Running the graph-cut algorithm on the boundary pixels provides a substantial computational savings over running it on full video frames.

To find the band of boundary pixels, the algorithm interpolates the particle assignments using a constrained Delaunay triangulation [54]. The resulting assignment map is blurred using a 9 by 9 box filter. Any pixel within the blurred boundary is defined to be part of the boundary region.

We then construct a pixel-based graph-cut problem in the boundary region. Let $w_{FG}(x,y)$ denote the weight of the edge from the foreground node to the pixel at (x,y) and $w_{BG}(x,y)$ denote the corresponding background weight. (We omit the t indexing, because this operation deals with a single frame at a time.) We set the weights according to the pixel probabilities:

$$\begin{aligned} w_{FG}(x,y) &= P_{FG}(x,y), && (5.9)\\ w_{BG}(x,y) &= 1 - P_{FG}(x,y). && (5.10) \end{aligned}$$

Pixels along the foreground edge of the boundary region are locked to the foreground by setting $w_{FG}(x,y) = 1000$. Similarly, pixels along the background edge are locked to the background by setting $w_{BG}(x,y) = 1000$.

The algorithm then adds edges between diagonally, horizontally, and vertically adjacent pixels. Let $w(x_1,y_1,x_2,y_2)$ denote the weight of the edge between the node for pixel (x_1,y_1) and the node for pixel (x_2,y_2). The cost of separating a pair of pixels is related to the appearance difference between the pixels:

$$w(x_1,y_1,x_2,y_2) = s_{pixel} \cdot \left(1 - \frac{g(x_1,y_1,x_2,y_2) - g_{min}}{g_{max} - g_{min}}\right) \tag{5.11}$$

Here $g(x_1,y_1,x_2,y_2)$ denotes the gradient between the pixels (averaged over the r,g,b color channels), clamped to lie between $g_{min} = 5$ and $g_{max} = 15$. The parameter s_{pixel} is a smoothness factor, which we set to 0.01.

The pixel assignment provided by the graph cut is binary. For many purposes, this is sufficient, but for some compositing operations, partial assignments are desirable to represent transparency, motion blur, fine structures, and otherwise mixed pixels. Standard methods for this include Bayesian matting [22] and poisson matting [76].

5.6 Discussion

This application is useful for several key problems in visual effects, but some work remains before these tools are ready for widespread usage. The algorithm should be easier to use, more robust, and have fewer parameters.

In the future, we plan to explore interfaces for drawing meta-masks that apply operations to the underlying region mask. These operations will include filling holes, expanding/contracting the boundaries, and forcing areas to foreground or background assignments. These meta-masks could themselves follow the particles. The main challenge is providing a simple and coherent way of letting the user working with all these interacting masks.

Additionally, advanced compositing techniques could be incorporated with these algorithms. We could use the particles to enforce temporal coherence of texture synthesis for hole filling or inpainting [92, 77, 24]. Particles could also improve the temporal coherence of poisson compositing methods [64].

A system that combines these tools would be very useful to a large number of people working in creative video fields. The particle video representation provides an effective foundation for such a system.

Chapter 6

Evaluation

In this chapter we evaluate the algorithm on a variety of videos, including footage of challenging real-world scenes and contrived cases designed to test the limits of the algorithm. We discuss quantitative evaluation measures and compare results obtained from different algorithm configurations.

6.1 Visualization

Viewing particle videos is an important part of developing, debugging, and evaluating the algorithm. Our system provides visualizations both of particle videos and the steps used to create them.

The basic particle video viewer displays particles and links overlayed onto video frames that can be browsed via a temporal slider interface. The user can choose to have particles colorized by data energy, distortion energy, combined energy, active channels, proximity to termination, or original frame color (as shown in the figures and videos for this thesis). Particle links can be colorized by length, weight, or distortion energy. The background can switched from the video frame to the flow field, flow gradient magnitude map, or scale map. The user can zoom in to investigate the placement of particles and links with respect to individual pixels.

The algorithm also provides space-time plots of particle trajectories. The user selects particles by drawing a mask onto one or more video frames. All of the particles that pass through the selection mask are plotted in space and time (either x vs t or y vs t). The user can also choose to plot a single particle and its linked neighbors (Figure 6-1). The same colorization modes are available for these plots.

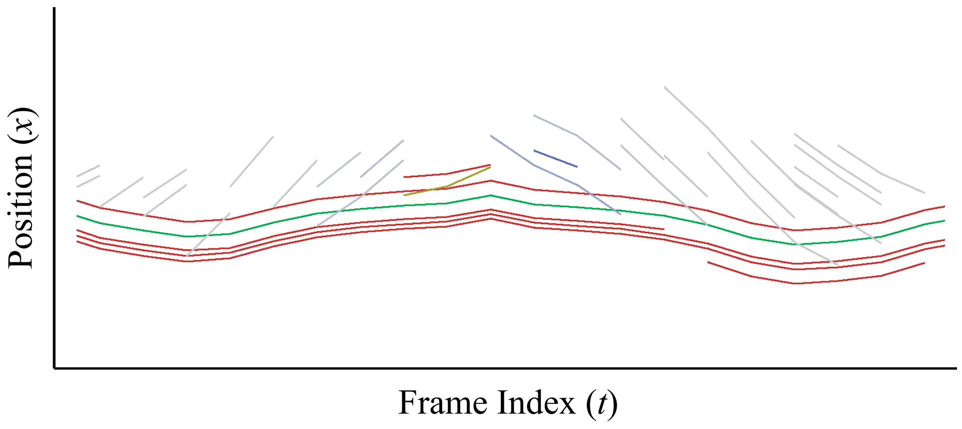


Figure 6-1: This space-time plot shows a single particle (green) near an occlusion boundary and other particles linked to this particle. The linked particles are shown only for frames in which the links are active. They are colored by link weight; red indicates a high weight and gray indicates a low weight.

6.2 Evaluation Measures

Objectively evaluating the algorithm's correctness is difficult given the lack of ground-truth data. The ideal evaluation measurement should allow comparison with future particle video algorithms and with non-particle approaches to long-range motion estimation.

One option is computing pixel value differences between distant frames according the estimated correspondences. The algorithm can produce dense correspondence fields by interpolating particles that exist in common between the frames. We can then measure the pixel difference between the frames according to this projection. Unfortunately, this measure is not monotonically related to correctness; the algorithm can obtain a lower error by incorrectly deforming the correspondence field to reduce occlusions. If we allow the algorithm to label pixels as occluded, the algorithm can obtain a lower error by errantly labelling non-occluded pixels as occluded.

Alternatively, we can evaluate the algorithm by building a particle video for a given test video, then independently building another particle video for a temporally reversed copy of the test video. If the algorithm is successful, we expect consistent correspondences between the videos. However, this approach is also foiled by occlusion. Without ground truth occlusion labels, we do not know which correspondences should be consistent. The algorithm can once again obtain a better consistency score simply by marking difficult areas as occluded.

One solution is rendering synthetic videos with known correspondences. To mimic challenging real-world videos, these rendered videos should include deforming objects, complex reflectance, detailed geometry, motion blur, unstable camera motion, optical artifacts, and video compression. All of these factors can be obtained using modern commercial rendering software, but setting up a wide variety of photo-realistic scenes would require substantial effort. In the future we envision that rendering such scenes will be easy enough that researchers will create a diverse set of ground-truth videos.

The particle video algorithm can also be evaluated using applications such as noise removal [10] and super-resolution [95]. We can create noise-removal evaluation datasets by adding synthetic noise to a low-noise video. Similarly, we can create super-resolution ground-truth test sets by reducing the size of high-resolution videos. We can then numerically evaluate particle-based solutions to the noise-removal and super-resolution problems. Unfortunately, these results would be strongly dependent on the particular selection of noise-removal and super-resolution methods, introducing a number of additional parameters and uncertainties.

For the purposes of this thesis, we quantify the algorithm's performance using videos that are constructed to return to the starting frame. We replace the second half of each evaluation video with a temporally reversed copy of the first half. This is similar to the forward/backward evaluation described above, but we construct only one particle video for each dataset, rather than two. We then compute the fraction of particles that survive from the start frame to the end frame (which is identical in appearance to the start frame). For each of these particles, we compute the distance between its (x, y) position in the start frame and (x, y) position in the end frame. This spatial error value should be near zero.

Like the methods described above, this evaluation scheme is flawed. The algorithm can easily obtain a lower spatial error by pruning more particles (at the cost of a lower particle survival rate). Furthermore, by allocating less particles near occlusions and more particles in other regions, the algorithm can both increase the survival rate and decrease the spatial error.

Another problem with this return-to-start evaluation is that the algorithm may be able to unfairly recover from mistakes. This prevents a comparison with techniques that refine concatenated flow fields; a good refinement algorithm should be able to find the trivial (zero flow) field mapping the first frame to the last frame, even if it has trouble with intermediate frames.

Because of these issues, we provide the evaluation for descriptive purposes only. These measures should not be used to compare the algorithm with future particle video algorithms.

6.3 Evaluation Videos

Our evaluation dataset consists of 20 videos, representing a range of real-world conditions and contrived test cases. These videos together include a variety of scenes, lighting conditions, camera motions, and object motions.

The videos are recorded at 29.97 non-interlaced frames per second in the MiniDV format using a Panasonic DVX100 camera. The video frames are 720 by 480 pixels with a 0.9 pixel aspect ratio (width/height). Before constructing a particle video, we crop four pixels from the left and right of each frame to remove sensor artifacts.

The videos are summarized in Table 6.1 and described here:

- **VBranches.** A hand-held camera records trees with many branches of different thicknesses (Figure 6-4).
- **VCars.** Several cars move through the scene (Figure 6-4).
- **VHall.** The camera operator walks down an office hallway (Figure 6-4).
- **VHand.** The hand deforms as it moves in front of the background, observed by a hand-held camera (Figure 6-4).

- **VMouth.** The head changes orientation while the mouth changes shape (Figure 6-4).
- **VPerson.** A person walks past the camera as it pans on a tripod (Figure 6-4).
- **VPlant.** A hand-held camera observes a plant and office clutter (Figure 6-5).
- **VShelf.** The camera moves vertically on a small crane device (Figure 6-5).
- **VTree.** The leaves on a tree flutter in the wind, observed from a moving hand-held camera (Figure 6-5).
- **VTreeTrunk.** The hand-held camera motion induces large-scale occlusions around a tree trunk (Figure 6-5).
- **VZoomIn.** The camera zooms in on a test pattern (Figure 6-6).
- **VZoomOut.** The camera zooms out from a test pattern (Figure 6-6).
- **VRotateOrtho.** The test pattern rotates, roughly parallel to the image plane (Figure 6-6).
- **VRotatePersp.** The test pattern rotates, viewed from off the axis of rotation (Figure 6-6).
- **VRectSlow.** A rectangular solid, covered in a test pattern, rotates in front of a patterned background (Figure 6-6).
- **VRectFast.** The sequence is twice the rate of VRectSlow. (Figure 6-7).
- **VRectLight.** A rectangular solid, covered in a test pattern, rotates in front of a patterned background, under one-sided lighting (Figure 6-7).
- **VCylSlow.** A cylinder covered in a test pattern rotates in front of a patterned background (Figure 6-7).
- **VCylFast.** This sequence is twice the rate of VCylSlow (Figure 6-7).
- **VCylLight.** A cylinder covered in a test pattern rotates in front of a patterned background, under one-sided lighting (Figure 6-7).

Name	Camera Motion	Occlusion	Object Motion	Figure	Length
VBranches	hand-held R+T	yes	none	6-4	50
VCars	hand-held R+T	yes	R+T	6-4	50
VHall	hand-held R+T	yes	none	6-4	50
VHand	hand-held R+T	yes	R+T; deformation	6-4	70
VMouth	static	yes	R+T; deformation	6-4	70
VPerson	tripod R	yes	R+T; deformation	6-5	50
VPlant	hand-held R+T	yes	none	6-5	70
VShelf	crane T	yes	none	6-5	50
VTree	hand-held R+T	yes	R+T; deformation	6-5	70
VTreeTrunk	hand-held R+T	yes	none	6-5	50
VZoomIn	static	no	none	6-6	40
VZoomOut	static	no	none	6-6	40
VRotateOrtho	static	no	R	6-6	90
VRotatePersp	static	no	R	6-6	90
VRectSlow	static	yes	R	6-6	80
VRectFast	static	yes	R	6-7	80
VRectLight	static	yes	R	6-7	80
VCylSlow	static	yes	R	6-7	50
VCylFast	static	yes	R	6-7	50
VCylLight	static	yes	R	6-7	50

Table 6.1: The evaluation videos include various camera motions and object motions. R denotes rotation and T denotes translation.

For the videos of planar surfaces (VZoomIn, VZoomOut, VRotateOrtho, and VRotate-Persp), we replace the optical flow estimation with global parametric motion estimation.

6.4 Particle Video Configurations

We evaluate several configurations of the particle video algorithm:

- **PVBaseline.** This uses all of the parameter settings described in Chapter 4 and summarized in Table 6.2. The following configurations are modifications, as specified, of this configuration.
- **PVSweep1.** This configuration performs a single forward sweep (whereas the baseline algorithm performs a forward sweep followed by a backward sweep).
- **PVSweep4.** This sweeps forward, backward, forward again, then backward again.

Variable	Description	Value	Units	Section
σ_l	motion difference prior for link weight	1.5	pixels per frame	§4.4
α	particle objective distortion factor	1.5	N/A	§4.5.1
σ_c	channel filter size	5	frames	§4.5.1
σ_t	pruning energy filter size	1	frames	§4.6
δ	pruning energy threshold	5	N/A	§4.6

Table 6.2: These parameter settings are used for the PVBaseline configuration.

- **PVNoOcc.** This configuration ignores the occlusion maps (provided by the optical flow algorithm) during particle propagation (Section 4.3).
- **PVPruneMore.** This configuration lowers the pruning threshold to $\delta = 5$, resulting in more pruning.
- **PVPruneLess.** This configuration raises the pruning threshold to $\delta = 20$, resulting in less pruning.
- **FlowConcat.** This is a simple concatenation of flow fields (computed as described in Chapter 3) for each particle position in the first video frame (according to the PVBaseline configuration). The flow trajectories are terminated when they enter an occluded region, as determined by the flow algorithm.

6.5 Evaluation Results and Discussion

The return-to-start evaluation is summarized in Figures 6-2 and 6-3. In each case, the particles return to their starting positions with lower error than the trajectories formed by concatenating flow vectors. As expected, concatenated flow vectors drift. Ideally the plots should be perfectly symmetrical (since the videos are temporally symmetrical); in some cases, the particle trajectories deviate from this symmetry, suggesting occasional failures.

The yellow lines indicate the fraction of surviving particles. For each video, particles disappear because they leave the frame boundaries or become occluded. A roughly constant survival fraction across the second half (returning to the start) indicates that few particles are lost for other (spurious) reasons.

Configuration	Return Fraction	Return Error	Mean Count	Mean Length	Run Time
FlowConcat	0.81	4.05	N/A	N/A	N/A
PVBaseline	0.65	1.12	13260	31.68	40.53
PVSweep1	0.71	0.99	11468	28.96	15.73
PVSweep4	0.66	1.24	14644	30.51	73.65
PVNoOcc	0.66	1.17	13178	32.90	57.47
PVPruneMore	0.43	0.83	14684	23.11	71.69
PVPruneLess	0.75	1.73	13304	37.15	20.11

Table 6.3: For each configuration, we evaluate the algorithm on videos that are constructed to return to the start frame (Section 6.2). We report the mean fraction of particles that survive to the end frame and the mean spatial distance between the each surviving particle's start and end frame positions. We also report the mean particle count, mean particle length, and mean per-frame running time. The running time does not include optical flow computation; it is a pre-process shared by all the algorithms. All statistics are averaged over the 20 videos described in Section 6.3.

Table 6.3 provides a comparison of the algorithm configurations described in Section 6.4. As expected, ignoring the occlusion masks provided by the flow algorithm results in higher error and a larger fraction of surviving particles. Also, as expected, additional pruning raises the accuracy while lowering the survival fraction.

Additional sweeps across the video add more particles, mostly in areas where other particles were previously pruned (the more difficult regions of the video). Thus, even though a single sweep has lower error, not necessarily providing a better model of the motion. (This is why, as discussed in Section 6.2, the return-to-start measure should not be used alone to evaluate particle videos.)

Table 6.4 gives a breakdown of the running time for each configuration. In each configuration, almost half the running time is consumed by running the sparse linear system solver. The remaining time is mostly spent constructing the linear system. The computational costs of adding, linking, and pruning particles are all relatively small.

All of the data used to generate these results, including the videos, plots, and particle trajectories are available online at `http://rvsn.csail.mit.edu/pv/`.

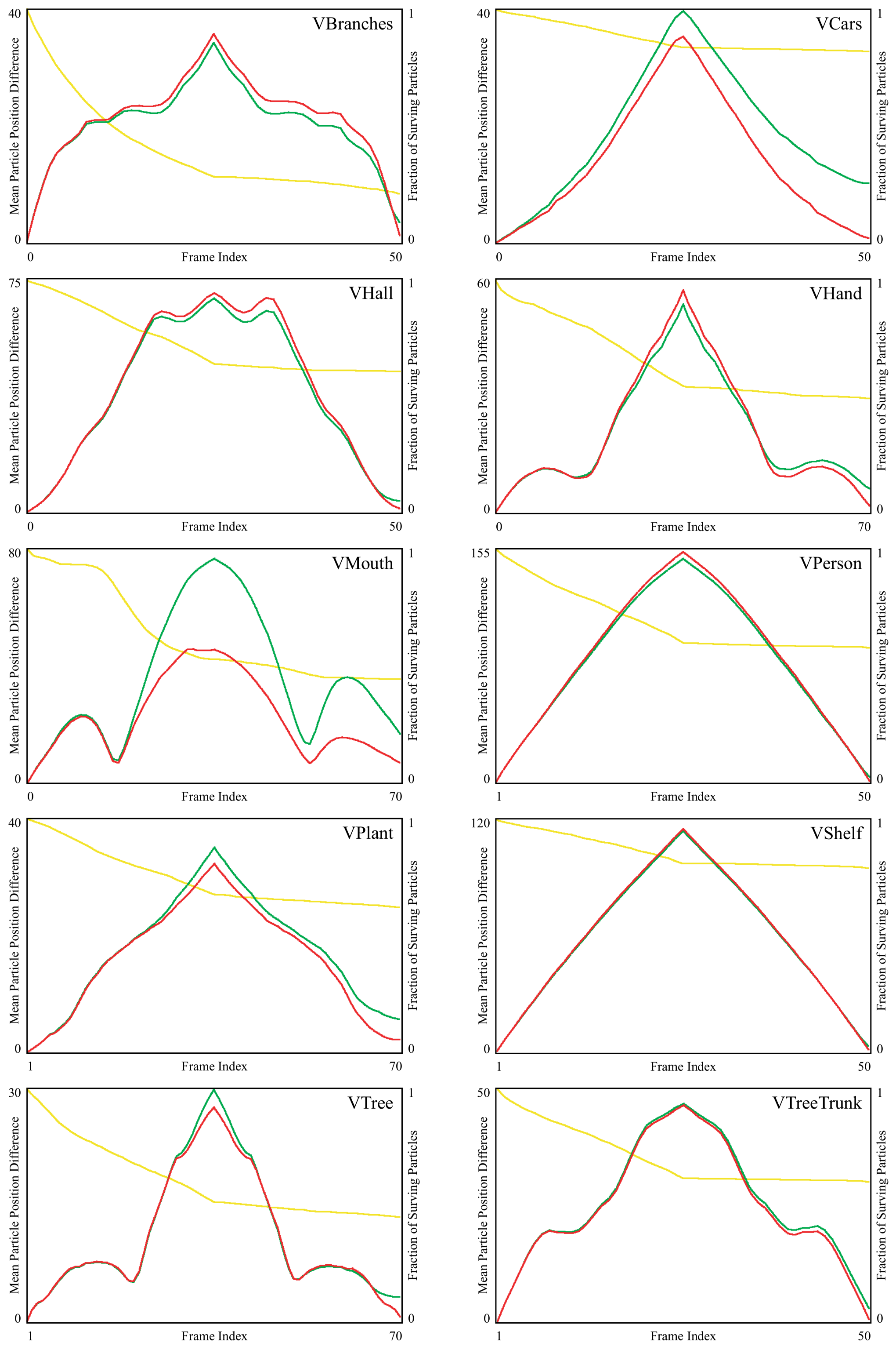


Figure 6-2: Each plot shows the fraction of surviving particles (yellow, right axis) and mean distance (red, left axis) of these particles from their positions in the start frame. The green lines denote concatenated flow vectors. As described in Section 6.3, the videos are temporally mirrored, so we expect all unoccluded particles to return to their start positions.

Configuration	Add Time	Link Time	Opt. Time	Solver Time	Prune Time	Total Time
PVBaseline	2.99	1.02	11.57	18.54	1.39	40.53
PVSweep1	1.29	0.41	4.14	7.27	0.40	15.73
PVSweep4	6.14	2.05	21.06	31.55	3.01	73.65
PVNoOcc	3.62	0.79	17.36	27.47	2.01	57.47
PVPruneMore	4.21	3.39	21.01	32.42	3.58	71.69
PVPruneLess	2.16	0.83	4.70	8.57	0.45	20.11

Table 6.4: For each configuration, we report the mean per-frame running time in seconds. The **Opt.** time includes optimization overhead but not the execution of the solver or the update of the energy values at the end of the optimization (which are reported in their respective columns). The total time includes some additional overhead, such as computing the adaptive scale map factor (Section 4.7).

6.6 Future Work

The particle video algorithm has some limitations, mostly related to the interpretation of occlusion. We discuss these problems and possible solutions in the following sub-sections.

6.6.1 Linking and Distortion

The largest difficulty in creating a particle video is handling occlusion boundaries. The current implementation represents occlusion boundaries using weighted links between particles. This linking scheme fails because it occasionally allows incorrect distortion or prevents correct distortion (such as that caused by non-rigid object deformation or changes in viewpoint). We hope to explore statistical and/or geometric methods for distinguishing correct and incorrect distortion.

The best approach may involve a hybrid of flow-based and particle-based occlusion handling. Flow methods provide the advantage of accounting for subtle image details, while the particle methods provide easier handling of long temporal ranges (and indeed we expect occlusions to be clearest in long temporal ranges). A single optimization could include both flow and particle objectives, possibly estimating flow over a range of different temporal scales. This optimization could be directed toward occlusions by identifying high-error manifolds in the spatiotemporal video volume.

Another possible way to handle occlusions is to create particle motion spaces. Each particle could be projected into a space such that particles with similar motion are close to one another and particles with different motions are not. This would allow efficient querying to find a set of particles with motion similar to a given particle (extending beyond the set of particles linked to the given particle). One option would be assigning a motion trajectory distance to each link (as is currently done in Section 4.4) then running Isomap [80] to project all of the particles into a low-dimensional (perhaps 2D or 3D) space. Hopefully independently moving objects would appear as distinct clusters in the space (certain motion patterns would appear as filaments or manifolds running through the motion space). This low-dimensional motion description could be used to set link weights and as additional feature dimensions for particle clustering (Section 5.2).

This motion space could also be used to help re-acquire image regions that are briefly occluded. To do this, the algorithm could assign a hypothetical trajectory to each recently pruned particle as it sweeps through the video. The algorithm could compute hypothetical positions by fitting a regression model to the motion of active particles that are nearby in the motion space. At each frame, the algorithm would then check whether the particle appearance matches the image at the hypothetical particle position; if so, the particle can be re-activated.

6.6.2 Particle Density

Another aspect of occlusion handling is deleting and creating particles in areas that become occluded or disoccluded. This process is less complicated than link weighting, but still rather difficult.

The current algorithm uses a scale map to create particles, but not to delete particles. We experimented with a method for pruning particles in over-dense areas according to the same scale map. To do this, we defined an *age* for each particle: the number of frames between the current frame and the furthest frame in which the particle is active. We then added the particles to the scale map in order of decreasing age. If the particle is added to a location that already has a particle (according to the scale occupancy maps), we deactivate it in the current frame. (In other words, in the event of overcrowding, old particles are allowed to stay while young particles are pruned.) To obtain some hysteresis, we check a scale that is two levels lower, so that the algorithm has a range of permissible particle densities. This approach did eliminate over-crowding that sometimes occurred when an image structure moved across a uniform background. However, it also resulted in a many short-lived particles. Further research could attempt to find a better trade-off between these effects.

In the future, we could also explore spatial regularization of addition and pruning, based on the observation that particles tend to be pruned or added when their neighbors are added or pruned. A set of particles with similar motions should have similar lifetimes.

One difficulty with regularizing particle lifetimes is that the regularization should be quite weak; a slow-moving occlusion boundary may result in only a few particles being added/deleted in any given frame. In fact, we should allow singleton additions and deletions.

Also this regularization would require several new parameters describing the strength and spatiotemporal extent of the regularization and how it interacts with the standard addition and pruning processes. Perhaps the best solution would be an entirely new approach to managing particle density. For example, we could use the gradient of the particle motion field to modulate the density (placing more particles near occlusions boundaries and fewer in areas of uniform motion), rather determining particle density solely by image scale.

6.6.3 Theoretical Framework

In the future we expect researchers to develop better theoretical frameworks for particle video representation and estimation. This could allow the current algorithm to be reformulated in a simpler and more coherent way, reducing the number of parameters and rules used to construct a particle video.

The main components of the algorithm that could use a stronger theoretical foundation are particle linking and particle creation/deletion. These are crucial parts of the algorithm, but they currently are undesirably complex and do not fully provide the desired behavior (as discussed in Sections 6.6.1 and 6.6.2).

To unify the algorithm, we could develop a set of physical rules that describe or constrain particle motion. This could be analogous to the brightness constraint equation that is the basis of most optical flow algorithms. However, physically characterizing long-range motion is difficult. The brightness constraint equation is only valid over short ranges (when the surfaces can be approximated as Lambertian). Geometric constraints, such as the fundamental matrix and trifocal tensor, handle longer temporal ranges, but do not apply to non-rigid objects.

Mathematics could provide another theoretical basis for particle video estimation. This could be analogous to how optical flow algorithms are derived as discrete approximations to continuous variational problems. One could imagine a mathematical representation that behaves like particles of infinite density. One could then formulate methods for particle estimation that are approximations to the desired behavior of the continuous particles fields.

Finding a continuous representation that captures the long-range characteristics of particles may be difficult. For a specific reference frame we can represent particle motion as a function with an x, y domain that maps each point to a trajectory $(x(t), y(t))$. However, we need a parameterization that does not depend on a specific choice of reference frame; we want to be able to characterize the motion of scene points that appear in frames other than the reference frame.

One option is to return to a flow-based representation, which can be viewed as the derivative of a particle-based representation. The main goal of the particle video algorithm is to move beyond a flow-based representation, but it may be that our theoretical reasoning about particles will have to occur in the derivative/flow domain. Much theoretical work has already been done in the area of flow. The challenge would be augmenting this with long range constraints that make statements about video properties along trajectories obtained from integrals of flow-based representations. This approach could borrow mathematical machinery from differential equations and applications of differential equations, such as fluid dynamics.

6.6.4 Other Areas of Future Research

Particles are intended to be small point features in order to reduce the likelihood of straddling occlusion boundaries, but for particles far from occlusion boundaries, we could use larger areas of support. These areas of support could be characterized using invariant feature descriptors, such as SIFT descriptors [17], which would allow larger changes in scale and reflectance. (The current model allows slow changes in appearance for a particle, but does not obtain the level of invariance provided by some of these descriptors.) Invariant feature descriptors could also be used to re-acquire previously occluded regions.

We could also explore world-space constraints for particle optimization. We have avoided geometric constraints because the algorithm must be good at handling non-rigid cases. However, once the non-rigid cases are well-modelled, we can obtain further performance gains by using geometric constraints to improve the rigid cases.

Another area of future investigation is segmentation-based representations of long-range motion. Several algorithms [96, 87, 85] use segments as simple, small (but adaptively-sized), spatiotemporal primitives. Combining elements of a segmentation-based approach with a particle-based approach could provide better handling of appearance changes and occlusion boundaries.

6.7 Conclusion

The particle video algorithm provides a new approach to motion estimation, a central problem in computer vision. Dense long-range video correspondences could improve methods for many existing vision problems, in areas ranging from robotics to filmmaking.

Our particle representation differs from standard motion representations, such as vector fields, layers, and tracked feature patches. Some existing optical flow algorithms incorporate constraints from multiple frames (often using a temporal smoothness assumption), but they do not enforce long-range correspondence consistency. Our algorithm improves frame-to-frame optical flow by enforcing long-range appearance consistency and motion coherence.

Current limitations of the particle video algorithm arise from our methods for positioning particles, rather than a fundamental limitation of the particle representation. Starting with the particle tools presented in this thesis, we believe researchers will soon develop better particle video algorithms. By making our data and results available online, we hope others will explore the particle video problem.

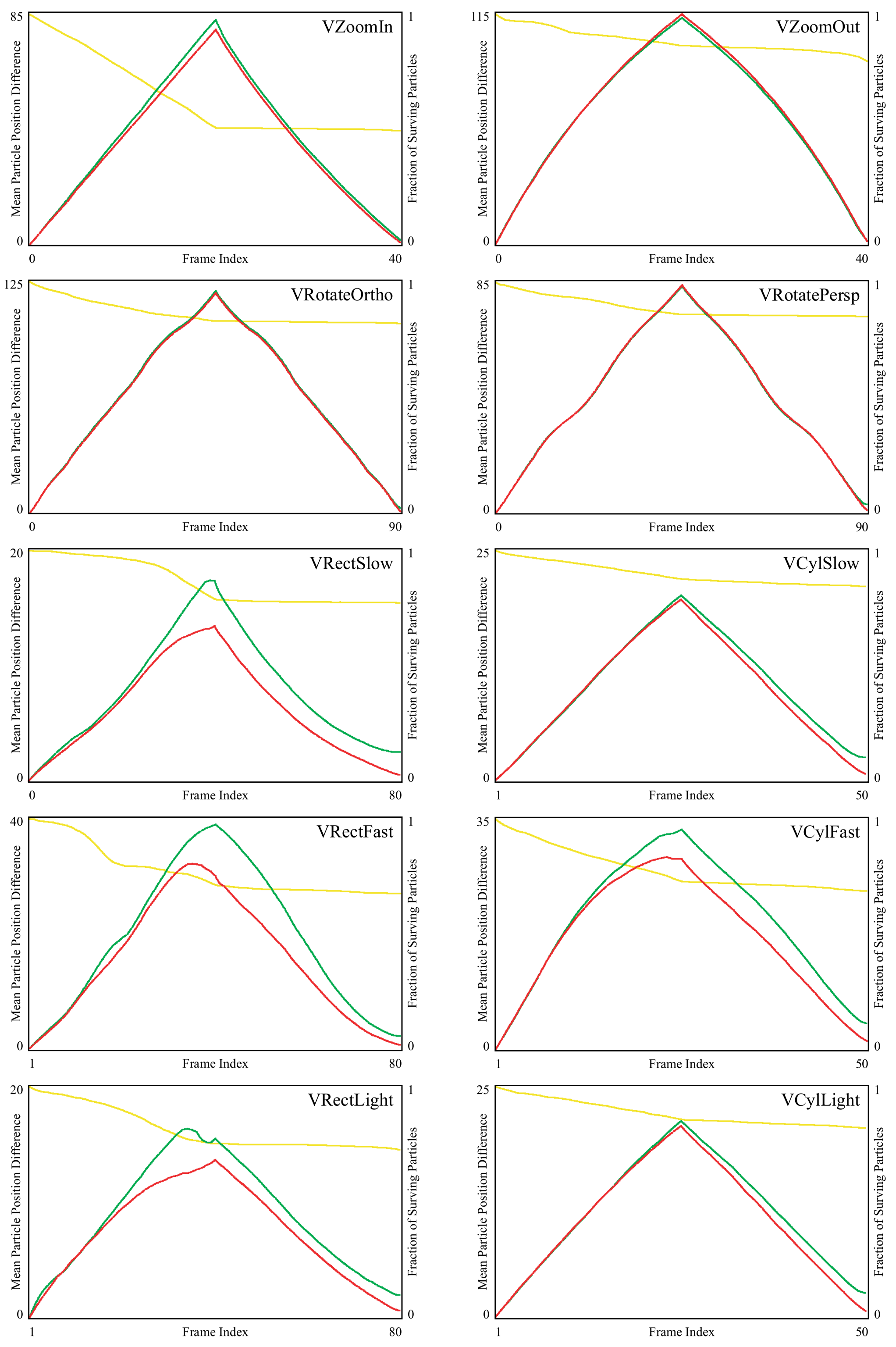


Figure 6-3: Each plot shows the fraction of surviving particles (yellow, right axis) and mean distance (red, left axis) of these particles from their positions in the start frame. The green lines denote concatenated flow vectors. As described in Section 6.3, the videos are temporally mirrored, so we expect all unoccluded particles to return to their start positions.

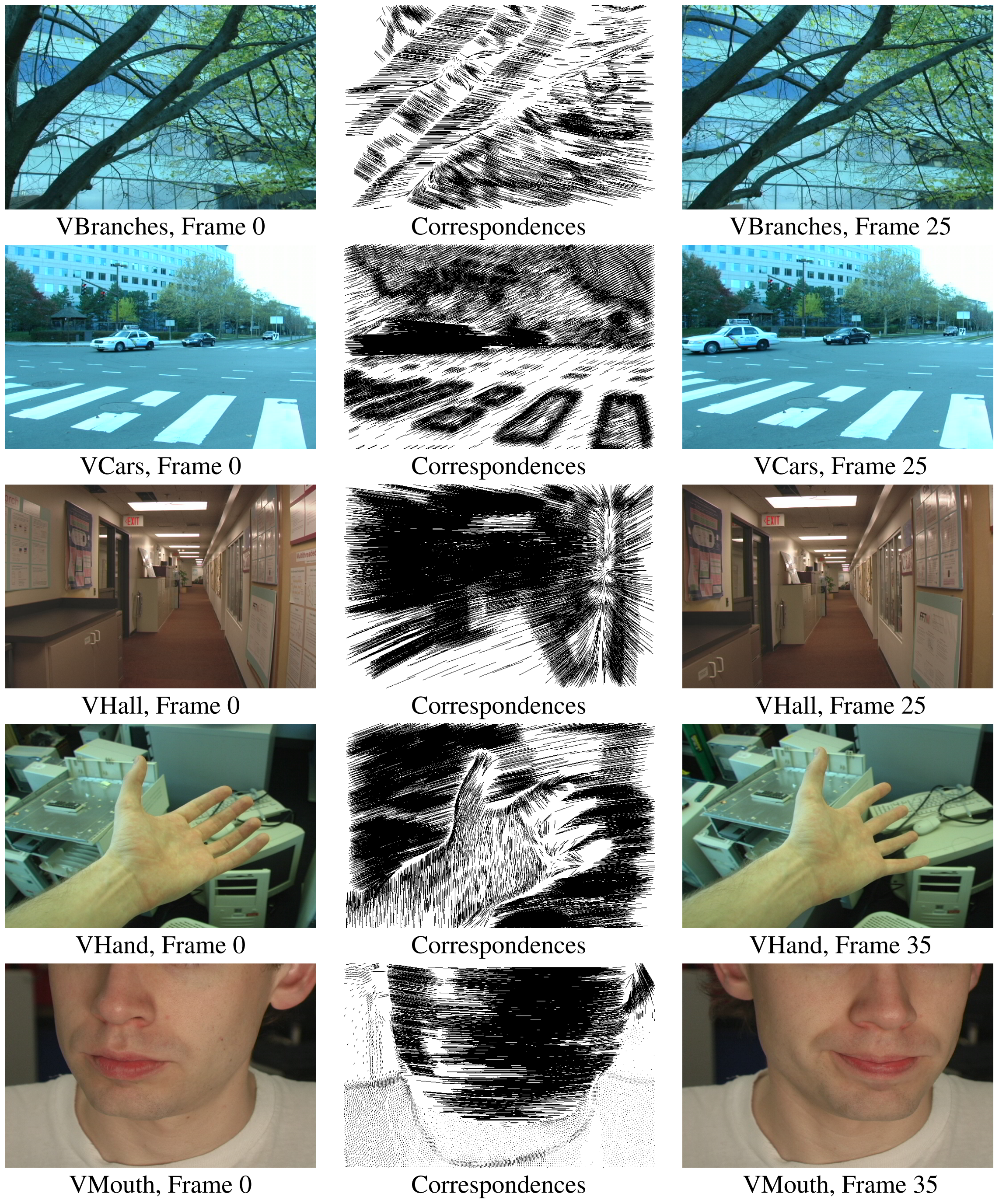


Figure 6-4: Each row shows a frame pair from one test video. Correspondences are shown for particles in common between the frames.

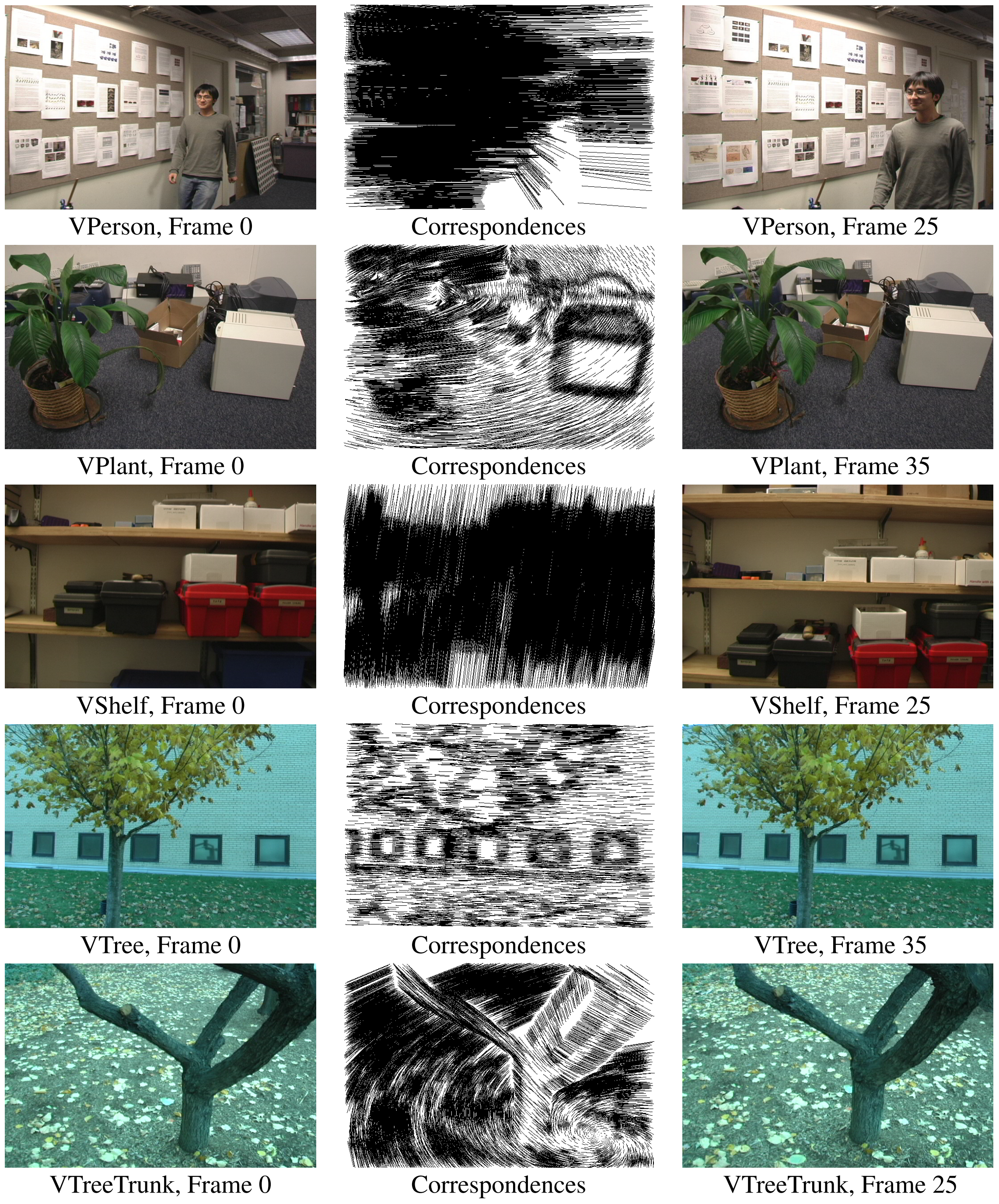


Figure 6-5: Each row shows a frame pair from one test video. Correspondences are shown for particles in common between the frames.

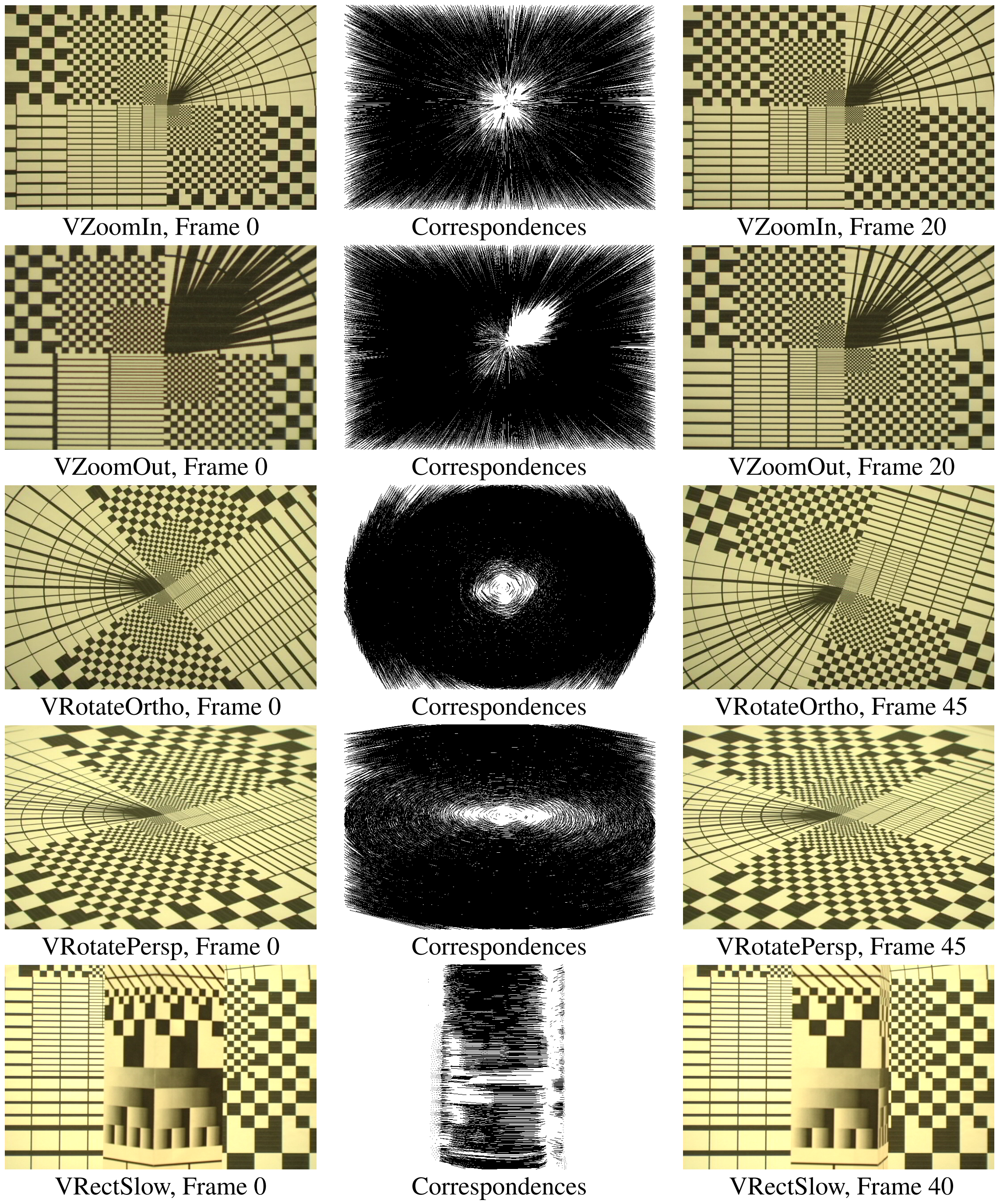


Figure 6-6: Each row shows a frame pair from one test video. Correspondences are shown for particles in common between the frames.

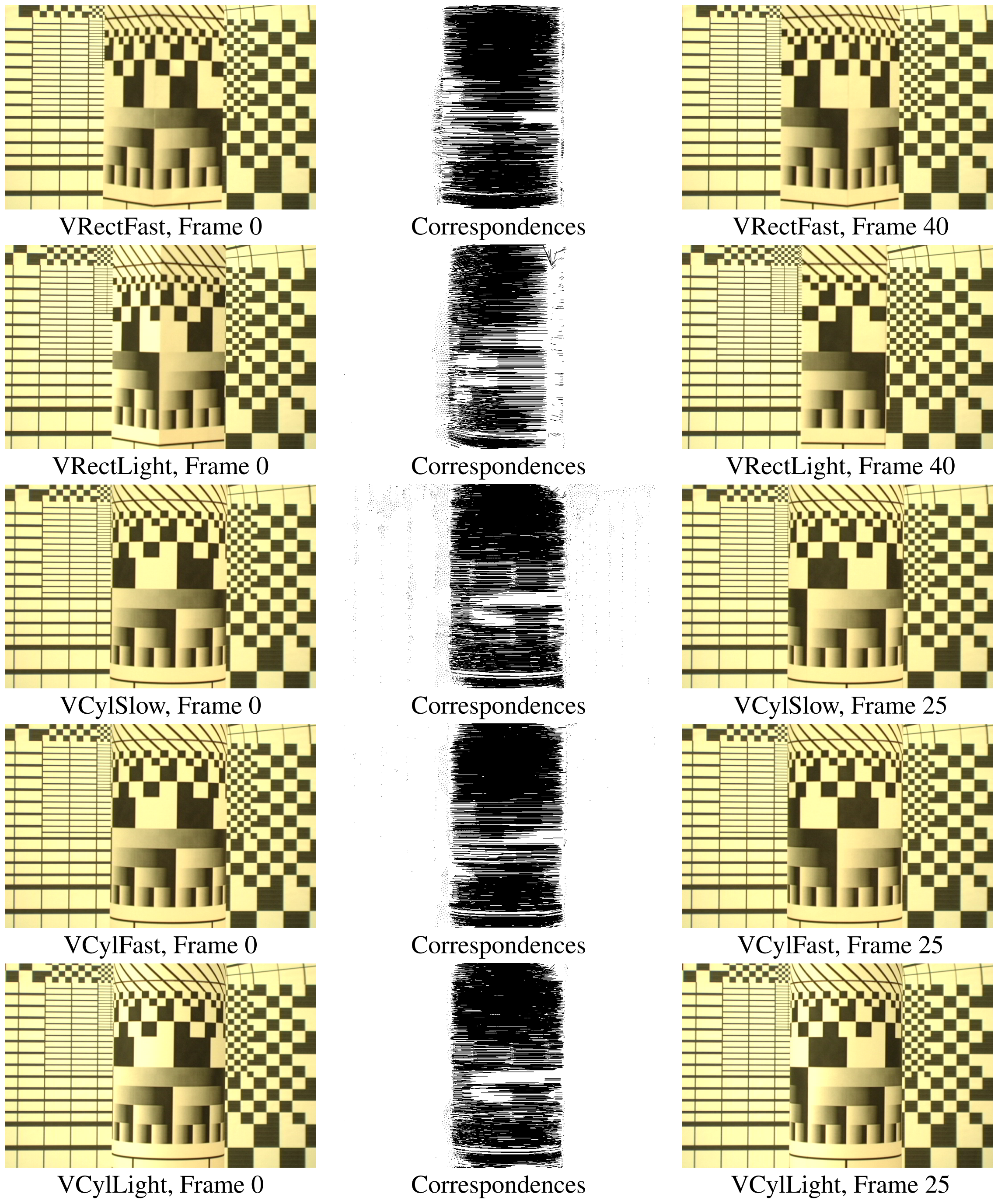


Figure 6-7: Each row shows a frame pair from one test video. Correspondences are shown for particles in common between the frames.

Bibliography

[1] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Trans. Graph.*, 23(3):584–591, 2004.

[2] L. Alvarez, R. Deriche, T. Papadopoulo, and J. Sanchez. Symmetrical dense optical flow estimation with occlusion detection. In *ECCV*, pages 721–735, 2002.

[3] T. Amiaz and N. Kiryati. Dense discontinuous optical flow via contour-based segmentation. In *ICIP*, pages 1264–1267, 2005.

[4] N. E. Apostoloff and A. W. Fitzgibbon. Bayesian video matting using learnt image priors. In *CVPR*, pages 407–414, 2004.

[5] S. Baker and T. Kanade. Limits on super-resolution and how to break them. *PAMI*, 24(9):1167–1183, 2002.

[6] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *IJCV*, 56(3):221–255, 2004.

[7] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, 1994.

[8] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *IJCV*, 12(1):43–77, 1994.

[9] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Computing Surveys*, 27(3):433–467, 1995.

[10] E. P. Bennett and L. McMillan. Video enhancement using per-pixel virtual exposures. *ACM Trans. Graph.*, 24(3):845–852, 2005.

[11] R. Bhotika, D. J. Fleet, and K. N. Kutulakos. A probabilistic theory of occupancy and emptiness. In *ECCV*, pages 112–132, 2002.

[12] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *PAMI*, 20(4):401–406, 1998.

[13] M. Black and P. Anandan. Robust dynamic motion estimation over time. In *CVPR*, pages 296–302, 1991.

[14] M. J. Black. Recursive non-linear estimation of discontinuous flow fields. In *ECCV*, pages 138–144, 1994.

[15] M. J. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[16] M. Brand. Morphable 3D models from video. In *CVPR*, pages 456–463, 2001.

[17] M. Brown and D. G. Lowe. Invariant features from interest point groups. In *British Machine Vision Conference*, pages 656–665, 2002.

[18] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, pages 25–36, 2004.

[19] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnorr. Real-time optic flow computation with variational methods. *Computer Analysis of Images and Patterns*, 2756:222–229, 2003.

[20] T. M. Chin, W. C. Karl, and A. S. Willsky. Probabilistic and sequential computation of optical flow using temporal coherence. *IEEE Trans. Image Processing*, 3(6):773–788, 1994.

[21] Y.-Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski. Video matting of complex scenes. *ACM Trans. Graph.*, 21(3):243–248, 2002.

[22] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *CVPR*, pages 264–271, 2001.

[23] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *PAMI*, 24(5):603–619, 2002.

[24] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *CVPR*, pages 721–728, 2003.

[25] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH*, pages 369–378, 1997.

[26] K. Eismann. *Photoshop Masking & Compositing*. New Riders Press, 2004.

[27] M. Elad and A. Feuer. Recursive optical flow estimation–adaptive filtering approach. *Visual Communication and Image Representation*, 9(2):119–138, 1998.

[28] G. Farnebäck. Fast and accurate motion estimation using orientation tensors and parametric motion models. In *ICPR*, pages 135–139, 2000.

[29] V. Ferrari, T. Tuytelaars, and L. Van Gool. Real-time affine region tracking and coplanar grouping. In *CVPR*, pages 226–233, 2001.

[30] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[31] L. Ford and D. Fulkerson. *Flow in Networks*. Princeton University Press, 1962.

[32] W. T. Freeman and E. Pasztor. Markov networks for super-resolution. In *Proceedings of the Conference on Information Sciences and Systems*, 2000.

[33] A. Fusiello, E. Trucco, T. Tommasini, and V. Roberto. Improving feature tracking with robust statistics. *Pattern Analysis and Applications*, 2(4):312–320, 1999.

[34] M. Han and T. Kanade. Homography-based 3D scene analysis of video sequences. In *Proceedings of the DARPA Image Understanding Workshop*, pages 123–128, 1998.

[35] C. Harris and M. Stephens. A combined corner and edge detector. In *Fourth Alvey Vision Conference*, pages 147–151, 1988.

[36] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[37] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge, UK, 2000.

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

[39] B. K. P. Horn and E. J. Weldon. Direct methods for recovering motion. *IJCV*, 2:51–76, 1988.

[40] M. Irani. Multi-frame optical flow estimation using subspace constraints. In *ICCV*, pages 626–633, 1999.

[41] M. Irani and P. Anandan. A unified approach to moving object detection in 2D and 3D scenes. *PAMI*, 20(6):577–589, 1998.

[42] M. Irani and P. Anandan. All about direct methods. In *International Workshop on Vision Algorithms: Theory and practice*, pages 267–277, 1999.

[43] M. Irani, B. Rousso, and S. Peleg. Computing occluding and transparent motions. *IJCV*, 12(1):5–16, 1994.

[44] A. Jepson and M. Black. Mixture models for optical flow computation. In *CVPR*, pages 760–761, 1993.

[45] S. Kang, R. Szeliski, and J. Chai. Handling occlusions in dense multi-view stereo. In *CVPR*, pages 103–110, 2001.

[46] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High dynamic range video. *ACM Trans. Graph.*, 22(3):319–325, 2003.

[47] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *CVPR*, pages 511–517, 2004.

[48] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV*, pages 82–96, 2002.

[49] K. N. Kutulakos. Approximate N-view stereo. In *ECCV*, pages 67–83, 2000.

[50] S. H. Lee and M. G. Kang. Spatio-temporal video filtering algorithm based on 3-d anisotropic diffusion equation. In *ICIP*, pages 447–450, 1998.

[51] J. J. Leonard, R. J. Rikoski, P. M. Newman, and M. Bosse. Mapping partially observable features from multiple uncertain vantage points. *International Journal of Robotics Research*, 21:943–975, 2002.

[52] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. *ACM Trans. Graph.*, 24(3):595–600, 2005.

[53] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, 2004.

[54] D. Lischinski. *Graphics Gems IV*, chapter Incremental Delaunay Triangulation, pages 47–59. Academic Press, 1994.

[55] C. Liu, A. Torralba, W. T. Freeman, F. Durand, and E. H. Adelson. Motion magnification. *ACM Trans. Graph.*, 24(3):519–526, 2005.

[56] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[57] A. Mansouri, B. Sirivong, and J. Konrad. Multiple motion segmentation with level sets. In *Proc. SPIE Image and Video Communications and Processing, vol. 3974*, pages 584–595, 2000.

[58] D. D. Morris, K. Kanatani, and T. Kanade. Uncertainty modeling for optimal structure from motion. In *Workshop on Vision Algorithms*, pages 200–217, 1999.

[59] N. Nguyen, P. Milanfar, and G. Golub. A computationally efficient superresolution image reconstruction algorithm. *IEEE Trans. on Image Processing*, 10(4):573–583, 2001.

[60] A. Noble. *Descriptions of Image Surfaces*. PhD thesis, Oxford University, Oxford, UK, 1989.

[61] J. Oliensis. Direct multi-frame structure from motion for hand-held cameras. In *ICPR*, pages 1889–1895, 2000.

[62] M. Paolini. *Apple Pro Training Series: Shake 4 (2nd Edition)*. Peachpit Press, 2005.

[63] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert. Highly accurate optic flow computation with theoretically justified warping. *IJCV*, 2006 (to appear).

[64] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.

[65] M. Pollefeys and L. Van Gool. From images to 3D models. *Communications of the ACM*, 45(7):50–55, 2002.

[66] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004.

[67] S. Roy and I. J. Cox. Motion without structure. In *ICPR*, pages 728–734, 1996.

[68] H. S. Sawhney, Y. Guo, K. Hanna, R. Kumar, S. Adkins, and S. Zhou. Hybrid stereo camera: an IBR approach for synthesis of very high resolution stereoscopic image sequences. In *SIGGRAPH*, pages 451–460, 2001.

[69] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1-3):7–42, 2002.

[70] J. Shi and J. Malik. Motion segmentation and tracking using normalized cuts. In *ICCV*, pages 1154–1160, 1998.

[71] J. Shi and C. Tomasi. Good features to track. In *CVPR*, pages 593–600, 1994.

[72] C. Silva and J. Santos-Victor. Robust egomotion estimation from the normal flow using search subspaces. *PAMI*, 19(9):1026–1034, 1997.

[73] C. Silva and J. Santos-Victor. Motion from occlusions. *Robotics and Autonomous Systems*, 35(3–4):153–162, 2001.

[74] A. R. Smith and J. F. Blinn. Blue screen matting. In *SIGGRAPH*, pages 259–268, 1996.

[75] C. Strecha, R. Fransens, and L. V. Gool. A probabilistic approach to large displacement optical flow and occlusion detection. In *Statistical Methods in Video Processing*, pages 71–82, 2004.

[76] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum. Poisson matting. *ACM Trans. Graph.*, 23(3):315–321, 2004.

[77] J. Sun, L. Yuan, J. Jia, and H.-Y. Shum. Image completion with structure propagation. *ACM Trans. Graph.*, 24(3):861–868, 2005.

[78] R. Szeliski and P. Golland. Stereo matching with transparency and matting. *IJCV*, 32(1):45–61, 1999.

[79] R. Szeliski and D. Scharstein. Symmetric sub-pixel stereo matching. In *ECCV*, pages 525–540, 2002.

[80] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2322, 2000.

[81] W. Thompson. Exploiting discontinuities in optical flow. *IJCV*, 30(3):163–174, 1998.

[82] S. Thrun. Robotics mapping: A survey. Technical Report CMU-CS-02-111, Carnegie Mellon University, 2002.

[83] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, pages 836–846, 1998.

[84] R. Vidal and Y. Ma. A unified algebraic approach to 2D and 3D motion segmentation. In *ECCV*, pages 1–15, 2004.

[85] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. *ACM Trans. Graph.*, 24(3):585–594, 2005.

[86] J. Wang and M. F. Cohen. An iterative optimization approach for unified image segmentation and matting. In *ICCV*, pages 936–943, 2005.

[87] J. Wang, B. Thiesson, Y. Xu, and M. Cohen. Image and video segmentation by anisotropic kernel mean shift. In *ECCV*, pages 238–249, 2004.

[88] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. Video tooning. *ACM Trans. Graph.*, 23(3):574–583, 2004.

[89] J. Weickert, A. Bruhn, N. Papenberg, and T. Brox. Variational optic flow computation: From continuous models to algorithms. In *International Workshop on Computer Vision and Image Analysis*, pages 1–6, 2004.

[90] J. Weickert and C. Schnörr. A theoretical framework for convex regularizers in pde-based computation of image motion. *IJCV*, 3(45):245–264, 2001.

[91] T. Werner and A. Zisserman. New techniques for automated architectural reconstruction from photographs. In *ECCV*, pages 541–555, 2002.

[92] Y. Wexler, E. Shechtman, and M. Irani. Space-time video completion. In *CVPR*, pages 120–127, 2004.

[93] J. Wills and S. Belongie. A feature-based approach for determining dense long range correspondences. In *ECCV*, pages 170–182, 2004.

[94] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi. Bilateral filtering-based optical flow estimation with occlusion detection. In *ECCV*, 2006 (to appear).

[95] W. Y. Zhao and H. S. Sawhney. Is super-resolution with optical flow feasible? In *ECCV*, pages 599–613, 2002.

[96] C. L. Zitnick, N. Jojic, and S. B. Kang. Consistent segmentation for optical flow estimation. In *ICCV*, pages 1308–1315, 2005.

[97] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. A. J. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23(3):600–608, 2004.