Creating Realistic Animations Using Video

Kiran S. Bhat

CMU-RI-TR-04-54

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Robotics.

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

December 10, 2004

Abstract

Generating realistic animations of passive dynamic systems such as rigid bodies, cloth and fluids is an important problem in computer graphics. Although several techniques for animating these phenomena have been developed, achieving the level of realism exhibited by objects in the real world has proved to be incredibly hard. We argue, therefore, that an effective method for increasing the realism of these types of computer animations is to infer the behavior of real phenomena from video. This thesis presents two techniques for using video to create realistic animations of phenomena such as rigid bodies, cloth, waterfalls, streams, smoke and fire.

The first technique, inverse simulation, estimates the parameters of physical simulations from video. Physical simulation techniques are widely used in computer graphics to generate animations of passive phenomenon using physical laws and numerical integration techniques. The behavior of a physical simulator is governed by a set of parameters, typically specified by the animator. However, directly tuning the physical parameters of complex simulations like rigid bodies or cloth to achieve a desired motion is often cumbersome and nonintuitive. The inverse simulation framework uses optimization to automatically estimate the simulation parameters from video sequences obtained from simple *calibration* experiments (e.g., throwing a rigid body, waving a swatch of fabric). This framework has three key components: (1) developing a *physical model* that accurately captures the dynamics of the phenomena, (2) developing a *metric* that compares the simulated motion with video and (3) applying *optimization* to find simulation parameters that minimize the chosen metric. To demonstrate the power of this approach, we apply this framework to find the parameters for tumbling rigid bodies and for four different fabrics.

The second technique presents a video editing framework for creating photorealistic animations of natural phenomena such as waterfalls by *directly* editing reference video footage. Our algorithm analyzes the dynamics and appearance of textured particles in the input video along user-specified *flow lines*, and synthesizes seamless infinite video by continuously generating particles along the flow lines. The user can then edit the video by manipulating flow lines from the original footage. The algorithm is simple to implement and use. We applied this technique to perform significant edits to the video. For example, we were able to change the terrain of a waterfall, add obstacles along the flow and add wind to smoke and flames, to demonstrate the editing capability of our approach.

The results from these two techniques demonstrate the effectiveness of using video to improve the realism of computer animations. Our experience with applying inverse simulation for cloth resulted in improvements to existing cloth models and produced animations that matched the realism of real fabrics. Our research on flow-based modelling produced simple one dimensional particle models that enable animating a wide variety of natural phenomena. We hope that the methods developed in this thesis provide useful insights into how realistic animations for other domains might be designed.

Acknowledgments

I would especially like to thank my advisors, Professors Steve Seitz, Pradeep Khosla and Jessica Hodgins for their inspiration, support and guidance throughout my career in graduate school. This thesis would not have been possible without their guidance and resources, and I am obliged for the opportunity to learn so much from them. Pradeep was an exceptional source of support and leadership, and encouraged me to tackle tough research problems. Steve and Jessica taught me the important skill of doing independent research, shaped my research ideas and values, encouraged me to develop my writing and speaking skills, and trained me to critically evaluate my research. They were instrumental in helping me pick my thesis problem and also helped me at each step of the way.

Many, many others have contributed to my effort. I had an opportunity to work with Dr. Steve Sullivan from Industrial Light + Magic, who gave me valuable insights from the F/X industry. Christopher Twigg, Professor Jovan Popović and Professor Zoran Popović collaborated with me on the first part of my thesis work, and I am thankful for their guidance and help. Dr. Al Rizzi and David Tolliver have helped me on numerous occasions with insightful comments and feedback. The weekly graphics lab meeting has been a continual source of inspiration for clear thinking and academic camaraderie. My colleagues at the CMU graphics lab, the Robotics Institute and University of Washington have helped me in innumerable ways, technical and social, and made my graduate school experience very enjoyable.

I am eternally indebted to my parents, Srinivas, Jayasree and my sister, Preethi for their endless love and support. My parents nurtured my interest in science, and inspired me to pursue a PhD. I am also deeply grateful to Payal for her companionship and encouragement, and for making the last year of my graduate study truly enjoyable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Creating realistic animations has been a great challenge in computer graphics for the last two decades. Despite many advances in modelling, animation and rendering techniques, generating realistic animated motion has proved to be incredibly hard. In order to create believable computer animations, we need realistic *secondary* motions of common background elements such as rigid bodies, cloth, leaves, and smoke [OZH00]. Compelling secondary motions add richness to the scene, naturalness to the action and give a fuller dimension to the personality of the character [TJ84]. This thesis focusses on techniques for creating realistic animations of a wide range of secondary motions: tumbling rigid bodies, fabrics of different materials, waterfalls, fire and smoke blowing in the wind.

We argue that a direct way of creating realistic animations of such phenomena is to design algorithms that infer the behavior of real phenomena (e.g., cloth, fluids) captured in video. In its most basic form, the idea of using video as reference is known as rotoscoping [Fle17], where an animator traces over film or video frames captured as reference. But beyond reproducing the original motion, our objective is to generate new animations that capture the characteristics of the original phenomenon, but can be controlled to meet an artist's or director's goals. For example, we would like to create animations of a dancer wearing a fleece skirt from a video clip of a fleece swatch (figure 1.1(a)). In a different setting, shown in figure 1.1(b), we would like to modify the terrain of a real waterfall in video to create a new waterfall with two extra channels. This thesis presents two methods that use video to create realistic animations for these two different scenarios.

Physically based simulation techniques have been widely used for automatically generating the secondary motions of passive phenomena such as rigid bodies and cloth. One technique for creating realistic animations for the scenario in figure 1.1(a) is to tune the parameters of a physical simulation to match real video footage. We call this technique *inverse simulation*. An alternative technique is to directly model

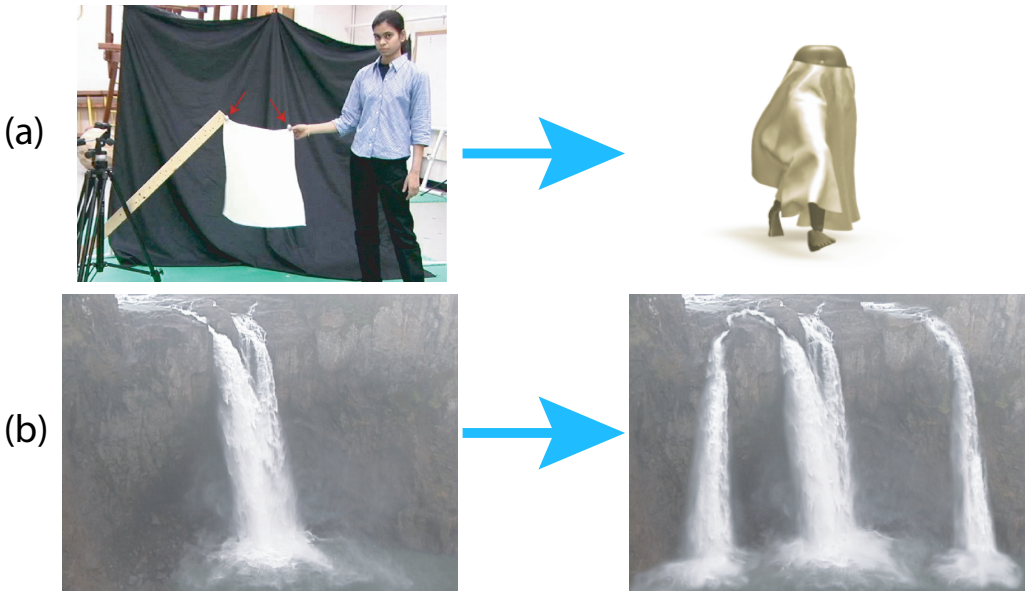


Figure 1.1: Capturing the behavior of real phenomena in video for creating new animations. The top row shows a realistic skirt simulation (on the right) created from a video of an artist waving a swatch of real fabric. The bottom row shows an edited waterfall (on the right) created from a short clip of a real waterfall on the left.

the motion of pixels in video, and use this model to synthesize new video. This technique is ideally suited for examples of natural phenomena such as the waterfall in figure 1.1(b), where the correct dynamics are difficult to simulate, but it is easy for a person to sketch out rough flow lines on the image and manipulate these flow lines to produce new video. We call this technique *flow-based video editing*.

1.1 Inverse Simulation

This thesis is organized into two parts corresponding to these two techniques: inverse simulation and flow-based video editing. The first part develops a general framework for estimating parameters of physical simulations from video clips of real phenomena. With the right set of parameters, good simulators produce very realistic looking motion. However, manually choosing the parameters that will provide a particular appearance is still a time consuming task because the animator has to re-run the simulation and view it for each set of parameters. Even harder is the task of exactly matching a simulation to a video sequence of the real world. For example, some parameters in a cloth simulation can be chosen based on the animator's intuition about

the fabric—a knit fabric is more stretchy than a woven fabric such as linen, for example. But not all the parameters of a cloth simulator are intuitive or map directly to measurements that can made by a system such as the Kawabata system [Kaw80]. For example, guessing the dynamic parameters such as damping or air drag is very difficult because the simulated motion is affected by the choice of numerical integration scheme. On the other hand, it is easy to film a video sequence of different types of fabrics in motion. We present an inverse simulation framework for automatically estimating the parameters of physical simulations from video. We used this framework to identify the parameters of rigid body and cloth simulations.

Parameter identification from video is a challenging problem because video does not directly provide three dimensional (3D) shape or motion information of real phenomena. While recent advances in computer vision enable estimating geometry rapidly and efficiently [TK92, MT93, ZCS02, HK03], existing methods do not allow the computation of physical parameters such as stiffness or damping in this manner. Our research shows that a set of perceptual features extracted from video has sufficient information to estimate the parameters of physical simulations.

The inverse simulation framework, based on optimization, has three key steps (figure 1.2): (1) developing a *physical model* for the system dynamics that captures the complexity of the motion, (2) developing a *metric* to compare the simulated motion with video and (3) searching the parameter space using *optimization* to find the best set of parameters. We validate the estimated parameters by applying them to new dynamic conditions in simulation and comparing the simulated results to video. For example, we use our parameter estimates—obtained by waving a small swatch of fabric in front of a camera—to create an animation of a dancer wearing a skirt made of the same material.

1.2 Flow-Based Video Editing

The second part of the thesis focusses on techniques for animating fluid-like phenomena such as waterfalls, streams, smoke and fire from reference video clips. Given a video sequence of a real phenomena like a waterfall (or smoke or fire), our objective is to capture the motion and appearance of the waterfall and create new video of waterfalls that we can control. We present a flow-based video editing framework for creating photorealistic animations by *directly* editing the reference video footage. For example, a video sequence of Niagara falls can be edited to create a new waterfall that is twice as big as Niagara.

Editing reference footage to create photorealistic animation is quite challenging. Although several good techniques exist for editing images [HJO+01, KSE+03, EF01], none of them extend well to video because these techniques, applied frame-

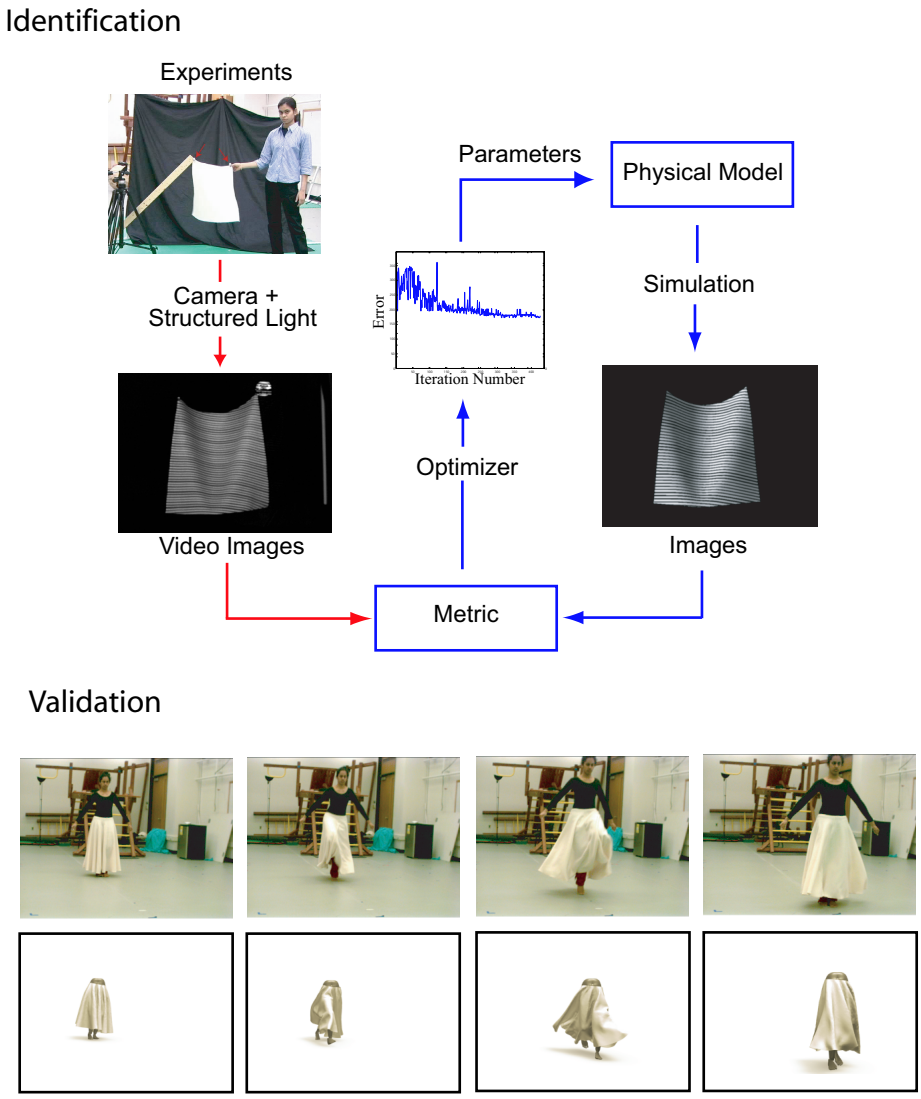


Figure 1.2: The steps of the inverse simulation framework applied to cloth. The top figure shows a block diagram for parameter identification using optimization. Images from real experiments (waving a swatch of fabric) are compared to images of a simulation using a metric that penalizes the mismatch in their folds. The parameters of the simulation are updated by an optimization routine (simulated annealing) until the metric error is minimized. The bottom figure shows the validation results for skirt simulations. The skirt simulations use the optimized parameters obtained from the fabric swatch. The corresponding frames from a real skirt made of the same material, driven by approximately the same human motion are shown for comparison.

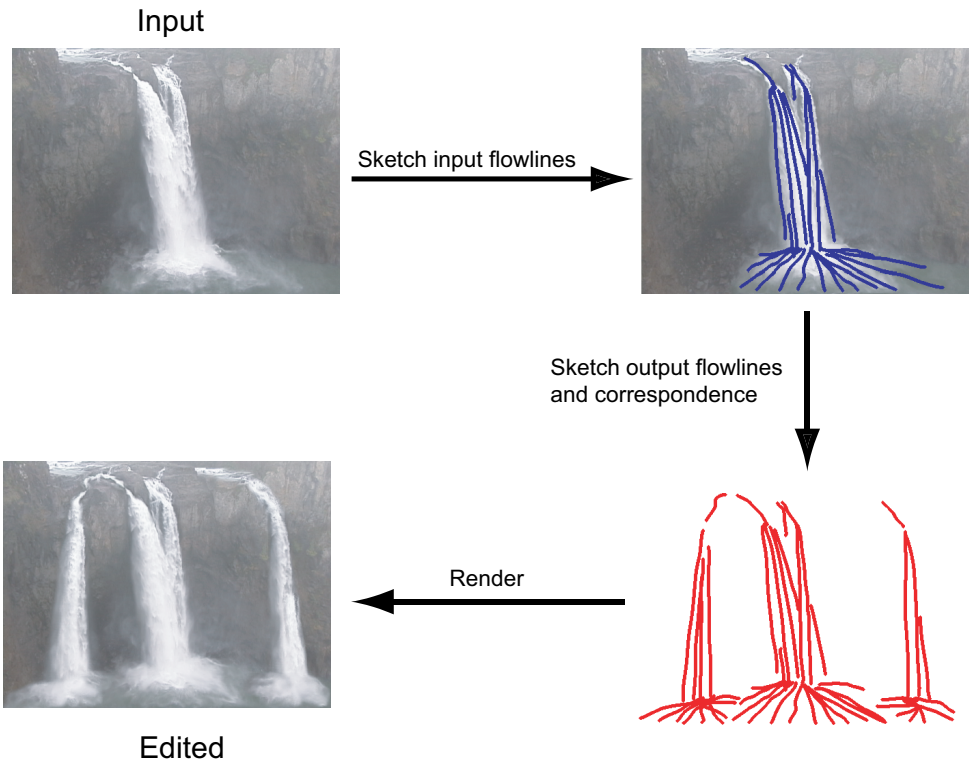


Figure 1.3: The steps of our video editing framework for creating realistic animations. First, the animator sketches a set of blue flow lines on the input video. These flow lines capture the animator's intuition about the flow of particles in the video. To create a new waterfall with two extra channels, the animator sketches the output flow lines shown in red, and specifies the correspondence between the output and input flow lines. Our algorithm automatically generates an infinitely long edited video, one frame of which is shown.

by-frame, do not guarantee temporal consistency (pixel colors in consecutive frames may vary significantly). Most prior work in video synthesis [DCSW03, KSE+03, SSSE00] has focussed on creating infinite sequences from video, with very little effort on editing video, barring a few exceptions [DS03]. Our editing framework leverages the animator's intuition about the flow of particles in video to model the input sequence and specify edits.

The two main steps of the video editing framework are (1) modelling the input video as a particle system along input flow lines and (2) synthesizing the edited sequence by generating particles along the output flow lines (figure 1.3). In our implementation, the input and output flow lines are specified by an animator. This technique works on complex natural phenomena such as waterfalls, streams, fire and

smoke that exhibit a continuous flow of particles along constant flow directions. We demonstrate the power of this approach with a wide range of edits: changing the terrain of a waterfall, adding wind to sequences of smoke and fire and adding obstacles to a stream.

The two video-based techniques presented in this thesis are appropriate for different classes of problems. The inverse simulation framework is ideally suited for phenomena such as rigid bodies and cloth whose dynamics are accurately modelled using a small set of parameters. Moreover, it is easy to obtain video footage for these phenomena from controlled experiments. However, extending the inverse simulation framework to more complex phenomena is difficult because we need custom simulators for each type of phenomena (e.g., smoke, water, fire). Moreover, in order to match real world measurements, physical simulators would require precise models for wind and other external forces, significantly increasing the number of parameters to be optimized. For these phenomena, flow-based video editing presents a simple and effective alternative to physical simulation. Our flow-based interface provides an intuitive interface for creating controllable animations of natural phenomena. However, the information learned from one sequence using the flow-based approach may not generalize to other sequences or new scenario.

1.3 Applications

Creating realistic animations of common passive objects using video footage has direct applications in entertainment, science, technology and medicine. Here, we discuss a few of the potential long term applications:

Visual Effects and Computer Games: Physically based simulation techniques are very popular techniques for animating secondary motion in movies and games. For example, the motion of Yoda's costume in *Star Wars: Episode II* was created using physical simulations. Automatic techniques for parameter identification from real footage should increase the realism of physical simulations, and will reduce the manual time spent on tuning their parameters. Our flow-based framework could be potentially used to increase the realism of *scale models* of natural phenomena such as waterfalls in studios, for example, by adding mist from real waterfalls.

Medicine: Mathematical models of human tissue have been developed to simulate organs and analyze the conditions for various disorders. Applying inverse simulation techniques to obtain accurate simulation parameters should help doctors to locate the plausible stresses in the tissues they are surveying. For example, Eysholdt et al. [ERH03] use endoscopy video of the vocal chords to analyze speaking disorders

in humans by estimating the force production in the larynx. Moreover, realistic rendering of human tissue using data from real video footage could significantly increase the usefulness of simulations for surgical training.

Web commerce: Accurate physical models of real objects will be beneficial to web-based commerce and interaction. One potential application is online shopping stores where customers can try out fabrics of different material on virtual avatars. The material parameters for the different fabrics could be obtained a priori using our inverse simulation framework.

Home video editing: Techniques for creating compelling animations by video editing could allow amateur artists to prototype their ideas by modifying existing video footage. Such software would also have potential for home video editing, allowing users to create new video by selectively cutting and pasting portions from different clips.

This thesis is organized as follows: chapter 2 presents the inverse simulation framework for capturing simulation parameters of rigid bodies and cloth. Chapter 3 discusses the video editing framework for creating photorealistic animations from short video clips. Finally, in chapter 4, we conclude with a discussion on the contributions and limitations of the two techniques, and directions for future research.

Chapter 2

Inverse Simulation Techniques

Physically based simulation techniques are often used for generating realistic animations of complex physical behavior such as colliding rigid bodies [Bar94, MC95], folding cloth [BW98, BFA02, BWK03, CK02], rising smoke [FSJ01], burning fire [LF02] and splashing water [Sta99, FF01]. An attractive feature of physical simulation is that the animation is generated automatically—the animator only needs to specify a few physical parameters. This feature has made simulations a very popular tool for animating secondary motions in computer games and movies.

The primary drawback of physical simulations, however, is the lack of intuitive control over the resulting animation. For example, an animator might have to simulate a skirt that moves and responds like satin. But, there is no easy way to guess the static and dynamic parameters of a cloth simulator that might achieve this effect. Adjusting the simulation parameters manually to achieve a desired outcome is often tedious and nonintuitive. An automatic solution to the parameter estimation problem is one of the main challenges in computer graphics [TPB+89]. The objective of the first part of this thesis is to automatically infer the simulation parameters from video.

Figure 2.1(a) shows a block diagram representation of a physical simulation. The term *forward simulation* refers to the process of numerically integrating the mathematical model for a given set of parameters. The inverse of this step, where the parameters of the simulation are estimated from state measurements, is typically referred to as *inverse simulation* (figure 2.1(b)). This thesis presents an inverse simulation framework based on optimization that uses video measurements. We have applied this technique to estimate the model parameters of rigid body and cloth simulators. The video used in our framework is obtained from laboratory experiments (using tumbling rigid bodies or swatches of fabrics) that are designed to exercise the different behaviors of the physical system. Our framework uses a temporal sequence of image features (silhouette and shape) derived from video for parameter identification. We validate the estimated parameters to ensure that they generalize to new

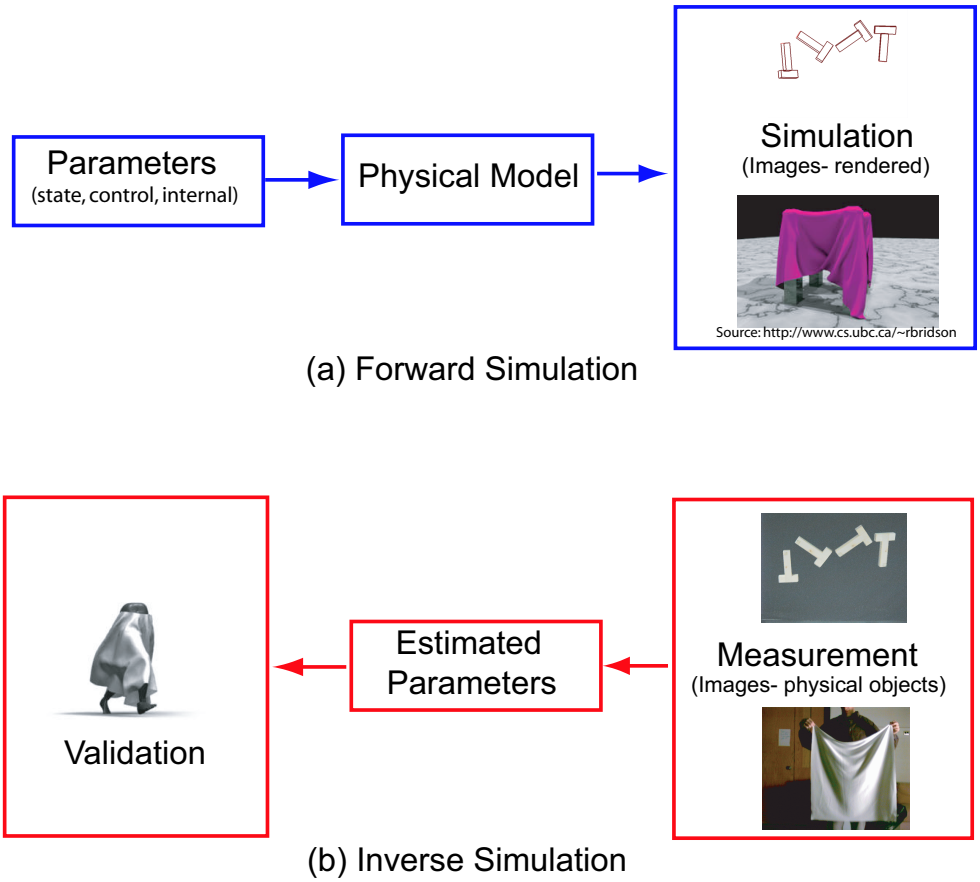


Figure 2.1: Forward and inverse simulation techniques.

simulation settings.

The parameter identification algorithm optimizes the simulation parameters such that the rendered simulation matches the video captured from the controlled experiments. We use perceptually motivated metrics to evaluate the match between video sequences from simulation and real experiments. The design of these metrics from video is an important contribution of our thesis. Metrics evaluate the visual realism of simulations by scoring them based on their match with real video. Equally important is the design of the (real) experiments, which provides the reference video. The experiments are designed such that they are simple, repeatable and easy to set up (in reality and in simulation), yet capture the complexity of the dynamics. For example, we waved swatches of fabrics of different material in front of a camera to capture the simulation parameters of different fabrics.

The next section presents a brief overview of the inverse simulation framework

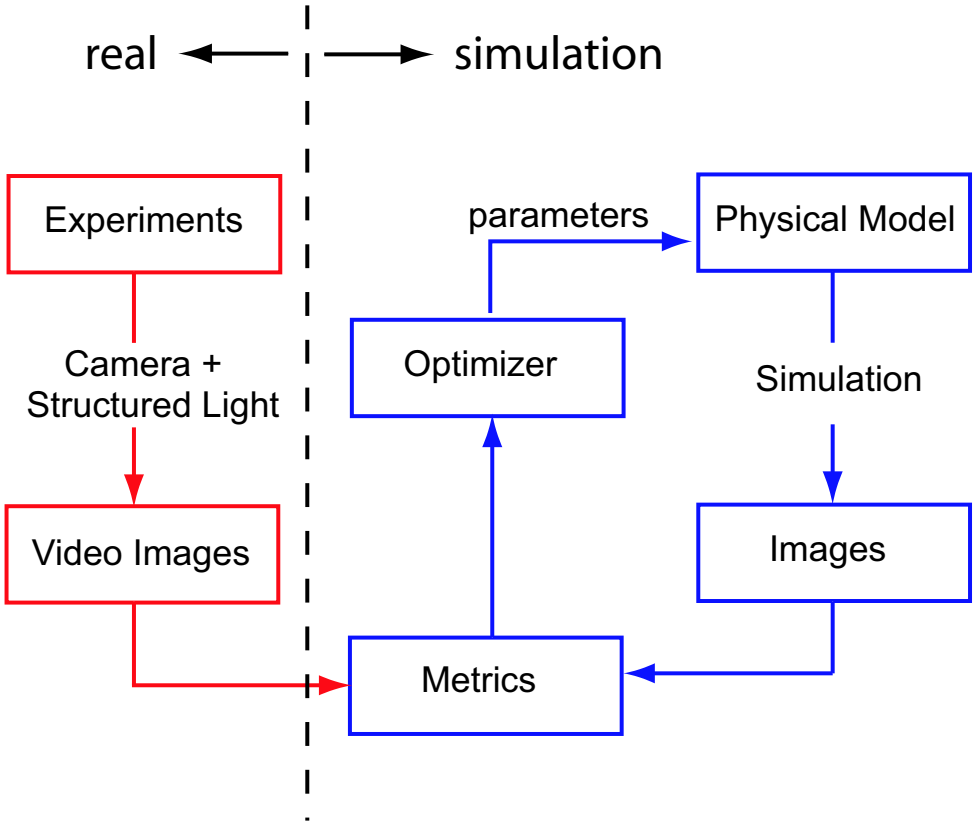


Figure 2.2: Block diagram of parameter identification using the inverse simulation framework.

for capturing parameters using optimization. We introduce the various components of the framework, focussing mainly on the specifics for rigid bodies and cloth.

2.1 Overview

Optimization is a common technique for estimating the parameters of non-linear dynamic systems. We pose the parameter estimation as an unconstrained minimization problem, with an objective function that minimizes a metric computed on rendered and video images. We used a gradient-based optimization routine for estimating rigid body parameters, and simulated annealing for estimating cloth parameters. Figure 2.2 shows the process diagram of our parameter identification framework, where the blocks on the left are obtained from real measurements and the blocks on the right are computed in simulation. For a given set of parameters, the metric compares the image sequences rendered from simulation against images captured from real ex-

periments. The optimizer continuously tunes the model parameters until the metric error is minimized. We now briefly discuss a few key elements of figure 2.2.

Physical Model: We applied our algorithm on two types of simulators—rigid bodies and cloth—that are widely used in computer graphics. Our first experiment was to estimate free-flight parameters of a tumbling rigid body from video. The tumbling dynamics of a rigid body is governed by the Newton-Euler equations of motion [Sym71]. Because gravity is the only external force acting on the body, the body rotational dynamics is completely governed by its inertia and initial conditions (position, orientation, linear and angular velocities). The goal of our second experiment was to estimate the static and dynamic parameters of the Baraff and Witkin [BW98] cloth simulator for different fabrics. We improved this model to handle object and self-collisions, added a non-linear air drag term and used a explicit integration scheme to improve the realism of the generated cloth motion.

Metric: One of the key elements of our framework is the design of metrics to compare video rendered from physical simulations with video from real experiments. Mathematically, the metric M takes two video sequences $\mathbf{A}$ and $\mathbf{B}$ and returns a number c that captures the difference between the two sequences.

$$M(\mathbf{A}, \mathbf{B}) = \mathbf{c} \tag{2.1}$$

Our goal is to design the function M such that the difference c measures the perceptual dissimilarity between the two sequences. For rigid bodies, our metric computes the frame by frame difference in silhouettes between the simulation and video. Similarly, for cloth, our metric measures the frame by frame difference between the folds in simulation and video. Both these metrics can be computed robustly and very quickly from video using standard computer vision algorithms, which make them suitable for inner loop computations in our optimization framework.

Optimizer: Our choice of optimization technique for a given problem was guided by three factors: noise of the error space, availability (or not) of analytical gradients and ease of initialization. We used analytical gradients to optimize the simulation parameters of a tumbling rigid body using a gradient descent algorithm. However, for cloth, the error space was extremely noisy, and it was very hard to initialize the optimizer with a good starting value for the parameters. So we chose a global optimization scheme, simulated annealing, which explores the space before settling for a particular minima. Although simulated annealing is several orders of magnitude slower than gradient descent, it performs well on noisy error spaces where the gradient information may not be reliable for optimization.

The rest of this chapter is organized as follows: first, we present related work in parameter identification techniques for dynamic systems. The next section presents a detailed discussion of estimating the parameters of rigid body and cloth simulations. For each simulation type, we present the related work, and details of the mathematical model, identification experiments, perceptual metrics and the optimization routine. Then, we validate the estimated parameters by creating new animations and evaluating their visual realism. Finally, we conclude with a summary that highlights the challenges of achieving realism in simulation, and justifies the various engineering decisions that we made along the way. We also present a couple of examples of metrics for cloth that did not work, and analyze the reason for their failure.

2.2 Related Work

This section summarizes the prior work in parameter identification and optimal control applied to physics based animation. The later sections on rigid bodies and cloth will present a detailed survey of the related simulation and video based techniques specific to these two systems.

In recent years, the computer graphics community has made great progress in modelling the complex secondary motions of passive phenomena through physical simulations [Bar94, MC95, BW98, BFA02, BWK03, CK02, Sta99, FSJ01, LF02, FF01]. These simulation techniques can generate very realistic results, but the animations that they produce are often difficult to control. The problem of controlling physical simulations has received considerable interest lately. Popović and his colleagues [PSE03] presented a sketch-based interface for controlling rigid body motions. They pose the control problem in a parameter estimation framework, and identify the initial conditions and other physical parameters for a desired sketch. An alternative approach for controlling passive systems like smoke [TMPS03, FL04] solves for custom wind fields that are continuously applied to guide the simulation towards desired key frames. Our inverse simulation framework draws from the work on interactive control and parameter estimation for rigid bodies [PSE+00, PSE03].

Constrained optimization techniques from optimal control theory [Ste94] have been used for inverse simulation of active dynamic systems. These techniques, called spacetime constraints in the graphics community [WK88], compute the motion of *active* dynamic systems by continuously applying actuating forces/torques $\mathbf{f}(\mathbf{t})$ at the active joints. The motion of *passive* objects like rigid bodies and cloth are fully determined by a finite number of simulation parameters $\mathbf{p}$ and therefore present a simpler problem. Our framework uses video from real world measurements to estimate the simulation parameters $\mathbf{p}$.

Parameter identification of linear dynamic systems has been used successfully

to a wide range of problems in engineering, chemistry, biology and other sciences [Kai79]. However, the equations of motion of most passive dynamic systems are typically described by non-linear differential equations. Recently, several techniques for non-linear system identification have also been developed (see [Nel00] for an excellent survey). Of these techniques, parameter identification using optimization is very popular and fairly easy to use, with excellent toolboxes and resources [Gra92, GMW82]. Optimization-based algorithms typically consider the simulation as a black box and therefore generalize well for different physical systems. Moreover, the objective function can be designed to minimize image measurements, which make them better suited for our framework in comparison to standard estimation techniques (e.g., non-linear least squares) that require state measurements.

2.3 Capturing Rigid Body Parameters

Our first experiment was to estimate the physical parameters of a rigid body in free flight (called *tumbling* motion) from a video sequence of its motion. Rigid bodies are ideal candidates for testing our inverse simulation framework because their complex tumbling motion is constrained by a small set of physical parameters and initial conditions [Sym71]. The parameters governing the tumbling motion of a rigid body in video can be categorized into two groups:

- Object Parameters: *intrinsic* (object shape, mass distribution, location of center of mass), *extrinsic* (initial position, orientation, velocity, angular velocity).
- Environment Parameters: *gravity direction, air drag*.

The objective of our first experiment is to extract the object parameters and the gravity direction from a video sequence of a rigid body in free flight. For the sake of simplicity, we assume that the object shape and inertial matrix are known and the effect of air drag is negligible.

2.3.1 Prior Work

Techniques for simulating rigid body motion have matured tremendously in the last few years. Modern rigid body simulators [GBF03, MC95, Bar94] can robustly handle complex interactions such as collisions, contact, friction and stacking involving thousands of rigid bodies. We are interested in estimating the parameters of rigid body simulations from video.

Prior work in computer vision has mainly focussed on the problem of extracting the shape of rigid bodies and tracking them in video [TK92, HZ00, FLP01, TWK88].

There is surprisingly little work on extracting the underlying physical properties of the object or the environment. Furthermore, prior video-based techniques for parameter estimation make simplifying assumptions about the shape and properties of the rigid bodies. For example, several researchers [RK96, BS99] make point mass assumptions to design robot controllers for juggling or playing air hockey using video measurements. Ghosh et al. [GL95] estimate the shape and dynamics of planar objects from optical flow measurements. The problem of simultaneously recovering the physical parameters of a 3D rigid object and its environment from a single camera has not been previously addressed.

More closely related to our identification framework is prior work on physics based reconstruction and tracking in computer vision [TWK88, Har92, CC91, SD91, Bre97, DF99, Wre00, PH91, MT93]. Most of these tracking techniques (in computer vision) define a dynamic model that is controlled via forces and torques. Their objective is to compute the set of forces and torques that give rise to the shape or trajectory that best fit the image data. An important limiting feature of most model based tracking techniques is that real world dynamics, including important effects due to gravity and impact, are not fully modelled. Consequently, the forces and torques computed through physically based approaches in computer vision are physically inspired but do not capture real world physics. In many cases, the dynamic model that relates the current and previous states is approximated using a linear Kalman filter [Har92, CC91], which works well for applications involving rigid body tracking [Har92] or navigation [CC91, SD91].

Our dynamic model captures the true rotational physics of a tumbling rigid body. This aspect distinguishes our work from prior work in motion tracking and analysis where the focus is on identifying object kinematics, i.e., motion trajectories. Moreover, our algorithm uses information from all frames in the video sequence simultaneously, unlike feed-forward filter based methods. Figure 2.3 compares the results of our offline batch algorithm with an online Kalman filter applied to a synthetic example of 2D ballistic motion with gaussian noise. Our algorithm tries to find the best fit parabola to the data, whereas a Kalman filter fits a higher order curve to the data. Although the Kalman filter approach *tracks* the data better, our algorithm finds the parameters describing the physics of the ballistic motion. These parameters can now be used to *animate* a particle in simulation, that moves like the given data.

2.3.2 Model

The tumbling motion of a rigid body in free flight is characterized by its position, orientation, linear and angular velocity. Let $\mathbf{q}(t)= [\vec{X}(t),\vec{\theta}(t),\vec{V}(t),\vec{\omega}(t)]$ be the state of the rigid body at time t. $\vec{X}(t) \in \mathbf{R}^3$ and $\vec{\theta}(t) \in SO(3)$ specify the position and orientation in a world coordinate system. We use a quaternion to represent the orien-

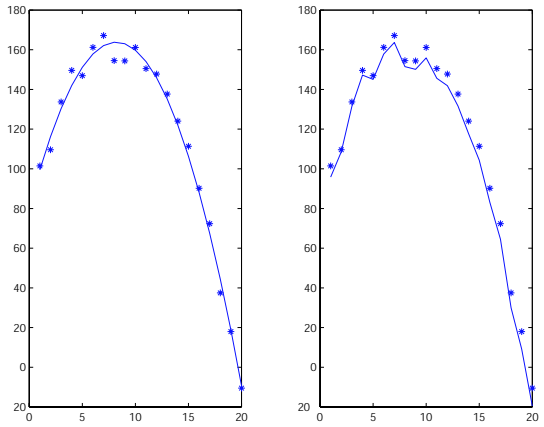


Figure 2.3: Illustrating the differences between parameter estimation using optimization (left) and a Kalman filter (right) on a simplistic 2D example. The input data is the trajectory of a point mass in free flight, corrupted with gaussian noise. Our optimization algorithm uses all the data points simultaneously to fit a path that globally satisfies physical laws; in this simple case, a parabolic path. The Kalman filter processes the data sequentially and fits a higher order curve to the data.

tation with four parameters. $\vec{V}(t) \in \mathbf{R}^3$ and $\vec{\omega}(t) \in \mathbf{R}^3$ specify the linear and angular velocity coordinates. The time derivative of the state $\mathbf{q}(t)$, called the simulation function $\vec{F}(t,\mathbf{q}(t))$, is governed by the ordinary differential equation (ODE):

$$\vec{F}(t, \mathbf{q(t)}) = \frac{\mathbf{d}}{\mathbf{dt}}(\mathbf{q(t)}) = \frac{\mathbf{d}}{\mathbf{dt}} \begin{pmatrix} \vec{X}(t) \\ \vec{\theta}(t) \\ \vec{V}(t) \\ \vec{\omega}(t) \end{pmatrix} = \begin{bmatrix} \vec{V}(t) \\ \frac{1}{2}(\vec{\theta}(t) * \vec{\omega}(t)) \\ \vec{g} \\ -\vec{I}(t)^{-1} \left(\frac{d\vec{I}(t)}{dt} \vec{\omega}(t) \right) \end{bmatrix} \tag{2.2}$$

Here, $\vec{I}(t)$ is the inertia matrix in world coordinates, and $\vec{g} \approx (0, -9.81m/s^2, 0)$ is the acceleration due to gravity. The product $*$ refers to quaternion multiplication. The state $\mathbf{q(t)}$ at any time instant $t = t_f$ is determined by integrating Equation 2.2:

$$\mathbf{q(t_f)} = \mathbf{q(t_0)} + \int_{\mathbf{t_0}}^{\mathbf{t_f}} \vec{\mathbf{F}}(\mathbf{t}, \mathbf{q(t)})\mathbf{dt} \tag{2.3}$$

The state at any time $\mathbf{q(t)}$ depends on the initial state at time $t = t_0$ and inertial matrix. In this thesis, we assume that the inertia matrix is known. Consequently, it is sufficient to solve for the initial state, which we denote by $\mathbf{p_{obj}}$. Because we are interested in estimating the simulation parameters from video, it is convenient to place the origin of the world coordinate system at the camera center, with the

principal axes aligned with the camera coordinate system. Due to the choice of this coordinate system, the direction of gravity depends on the orientation of the camera, and is not necessarily vertical. The gravity direction in our framework is encoded by two angles (tilt and roll). We use the symbol $\mathbf{p_{env}}$ to indicate the direction of gravity in simulation, and the symbol $\mathbf{p}$ to denote the total set of simulation parameters: $\mathbf{p} = (\mathbf{p_{obj}}, \mathbf{p_{env}})$.

2.3.3 Metrics from video

We use the mismatch between the object silhouettes extracted from the simulation and video as the metric for comparing a rigid body simulation with video. Silhouettes are an appropriate choice for comparing videos of tumbling rigid-bodies. Because the motion of a tumbling rigid body in video is complex with many self-occlusions, traditional feature trackers or optical flow algorithms may produce unreliable feature estimates. On the other hand, silhouettes can be computed quickly and reliably from video.

Silhouette Comparison: We begin by preprocessing the input video sequence to compute the silhouettes at each frame. Similarly, we render a silhouette sequence in simulation using the current parameter values. We present two techniques to compare silhouette images, as shown in figure 2.4. The first metric (left) counts the number of mismatched pixels between the two silhouettes. We used this metric in our implementation. Define A_k as a binary silhouette image of size (S_x, S_y) at frame k.

$$A_k(i,j) = \begin{cases} 1, \ inside\ silhouette \\ 0, \ otherwise \end{cases} \tag{2.4}$$

The error at frame k is given by:

$$E_k^{silh} = \sum_{i=1}^{S_x} \sum_{j=1}^{S_y} \| A_k^{real}(i,j) - A_k^{sim}(i,j) \| \tag{2.5}$$

The second metric computes the distance between the corresponding edge pixels (right side of figure 2.4) of the two silhouettes. To evaluate this metric, we first generate a *silhouette edge* from the silhouette image using a Canny edge detector. For each pixel in the first silhouette edge, we compute the closest point (L2 norm) in the second silhouette edge. The final value of the metric is obtained by accumulating the (square of) distance values across all pixels:

$$E_k^{silh} = \sum_{m=1}^{M} (i_m - i_m^*)^2 + (j_m - j_m^*)^2 \tag{2.6}$$

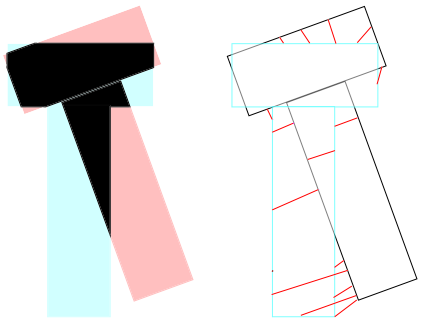


Figure 2.4: Two different methods of comparing silhouettes. Left: (a) Counting the number of mismatched pixels corresponds to the area within the silhouettes but outside the black region. Right: (b) Computing the distance between closest pixel pairs (minimum L2 distance) from the two silhouette edges (shown as red stripes in the figure). We show only a few closest pixel pairs in this figure (for clarity)—the metric computes the distance for all the pixel pairs from the two silhouette edges.

where (i_m, j_m) and (i_m^*, j_m^*) are corresponding pixels in the two silhouettes and M is the total number of pixels in the first silhouette edge. The error across the whole sequence of N frames, is given by:

$$E = \sum_{k=1}^{N} E_k^{silh} \tag{2.7}$$

2.3.4 Optimization

The optimization algorithm solves for the object and environment parameters simultaneously to match the silhouettes from simulation with those from video. Recall that the motion of a rigid body is fully determined by the parameters $\mathbf{p}$. For a given set of parameters, the optimizer simulates the motion to compute the silhouette images at each frame. It then computes the gradients of the silhouette metric and uses it to update the parameters of the optimizer. The goal of the optimization algorithm is to minimize the objective function defined by equation 2.7. We provide an initial estimate for the parameters $\mathbf{p}$ and update it at each step using gradient descent. Gradient based methods are fast, and with reasonable initialization, converge quickly to the correct local minima. The update rule for parameters is

$$\mathbf{p} = \mathbf{p} - \lambda \frac{\partial E}{\partial \mathbf{p}} \tag{2.8}$$

where λ is the magnitude of the step in the gradient direction.

Gradient Computation. The optimization algorithm requires computing the gradients of the objective function. This in turn, requires computing the gradients of the silhouette at each frame. Although the state $\mathbf{q}(t)$ is a continuous function of the parameters $\mathbf{p}$, quantities such as silhouettes are not continuously differentiable functions of parameters $\mathbf{p}$. One straightforward approach is to compute the gradients of the error function $\partial \mathbf{E}(\mathbf{q})/\partial \mathbf{p}$ numerically, using finite differences. This approach, however, has two major drawbacks. First, computing the gradients of the metric with respect to $\mathbf{p_{obj}}$ (initial conditions) using finite differences is extremely slow, because the simulation function has to be evaluated several times during the gradient computation. Second, determining step sizes that yield an accurate finite difference approximation is difficult. We resort to a hybrid approach for computing the gradients. We analytically compute the gradients of the state with respect to parameters $\partial \mathbf{q}(\mathbf{t})/\partial \mathbf{p}$. We then compute the derivative of the error with respect to the state using finite differences, e.g. $\partial \mathbf{E}(\mathbf{q})/\partial \mathbf{q}$. We use the chain rule to combine the two gradients:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{p}} = \frac{\partial \mathbf{E}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{p}} \tag{2.9}$$

Because the metric (e.g., silhouette $\mathbf{S}$) depends only on the position and orientation terms of the state $\mathbf{q}$, the gradient $\partial \mathbf{S}(\mathbf{q})/\partial \mathbf{q}$ can be computed quickly and accurately using finite differences. The camera parameters do not depend on the 3D state of the object, which allows us to use finite differences to compute the gradients with respect to the gravity vector $\mathbf{p_{env}}$.

Jacobian for Free Flight. The motion of a rigid body in free flight (in 3D) is fully determined by the control vector $\mathbf{p_{obj}}$. Rewriting equation 2.2 to show this dependence explicitly yields:

$$\frac{d\mathbf{q}(t)}{dt} = \mathbf{F}(t, \mathbf{p_{obj}}) \tag{2.10}$$

We evaluate the Jacobian $\partial \mathbf{q}(t_f)/\partial \mathbf{p_{obj}}$ at time t_f by numerically integrating the equation

$$\frac{d}{dt}\left(\frac{\partial \mathbf{q}(t)}{\partial \mathbf{p_{obj}}}\right) = \frac{\partial \mathbf{F}(t, \mathbf{p_{obj}})}{\partial \mathbf{p_{obj}}} \tag{2.11}$$

until time t_f with the initial condition $\partial \mathbf{q}(t_0)/\partial \mathbf{p_{obj}}$. We use a fourth order Runge-Kutta method with fixed step size to perform this numerical integration.

Derivatives of Error function. We use finite differences to compute the derivative of error function with respect to the current state $\mathbf{q(t)}$. We compute the silhouette at

the given state by rendering the simulated object. The derivative of the error function with respect to a scalar component q_i of the state $\mathbf{q}$ has the form:

$$\frac{\partial \mathbf{E}}{\partial q_i} = \lim_{\Delta q_i \to 0} \left(\frac{\mathbf{E}(\mathbf{q} + \Delta q_i) - \mathbf{E}(\mathbf{q})}{\Delta q_i} \right) \tag{2.12}$$

The error Jacobian is obtained by applying chain rule

$$\frac{\partial \mathbf{E}}{\partial \mathbf{p_{obj}}} = \frac{\partial \mathbf{E}}{\partial \mathbf{q}} \frac{\partial \mathbf{q}}{\partial \mathbf{p_{obj}}} \tag{2.13}$$

Finally, we compute the error Jacobian with respect to a scalar component p^i_{env} of the gravity direction $\mathbf{p_{env}}$ using finite differences, as shown:

$$\frac{\partial \mathbf{E}}{\partial p^i_{env}} = \lim_{\Delta p^i_{env} \to 0} \left(\frac{\mathbf{E}(\mathbf{p_{env}} + \Delta p^i_{env}) - \mathbf{E}(\mathbf{p_{env}})}{\Delta p^i_{env}} \right) \tag{2.14}$$

The overall Jacobian matrix is given by:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial \mathbf{E}}{\partial \mathbf{p_{obj}}} \\ \frac{\partial \mathbf{E}}{\partial \mathbf{p_{env}}} \end{pmatrix} \tag{2.15}$$

2.3.5 Results

Our system has three main modules: preprocessor, rigid body simulator and optimizer. We first preprocess the video sequence to compute the silhouettes of the rigid object. We build a background model for the scene and use autoregressive filters [BSK00] to segment the moving object from the background. We then compute the silhouette metrics from the segmented image at each frame. Our tumbling video sequences are typically 35-40 frames long when captured with a digital camera operating at 30 Hz. The optimizer typically takes a couple of minutes to compute the parameters from the sequence on a SGI R12000 processor.

Estimation: Figure 2.5 shows a few frames of a T-shaped object in simulation matched with video. This sequence was obtained by tossing the object in the air in front of a camera. Our user interface lets the user specify an approximate value for the initial positions and velocities of the body. The algorithm robustly estimates the initial position and linear velocity, even with a poor initial guess. However, it is very sensitive to the initial estimate of the orientation and angular velocities. From numerous experiments, we have found that the error space of the silhouette metric contains many local minima. However, with a reasonable initialization for the orientation and angular velocity parameters, we find that our algorithm converges to a

reasonable local minima. This convergence is seen in figure 2.5(a), where the overall motion of the simulation closely matches the motion in video. We superimpose the bounding boxes obtained from simulation onto the frames from video (the white boxes in the first row) to show the match. We also show the match between the trajectory of a corner point in video with the corresponding trajectory in simulation. The small square boxes show the trajectory of a corner point, identified by hand, from the video sequence. These trajectories are shown for visualization purposes only and are not used in the optimization. As the algorithm proceeds, the trajectory of the 3D corner point in simulation (black line) overlaps with these boxes. This sequence also highlights some limitations with our optimization framework and metrics. Row (c) shows an example where the simulated and the real object have different orientations but silhouettes that appear similar.

Prediction: The objective of this experiment is to predict the motion of a rigid body in a long video sequence of free flight from a subset of frames of the same sequence. Figures 2.6 and 2.7 show two different video clips of a rigid body in free flight, with different motion trajectories. We match the motion of the shorter clip to a simulation and use the simulation parameters to predict the motion of the longer clip.

In figure 2.6, the object tumbles about its major axis—the physics of this motion is captured in a shorter subsequence. Hence the simulation parameters computed by matching this small sequence correctly predict the motion in the overall clip. However, the motion of the object in figure 2.7 is about the body's intermediate axis, which degenerates into complex motions about all three axis [Sym71]. Small errors in the estimated values of simulation parameters result in a large orientation error in the predicted frames, as time increases. We see this effect in the results obtained in figure 2.7.

Roll Correction: Because the orientation of the camera is not known, our algorithm optimizes the direction of the gravity vector along with the motion parameters from a video sequence. Figure 2.8 shows results of roll correction performed using the parameters obtained from our optimization algorithm. The first row shows four frames of a video sequence captured from a camera with significant roll distortion. We compute the camera pitch and roll parameters which minimize the silhouette error. These parameters are used to warp the original sequence such that the camera vertical axis aligns with the gravity vector. The last row shows the result of this rectification using the estimated roll angle (the pitch angle was estimated as well in this experiment, but was not used for rectification). The people in the first row are in a slanted pose, whereas they are upright in the last row.

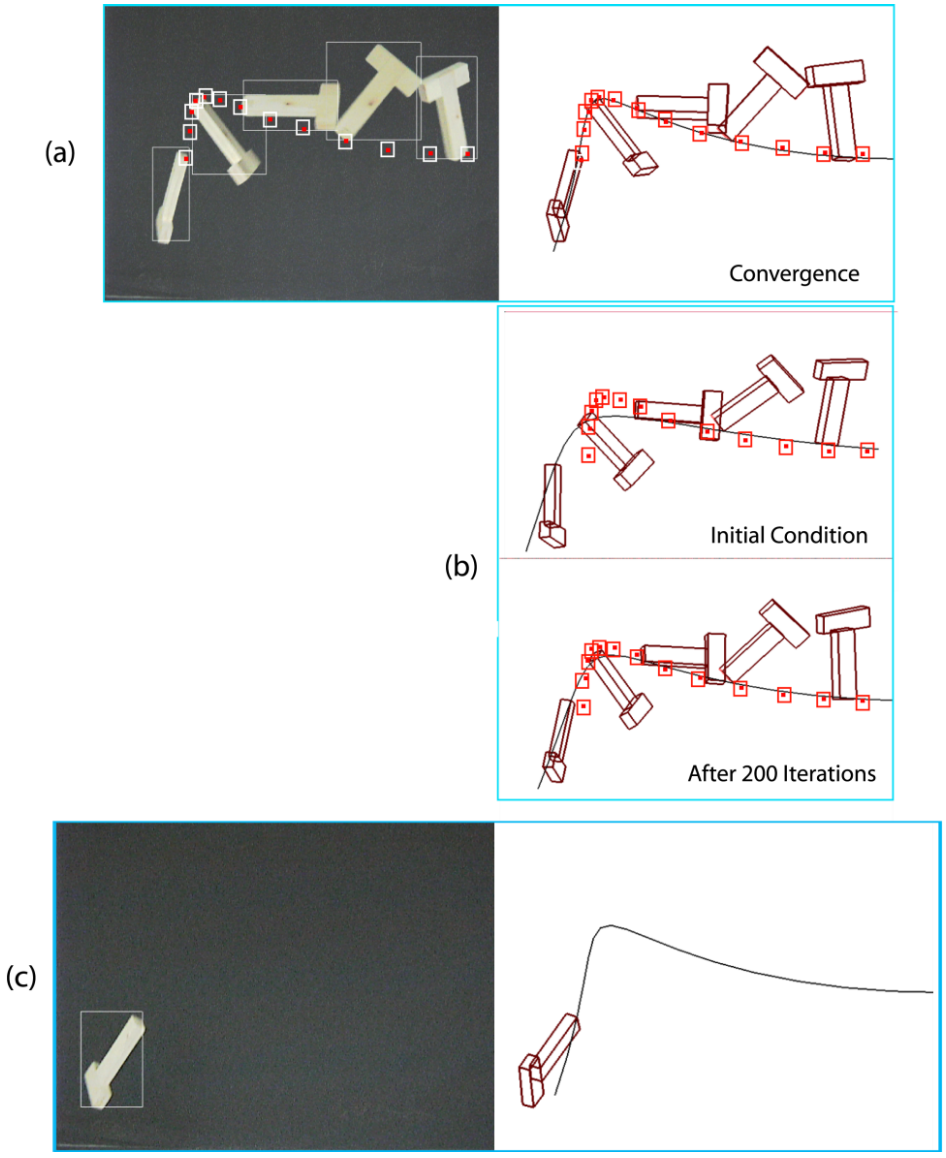


Figure 2.5: Computing the parameters of a T-shaped object thrown in the air. The first row (a) shows a few frames from the video (right to left) and the corresponding physical simulation generated by our algorithm. The bounding boxes from simulation are superimposed over the video to show the match. The small square boxes indicate the trajectory of a corner point in video. The thin line indicates the motion of the corresponding corner in simulation. Row (b) shows the results of the algorithm at different stages of optimization. Notice that the match improves as the number of iterations increases. Row (c) highlights a limitation of the silhouette based metric. Note that although the orientation of the simulated object is flipped relative to the real object, they both have similar silhouettes.

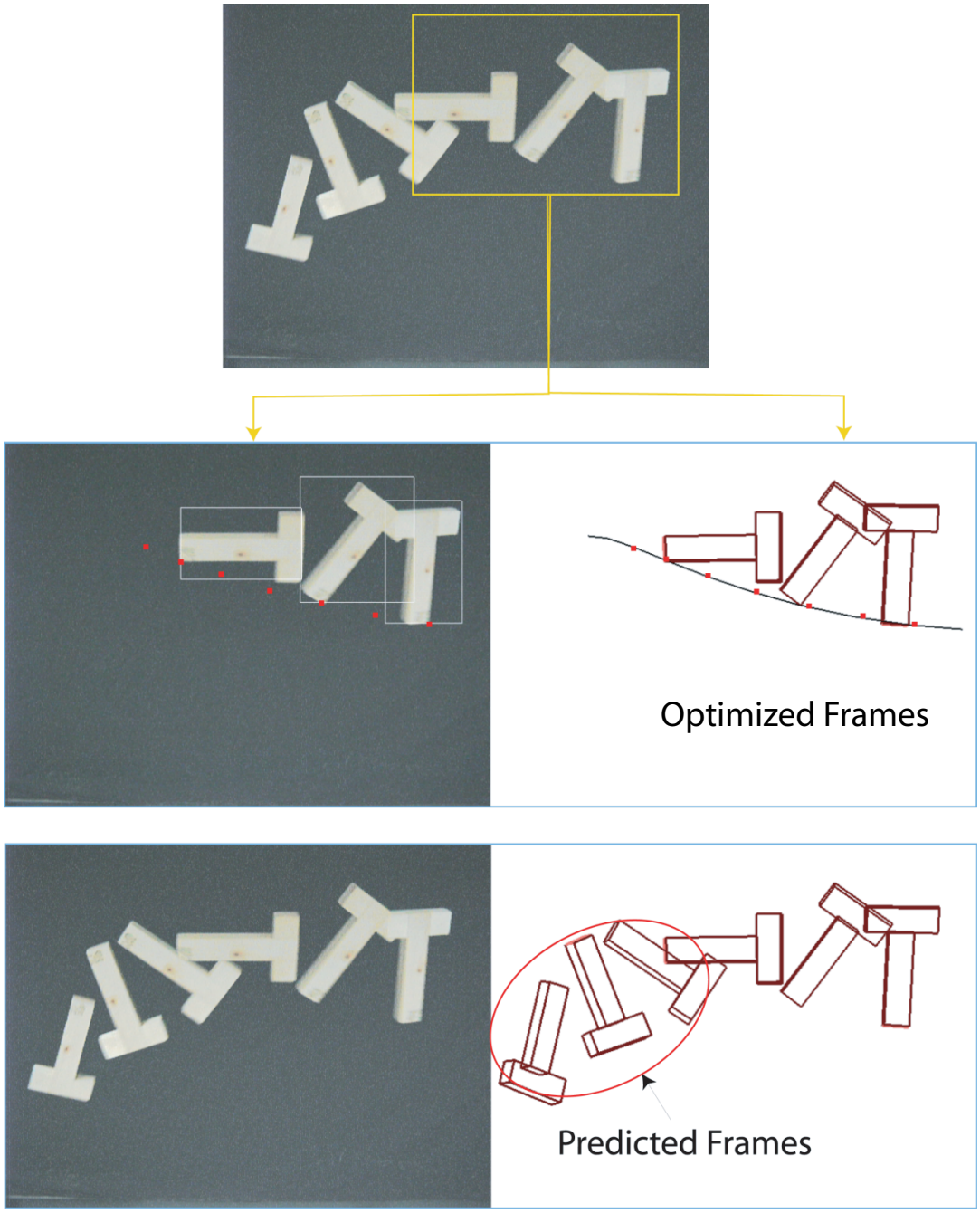


Figure 2.6: Predicting the motion of objects in free flight in video. The first row shows a long sequence of an object thrown from right to left. We select a portion of this sequence and match its motion in simulation. The second row shows a portion of the original clip contained inside the yellow box, and the corresponding frames of a simulation. We use these parameters to predict the motion of the tumbling object across the whole sequence.

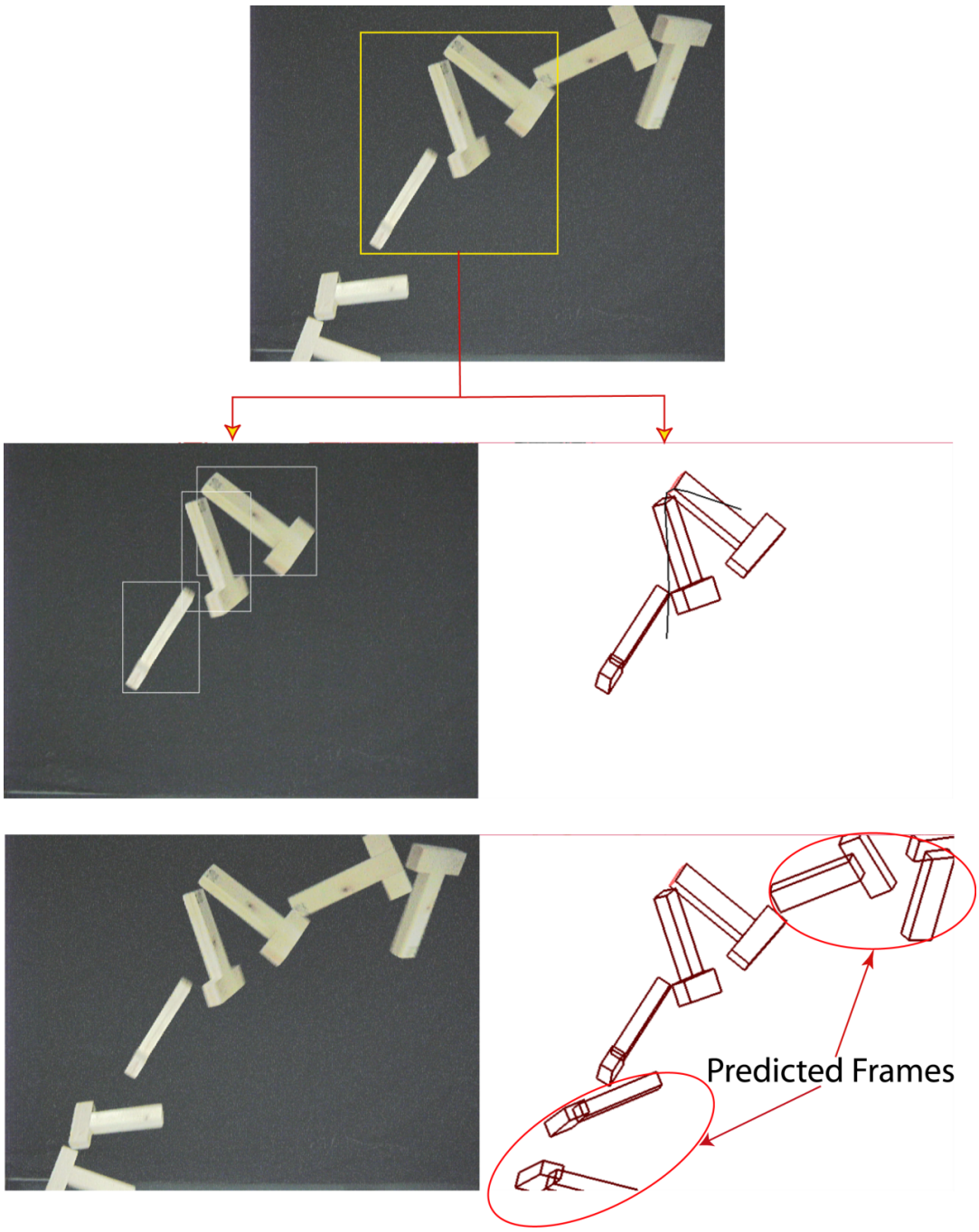


Figure 2.7: Predicting the motion of an object in a complicated tumbling pattern (top row). The second row shows the match between a small portion of the video clip and a simulation. The simulation matches the video quite well for the frames on which it optimized, but the small errors propagate to larger errors in the predicted frames.

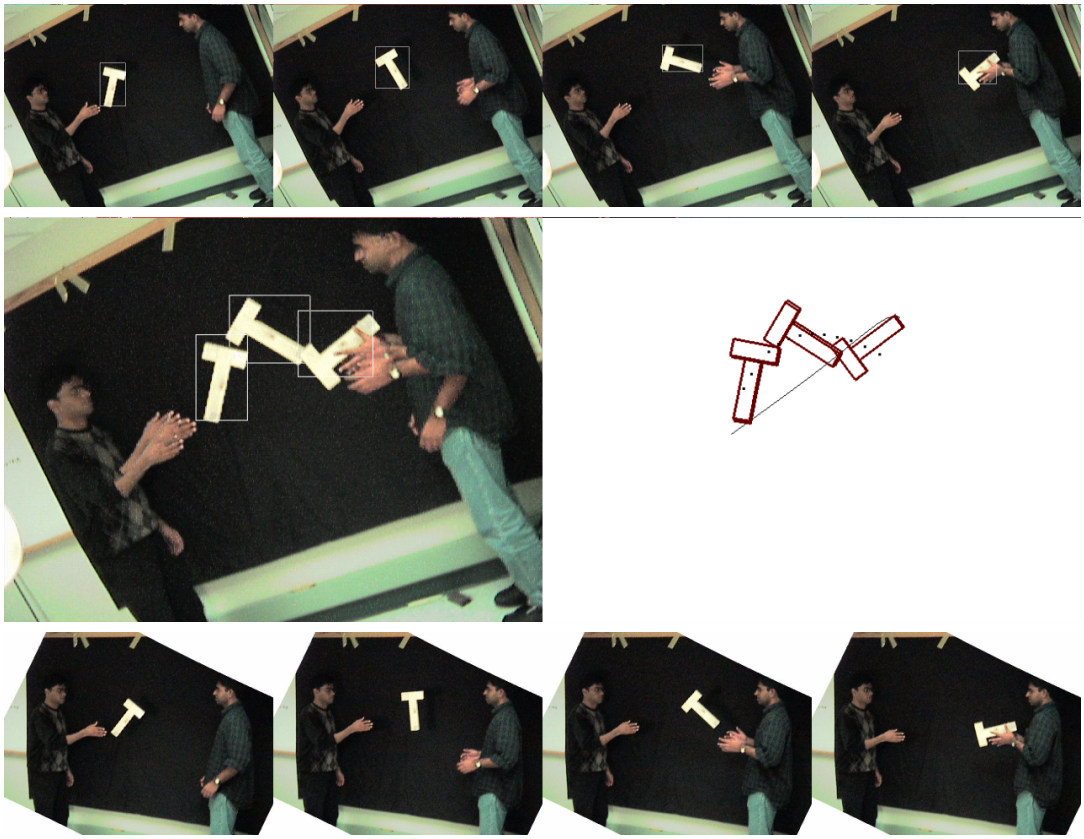


Figure 2.8: Using rigid-body motion estimation to correct for camera roll. An object is thrown in the air from left to right (top row) and captured from a video camera with significant roll. The video is difficult to watch because of the image rotation. The motion of the object is estimated (middle row), as well as the camera roll (defined by the orientation with respect to the gravity direction). The video frames are automatically rectified to correct for the camera roll.

2.4 Capturing Cloth Simulation Parameters

The objective of our second experiment was to create realistic cloth simulations that matched the behavior of four real fabrics—linen, fleece, satin and knit. The ubiquity of cloth makes it an indispensable element in creating compelling animations of human figures. Realistic virtual clothing is now possible because of recent advances in cloth simulation techniques[BW98, CK02, BHW94, CVMT95, BFA02].

The motion of fabric is determined by resistance to bending, stretching, shearing, external forces, aerodynamic effects, friction and collisions. We used the cloth simulator developed by Baraff and Witkin [BW98] for our experiments, and estimated the static and dynamic parameters of this model from video. The parameters of the Baraff-Witkin model can be characterized into three groups:

- Static Parameters: *mass, stiffness (stretch, shear and bend)*. Our model has stiffness parameters to control the stretch, shear and bend forces acting on the fabric. The stretch and shear stiffness parameters have two components along the two principal directions, and the bend stiffness has one component.

- Dynamic Parameters: *damping, air drag*. We have a damping term corresponding to each stiffness term, namely stretch damping, shear damping and bend damping. Additionally, we model the air drag with a three parameter model.

- Collision parameters: stiffness terms for *self collisions* and *object collisions*. The collision parameters were selected manually in our experiments.

Our algorithm estimated the static and dynamic parameters on a set of static shots and motion clips of a small swatch of a particular fabric and then tested on a simulation of a full skirt made from that fabric. We designed the swatch tests to span the space of behaviors that we expect to see in the final sequences of motion with the skirt so that all parameters can be tuned appropriately. We use simulated annealing for the optimization step with an optimization function that assesses the extent to which the folds in the simulated and physical fabric match. This match is evaluated by means of a shape metric that uses projected light to detect surface orientation in real and simulated fabrics. The metric is tuned to be most sensitive along folds and to discount planar regions.

We use the system to find the parameters for four different fabrics. We show the match between the video footage and the simulated motion on the calibration experiments, on additional video sequences for the swatches, and on a simulation of a full skirt.

2.4.1 Prior Work

Cloth modelling has a long history, dating back to work in the textile community from the mid-1930s by Peirce [Pei37]. Work on cloth modelling in computer graphics has focused on developing dynamic simulation techniques that are both realistic and fast. Baraff and Witkin describe a cloth model that uses stiff springs with implicit time integration [BW98]. This model was subsequently adapted to reduce the overdamping due to implicit integration [CK02]. Explicit time integration approaches [HDB96] use weaker springs for stretching and shearing, often explicitly limiting the amount of stretching [Pro95, BFA02]. Choi and Ko introduced a bending energy model that more accurately captures the fine creases and bends of cloth [CK02]. Lahey provides a comprehensive overview of cloth hysteresis models from the perspective of computational fabric mechanics [Lah02]. Extensive work has also been done on modelling collisions and friction. Cloth self-collision is handled either by untangling the cloth [CVMT95, VMT00, BWK03] or by preemptively avoiding collisions [Pro97, HMB01, BFA02]. Various potential field methods have been used for general collision detection and response [TPBF87, SP91].

Despite this large body of work on cloth simulation models, little work has appeared in the computer graphics literature on estimating the parameters of these models so that they match the behavior of real fabrics. Cloth parameter estimation has been studied in the textile community (for an overview, see Breen and colleagues [HB00]), but such methods have not yet enjoyed widespread use in the computer graphics community. An important exception is the work by Breen [BHW94] who used the Kawabata system [Kaw80] to measure bending, shearing, and tensile parameters by subjecting a swatch of fabric to a series of mechanical tests and measuring the force needed to deform it into a standard set of shapes. Although the Kawabata system can provide accurate measurements, these measurements are problematic for computer graphics cloth simulation problems for two reasons. First, there might not be a direct and simple mapping between the parameters for a particular cloth model and the Kawabata parameters. Second, the Kawabata system does not measure *dynamic* cloth parameters, e.g. air drag or damping, which are essential for computing the motion of cloth.

One promising approach for modelling cloth parameters is to automatically search for parameters that match real cloth. Jojic and Huang fit parameters of a particle-based cloth model to fit a range scan of real cloth in a static rest configuration, draped over a sphere [JH97]. However, they did not treat the problem of measuring dynamic parameters or demonstrate accurate results across a range of fabric types. Instead, they explored the problem of measuring the 3D geometry of an object from the resting shape of a piece of cloth draped over it, a challenging problem that we do not consider in this paper.

More distantly related are techniques for computing the geometry of cloth from images. Coarse estimates of the time-varying geometry of cloth can be computed using traditional stereo matching techniques by using two or more cameras and treating each time instant independently (see Scharstein and Szeliski [SS02] for an overview). More accurate results may be obtained by projecting structured light patterns on the cloth (see Zhang et al. [ZCS02] for an overview). Rather than computing shape at every time instant independent from the next, it can be advantageous to integrate images over time to improve accuracy. Two examples of promising work along these lines are Carceroni and Kutulakos [CK01] and Torresani et al. [TYAB01]; both studies demonstrated reconstructions of moving cloth.

2.4.2 Cloth Model

Because our framework for estimating cloth simulation parameters is independent of the cloth model, we can, in principle, select a specific model that meets a set of criteria such as accuracy or simulation speed. Our choice of a cloth model was guided by two goals: realism and practicality. We wanted a model that was sophisticated enough to capture the detailed dynamic behavior found in real fabrics but still straightforward to implement. Because our intention was to apply the learned cloth model parameters to arbitrary garments with varying triangle resolution, it was also important that the cloth parameters scale correctly with the resolutions of the cloth simulation.

We used the model described by Baraff and Witkin as the basis for our cloth simulator [BW98]. This model has sufficient richness to produce a wide variety of cloth behaviors. The underlying mesh is triangular, making clothing modelling easier. More importantly, its input parameters are independent of meshing, so that parameters recovered on one mesh (the test swatch) can be transferred to another (the skirt). While nonlinear models such as the buckling behavior of Choi and Ko [CK02] could potentially capture more realistic details of cloth, there is no straightforward way to scale the parameters of these models to meshes of varying resolutions. We expect that future application of our parameter-estimation framework to other scale-invariant cloth models will provide even more realistic results.

The model developed by Baraff and Witkin formulates the energy of a particular triangle in terms of *condition functions* $\mathbf{C}(\mathbf{x})$ such that the total potential energy associated with the system is given by

$$\mathbf{E}_u = \frac{k_s}{2}\mathbf{C}(\mathbf{x})\mathbf{C}^T(\mathbf{x}) \tag{2.16}$$

where k_s is a stiffness coefficient associated with the particular condition function.

Forces are then simply calculated by

$$\mathbf{F} = \nabla_{\mathbf{x}} \mathbf{E}_u \tag{2.17}$$

Damping forces are similarly fomulated in terms of the $\mathbf{C}(\mathbf{x})$:

$$\mathbf{d} = -k_d \frac{d\mathbf{C}}{d\mathbf{x}} \dot{\mathbf{C}}(\mathbf{x}) \tag{2.18}$$

We thus associate a stiffness coefficient k_s and a damping coefficient k_d with each of the $\mathbf{C}(\mathbf{x})$. In their paper, Baraff and Witkin describe a set of $\mathbf{C}(\mathbf{x})$ consisting of an in-plane stretch term, an in-plane shear term, and an out-of-plane bending term, giving a total of six parameters we can use to tune the internal cloth model. We refer the reader to their paper for the full details[BW98]. We note, however, that (as they allude to in footnote 5) *energy* should scale *linearly* with triangle area to ensure scale independence. Therefore, we need to be careful when substituting $\mathbf{C}(\mathbf{x})$ for stretch and shear into eq. 2.16 that the resulting formula is linear in area rather than quadratic.

In the course of running our experiments, we discovered that a linear drag model such as that used in previous cloth work [BW98, CK02] was not able to capture dynamic aspects of cloth. In order to add additional air-drag degrees of freedom to our cloth model without resorting to fully modeling aerodynamics [Lin00], we developed a simple nonlinear alternative. Our new formulation for the drag force has a linear term for the tangential velocity component ($\mathbf{v}_T$) and a quadratic term for the normal velocity component ($\mathbf{v}_N$), with an additional term k_f that controls the degree of nonlinearity:

$$\mathbf{f}_{drag} = -a \left(\frac{k_N |\mathbf{v}_N|^2}{1 + k_f |\mathbf{v}_N|^2} \frac{\mathbf{v}_N}{|\mathbf{v}_N|} + k_T \mathbf{v}_T \right)$$

where a is the area of the given triangle. The linear term is merely Stokes's law [Ach90]; the quadratic term matches better the experimental behavior of macroscopic bodies in low Reynold's number flow[Fro84]. The addition of the $|\mathbf{v}_N|^2$ term in the denominator which makes the force asymptotic as $\mathbf{v}_N \to \infty$ was partially motivated by the observed phenomenon of *drag crisis* [Fro84], where under certain circumstances the drag coefficient can actually drop at the onset of turbulence [Ach90]. The optimizer is free to eliminate this behavior or other terms of this equation by setting the corresponding parameters to zero.

Initially, we used a first-order implicit Euler time integration scheme similar to the one described by Baraff and Witkin [BW98]. Unfortunately, we found that implicit integration introduced damping which could not be eliminated by optimizing cloth parameters. We had more success in matching realistic cloth motions by using higher-order explicit methods. The results in this thesis use an adaptive 4th-order accurate Runge-Kutta methods with embedded error estimation [AP98]. While

this method offers the advantages of simplicity and automatic bounding of error, it is rather slow, and recent work suggests that using 2nd-order backward differences [CK02] or Newmark schemes [BMF03] may be a better choice.

For collision handling, we use a model similar to that used by Bridson and colleagues [BFA02]. It combines repulsion forces with impulses to robustly prevent all collisions before they occur. However, separating repulsion forces from the cloth internal dynamics and applying them outside the Runge-Kutta solver affected stability and resulted in visible artifacts. Instead, we apply repulsion forces inside the solver loop, so that the solver's own internal error estimation can remove these artifacts. The drawback of this technique is speed, because the system must check for collisions every time it evaluates the state derivatives (as opposed to once every *collision timestep* as in Bridson et al. [BFA02]). To achieve acceptable performance, we used a number of collision culling algorithms, including hybrid top-down/bottom-up update [LAM01], fast triangle reject tests [Möl97], and a curvature-based criterion for rejecting self-collisions that was first introduced by Volino and Thalmann [VMT94] and later refined by Provot [Pro97].

2.4.3 Metrics from Video

We use a perceptually motivated metric to compare the motion of cloth in simulation with a video sequence of real fabric motion. Our metric captures the complex dynamics of cloth motion and also helps to distinguish between different fabrics.

Researchers in computational neurobiology hypothesize that the human perceptual system is sensitive to *moving edges* in video [Fie87, Fie94, vHR98]. Studies have shown that the receptive fields of simple cells in the macaque cortex act as edge or line detectors, responding to oriented edges or lines in natural scenes [HW68, VYH82, DOF93]. In cloth, these edges correspond to folds and silhouettes, which are regions of high variation in shape. Hence, our perceptually motivated metric for cloth compares two video sequences, one from simulation and one from the real world, and returns a number that measures the differences in their folds. The metric also penalizes the silhouette mismatch between the two sequences.

Fold Detection and Representation: Folds appear as soft edges in video whose appearance is dependent on material properties and lighting. Haddon and Forsyth [HFP98, HF98] describe a learning approach for detecting and grouping folds (and grooves) in images of fabrics. Their technique can handle lighting effects caused by diffuse inter-reflections in cloth. However, most fabrics have very complicated reflectance properties. In our experiments, we eliminate the effects of lighting and material reflectance by projecting a structured light pattern of horizontal stripes onto

the fabric.

From the light-striped video sequence, we compute the *dominant orientation* for each edge pixel by convolving it with a steerable filter bank [FA91]. In our implementation, we use the G2/H2 quadrature pair with kernel size 12 as the basis filters. Details of computing the dominant orientation from the coefficients of filter bank response are given in Appendix I of Freeman and Adelson [FA91]. We convolve the image with the filter bank, compute the filter coefficient responses, blur the coefficients using a gaussian kernel, and compute the dominant orientation from these coefficients. We name the resulting orientation image an *angle map*, shown in figure 2.9. The angle map, which measures the local orientation of the projected pattern, has a constant value when the surface is planar and varies at folds. We threshold the gradient of the angle map to get a *gradient mask* M_k for each frame of video (figure 2.9).

$$M_k(i,j) = \begin{cases} 1, & \|\delta(i,j)\| \geq \tau \\ 0, & \|\delta(i,j)\| < \tau \end{cases} \tag{2.19}$$

where τ is a user defined threshold and $\|\delta(i,j)\|$ is the magnitude of the gradient of the angle map at (i,j). The gradient mask is non-zero at regions of high gradients, corresponding to folds, and zero at planar regions.

Fold Comparison: Our metric computes the frame by frame sum of squared differences (SSD) between masked angle maps in simulation with video. We preprocess the input video sequence to compute the angle map at each frame. Similarly, in simulation, we render the cloth shape using the current parameter values and project the same striped pattern, to get a striped simulation sequence. We compute the angle map at every frame in simulation from this sequence. We then compute the SSD of the angle values for all overlapping points in the two angle maps. We multiply this difference with the gradient mask, which helps to emphasize the differences in fold regions over planar regions (figure 2.10). We sum the error across all frames to compute the overall error across the entire sequence. The error at any particular frame k along the sequence is

$$E_k^{fold} = \sum_{i=0}^{S_x} \sum_{j=0}^{S_y} M_k(i,j) \cdot (\theta_k^{real}(i,j) - \theta_k^{sim}(i,j))^2 \tag{2.20}$$

where (S_x, S_y) is the size of the angle maps and θ^{real}, θ^{sim} are the angle values from real and simulation angle maps respectively.

Silhouette Comparison: In addition to the angle map error, we penalize the silhouette mismatch between the simulation and the video of real cloth. This penalty

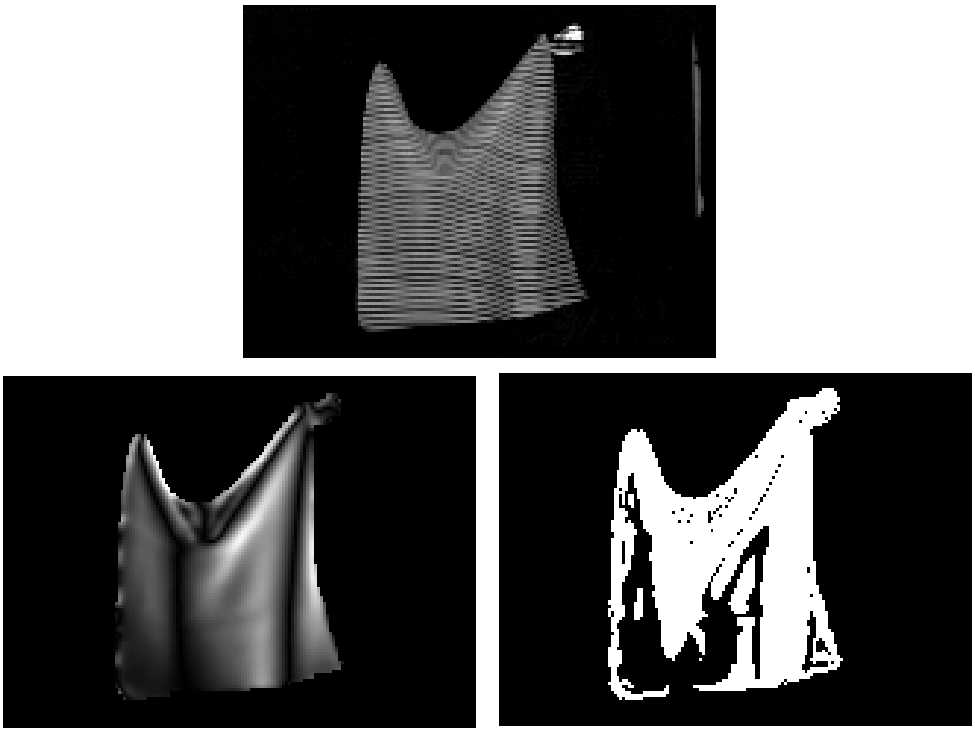


Figure 2.9: Top Row: Input light striped image. Bottom Row (left to right): angle map and gradient mask.

is proportional to the difference between the two silhouettes, i.e., the number of mismatched pixels:

$$E_k^{silh} = \sum_{i=0}^{S_x} \sum_{j=0}^{S_y} \mid A_k^{real}(i,j) - A_k^{sim}(i,j) \mid \tag{2.21}$$

where

$$A_k(i,j) = \begin{cases} 1, & inside\ silhouette \\ 0, & otherwise \end{cases} \tag{2.22}$$

The total error in frame k is

$$E_k = E_k^{fold} + \alpha E_k^{silh} \tag{2.23}$$

where α is a user-defined weight that controls the relative contribution of the two terms. We used a value of 0.1 for α in our experiments. The error across the entire sequence of length N frames is given by

$$E = \sum_{k=1}^{N} E_k \tag{2.24}$$

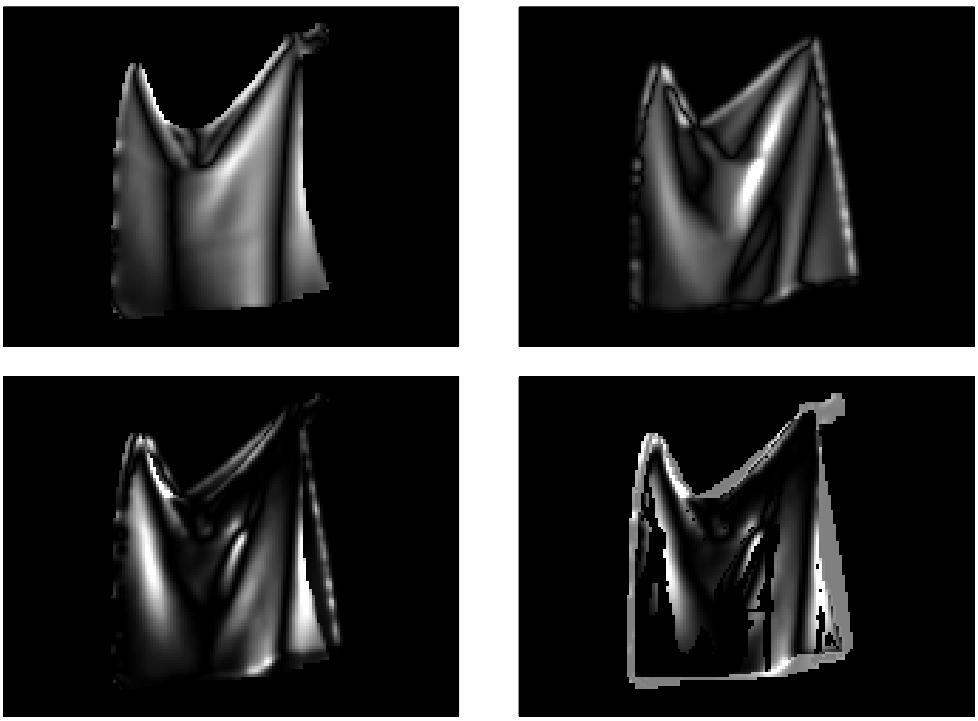


Figure 2.10: The stages in the metric pipeline. Top row (left to right): Angle map from video, angle map from simulation. Bottom row (left to right): angle map difference, final metric value for this frame (angle map difference multiplied by gradient mask from video).

2.4.4 Parameter Identification

We use optimization to estimate the parameters of the cloth simulator from video. Before we describe the details of the optimizer, it is informative to look at the error space of the angle map metric. This fairly noisy space motivates the use of a global optimization technique, like *Simulated Annealing* and also gives us useful insight about the parameters of the system. Figure 2.11 shows the variation of error for different values of bend and stretch stiffness coefficients for satin. Here, the angle map in video is obtained after the fabric settled down in a static test. The corresponding angle map in simulation is computed for a wide range of values for bend and stretch, spanning the space of these two parameters. From the figure, it is evident that the error space is very noisy with several local minima, motivating the need for a global optimization technique.

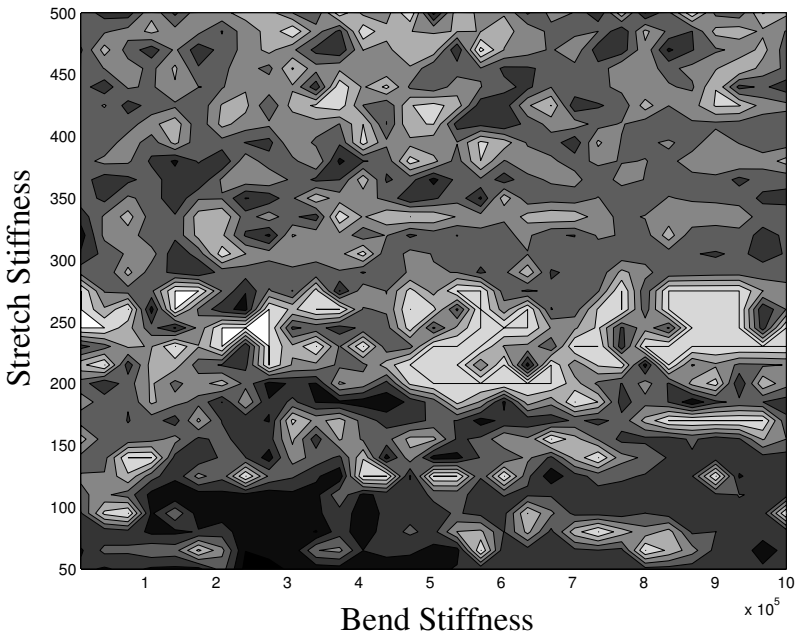


Figure 2.11: This plot shows angle map error as a function of bend and stretch stiffness parameters. Dark areas indicate regions of small error and bright areas correspond to large errors. Note that the space is fairly noisy. The minimum found by the optimizer is contained in the large dark region in the lower portion of the plot.

2.4.5 Optimization Framework

We use simulated annealing to find the parameters that minimize the error function given in eq. 2.24. Simulated annealing initially explores the space in a semi-random fashion and eventually takes downhill steps. The likelihood that it will take a step in a direction that is not locally optimal is a function of the *temperature* (figure 2.12). We use the continuous simulated annealing method presented in Press et al.[PFTV92], which combines the Metropolis algorithm with the downhill simplex method for continuous n-variable optimization. We reset the simplex with the current best solution when the temperature is reduced by a factor of 3. Prior to optimization, we perform an exhaustive search for each fabric, where we choose four values for each cloth parameter across its entire range. This procedure corresponds to a very coarse sampling of the parameter space. We simulate the fabric for all points in this coarse set (with 4^9 points – 9 parameters $\times$ 4 values per parameter) and compute the error for each point by comparing against the real fabric. We initialize the optimizer with the point corresponding to the minimum error. We have found that this strategy for initializa-

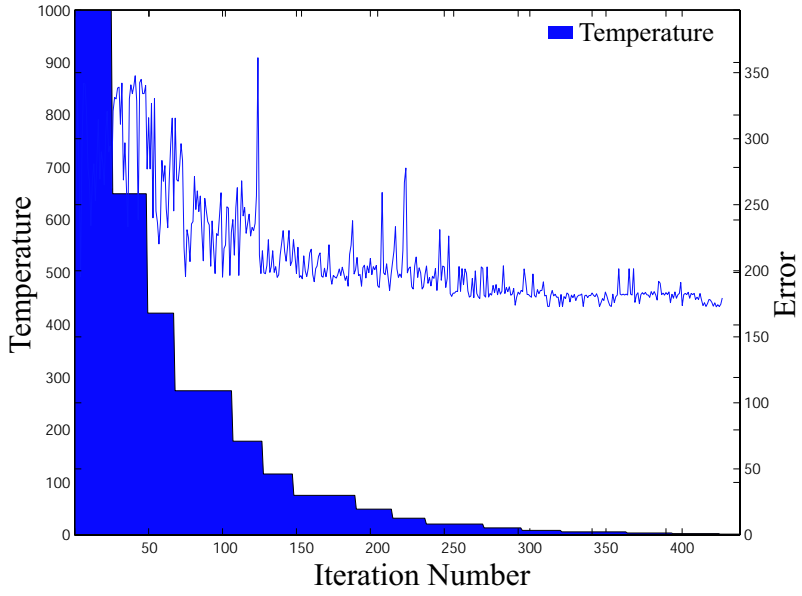


Figure 2.12: Progress of the simulated annealing optimizer as measured by error. The temperature decrease is governed by a geometric cooling schedule.

tion allows the optimizer to locate a better minimum of the space rather than using constant (arbitrary) values for the parameters for the initial guess.

2.4.6 Experiments

We designed simple experiments to capture the dynamics of the different types of fabrics and the air/cloth interaction. The experiments are easy to perform, capture, and repeat; yet they demonstrate the complex dynamics of cloth motion. The parameters obtained from the simple experiments were used to simulate skirts and other complex fabric motions. In essence, our experiments were designed to be a *calibration* setup for estimating the static and dynamic parameters of a cloth simulator.

We perform two estimation experiments for each fabric, a *static test* and *waving test*. We used four types of fabrics: linen, fleece, satin and knit. These fabrics exhibit a wide range of static and dynamic behavior and span a large range of real fabrics.

We perform the static and waving tests on a small swatch of each fabric. In the static test, the two top corners of the fabric are held stationary, and the fabric is allowed to sag under gravity. For a fixed separation between the top corners, different fabrics attain different static shapes as shown in figure 2.13. The static test gives a good estimate for the static stiffness and bend parameters.

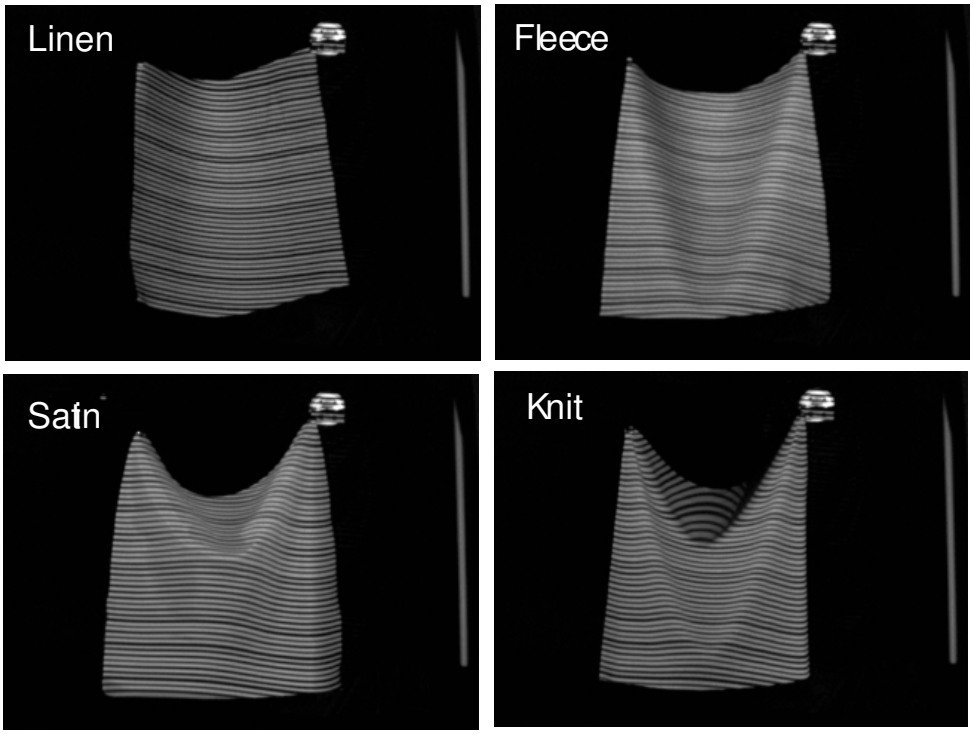


Figure 2.13: The static test with four real fabrics. Top corner separation is identical across all four fabrics.

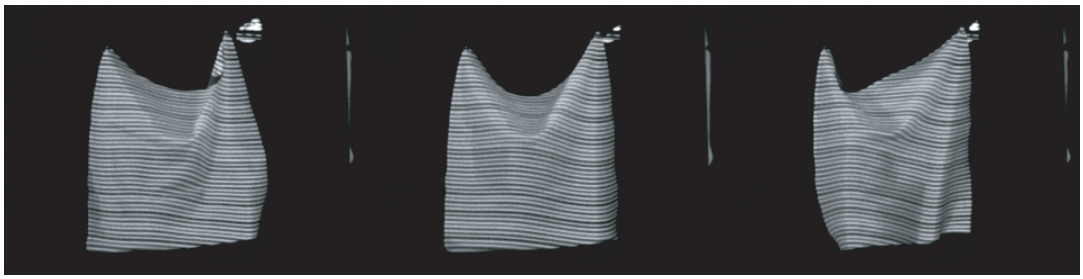


Figure 2.14: Three frames from the waving test for satin.

In the waving test, one of the top corners of the fabric is fixed and the other corner is moved back and forth (figure 2.14). The waving motion of fabrics in simulation is affected by their dynamic parameters. The videos posted on our website[1] that real fabrics exhibit a wide range of interesting motions even with the same input excitation. We designed the waving motion to approximately match the types of motion occurring in real garments such as skirts. This experiment gives reasonable estimates for cloth parameters while avoiding the need to optimize directly on complex fabric geometries (e.g. skirts) involving many collisions.

2.4.7 Results

In this section, we report the results of simulation parameters obtained using our technique applied to four fabrics: linen, fleece, satin and knit. We measured the mass and dimensions of the fabrics. We also accurately measure the position of the two top corners using a Vicon motion capture system. We compute the projection matrices for the camera and projector using a calibration grid with several motion capture markers. We performed two trials per experiment, each with slightly different initial conditions and optimized on the first 50 frames of video in each trial. Each trial took approximately 50 hours to converge on a 2.8GHz Intel Pentium 4 Xeon processor (approximately 600 iterations of simulated annealing). Because each optimization takes a long time to converge, we choose the final parameter estimate (for each fabric) from one of the two trials (that minimizes the total error on the two trials) instead of simultaneously optimizing across both the trials.

Static test. We performed optimization on two trials for each fabric; the results are shown in figure 2.15 and figure 2.16. The two trials have different separation distances between the top corners. For each fabric, we optimized for six parameters: stiffness and damping parameters for stretch, shear, and bend. The air drag parameters were set to the mid point of their range of values for this experiment.

The initial values and final values of the estimated parameters are summarized in table 2.1. Figures 2.15 and 2.16 show a very good visual match between the simulations with their counterpart real fabrics. However, there is a significant disparity in the optimized parameter values from the two trials. In order to understand this disparity, we performed a set of optimizations (on a single fabric) with very similar initial values. Table 2.2 shows the parameter values for satin from five optimizations where the initial conditions were randomly varied by $\pm 5\%$. From the table, we see that the final error values are very close. We get consistent estimates for parameters that have lower variability (e.g., bend, stretch). Parameters with high variability are

[1] http://graphics.cs.cmu.edu/projects/clothparameters

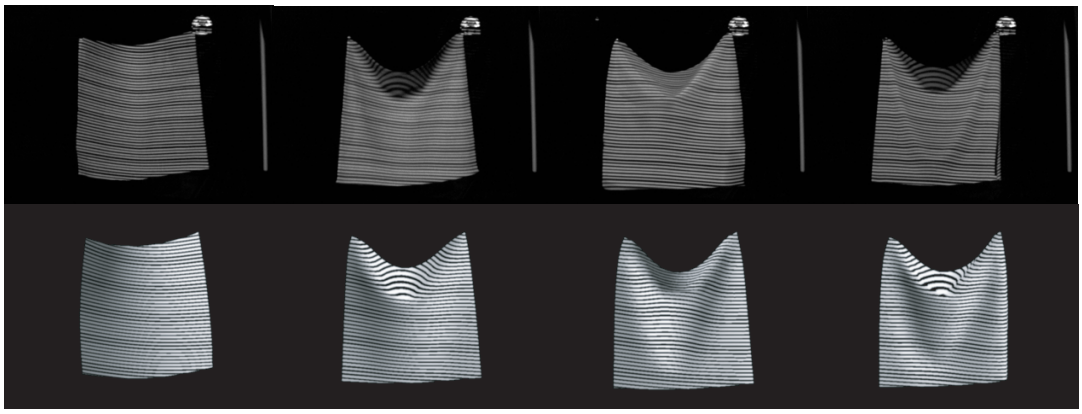


Figure 2.15: Results of optimization for the static test, trial 1. Top row: real fabrics (left to right) linen, fleece, satin and knit. Bottom row: Corresponding fabrics in simulation.

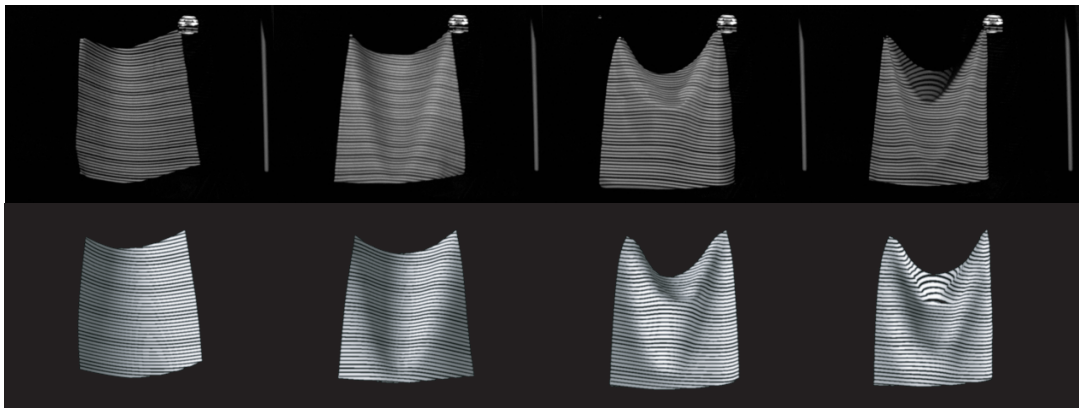


Figure 2.16: Results of optimization for the static test, trial 2. Top row: real fabrics. Bottom row: Corresponding fabrics in simulation.

	Linen			***Fleece***		
Parameters	Start	Exp 1	Exp2	Start	Exp 1	Exp 2
Bend	1e-3	0.009	0.0045	1e-4	0.0001	0.0001
Stretch	4000	404.9	3682.1	50	129.2	200.04
Shear	215.44	175.37	208.15	215.44	103.96	31.39
Bend damping	1e-7	9.92e-7	3.22e-7	2.15e-6	2.13e-7	4.11e-7
Stretch damping	10	12.16	10.17	10	4.78	0.064
Shear damping	10	2.19	13.17	10	13.86	3.75
Error (Exp 1)	75.0	62.8		136.8	121.5	
Error (Exp 2)	81.2		67.0	207.3		98.7
	Satin			***Knit***		
Parameters	Start	Exp 1	Exp2	Start	Exp 1	Exp 2
Bend	1e-5	1.106e-5	6.94e-6	1e-6	1.52e-6	1.51e-6
Stretch	50	19.58	19.38	50	27.97	28.36
Shear	50	76.81	69.65	50	1226.44	2693.07
Bend damping	1e-7	2.49e-7	3.98e-7	1e-7	1.01e-7	2.27e-7
Stretch damping	10	14.42	3.68	10	10.12	11.83
Shear damping	10	4.11	4.56	10	0.13	4.04
Error (Exp 1)	115.5	92.5		253.1	172.8	
Error (Exp 2)	194.7		104.7	179.7		136.1

Table 2.1: Tabulation of the static parameters from two experiments.

	Bend	Stretch	Shear	Bend Damp	Stretch Damp	Shear Damp	Error
Initial Values	1.0e-05	50	50	2e-07	10	10	179.026
Optimization 1	6.93766e-06	19.3832	69.653	3.98337e-07	3.67932	4.56238	104.747
Optimization 2	7.77204e-06	20.2884	32.6492	2.08755e-07	1.95807	10.6535	104.502
Optimization 3	8.75613e-06	19.8365	50.8304	2.56854e-07	7.08276	9.25576	103.501
Optimization 4	9.55647e-06	19.2745	74.7429	3.14821e-07	5.47909	1.06559	103.243
Optimization 5	8.47762e-06	20.1119	36.762	2.3997e-07	8.38981	11.9167	103.849
Variability (in %)	9.18	8.10	23.01	21.11	>100	>100	

Table 2.2: Performance of simulated annealing on several optimizations. All the optimizations start with values which are within $\pm 5\%$ of the initial values given in the first row. Some parameters (e.g., stretch damping) vary significantly across the different optimizations while others (e.g., bend) are consistent across multiple optimizations.

estimated poorly, because their values does not affect the final error value. This result is consistent with our intuition that static tests cannot be used to estimate dynamic parameters like stretch and shear damping or air drag and motivates the waving test, which exercises both the static and dynamic parameters.

Waving test. We optimized for nine parameters in the waving test: the six cloth stiffness and damping parameters and three air drag parameters (figure 2.17). As with the static test, we initialized the static parameters in this test from a coarse exhaustive search. The dynamic parameters were initialized using a random guess. We optimized on the first 50 frames of the sequence. The initial values and final values of the optimized parameters for two trials are reported in table 2.3. The final values of the parameters from the two trials differ in part because the variability of the parameters is still fairly high (figure 2.18). Different motions or longer sequences might further reduce the variability of the parameters. We choose the parameter set that minimizes the sum of the error from the two trials. For instance, in the following example of satin waving, we choose the parameters from experiment 2. This approach seems to produce plausible results with skirts and other validation experiments.

Optimization progress. Figure 2.19 shows the static shape of the simulation before and after optimization. Figure 2.20 shows the corresponding angle map comparison. These two figures show the progress of the optimization and indicate that the minimum corresponds to a visually compelling match.

Metric validation. We compare each of the four optimized angle maps from simulation (corresponding to the four fabrics) with the four angle maps computed from video. In figure 2.21, each curve shows one particular fabric compared with four simulations, corresponding to each fabric type. We see that each fabric in simulation has a minimum error when compared to its counterpart in reality. Figure 2.21 also demonstrates that our approach could be potentially useful for recognizing different types of fabrics in video.

2.5 Animation by Inverse Simulation

The parameters obtained using the inverse simulation framework can be used to create new simulations under different conditions. In this section, we present two examples of using the estimated cloth parameters to create new animations. In the first example, we simulate the fabric swatches with new input excitations (waving

Figure 2.17: Waving results for satin. The top picture in each block shows the real fabric and the bottom shows the corresponding frame from simulation.

	Linen			***Fleece***		
Parameters	Start	Exp 1	**Exp2**	Start	Exp 1	**Exp 2**
Bend	1e-3	0.001	0.0008	1e-4	1.1e-5	0.0001
Stretch	4000	2016.8	2935.3	50	82.6	89.3
Shear	215.4	167.8	465.7	215.4	255.2	296.9
Bend damping	1e-7	3.1e-7	4.7e-7	2.2e-6	1.4e-6	1.3e-6
Stretch damping	10	2.7	5.2	10	2.4	5.9
Shear damping	10	3.9	5.5	10	1.6	9.8
Linear drag	2	8.7	2.2	2	2.4	1.6
Quadratic drag	2	5.6	2.0	2	3.1	0.3
Drag degradation	2	0.4	1.3	2	4.3	1.2
E1 (Error from Exp 1)	94.2	85.9	93.1	208.8	179.6	222.2
E2 (Error from Exp 2)	115.7	113.0	100.9	233.2	230.2	180.2
E1+E2		198.9	194.0		409.8	402.4
	Satin			***Knit***		
Parameters	Start	Exp 1	**Exp 2**	Start	**Exp 1**	Exp 2
Bend	1e-5	6.4e-6	5.6e-6	1e-6	1.1e-6	1.2e-6
Stretch	50	26.4	32.4	50	69.7	12.7
Shear	50	97.7	74.2	50	37.5	60.0
Bend damping	1e-7	1.5e-6	1.2e-7	1e-7	1.0e-7	5.4e-7
Stretch damping	10	0.6	4.5	10	4.5	3.9
Shear damping	10	6.6	4.7	10	4.9	2.6
Linear drag	2	4.8	0.8	2	1.5	1.0
Quadratic drag	2	1.8	1.5	2	0.5	1.8
Drag degradation	2	0.9	0.8	2	1.2	0.3
E1 (Error from Exp 1)	124.0	106.4	114.1	230.7	208.8	246.3
E2 (Error from Exp 2)	280.8	272.8	178.6	255.1	261.8	225.3
E1+E2		379.2	292.7		470.6	471.6

Table 2.3: Parameters from two waving experiments. Line E1 shows the error for Experiment 1 with the initial conditions and after optimization. It also shows the error for experiment 2 when run with the parameters found for experiment 1 without further optimization. Similarly line E2 shows the initial error for experiment 2, the error after optimization, and the unoptimized result with those parameters on experiment 1. The parameter set from the experiment shown in bold is selected as the final estimate for each experiment because this parameter set minimizes the sum of the error from the two trials, E1 + E2. Satin has very different starting errors for the two experiments although the initial conditions are the same and the error values after optimization also differ significantly. This disparity is caused because the two experiments had significantly different waving motion.

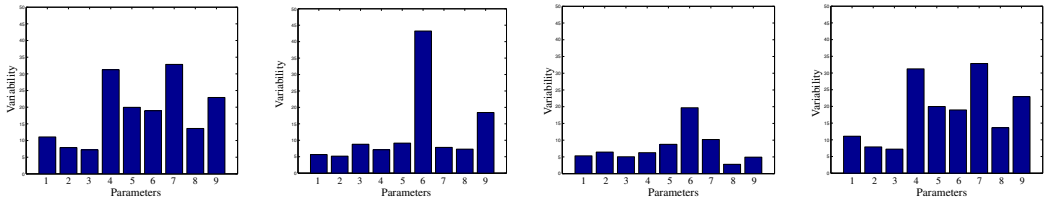


Figure 2.18: Bar charts showing the variability analysis results for the waving test. From left to right: linen, fleece, satin and knit. Legend: 1=bend, 2=stretch, 3=shear, 4=bend damping, 5=stretch damping, 6=shear damping, 7=linear drag, 8=quadratic drag, 9=drag degradation.

motions). In the second example, we created skirt simulations for a virtual character performing a hop-scotch jump. In both these examples, we validate the realism of the resulting simulations by comparing them against video of real swatches and skirts.

Animating waving fabrics. In order to validate the parameters obtained from optimization, we simulated four types of fabric swatches excited by the same input waving motion (a simple sinusoidal trajectory). This experiment allowed us to perform a side-by-side evaluation of the simulations for the different fabric material. For comparison with ground truth, we captured video sequences of the corresponding real fabrics waved using a Mitsubishi PA-10 robot arm. The robot was programmed to repeat the same sinusoidal trajectory for all the real fabrics, thereby ensuring the same input excitation for all fabrics in reality and simulation. Figure 2.22 and the accompanying videos[2] show a good visual match between corresponding frames in simulation and video.

Animating skirts. We used the optimized parameters to simulate the skipping motion of a human actor wearing a skirt (figure 2.23). Our skirt simulation used a high resolution mesh (about 5000 nodes) to generate detailed dynamic motions and wrinkles that were much more complex than the rectangular swatches of fabrics used for identification. To simulate skirt-body and skirt-skirt interactions, we extended our cloth solver to robustly handle self collision [BFA02] and collisions with the actor's body. We used data from a full body optical motion capture of the actor performing the skipping motion to drive the animated character for the cloth simulation.

We obtained a simple skirt pattern and had four matching skirts (of different ma-

[2]http://graphics.cs.cmu.edu/projects/clothparameters/

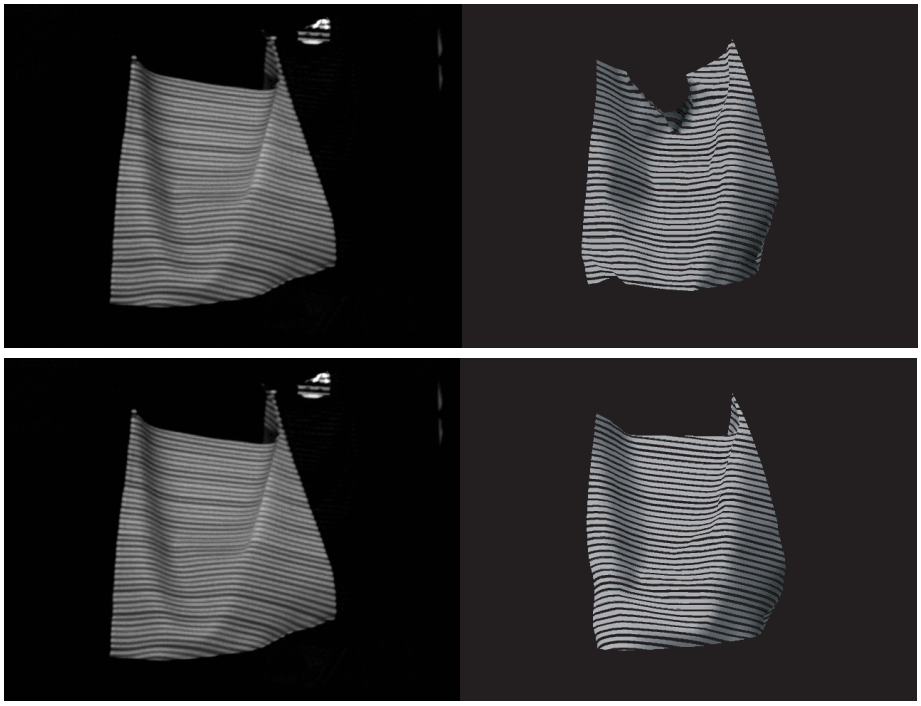


Figure 2.19: Showing the improvement in shape match after optimization. The top row compares a video frame of fleece with simulation before optimization. The bottom row shows the corresponding video/simulation pair after optimization.

terial) tailored. To create matching skirts in simulation, we require a cloth mesh that has the same basic shape as the real skirts. We generated this mesh from the silhouettes of the skirt pattern using the Maya package[3]. Maya includes the capability to generate a cloth mesh bounded by an arbitrary set of curves in the plane. These meshes are optimized to work with Maya's own Baraff/Witkin style solver. We exported these meshes for use in our own cloth solver.

We used piecewise rigid geometry for the upper and lower legs, feet and hips of the actor, which simplifies collision checking in our solver and eliminates the difficulty of properly skinning the model. To attach the skirt to the body, we selected a set of attachment points for the mesh vertices along the top of the skirt. We ensured that attachment points were well-separated around the waist by choosing them based on arc-length. We created an appropriate initial condition for the animation by simulating the skirt to wrap around the waist by allowing the skirt to fall under its own weight and simultaneously pulling the designated skirt points toward their attachment points (using a simple damped spring). This pre-simulation step ensured that

[3]http://www.alias.com

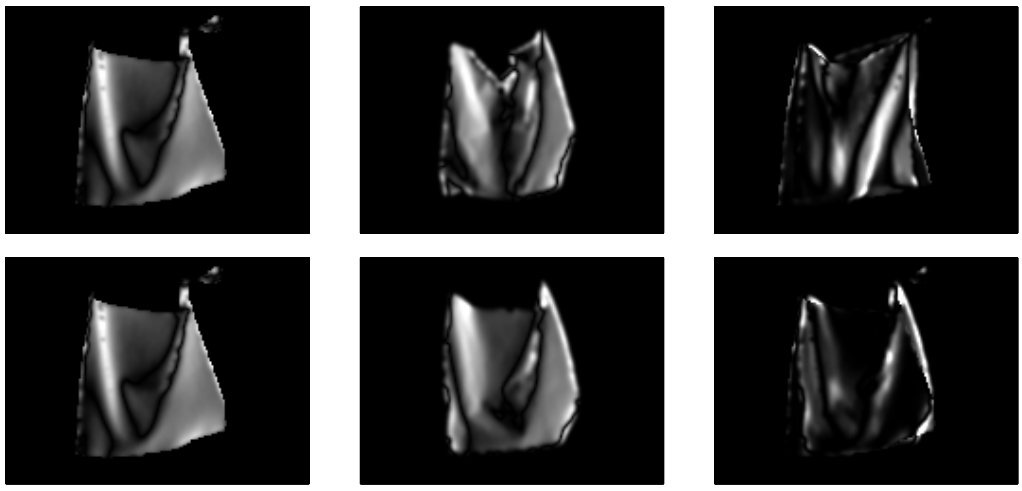


Figure 2.20: Comparison of angle maps for the shapes shown in figure 2.19 before and after optimization. Top Row (Before Optimization, from left to right): Angle map from video, angle map from simulation, angle map SSD. Bottom Row: The corresponding angle maps after optimization.

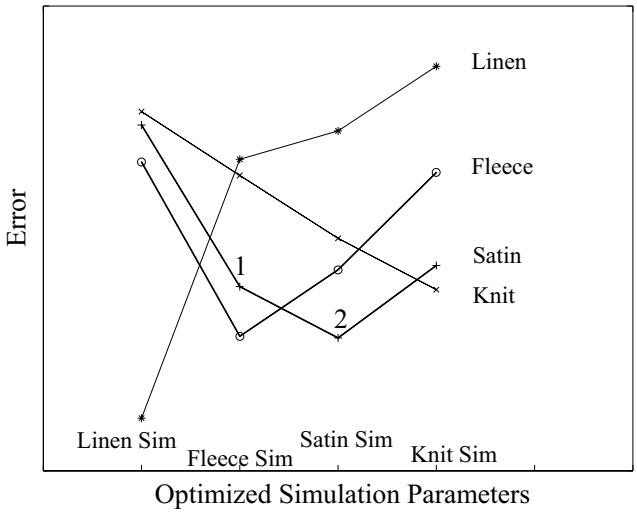


Figure 2.21: Comparing the optimized parameters in simulation for each fabric with the four real fabrics. For example, point 1 in the graph shows the error when a simulation with fleece parameters is compared with video of satin. Similarly, point 2 is the error when the satin simulation is compared with real satin. The four curves have a minimum when they are compared to their correct counterparts.

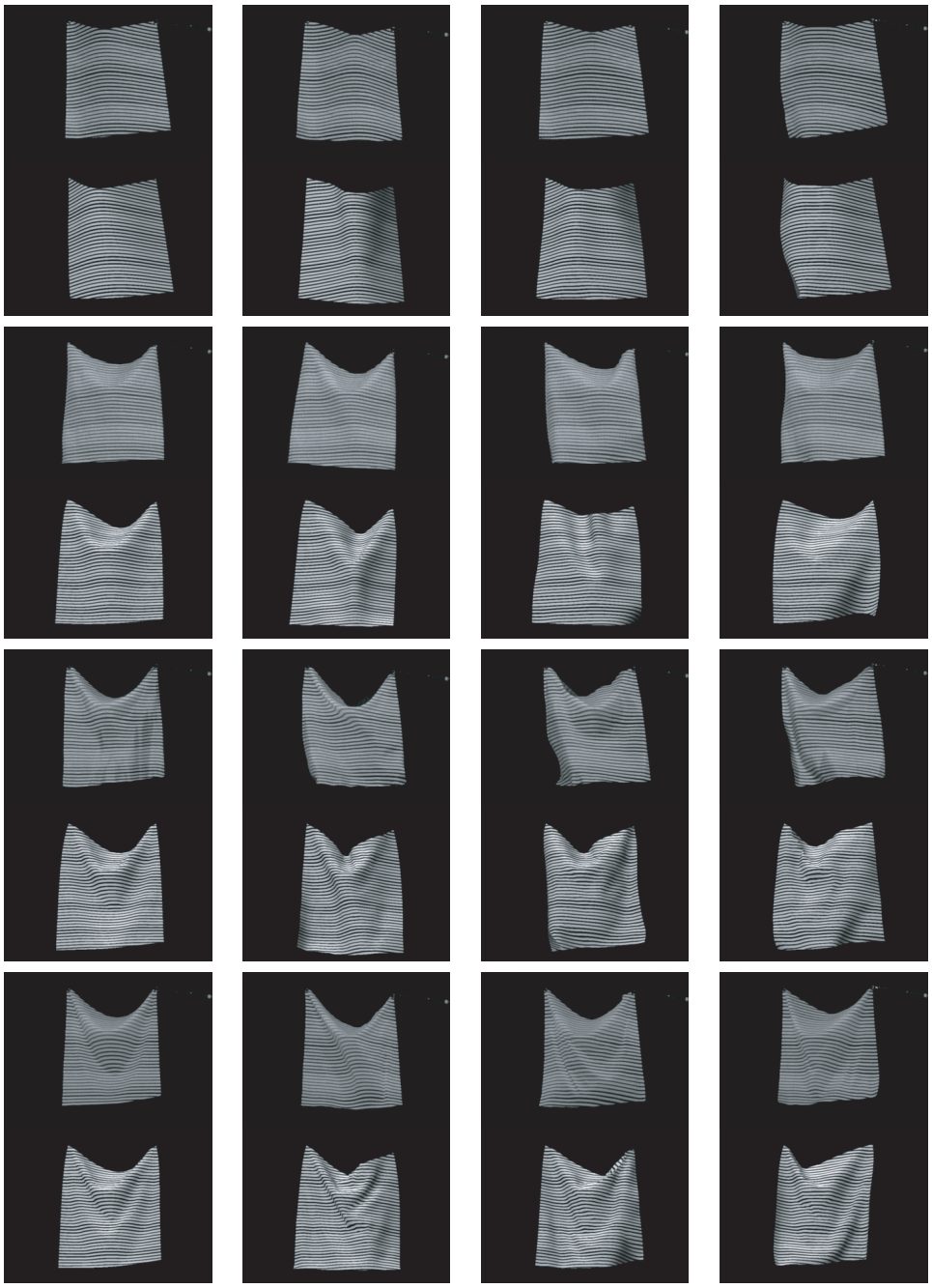


Figure 2.22: Validating the estimated parameters using the same input excitation. The top right corner of the fabric is actuated using a Mitsubishi PA-10 robot. Each row shows the match between video (top) and simulation (bottom) at four frames chosen from a 100 frame sequence. The fabrics, from top to bottom, are linen, fleece, satin and knit.

the attachment points reached their destinations and the skirt is well-settled (velocity was zero) before simulating with the motion capture data. During the simulation, skirt points along the waistline were moved based on the mocap data at the hip. The remaining points were simulated to follow the motion of the skirt points, after resolving collision and other physical forces. Stiffness was not a major problem with skirts because the simulation time steps were quite small because of collision handling.

To generate the motion data for figure 2.23[4], the actor repeated the same hopscotch motion (approximately) for the four different skirts. The videos from our web page show that the parameters obtained from our optimization approach successfully captures the static shape and dynamic properties of skirts of different materials. Because the simulation parameters were tuned to match folds, our algorithm works especially well on softer fabrics like knit and satin that exhibit a lot of static and dynamic folds. However, the linen skirt appears to be stiffer in simulation than the real skirt. We attribute it to two factors – first, the seamstress cut the fabric along the bias to stitch this skirt whereas the swatch used for estimation was cut along the normal direction (not along the bias). Although we tried to compensate for the bias effect by rotating the coordinate system of the skirt simulation (*uv* coordinates in Baraff/Witkin solver) by 45 degrees, this approximation did not work well for stiff fabrics. Second, our fold-based metric was probably not very effective for stiffer fabrics like linen, which do not have any folds in the static and waving tests. We believe, however, that our technique can easily be tuned to handle stiffer fabrics by assigning higher weight to silhouettes over folds in the metric. In conclusion, our fold-based metric is ideally suited for softer fabrics like satin or knit, which are in general harder to simulate accurately when compared to stiffer fabrics like linen.

2.6 Discussion

This chapter described the inverse simulation framework for identifying the parameters of physically based simulations of passive objects from video. Our technique has been applied to estimate the free flight parameters of rigid bodies and the material parameters for different types of cloth. We showed that the estimated parameters can be used to create realistic animations under new simulation conditions.

We first experimented with rigid body simulations, which was a simple yet illuminating example for testing our inverse simulation framework. We simplified the estimation problem by focussing only on the free flight parameters. Our optimization approach is well suited for rigid body simulations for several reasons—the free flight dynamics is easy to model accurately, experiments with real objects (of reasonable

[4]http://graphics.cs.cmu.edu/projects/clothparameters/

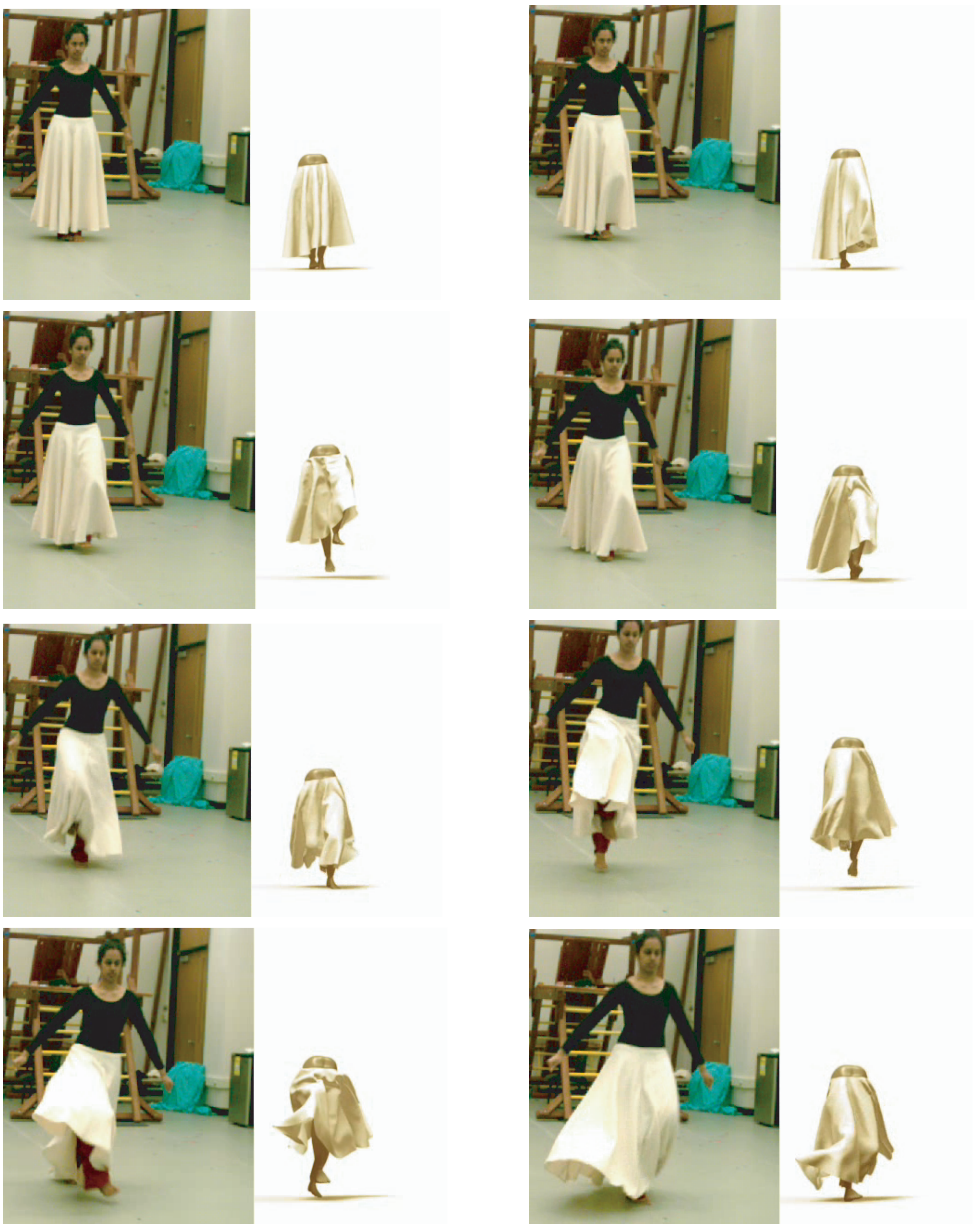


Figure 2.23: Validating the estimated parameters on a more complicated motion and garment. We show (from left to right, top to bottom) several frames of an actor skipping while wearing a fleece skirt. The corresponding frames of the skirt in simulation show that our technique captures the approximate shape and dynamics of the real skirt. These frames were equally spaced in time (0.5 seconds apart). The videos in the web page show the validation results on all four skirts.

size) are easy to perform, silhouettes from video can be extracted robustly, and analytical gradients can be computed accurately. However, tuning tumbling rigid body simulations proved to be very challenging because the angular motions are extremely sensitive to initial conditions [Sym71]. Moreover, the error space of the silhouette-based objective function includes local minima. In our experiments, we noticed that several different combinations of initial orientations and angular velocities result in silhouette sequences that look *similar* while representing quite different orientations. This observation is especially true for short sequences, which do not have enough information to uniquely identify the true parameters. Nevertheless, with longer input sequences and reasonable initial guesses, our gradient based optimization successfully identified the simulation parameters for several tumbling rigid body sequences.

Next, we experimented with cloth simulations, where our objective was to identify parameters of four real fabrics. We captured the behavior of small swatches of fabric using a set of dynamic and static tests and demonstrated that the optimizer could identify appropriate simulation parameters from those tests. These parameters produced four distinct and recognizable fabrics when applied to a more complex simulation of a skirt as it was driven by motion capture data from a human figure.

However, cloth simulations were much harder to match with real fabrics. To match a video sequence accurately using our framework, the cloth physics model as well as the collision algorithms had to be chosen carefully. The choice of integration scheme is crucial because extra damping introduced by implicit methods (e.g., Baraff and Witkin [BW98]) makes crisp folds impossible to match. Parameters must be independent of the resolution of the mesh so that they can be identified on low resolution swatches and applied to higher resolution garments. Several cloth models do not satisfy this requirement. For example, the non-linear bend model used by Choi and Ko [CK02], does not scale with mesh resolution. Progress is being made in these areas, for example, Bridson et al. [BMF03] introduced a scale-independent bend model with encouraging results.

During the course of this thesis, we experimented with a several cloth models (Choi and Ko [CK02], Baraff and Witkin [BW98]) different numerical integrators (implicit, explicit—RK4, leapfrog). We used the Baraff/Witkin model with an explicit integrator (RK4) for all our simulations. A major addition to this model was a nonlinearity in the drag model. We also modified the model so that the parameters scale correctly with the resolutions of the cloth simulation.

En route to the metric used in the experiments described here, we tried a number of other metrics: comparing the overlap of the silhouettes, the distance function between silhouette edges, and using information from internal edges marked on the fabric. Figure 2.24 shows a static optimization result that failed to capture the fabric shape and folds even though the internal edges matches well. The metric that measures folds and silhouettes, in concert with the projector for the light stripes, proved

to be a simple and effective metric that far out-performed our earlier attempts. Our angle map metric also performs better than metrics that operate on fold edges (e.g., figure 2.25). We believe that the error space of the angle map is much smoother than the error space of the fold edges because these edges depend on a discrete threshold. Although our anglemap metric also uses a threshold to compute the gradient mask from reality, this threshold just alters the weights of the anglemaps (higher weight to folds over planar regions) and hence does not affect the noise of the error space.

An effective strategy for debugging the blocks of the inverse simulation framework was to test them on synthetic input sequences. This test factors out any problems that might arise due to the model's inability to accurately mimic the real world. For example, we could simulate the input waving motion with a pre-defined set of parameters, and determine whether the metric steers the optimization towards the correct solution. Repeating the optimization with different values for the parameters and a different driving motion for simulation would also give us a good understanding of the convergence properties of the optimizer. After determining that a given metric worked on synthetic input, we would know that problem with real input were either due to inaccuracies in the model or computation of the metric on real video images.

The space of possible metrics is vast, of course, but one class of metrics that we did not experiment with are statistical metrics that compute a function of the shape of the fabric across time rather than evaluating the match on a frame-by-frame basis. The experiments with the swatches were carefully controlled to be short and have initial conditions for the simulation that matched those seen in the video. If instead, we were to optimize on more complicated garments or longer sequences, then such tight control of the initial conditions is unlikely and a statistical metric might be preferable. Such a metric might, for example, compute the average number of folds across a time sequence rather than looking for a fold to appear at a particular location on the swatch in a particular frame.

Because the main focus of this work was to achieve visual realism in simulation, we qualitatively evaluated the estimated parameters by performing a side by side comparison of our simulation results with real fabric motion. An alternate idea for evaluating our results would be to perform a *blind* test by presenting users with the four unlabelled simulations (for the four swatches), and asking users to match each simulation to its counterpart from reality. Performing the blind test for new experiments (e.g., draping) would allow us to visually evaluate the generality of the simulation parameters in new situations.

Another idea to evaluate the parameter estimates obtained from optimization is to perform statistical tests such as N-fold cross validation using data from multiple trails [Ste94]. For example, we could perform the waving test with several initial conditions to obtain a representative data set for each type of fabric. The cross val-

idation technique estimates the parameters from a portion of this data set (training set) and evaluates the estimated parameters on the remaining data set (test set). Varying the size and elements of the training and test set would allow us to compute the confidence intervals on the estimated parameters [Nel00]. Moreover, using data from different experiments in the training set (e.g., waving, static draping, dragging fabrics over different surfaces) would result in parameter estimates that would generalize to a wide range of fabric behavior.

Although the skirt is far more complex than the swatches that were used to determine the parameters, it is not as complex as many garments, for example, a form-fitting pair of pants or a tailored blazer. For more complex garments, choosing the parameters via optimization on small, flat swatches may not be sufficient because the shape of the garment is determined by darts, pleats and by the interplay of different fabrics (wool, lining, and interfacing, for example). More complex garments may require the hand design of additional tests that mimic particular behaviors or elements of the garment in isolation. Moreover, the model might need extra parameters to handle anisotropic effects, hysteresis and coupling effects (stretching along one direction causing shrinking in the other direction), all of which would need specialized tests.

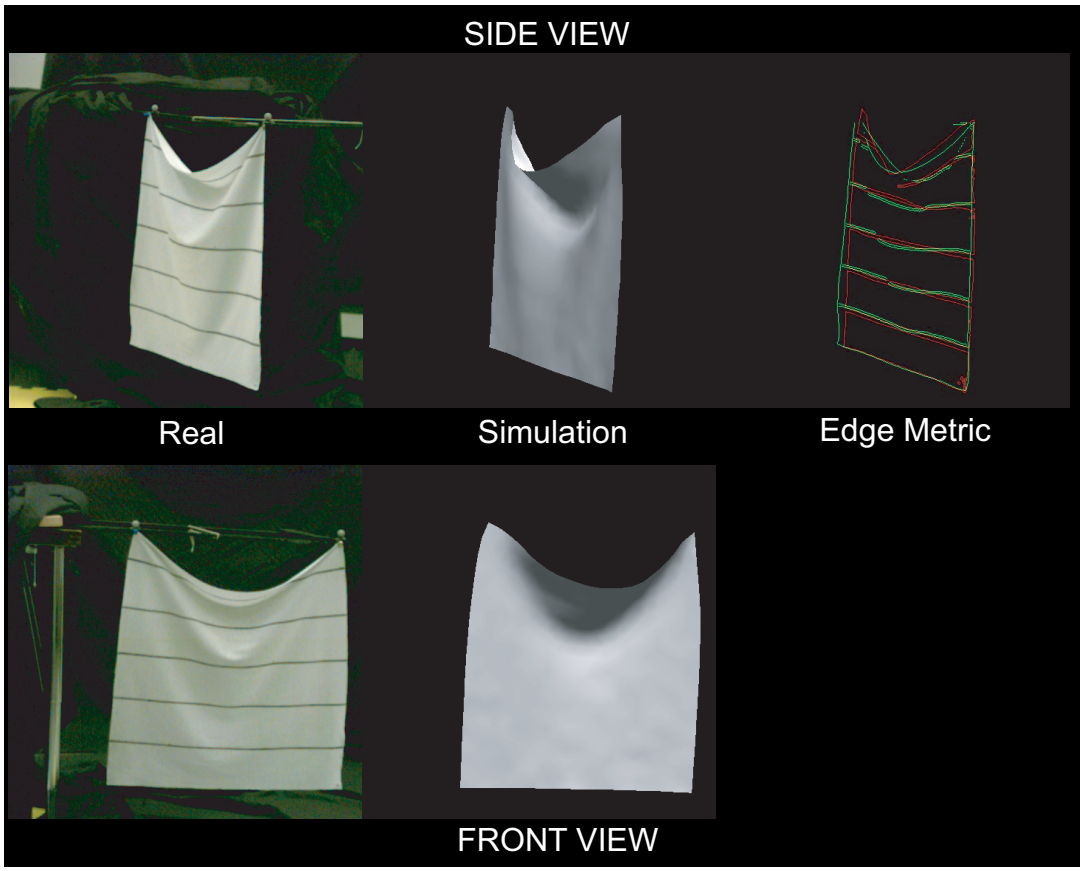


Figure 2.24: Example of a failed metric. The top row shows the optimization results for knit using a pattern of horizontal lines marked on the fabric. Only the static parameters were tuned in this experiment. The green lines (in the third column) show the edges extracted from the real fabric and the red lines are the edges extracted from simulation. (The same pattern was texture mapped in simulation, it is not shown in this figure). The edges from simulation match well with the edges from video after optimization. However, this metric does not guarantee a good visual match. The bottom row shows the *same* result from a different camera view, which highlights the mismatch in folds between the simulated and real fabric. This example illustrates that matching patterns drawn on the fabric does not guarantee a perceptual match, which motivates our fold based metric that matches folds directly.

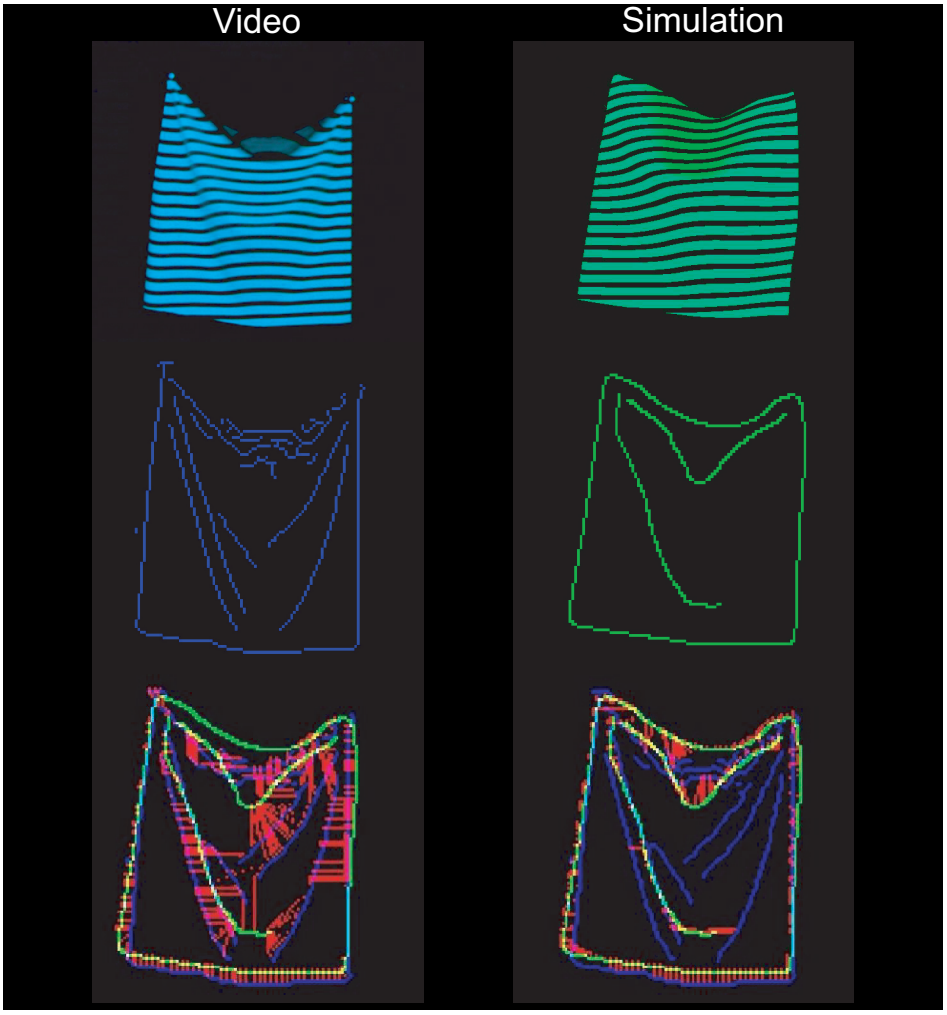


Figure 2.25: Another edge based metric that did not work well for parameter identification. In this case, these *fold* edges were obtained by thresholding the angle maps. The top column shows an image of the fabrics (in video and simulation) with a projected pattern. We computed the angle maps for both these images and applied an edge detector to obtain the fold edges in the middle row. The threshold for edge detection and the scale was chosen manually, based on the angle map from video. The last column shows the mismatch between the two fold edges, where the red lines show the distances between corresponding points in the two edge images (left image shows correspondences from edge pixels in blue to their closest green edge pixel, right image shows correspondences from green to blue) . The main problem with this metric was its sensitivity to the manually selected threshold. A threshold chosen to detect folds in stiff fabrics results in erroneous folds in softer fabrics. This dependence on threshold makes the search space very noisy and hence makes the metric unreliable.

Chapter 3

Animation by Video Editing

Creating photorealistic animations of such natural phenomena as water, fire and smoke is an important and challenging problem with many applications in computer graphics. Physics-based techniques have become a popular choice for simulating fluid flows with the advent of new algorithms and faster hardware. However, despite enormous advances in simulation and rendering techniques, synthetically generated video of natural phenomena rarely matches the complexity and beauty of real world video footage. Video captures the complex variations in the motion and texture of real fluids interacting with their environment. Thus in this chapter, we explore how to directly model the dynamics and appearance of video and use it to create new animations.

Inverse simulation techniques from chapter 2 could potentially be applied to create realistic simulations of natural phenomena using video data. However, extending the inverse simulation framework to natural phenomena is difficult because we need custom simulators for each type of phenomena. For example, models for simulating water [FF01] vary significantly from models for smoke and gases [FSJ01, Sta99], which in turn differ from models for fire [NFJ02]. Moreover, in order to match with real world measurements, physical simulators would require precise models for wind and other external forces. Although certain types of external forces (e.g., wind, vortices) can be easily parameterized [TMPS03], incorporating their parameters increases the dimensionality of the search space, making the optimization process much harder.

In this thesis, we would like to develop an general framework for animating a wide range of natural phenomena from real video footage. Given a video sequence of a real phenomena like a waterfall (or smoke or fire), our objective is to capture the *essence* of the waterfall and create new video of waterfalls that have characteristics that we can control. Here, essence refers to the appearance and dynamics of the natural phenomena in video. This chapter presents a novel solution—a flow-based

(a) Input (b) Edited

Figure 3.1: Given a video of a waterfall on the left, our objective is to synthesize a new waterfall with two extra channels shown on the right. We propose a flow-based algorithm that directly edits the input video to create the output.

video editing algorithm, that generates controllable animations of fluidic phenomena by editing the reference video clip.

To motivate this approach, consider that we have a short reference footage of a waterfall shown in figure 3.1(a). Leveraging such footage for special effects is difficult, however, because a natural scene may not match the director's intentions. He might want a waterfall with four channels, for example, figure 3.1(b). Modifying the physical setting to achieve this effect in the real world may be expensive or impossible. Our technique allows the user to modify real world footage while maintaining its natural appearance and complexity. We demonstrate this capability by modifying scenes of waterfalls, a river, flames, and smoke.

Our algorithm works in two steps, first creating an infinite sequence from the input and then using it to create infinite edited sequences. In the first step, our algorithm captures the dynamics and texture variation of the particles along user-specified flow lines in the input video. Figure 3.2(b) shows a set of user-defined flow lines (in blue) sketched over the input video (figure 3.2(a)). The algorithm generates an infinite sequence by continuously generating particles along the input flow lines. To create the edited video, the user specifies a new set of flow lines in red, shown in figure 3.2(c), and specifies the correspondence between the output (red) flow lines and the input (blue) flow lines. Our algorithm generates the edited video by continuously generating particles along the output flow lines (figure 3.2(d)).

Our approach works on natural phenomena such as waterfalls whose behavior can be characterized as motion of textured particles along well-defined flow lines. The key to our synthesis step is an insight about the continuity of flows that makes

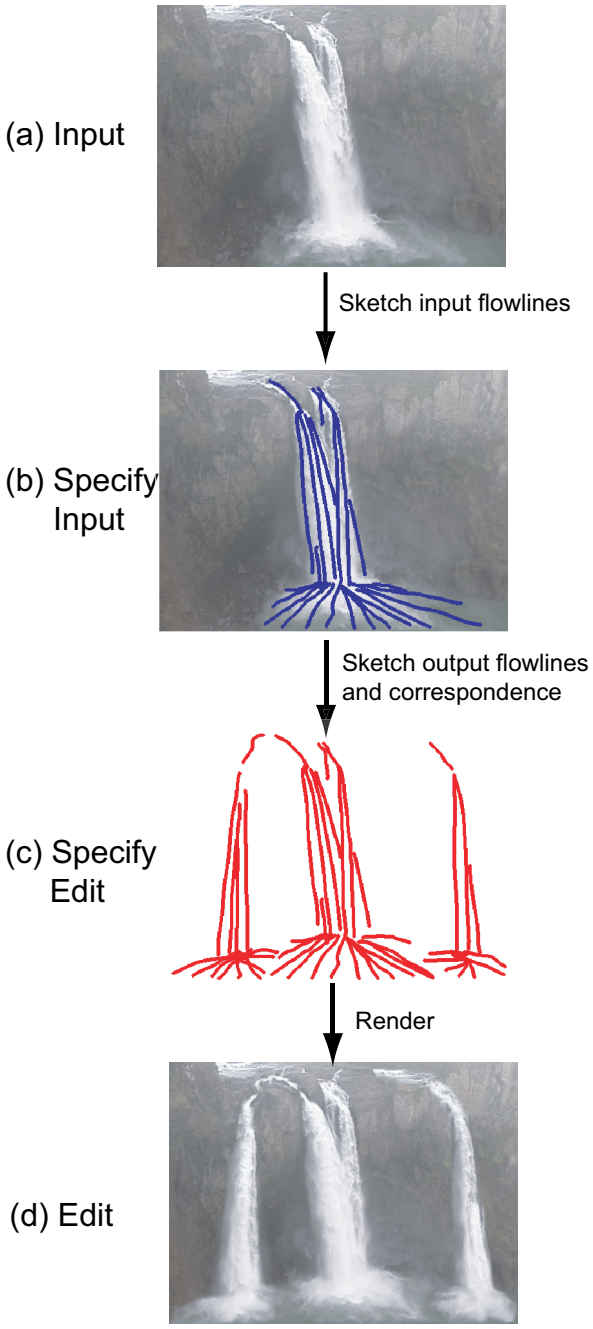


Figure 3.2: A visual flow chart of our synthesis/editing algorithm. An artist sketches the blue input flow lines, the red output flow lines, and specifies the correspondence between the two pairs of flow lines. The algorithm automatically synthesizes an infinitely long edited sequence, one frame of which is shown.

it extremely simple to create infinite looped video without apparent discontinuities. Our synthesis procedure enforces temporal continuity by forcing the particles to complete their path along the flow lines. In contrast, simply looping a short video clip to produce an infinite sequence produces a jarring artifact because all particles are interrupted simultaneously in their path along the flow line.

We have tested our algorithm on a variety of sequences such as waterfalls, streams, smoke and flames. Our technique can be applied to any phenomena that exhibits continuous flow of pixels along constant flow directions. We leverage the user to define flow lines on both the input and output video. Our framework allows the user to perform significant edits to the scene—changing the terrain of a waterfall, adding obstacles along the the flow or adding wind to smoke and flames. We show that flow lines provide an intuitive and effective user interface to edit video sequences of natural phenomenon.

The rest of this chapter is organized as follows: we begin with a discussion of related work in creating photorealistic animations of natural phenomena. Then, we present our key insight for generating infinite sequences from short video clips, and describe an implementation using a particle-based algorithm. The next section describes our editing framework, where we present the algorithm details and show the results on several sequences. We then discuss the limitations of our current framework and show a few interesting extensions that addresses some of the limitations. Finally, we conclude with a list of contributions and directions for future research.

3.1 Related Work

Creating realistic animations of fluid flow is an active area of research in computer graphics. Physics-based simulation techniques have been successfully used to simulate and control fluids (e.g., [Ree83, Sta99, TMPS03]). Earlier work in simulating fluid-like phenomenon (fire, smoke, cloud, etc.) used dynamic particle systems, that was introduced into computer graphics by Reeves [Ree83, RB85]. Individual particles within a particle system move in three-dimensional space (under the influence of user-defined forces) and also change over time in color, transparency, and size. Particle systems provide a high degree of user control over the simulation and are very intuitive to use. The rendering process can also be parallelized [Sim90] because the individual particles have independent attributes, and hence can be rendered separately. However, specifying the force fields and color attributes for complex fluid motions is very laborious. Therefore, designing particle systems to match real video footage is labor intensive.

In contrast, modern physical simulators (e.g., [Sta99, FSJ01, FF01, LF02]) produce stunning photorealistic fluid animations by using sophisticated models for dy-

namics and light transfer. These techniques model fluid motion using the Navier-Stokes equations and integrate these equations over a 2D or 3D grid. Several of these also provide indirect methods for controlling the simulation, for example using collision objects [FF01]). However, these techniques are computationally expensive and are usually tailored for a single type of natural phenomena such as smoke, water, or flame. Moreover, directly controlling these simulations is very hard due to the chaotic nature of the underlying equations of motion. A few recent papers have addressed the problem of controlling fluids for specific domains. For example, Lamorlette and Foster [LF02] present a semi-automatic framework for producing controllable simulations of fire. Treuille et al. [TMPS03, MTPS04] use a gradient-based nonlinear optimization framework to control the simulations of smoke and water. Fattal et al. [FL04] present a closed-form solution for controlling smoke simulations by allowing the user to specify high-level directions. Once again, these techniques are specifically designed for controlling a particular simulation (e.g., smoke) and may not generalize to other types of natural phenomena. In contrast, our technique works on a wide range of natural phenomena characterized by continuous flow along well defined directions. Our approach that models video using particles automatically achieves photorealism, while allowing a wide range of interesting edits by manipulating the particle trajectories in video.

Recently, several researchers have attempted to model the textured motion of fluidic phenomena in video and synthesize new (and typically longer) image sequences. Wang and Zhu [WZ02] present a generative model for analysis and synthesis of natural phenomenon from video. Their algorithm uses a second order Markov chain to automatically model the motion of texture particles in video. They use the EM algorithm to infer the properties of the Markov chain. Doretto et al. [DCSW03] use Auto-Regressive filters to model and edit the complex motion of fluids in video. Non-parametric models for texture synthesis have been applied to create 3D temporal textures (e.g., [WL00]). The graph cuts algorithm combines volumes of pixels along minimum error seams to create new sequences that are longer than the original video [KSE$^+$03]. Another class of synthesis techniques creates longer videos from short clips by concatenating appropriately chosen subsequences, and allows for high level control over moving objects in video (e.g., [SSSE00],[SE02]). Most of these techniques focus on creating long video sequences from short clips with little or no editing capabilities. In this thesis, we are interested in performing substantial edits to the video, resulting in infinitely long edited sequences. However, editing has been well studied for images and textures. Image Analogies [HJO$^+$01] uses a non-parametric learning technique to edit images using the *texture by numbers* framework. The image quilting algorithm [EF01] and the graph cuts algorithm present several examples of texture editing. Our algorithm provides a flow-based technique for editing video.

Our synthesis approach is very simple and produces results that are comparable

to previous video synthesis algorithms on sequences with continuous flow. Additionally, our technique allows an artist to specify edits intuitively by sketching input and desired flow lines on top of an image. The next section describes the central idea of the synthesis step that allows us to generate infinitely long sequences with no visual discontinuities.

3.2 Synthesizing Infinite Sequences

In this section, we present a particle-based framework for modelling video sequences of such natural phenomenon as waterfalls, streams, smoke and fire. This model allows us to synthesize new sequences by varying the parameters of the particle system. Before discussing the details of our model, we first present a short tutorial on particle systems and their applications in computer graphics.

3.2.1 A Brief Tutorial On Particle Systems

Particle systems have traditionally been used in computer graphics for modelling fuzzy objects such as fire, clouds, smoke or water [Ree83]. A particle system is defined as a collection of particles whose motion, color and other attributes are controlled by user scripted rules (or forces). The particle attributes—positions, velocities, colors, transparencies, size, age, etc.—are typically stochastic in nature. Each particle goes through three distinct phases in its life cycle: generation, dynamics, and death. Particles in the system are stochastically generated from user specified emitters, and are destroyed after their lifetime exceeds a pre-defined threshold or if they go out of the view area.

Particle Dynamics. The motion of particles is governed by user-specified velocity or force fields. In a first order particle system, the user defines a velocity field over a grid (workspace) and the particles are advected through the velocity field. These velocity fields are typically generated procedurally (e.g., Flow Tiles [Che04]), or from physics (e.g., Stable Fluids [Sta99]). In a second order particle system, the user specifies a force field on the workspace and the particles move in response to these forces [Ree83, RB85]. Gravity, wind and collisions are typical ingredients of hand crafted force fields. In some cases, the particle dynamics can be purely procedural, for example, when particle position depends on positions and velocities of other particles, as seen in flocking behavior [Rey87]. Additionally, the attributes of each of the particles may vary over time. For example, the color of a particle in an explosion may get darker as it gets further from the center of the explosion, indicating

that it is cooling off. In general, each of the particle attributes can be specified by a parametric equation with time as the parameter.

Particle Rendering. Particles are typically represented as blobs, streaks, clouds, tubes or sprites. Although the specific details of rendering each particle type differs, most rendering algorithms have a few common features in order to scale to large number (tens of thousands) of particles. For example, most algorithms render each particle independently and later combine the (projected) colors in screen space. This simplification allows particles to be rendered in parallel and eliminates the need for hidden surface removal algorithms. For increased realism, sprite-based particle renderers use texture patches from real images [PDI01] for rendering stochastic phenomena like smoke and fire, instead of using blobs or streaks. These textures, either hand drawn or manually chosen from images, are varied procedurally as the particle evolves. Our algorithm, in contrast, automatically extracts these time-varying textures from video.

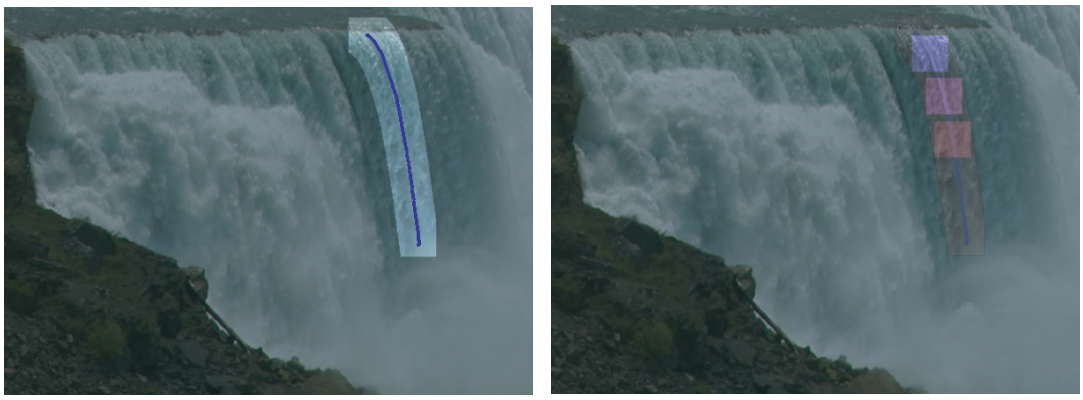
Particle systems are very intuitive to use and provide a high degree of user control. However, designing the dynamics and appearance of particle systems to match with real fluids is very laborious and often impossible. Our algorithm addresses these limitations by automatically estimating the particle dynamics and its time varying texture attributes from video. We evaluate the quality of our particle model by creating infinitely long simulated sequences and comparing them side by side with the corresponding input video.

3.2.2 Modelling Video Using Particles

Our synthesis algorithm uses particle systems to model video of natural phenomena. Rather than hand crafting the dynamics and appearance of particles, our algorithm derives these attributes from video using a combination of user interaction and motion estimation techniques.

Our technique is ideally suited for modelling phenomena that have time-varying appearance but roughly *stationary* temporal dynamics [DCSW03]. For example, the velocity at a single fixed point on a waterfall is roughly constant over time. Consequently, these phenomena can be described in terms of particles moving along fixed flow lines (possibly curved) that particles follow from the point at which they enter the image to the point where they leave or become invisible due to occlusions. The flow lines, which are sketched by a user, define the particle trajectories in the input video (figure 3.3). The particle attributes—velocity and texture (a rectangular patch of pixels, figure 3.3(b))—changes as the particle moves along the flow line.

Our video texture synthesis technique produces seamless, infinite sequences by



(a) Flow line (b) Particles

Figure 3.3: (a) A flow line sketched on Niagara falls. (b) Particles moving along the flow line.

modelling the motion and texture of particles along user-specified flow lines. We first describe the way in which the particles move in video, and then describe how they are rendered using texture patches.

Particle Dynamics. To begin, consider the case of a single flow line in the image, as shown in figure 3.4(a). Any particle that begins at the start of the flow line d_1 will pass through a series of positions $d_1, d_2, \ldots, d_n$ during its trajectory. The particle's velocity along the flow line may be time varying; thus the positions d_i need not be evenly spaced. The particle's texture may vary as it moves along the flow line (figure 3.4(b,c,d)).

We represent the temporal evolution of particles along this flow line as follows. Define as a matrix $M(d,t) = (p,f)$, where p refers to a specific particle, and f specifies the frame in the input sequence where that particle appears at d. Figure 3.5(a) plots $M(d,t)$ for the input sequence, where the number in each cell corresponds to f and the color to p. The first column of this matrix shows the particles and their positions on the flow line in frame 1 of the input (hence these entries have $f = 1$). The red numbers, for example, show the path of a single particle during the course of the input sequence.

When the sequence is looped, there is a discontinuity (vertical black line) between frames 5 and 6 because particles abruptly move to different locations or disappear altogether. This discontinuity appears simultaneously for every pixel in the image making it a very noticeable artifact.

We can reduce this discontinuity by a simple change to the entries of $M(d,t)$. The modified matrix is obtained by repeating the first K diagonals of the original

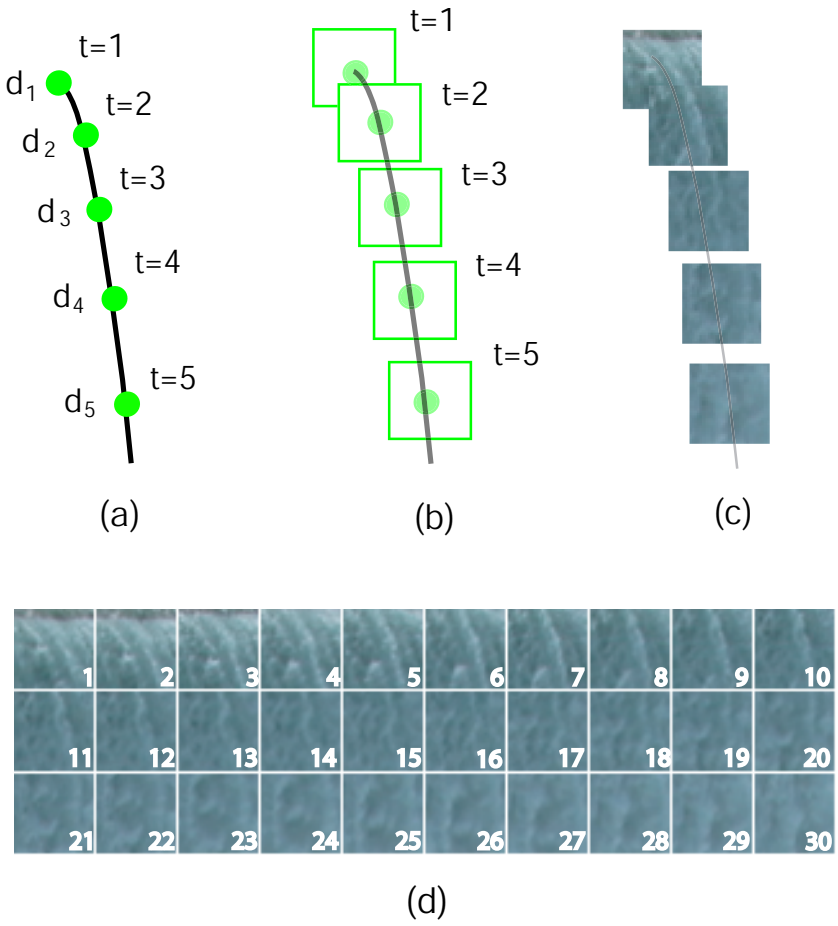
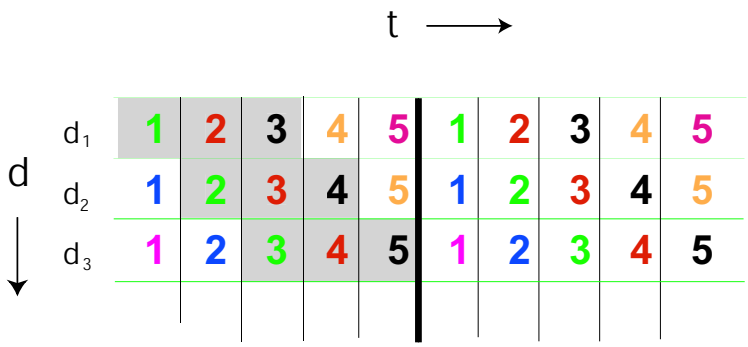
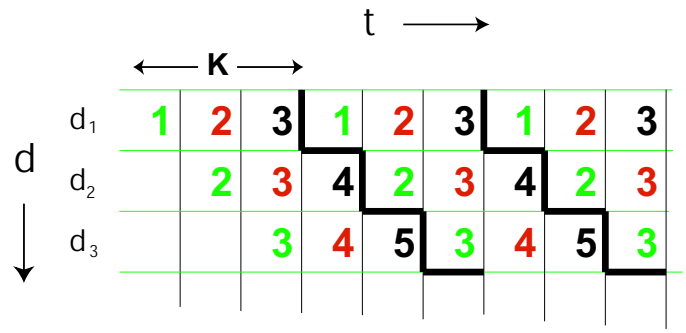


Figure 3.4: A particle-based representation for video. (a) A particle moving along its flow line. (b) Particle texture moving along the same flow line over time. (c) Texture variation of a *real* particle from the Niagara sequence. For clarity, we show the particle texture every 6th frame as it moves along the flow line. The particle velocity increases as the particle moves downward as would be expected due to gravity. (d) A filmstrip (left-right, top-bottom) showing the particle texture for each frame as the particle travels downward along the flow line. The texture of two adjacent cells is similar, which facilitates tracking. However, the texture varies significantly between the beginning and end of the flow line.



(a) Simple loop



(b) Our algorithm

Figure 3.5: (a) Plot of $M(d,t)$ showing the particles on a single flow line over time in the input sequence. Numbers specify frames, and colors specify particles. Note that there is a discontinuity for each particle along the flow line between time $t = 5$ and $t = 6$ when the input sequence is looped. (b) In contrast, our synthesis algorithm maintains temporal continuity along flow lines. Although there is a discontinuity along the diagonal stepped line, it is less noticeable because all particles complete their paths along the flow line.

Figure 3.6: Computing the particle texture and velocity from video. (a) shows a particle along the flow line. To compute its position in the next frame, we search along the flow line to find the most correlated block in the next frame. The white region in (b) shows the search window along the flow line. The best matching block is shown in (c).

matrix. For example, the shaded entries of figure 3.5(a) are repeated to produced figure 3.5(b). While the modified matrix is composed of a subset of the same entries as in (a), it enforces temporal continuity because each particle completes its full path along the flow line. The vertical discontinuity in figure 3.5(a) is replaced with a ladder-shaped discontinuity pattern in figure 3.5(b), with one spatial discontinuity in each column, corresponding to an abutting pair of patches in the output sequence that were not adjacent in the input sequence. These spatial discontinuities are difficult to detect, however, because the abutting pair of patches move together in the output sequence. This simple procedure lets us create a matrix of arbitrary width that preserves temporal continuity for all particles along the flow lines.

We implement the procedure in figure 3.5(b) by sequentially generating particles at d_1 from a subset of K input particles (shaded in figure 3.5(a)), and moving them along the flow line over time. First, an artist sketches a dense set of flow lines over the input video. Then, we compute and store the particle velocities and textures along these flow lines. The particle velocities are computed as follows. For a given particle location in the current frame, we search *along the flow line* for a corresponding location in the next frame that best matches the texture of the current particle (figure 3.6). This procedure performs a constrained form of optical flow along the flow line [TV98]. The rectangular patch of pixels around the *best match* location becomes the particle texture for the next frame. We repeat this procedure to compute the attributes for particles starting at different frames in the input video (for all flow lines). For example, figure 3.7 shows the texture attributes for four particles that start from the top of the flow line at different frames of the input video. All the synthesis parameters, including the number and spacing of flow lines, and the size of the particle

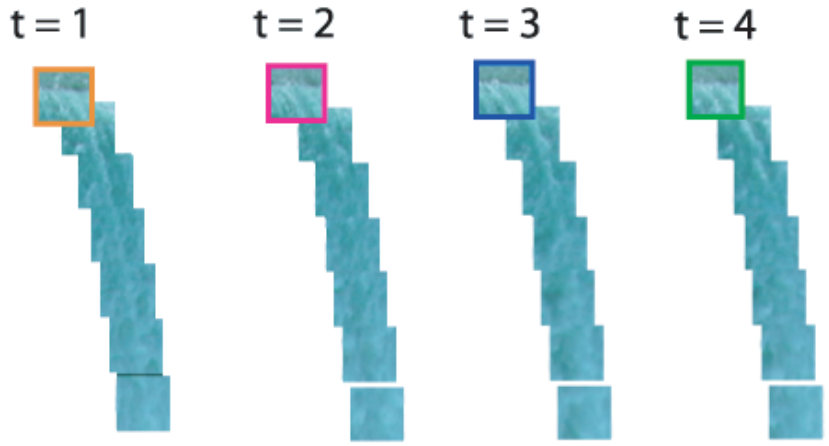


Figure 3.7: Showing the temporal history of four particles that start at different frames of the input video. The variation of the particle attributes along the flow line for all four particles are similar (but not identical). This sequence is a good example of a stationary process.

texture (rectangular region) are controlled by the artist.

The user plays a crucial role in the modelling step by specifying the input flow lines. These flow lines capture the user's intuition about the dynamics of particles in video. Such user interaction enables our algorithm to model (and synthesize) complex sequences like Niagara falls. Modelling such sequences using existing optical flow algorithms is challenging, especially at interfaces (e.g., water and mist).

Particle Rendering. Once the particle positions along the flow lines in each new image are determined, the system blends their textures to produce the rendered result. Recall that we store the particle texture as it evolves along the flow line in the input sequence. Rendering simply involves drawing the patch of texture around each particle, and using feathering to blend the overlapping regions from different patches. The feathering method assigns a weight to each pixel in the overlapping region that is inversely proportional to its distance to the edges of the overlapping region.

Generality. Our approach works best for input sequences that have stationary temporal dynamics. Waterfalls and rivers are examples of continuous phenomena where the velocity (magnitude and direction) at any fixed point in the image is roughly

constant over time. Phenomena like flames rising upward or smoke from a chimney (without wind) can also be modelled with a simple extension to our method for defining particle dynamics. For such input sequences, we allow the velocity magnitude at each fixed point on a flow line to vary over time. Unlike figure 3.4(a), particles that begin at the start of the flow line d_1 at different times t will pass through a *different* series of positions $d_1(t), d_2(t+1), \ldots, d_n(t+n-1)$ along the *same* trajectory. This modified procedure uses the steps for dynamics and rendering described in the previous paragraphs, however, particles with different frame indices f may now have different velocity magnitudes along the flow line. Although this extension is an approximation to input video with non-stationary dynamics, it seems to work well in practice for video of flames and smoke where the overall direction of flow remains almost constant.

3.2.3 Synthesis Results

We used our algorithm to synthesize long image sequences for natural phenomena such as waterfalls, rivers, smoke and flames[1]. For most of these examples, we create a long temporal sequence from a short clip of video (approximately 50 frames long). We used texture patches of 24×24 pixels for all sequences except flame, where the patch size was 100×100 pixels. Our rendering speeds in Matlab (frame size 320×240) were roughly 20 seconds per frame on a 2GHz Pentium PC with 512M RAM.

Niagara Falls: This sequence has several distinct texture regions, with clear water on the top, foamy water on the left and spray on the bottom (figure 3.8). Each region also exhibits a significant variation of texture along the flow lines. Our algorithm does not model the change in transparency of the foam rising from the bottom right portion of the image (where flow lines from these two regions overlap). This effect causes some visual artifacts at the interface between water and foam in the synthesized infinite sequences.

Waterfall: We sketch a set of vertical flow lines to capture the water moving down and another set of radial flow lines to model the collisions at the bottom of the fall (figure 3.9). The interface between the waterfall and the lake at the bottom causes interesting foam patterns, which are captured in the temporally extended sequences.

Stream: Here, the flow lines are mostly horizontal, except for regions where the water rises due to collisions with the underlying rock bed (figure 3.10).

Smoke: This example shows our algorithm applied to a sequence that exhibits intermittent motion along flow lines (figure 3.11). However, the overall direction of the smoke motion almost remains constant throughout the input sequence, which

[1] http://graphics.cs.cmu.edu/projects/flow/

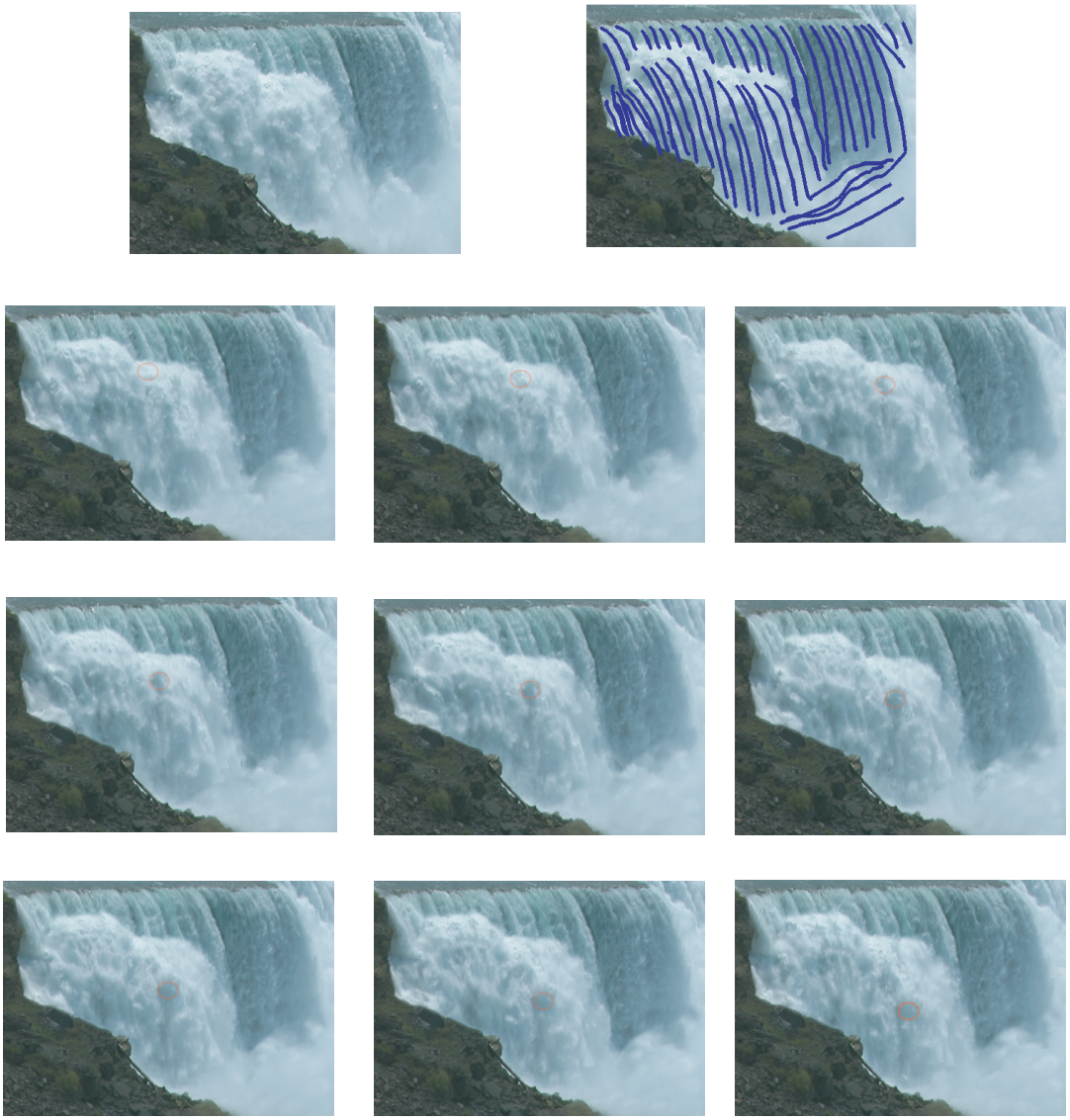


Figure 3.8: The synthesis results for Niagara falls. The first column shows one frame of the input sequence and the user specified flow lines superimposed on the input. The last three rows show nine consecutive frames (scan-line order) from a long synthesized sequence. The red circle drawn on the result shows the motion of the particles in the synthesized video.

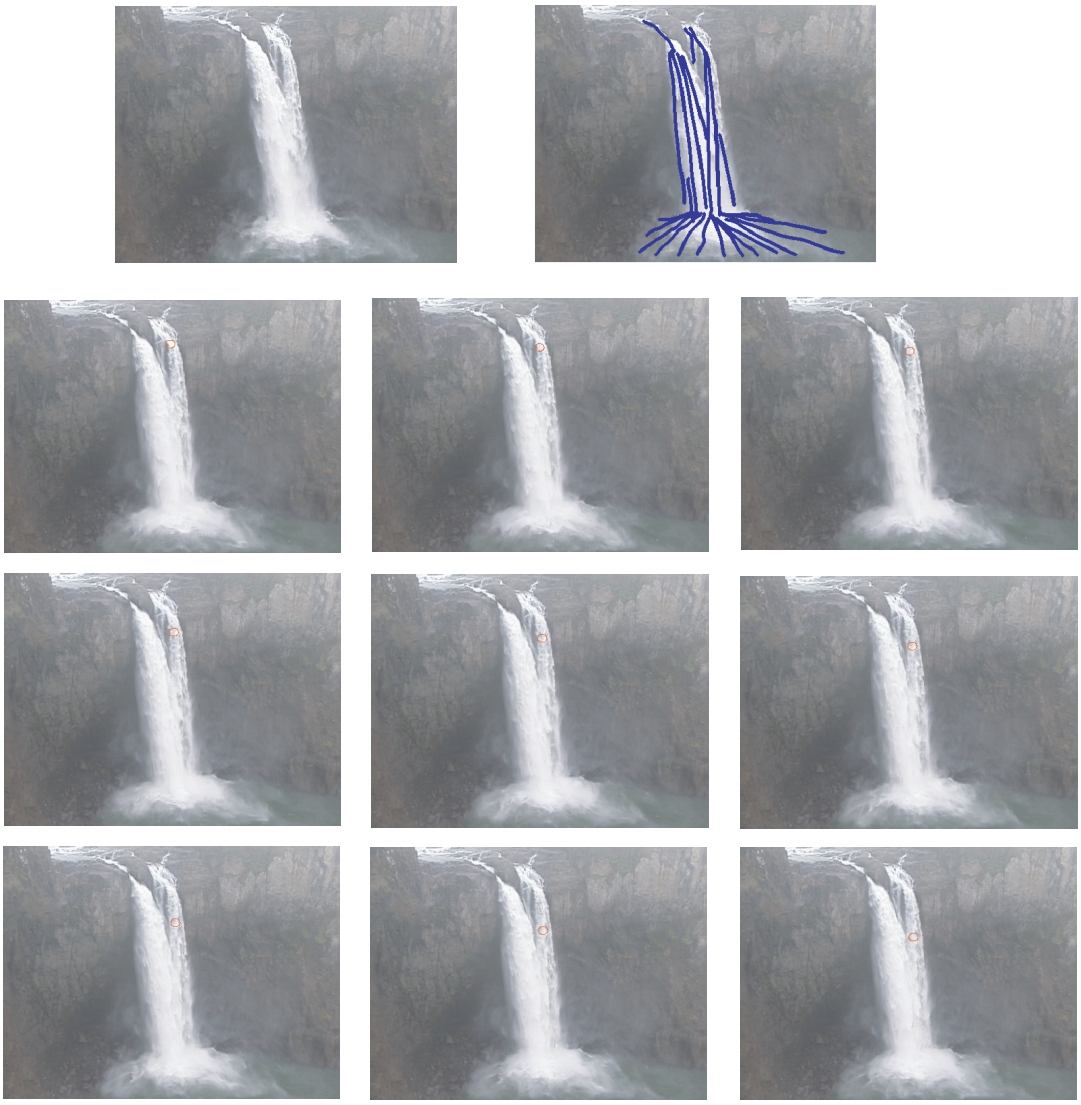


Figure 3.9: Synthesis results for a waterfall, with the input shown on the first row and nine consecutive frames of the output shown in the next three rows.

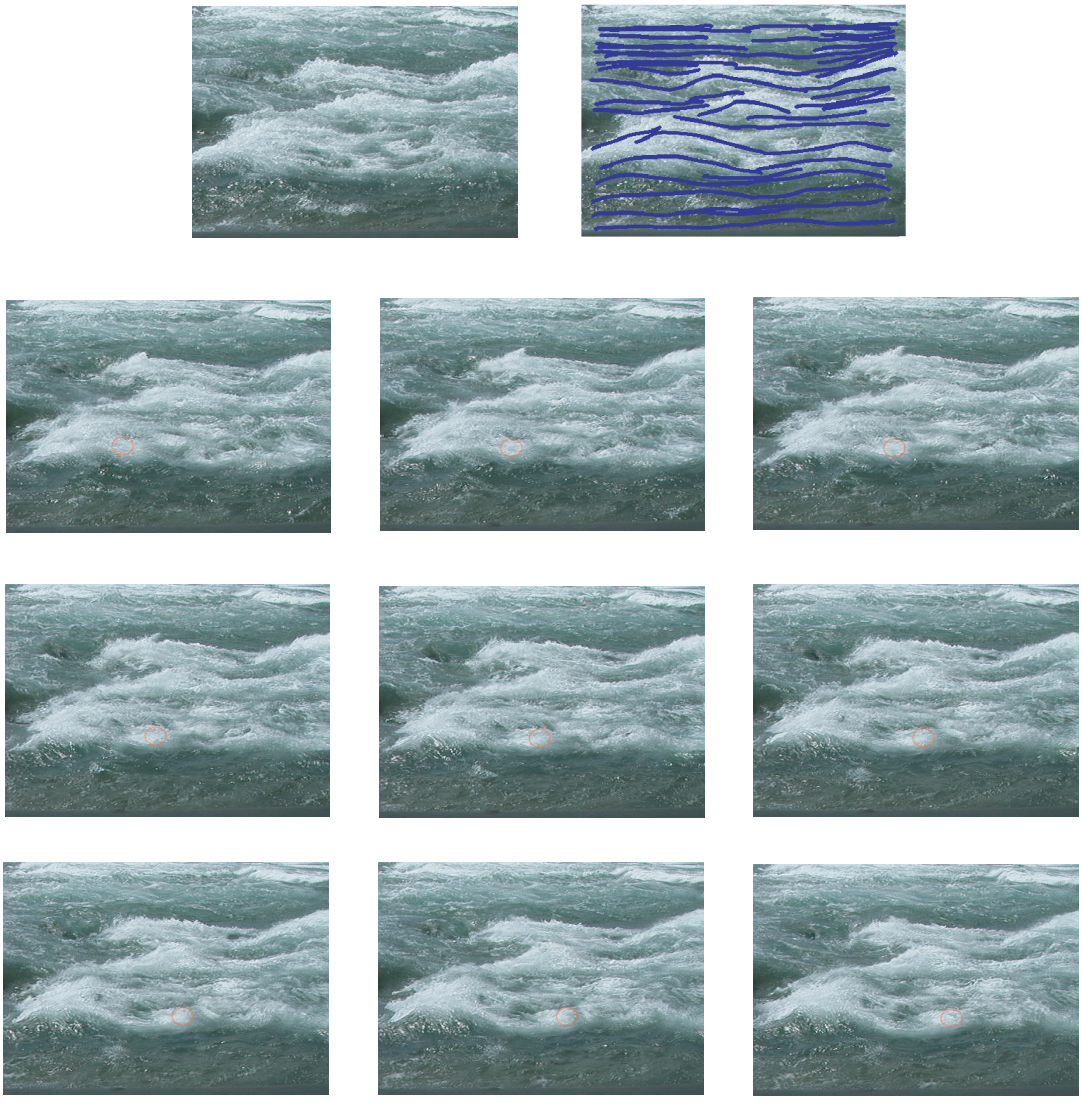


Figure 3.10: Synthesis results for a stream sequence, with the input shown on the first row and nine consecutive frames of the output shown in the next three rows.

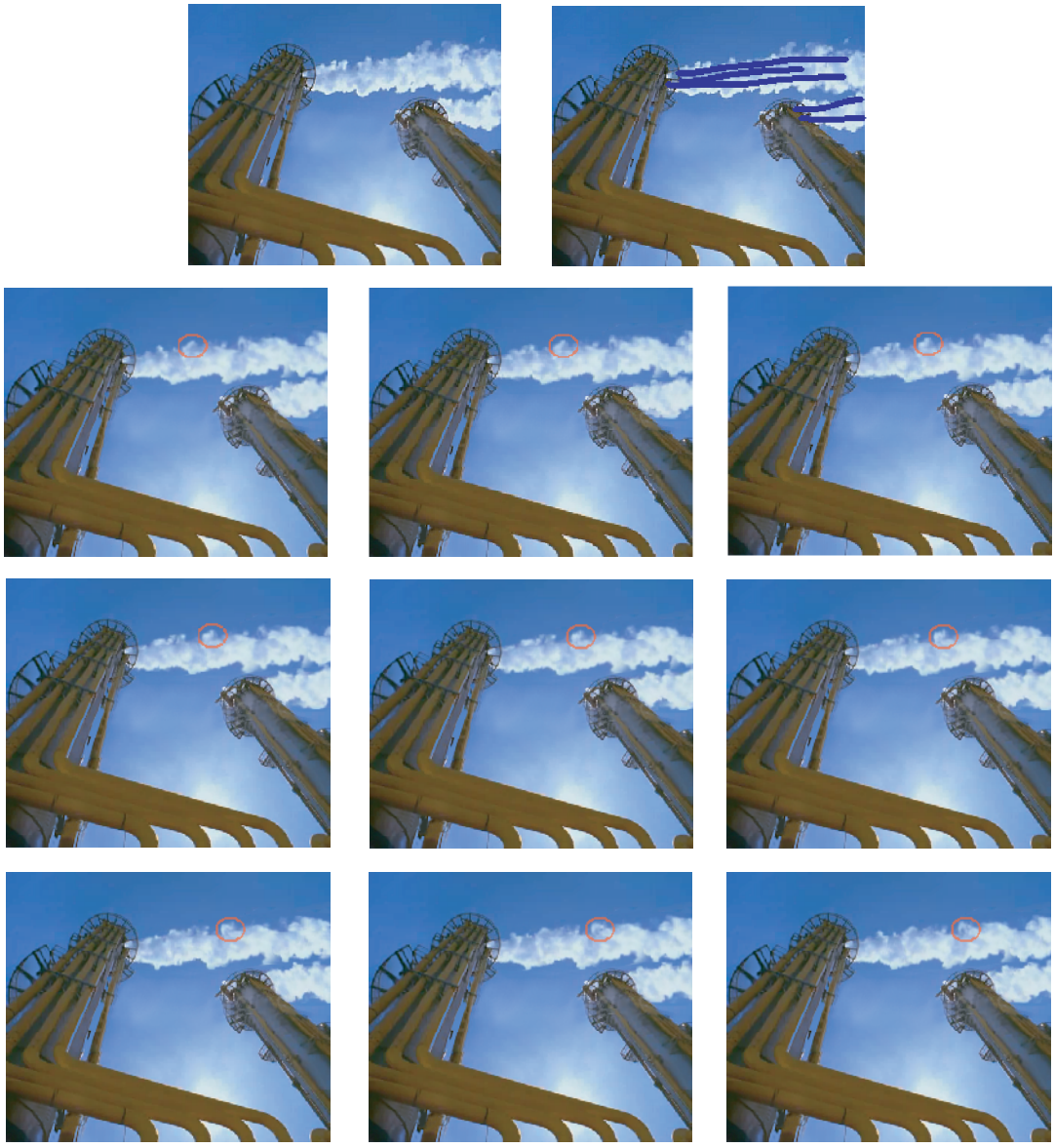


Figure 3.11: Synthesis results for a smoke sequence. The input is shown on the first row and nine consecutive frames of the output are shown in the next three rows.

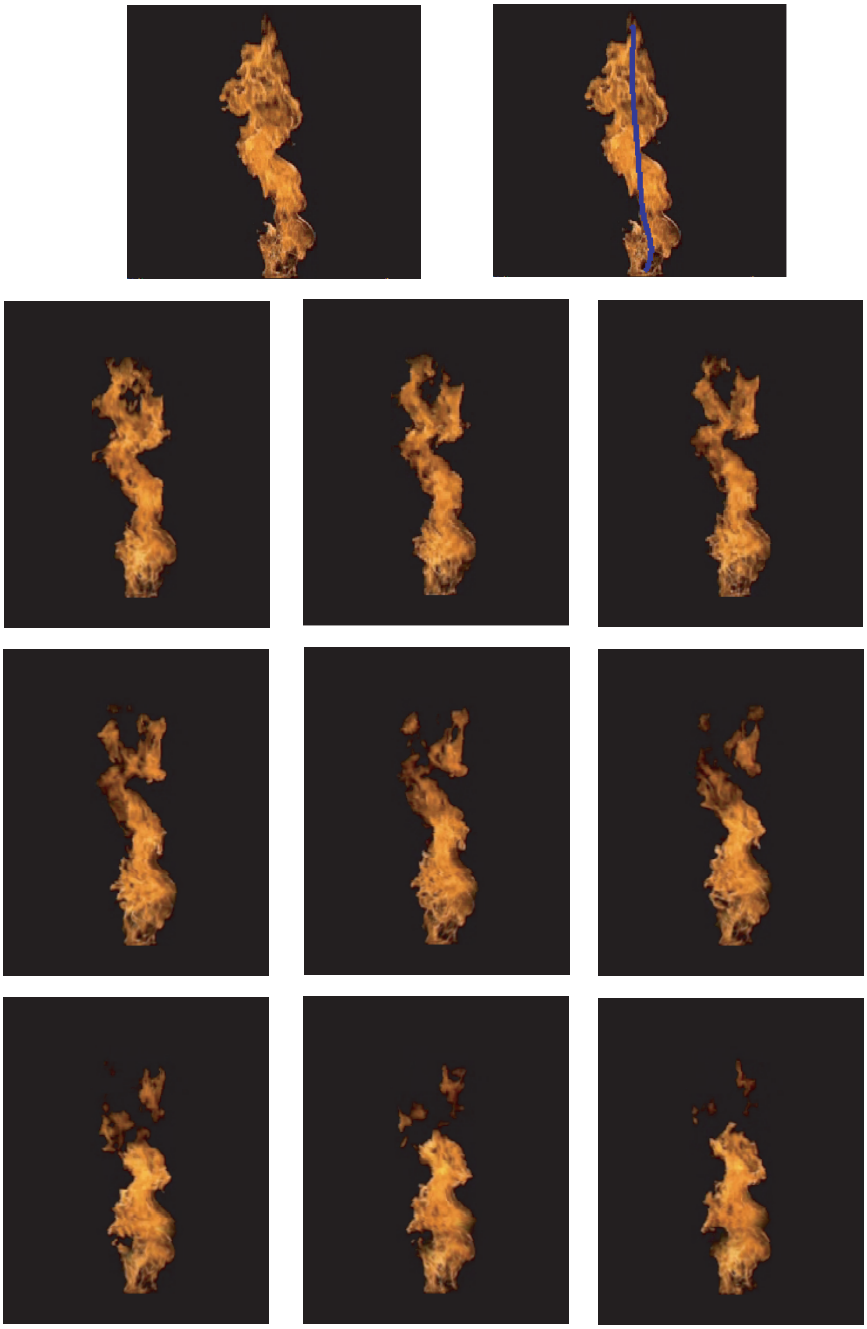


Figure 3.12: Synthesis results for a flame sequence. The input is shown on the first row and nine consecutive frames of the output are shown in the next three rows.

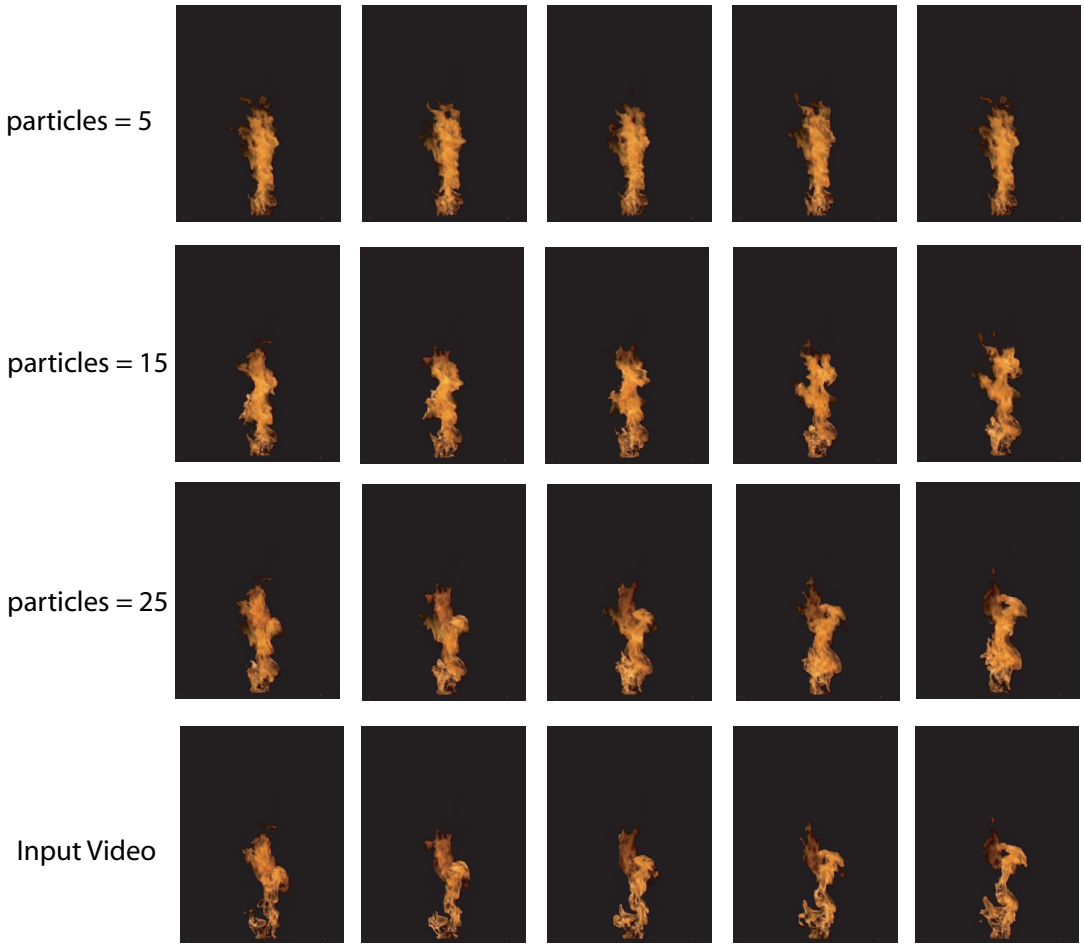


Figure 3.13: The effect of varying the number of particles used in the synthesis step. The first three rows show a few frames of the synthesized flame using different number of input particles, and the last row shows the input video. The texture patterns in the first row shows very little variation over time. However, as we increase the number of particles used in the synthesis, the synthesized flame begins to capture the dynamics and appearance properties of the input.

makes the static flow line assumption hold. Particles that start at different frames have different speeds (magnitude) along the flow line.

Flame: We model the flame using particles moving on a single vertical flow line (figure 3.12). Similar to the smoke example, the flame direction throughout the sequence is almost constant (vertical, in this example). However, the speed (magnitude) and texture of different particles (that start at different frames of the input) vary quite a lot for this example. This difference results in interesting artifacts (figure 3.13) when we vary the number of particles used in the synthesis. For example, using fewer particles causes texture repetition along the flow line, because our synthesis algorithm uses the (small number of) particles repeatedly to create the output. Using more particles reduces this effect and makes the synthesized result resemble the input flame.

3.3 Editing

The synthesis algorithm described in the previous section can be extended to support editing by synthesizing texture over a new set of flow lines. Our framework requires the user to perform the following two steps (figure 3.14):

- Sketch new (output) flow lines (figure 3.14(a)). This step specifies the direction of motion of particles in the resulting edit. The set of output flow lines are typically much sparser than the input.
- Specify correspondence between the input and output flow lines (figure 3.14(b)). This step specifies the texture along the output flow lines.

These two steps allow the user to creatively specify the desired edit. Our algorithm first interpolates the sketched flow lines (red) to create a dense set of flow lines (green lines in figure 3.14). To generate the final edited video, our algorithm performs the following steps:

- Compute the correspondence between the interpolated flow lines and the input flow lines. This assignment is achieved automatically using dynamic programming.
- Render output flow lines by warping the synthesized input texture to the output.

Correspondences for interpolated flow lines: We use dynamic programming to find a globally optimal set of correspondences between the interpolated flow lines and the input flow lines. Let Y be the dense set of input flow lines and X be the set of edited flow lines. For every pair $(x-1, x)$ of neighboring edited splines, we compute

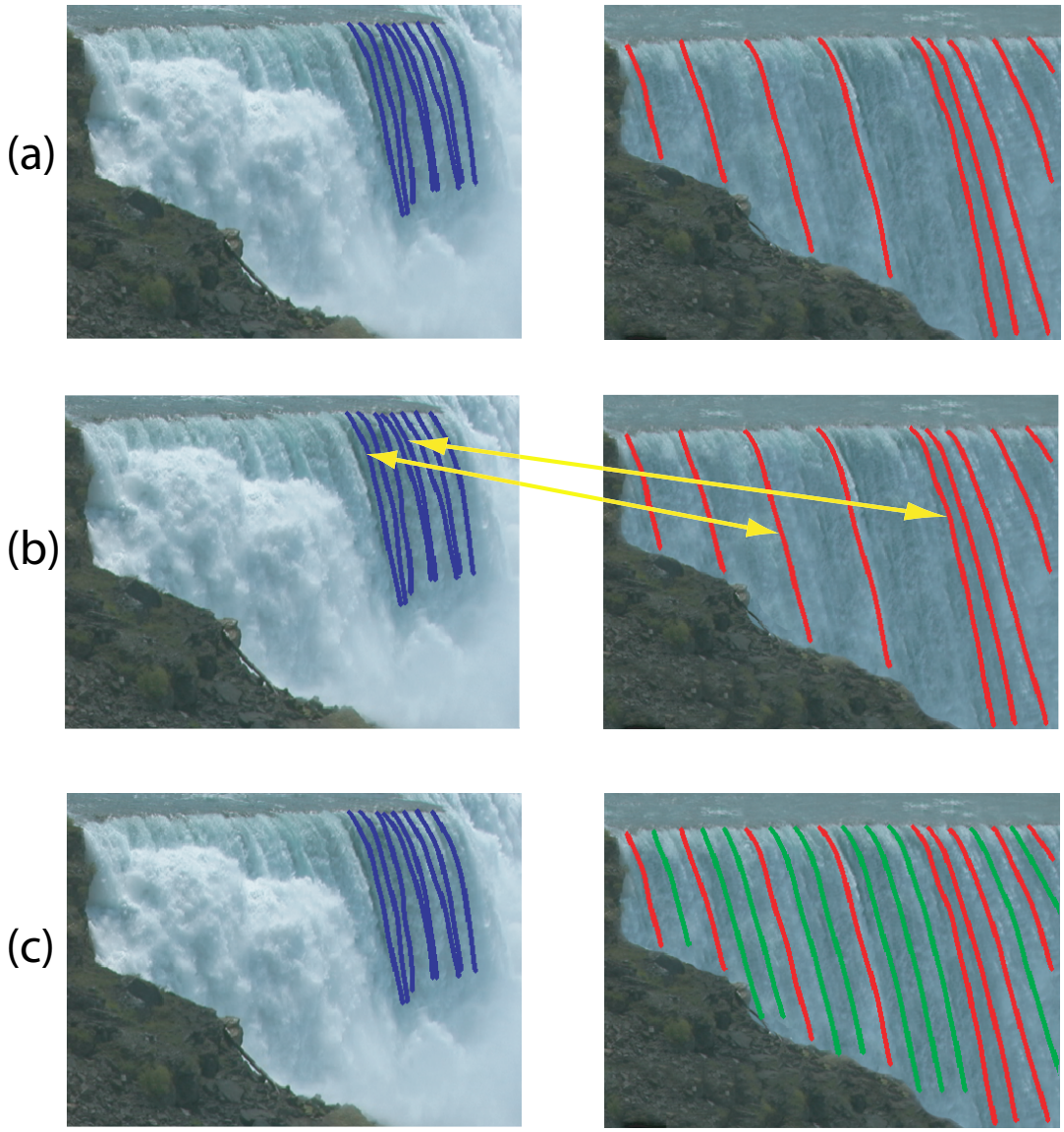


Figure 3.14: The left column shows an input waterfall sequence and the right column shows an edited waterfall created using our algorithm. (a) The animator draws the flow lines in the input image, shown in blue. To specify an edit, she draws a sparse set of flow lines in red. Then, she specifies correspondences between all edited flow lines and the input flow lines (two of these are shown in (b)). The system generates a set of interpolated flow lines (thick green lines in (c)) and computes a correspondence with the input flow lines automatically using dynamic programming.

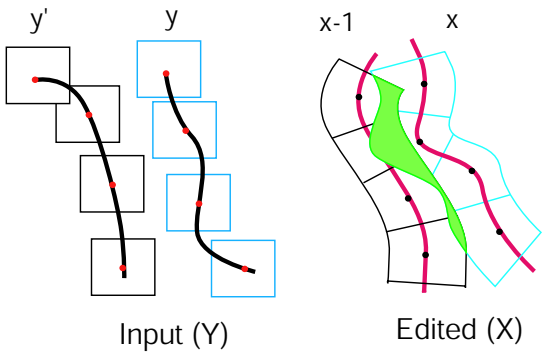


Figure 3.15: For a set of neighboring edited flow lines $(x-1, x)$ and their corresponding input flow lines (y', y), the patch overlap error computes a least square difference between the colors of overlapping regions, shown in green.

the *patch overlap error* $E(x-1, y', x, y)$ for all possible pairs (y', y) of input splines (figure 3.15). We evaluate the overlap error term by synthesizing the flow line $x-1$ using texture from y' and the neighboring flow line x using texture from y. The error is defined as the sum-squared error in color in the overlapping regions of the two textures, averaged over a fixed number of frames (30 in our experiments). To compute the correspondence, we minimize the following cost function:

$$C(x, y) = \min_{y'}[C(x-1, y') + E(x-1, y', x, y)] \tag{3.1}$$

where $x \in X$, $y \in Y$ and $y' \in Y$. Although the dynamic programming approach is an $O(N^2)$ algorithm (and hence slow), it produces a set of interpolated flow lines whose colors match well.

Rendering the output flow lines: We warp the texture along the input flow line to synthesize texture over the edited flow line, using the tangent information between corresponding points of the two curves (figure 3.16). A simple scheme for assigning corresponding points between the two flow lines is to choose points of equal arclengths (from the start of the flow line) along the two curves. An alternative is to add a term to the warping function that scales the arclengths of the input to be equal to the edited flow lines. This method produces edited textures that are scaled and warped versions of the texture from the input flow line. The animator chooses between these two schemes to create different effects in the edited sequence (figure 3.17).

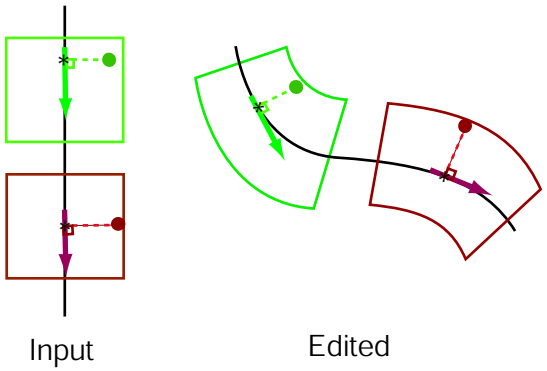


Figure 3.16: Schematic showing the transformations used to transfer texture from an input flow line to an edited flow line. For example, the green and red pixels in the edited flow line are obtained from the input particle texture using the tangent information at corresponding points.

Figure 3.17: Result of editing a smoke sequence by manipulating flow lines. Corresponding flow lines have the same color in this picture. The flow lines from the second and third chimney in the edited sequence (shown with letter S) are scaled because the input and edited flow lines have different lengths.

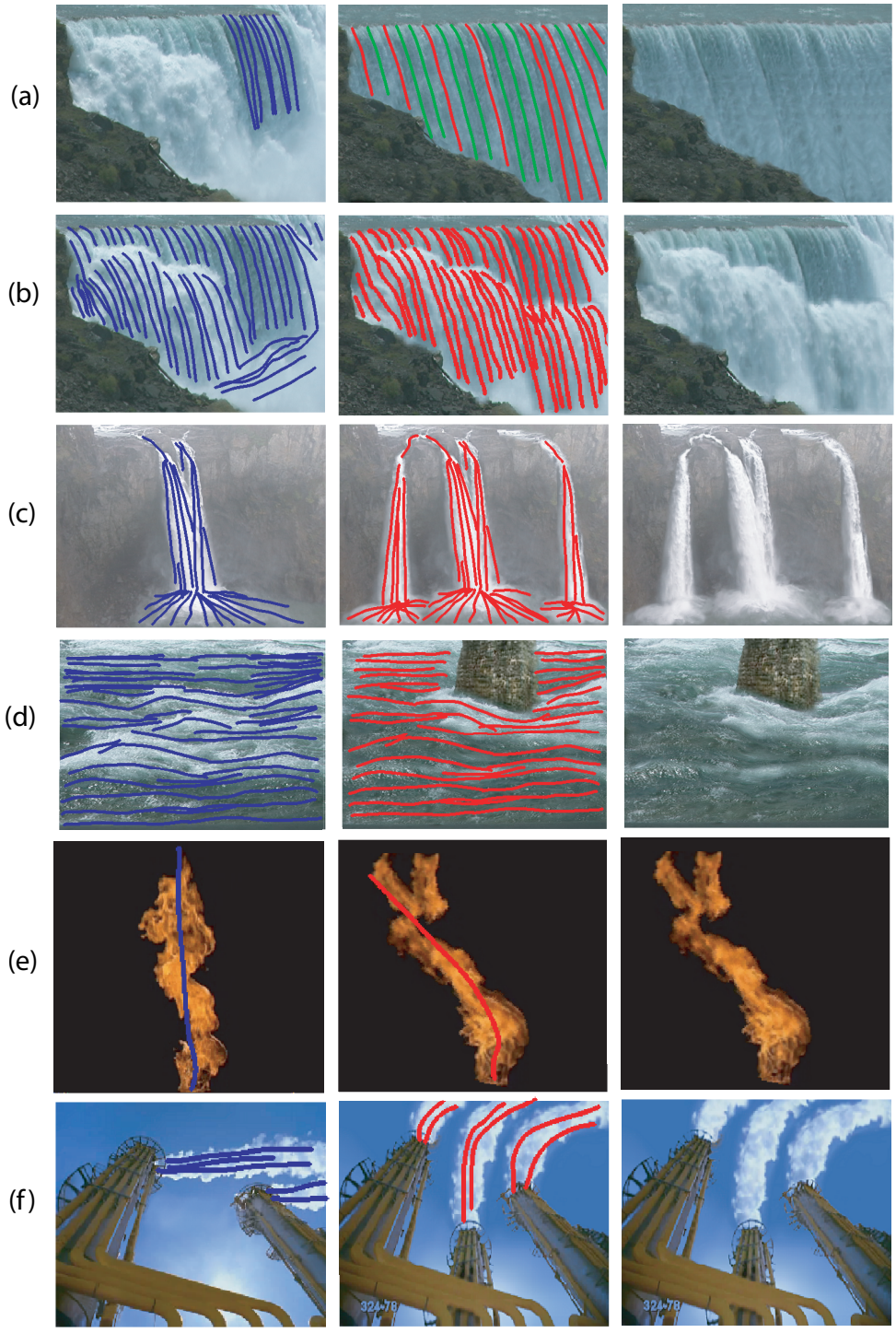


Figure 3.18: Various editing results obtained using our flow-based editing algorithm. The left column show a frame of the input video with the input flow lines, the middle column shows the edited flow lines and the right column shows a frame of the edited video synthesized by our algorithm.

3.3.1 Editing Results

We demonstrate a range of creative edits on video sequences of waterfalls, streams, smoke and fire by manipulating their flow lines in video. The input clips were the same ones that we used in the previous section to create infinite sequences. Because editing is a creative process, most of these examples took a few iterations of flow line sketching and correspondence specification before the desired edit was found. These edits demonstrate the effectiveness of our flow-based interface for creating dramatic edits to the scene.

Niagara Falls: We edited the Niagara sequence to change the landscape and remove the foamy portion of the waterfall. The resulting edited sequence and the corresponding flow lines are shown in figure 3.18(a). We then edited the waterfall to add some extra foam on the bottom right corner of the waterfall (figure 3.18(b)). This edit simulates the effect of changing the terrain under the waterfall, as if there were extra rocks causing the new foam to appear. In both the edits, we removed the mist (the wavy lines rising from the bottom left) from the input video.

Waterfall: In this example, we added two extra channels to a waterfall sequence (figure 3.18(c)). The terrain on the top of the waterfall was also edited to create an effect of water forking off to the four channels.

Stream: Here, we edit a stream sequence to simulate the effect of water colliding against a trunk (figure 3.18(d)). We edit the water flow lines to curve around the trunk and use the foamy texture (where water collides with rocks in the input video) to create the effect of water colliding against the trunk.

Flame: We model the input flame video using particles moving on a single vertical flow line. Figure 3.18(e) shows one frame of an edited sequence, where we simulate the effect of wind blowing to the left by specifying an edited flow line that curves left.

Smoke: This example exhibits intermittent motion along flow lines as smoke is emitted from a chimney (figure 3.18(f)). We add a extra chimney and change the direction of the smoke to create an edited footage.

3.4 Physics-Based Flow Lines

The flow-based editing algorithm presented in this thesis allows an artist to create photorealistic animations from video by specifying the input and output flow lines. Although our framework provides the artist with the freedom to sketch what he wants, he has to have the judgement to do things that are physically realistic. For example, the smokestack in figure 3.17 may not produce a convincing result because the smoke leaves the three stacks in very different directions in the edited sequence—something

that would not happen in real life. In such situations, it is easier (and more physically accurate) to generate the output flow lines automatically from simple physical simulations. Specifically, we show two editing examples that use a 3D particle system in Maya[2] to create the output flow lines. Varying the parameters of the particle system allows us to edit the input video in several interesting ways. Note that the physics model used here is very simple because its main purpose is to define the flow lines—the detailed particle dynamics and texture are still obtained from the input video.

Example 1: Controlling video with a 3D particle system In the first example, we create a controlled animation of a water jet colliding against an inverted bucket (figure 3.19(c)). This simple example illustrates the key steps for incorporating physics into our flow-based editing framework. The water jet in the input video follows a parabolic path in the air, collides against the bucket and curves downward as it flows down (figure 3.19(a)). We approximate the input water jet by manually tuning a 3D particle system such that the particle trajectory in simulation matches the water trajectory in the input video (figure 3.19(b)). This approximation gives us a good estimate of the simulation parameters such as the faucet pressure and the bucket collision coefficients (elasticity and friction). Varying the faucet pressure or the location/orientation of the bucket in simulation generates new output trajectories, that allows us to *control* the water jet in video (figure 3.19(c)). Continuously varying the simulation parameters results in output flow lines that change continuously over time. For example, the faucet pressure is continuously varied to follow the moving bucket in figure 3.19(d). To render this result, the video particles are synthesized along the time varying output flow lines. This example also highlights the advantage of incorporating physics because specifying time varying flow lines manually is very hard and inaccurate. Although the correspondence between input and output flow lines is trivial for this example, it has to be specified manually in the general case with multiple input and output flow lines.

Example 2: Creating a 3D city scene The second example uses our framework to create a fly-over animation of a CG city with a photorealistic waterfall (figure 3.20(c)). To create this animation, we first constructed a 3D model of the city with a large channel in the middle for a waterfall. Similar to the previous example, we use a 3D particle system to generate the output flow lines along the waterfall (green lines in figure figure 3.20(a)). The 3D particle trajectories in simulation are governed by the initial velocities, terrain interactions and gravity. The input flow lines and particle attributes of the waterfall are obtained from a video clip of Niagara falls (figure 3.20(b)). The

[2]http://www.alias.com

correspondence between the input and output flow lines is specified manually. We generate the fly-over effect by animating the camera in simulation, which causes the projected (output) flow lines to vary continuously with time. Once again, we render the resulting simulation by synthesizing video particles along the non-stationary output flow lines (figure 3.20(c)). This example clearly demonstrates the advantages of using a simple physics based simulator in conjunction with our video-based rendering technique to add photorealistic elements to synthetic scenes.

3.5 Discussion

The second part of this thesis presented a framework for creating and editing arbitrarily long videos from short input clips for a variety of natural phenomena. We used this technique on image sequences of waterfalls, rivers, flames, and smoke. For these phenomena, our synthesis results are comparable to the best existing video texture synthesis techniques. Our system also provides an intuitive interface for editing video.

A key contribution of this work is the simple algorithm for creating infinite sequences without significant visual discontinuities. We achieve continuity by ensuring that all particles complete their path along the flow lines without any interruption. We have found that our flow-based interface is intuitive to use and is ideally suited for sequences that exhibit nearly stationary dynamics. The algorithm is also quite simple, making it easy to understand and implement.

Our algorithm has a few limitations, some of which are specific to our current implementation and others that are more fundamental to our 2D technique. Because our algorithm does not model pixel transparencies, it can produce blending artifacts at regions of the video that have multiple transparent layers interacting with each other, such as the water-foam interface at the bottom portion of Niagara Falls (figure 3.21(a))[3]. At this interface, water particles travelling along the vertical flow lines are occluded by the travelling mist rising on the bottom right corner of the video. Extending the algorithm to handle transparency explicitly using video matting techniques [CAC+02] might address this problem.

Our simple blending scheme performs poorly on sequences with discrete objects moving along flow lines like cars on a highway (figure 3.21(b)). An interesting extension would be to use seaming techniques such as graph cuts [KSE+03] instead of linear blending to approximate the interface between the textures of neighboring particles. Separating the foreground portion of the particle texture from the background before merging should also improve the quality of the rendered result.

[3] http://graphics.cs.cmu.edu/projects/flow/

Although our approach has been successfully used to edit a number of sequences, it is limited to input sequences with nearly stationarity flow patterns. Manually specified flow lines may not be suitable for input sequences such as fire or smoke blowing in the wind (figure 3.22), where our assumption of constant flow lines (over time) does not hold. In order to handle general non-stationary process using our framework, the user has to specify temporally varying flow lines for all particles in video, which is a laborious task. Automatically computing flow lines from video is also very hard, because velocity estimates from general motion analysis algorithms like optical flow are not reliable in regions that do not have much texture. For example, figure 3.23 shows the flow field obtained from two consecutive images of Niagara falls using the robust incremental flow algorithm developed by Black et al. [BA96]. Although we get reasonable flow results for the top portion of the waterfall, the flow estimates for the lower portion of the scene with mist are not very reliable. Devising robust statistical techniques that compute flow lines automatically for general scenes of fluid motion [CEY02] might address this problem.

Our technique is primarily suited for 2D editing, because we use video as our source of particles. Consequently, our approach may not perform well for edits that require a large change in the view point. For example, our technique with 2D flow lines cannot be used to render the top view of the horse-shoe fall at Niagara using information from the side view (figure 3.24). A potential solution is to model the scene in 3D using the ideas presented in section 3.4 that combine our flow-based rendering approach with 3D particle systems. Techniques for combining our particle-based approach for appearance with general purpose fluid simulators for dynamics might allow the us to model and edit other complex phenomena such as waves or explosions using our flow based approach.

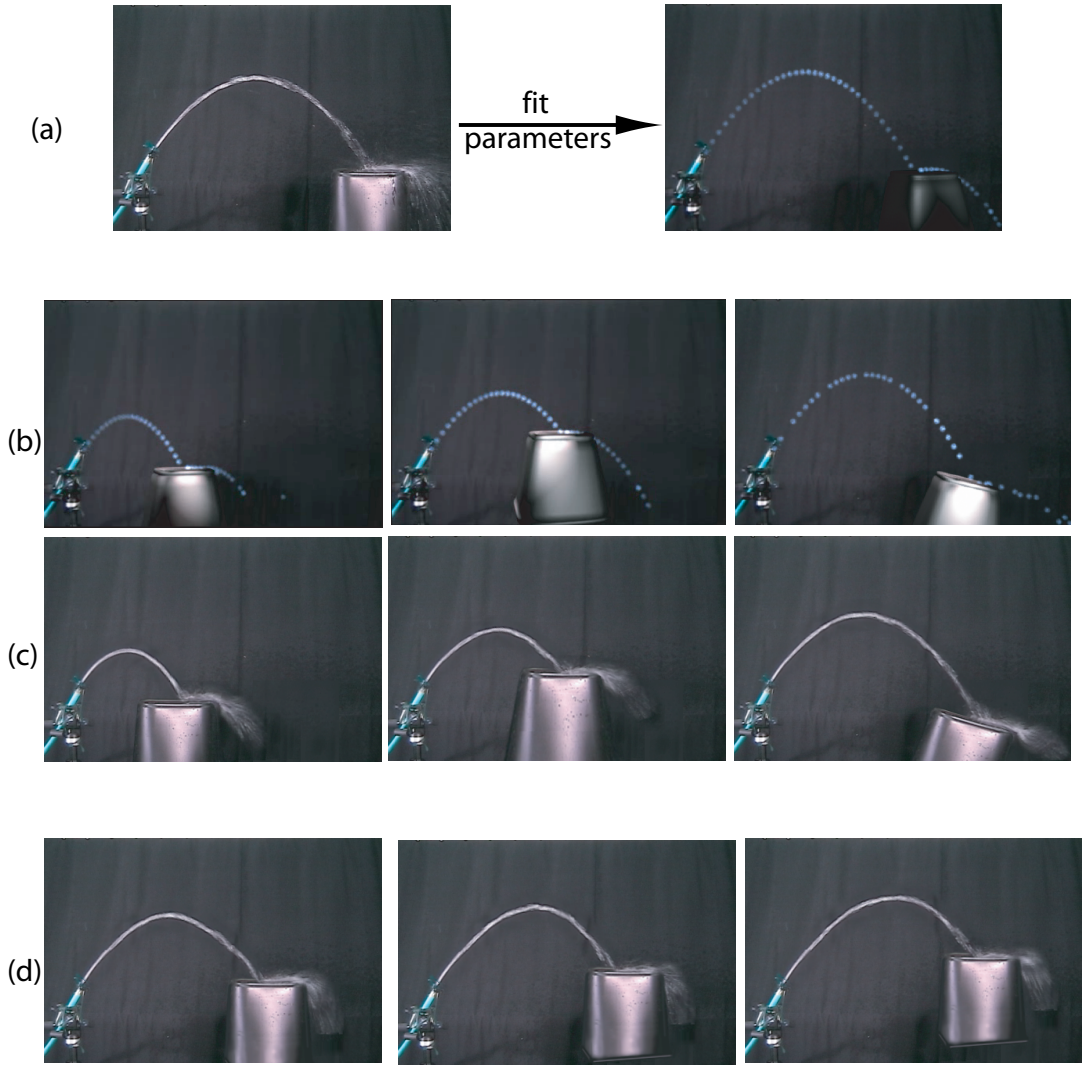


Figure 3.19: Using a 3D particle system to control video. (a) A particle system in Maya, fitted manually to match the input video on the left. Varying the parameters of the particle system in row (b) allows us to automatically generate the output flow lines, which are rendered using texture from the input as seen in row (c). (d) The water jet in the last row moves continuously to follow the moving bucket.

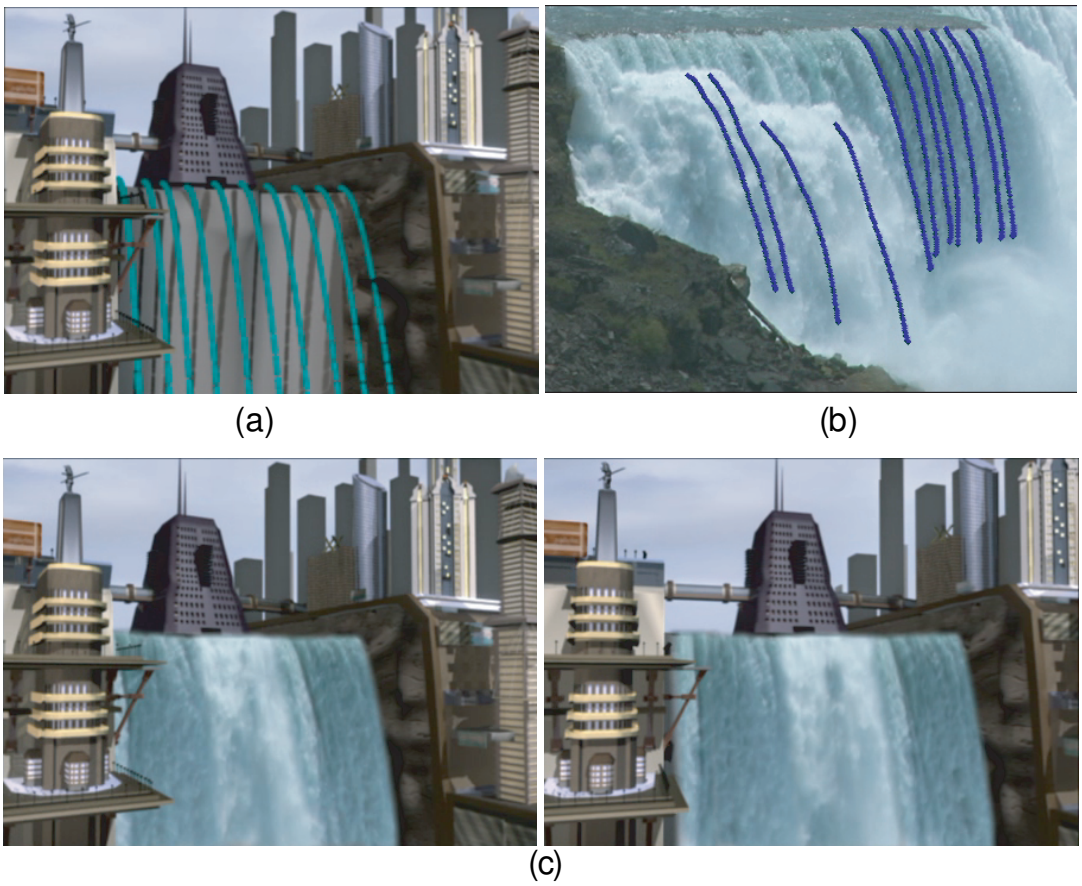


Figure 3.20: Building a virtual city using a 3D terrain and particle system in Maya. The top row shows the Maya terrain and desired flow lines created using a 3D particle system. The input flow lines are obtained from the Niagara fall clip. The bottom row shows two frames of the waterfall, rendered at different camera viewpoints using our technique.

Figure 3.21: (a) The blending artifacts created at the interface of water and mist in Niagara. The interface region is zoomed to show the blending artifact. This artifact is more perceptible in video. (b) A traffic scene with discrete objects where we expect our feathering algorithm to perform poorly.

Figure 3.22: Examples of non stationary phenomena where the constant flow line assumption fails.

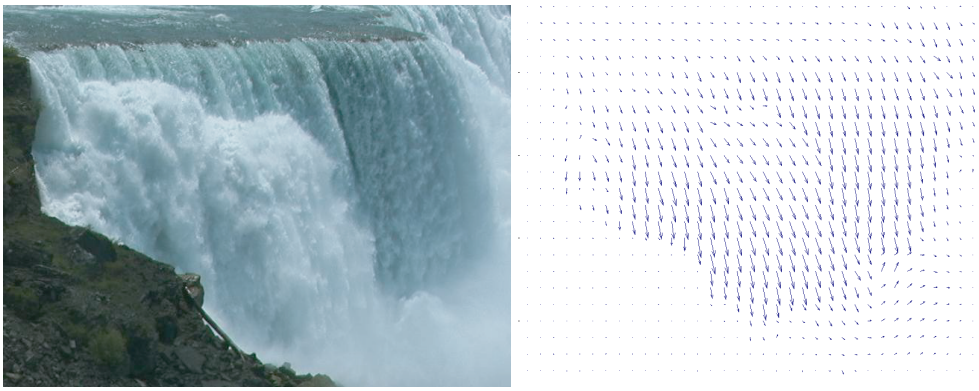


Figure 3.23: Optical flow results with two frames of the Niagara Falls sequence using Black et al. [BA96]. The algorithm gives accurate velocity estimates at the top portion of the fall, but performs poorly near the interface of water and mist at the bottom.

Figure 3.24: Our 2D technique will not be able to synthesize the top view from the side view.

Chapter 4

Conclusions

This thesis extends the state of the art of video-based computer animation by presenting new algorithms for creating compelling animations of passive dynamic systems like rigid bodies, cloth and fluids from video. Our first method, inverse simulation, uses video sequences of simple experiments (throwing rigid bodies and waving swatches of cloth) captured in a laboratory setting to identify parameters of rigid body and cloth simulators. The parameters estimated from simple experiments can be used to create realistic simulations under new conditions—for instance, to animate a dancer wearing a skirt of the same material. Our second method, flow-based video editing, creates realistic and controllable animations of fluids by manipulating and rearranging flow lines from a reference video clip. We tested the flow-based editing framework on several example clips: water, fire, smoke, streams and waterfalls. In all these examples, an artist controls the global dynamics and appearance characteristics of the resulting animation by specifying the input and output flow lines.

We review the contributions of our two video-based animation techniques in Section 4.1. In Section 4.2, we describe several possible extensions for the two techniques. Finally, in Section 4.3, we compare and contrast the inverse simulation framework with the video editing framework, and present ideas for extending video-based animation techniques to other domains.

4.1 Contributions

The main contribution of this thesis is the design, implementation and evaluation of two video-based techniques, inverse simulation and video editing, for creating realistic animations of a wide range of passive phenomena.

Inverse Simulation Framework. The inverse simulation framework computes the parameters of a simulation that match reference video footage of real phenomena. To the best of our knowledge, the inverse framework presented in this thesis is the first systematic technique in computer graphics for estimating the static and dynamic parameters of physical simulations from video measurements. In order to achieve this goal, we designed new algorithms (or significant extensions of existing methods) for various components of the inverse simulation framework. Additionally, we designed simple laboratory experiments that excited the important modes of the dynamic system for parameter identification.

We used the inverse simulation framework to estimate the free flight parameters of rigid bodies, and the static and dynamic parameters of cloth simulations. En route to designing this framework, we made a number of contributions for enhancing the realism of computer simulations:

- *Model refinement.* Matching simulations to video of real cloth required several improvements to existing cloth models. Specifically, we extended the Baraff-Witkin cloth model to make the cloth parameters independent of mesh resolution, and used an explicit integrator that produced accurate folds. Additionally, our three parameter model for air drag better approximates the non-linear behavior of air-cloth interactions.

- *Metric design.* An important contribution of this thesis is the design of perceptual metrics, for rigid bodies and cloth, that can be robustly and quickly estimated from video. Our results show that a metric which compares two sequences of 2D silhouettes successfully captures the tumbling parameters of rigid bodies. For cloth, we developed a novel representation, which we called the *anglemap*, to measure fabric folds in video. The anglemap representation allowed us to identify the parameters for four types of fabrics with sufficient accuracy to produce visually distinct animations for each type of fabric.

Video Editing Framework. The video editing framework creates realistic animations by modelling the flow patterns of natural phenomena in video and rearranging the flow lines to create edited sequences. The important contributions of this framework are summarized below:

- *Simple particle models for video.* We developed a new particle model for capturing the dynamics and appearance of video sequences of natural phenomena. Our algorithm simplifies the analysis of stationary phenomena like waterfalls to one dimension by modelling video via particles that move along constant flow lines. This model is simple to implement and successfully models a wide range of natural phenomena such as water, streams, fire and smoke.

- *Intuitive editing interface.* Another contribution of this thesis is the use of flow lines for editing video sequences of natural phenomena. Flow lines capture the user's intuition about the motion of particles in video, providing an highly effective mechanism for specifying edits. We demonstrated the effectiveness of the flow-based interface by creating dramatic changes to natural scenes: altering the terrain of a waterfall, adding obstacles in streams and changing the direction of smoke and fire.

- *Controlling video.* We incorporated physical simulations within our video-based rendering framework for automatically generating the output flow lines. Physically generated flow lines varied naturally in response to such animator controls as emitter velocities, collisions and external forces, resulting in compelling animations that are controllable and exhibit natural dynamics.

4.2 Future Work

In this section, we highlight a few interesting areas for future research in video-based animation, focusing first on the inverse simulation framework and second on the video editing framework.

4.2.1 Inverse Simulation

Extending the inverse simulation framework to capture the finer details of real cloth and other passive dynamic systems (e.g., deformable tissues) would require several advances to the existing framework. Our discussion is primarily focused on cloth simulations, although many of the ideas apply to rigid bodies and other phenomena as well.

Improved models. Although our current cloth model produces compelling fold patterns that are similar to real fabrics, it does not model the time-varying changes in cloth due to aging or wrinkling (figure 4.1). An important direction for future work is to design cloth models that handle non-linear hysteresis effects commonly observed in real fabrics. Additionally, incorporating complex aerodynamics [Pes77] and wind effects [WH91] into cloth models will allow us to simulate two way air-cloth interactions and billowing effects seen in real garments (twirling skirts). These improved cloth models would require more parameters to be identified from video, which in turn will necessitate the design of new experiments for exercising these additional parameters.

Figure 4.1: Examples of non-linear wrinkling and aging effects in cloth.

Statistical metrics. Our metrics perform a frame to frame comparison between features (e.g., folds, silhouettes) in simulation and video and are therefore ideally suited for short video sequences obtained from controlled experiments. However, such metrics may not be appropriate for matching longer sequences or complex garments like skirts, where an overall match in the folding behavior across the entire sequence might be desirable rather than a precise frame-to-frame match. For instance, a frame-to-frame match might require the accurate knowledge of external parameters such as collisions with limbs, which might be very hard to estimate. An alternative approach would be to model the fold propagation as a statistical process. Instead of matching the folds pixel by pixel, we could match the global characteristics across the sequence. For example, we might match the average life time of a fold, the average number and orientation of folds, or the average width (scale, sharpness) of folds.

Metric evaluation. Our metrics are only two of the many possible metrics for matching simulations with video. Performing carefully designed user studies to evaluate the different metrics (e.g., folds versus marked patterns, frame-by-frame versus statistical) would provide useful guidelines for designing perceptual metrics for cloth and other similar phenomena. Such a study could be conducted, for example, by estimating the simulation parameters using several different metrics, and having users rank the visual quality of the match with reality.

Experimental design. The waving test with swatches of fabrics was a simple test that worked well for identifying the stiffness, damping and drag parameters of a cloth simulation. However, estimating different cloth parameters, such as friction, would require new experiments. For example, friction might be measured by dragging the fabric over different materials. An important area for future work is to systematically

design experiments that best exercise all the modes of the simulation [Kho88]. A general estimation framework should also allow data from multiple experiments (and multiple trials) to be optimized simultaneously to obtain the best parameter estimates. Such a framework would also reduce the burden of experimental design because individual parameters would be exercised in multiple experiments.

Inverse draping. An interesting application of the inverse simulation framework is to explore the problem of inverse draping: given an image or video of a fabric draped on an object, estimate the object's underlying shape [JH97]. This analysis-by-synthesis approach might lead to improved human tracking algorithms that could accurately identify limb and joint configurations despite loose-fitting clothing.

Extensions to other phenomena. The inverse simulation framework can be potentially applied to other domains such as deformable tissue simulations. Obtaining realistic simulations of tissue models from real video measurements has tremendous potential in medicine and surgery for training and diagnosis. In principle, inverse simulation techniques could also be extended to simulations of more complex natural phenomena such as smoke, water, fire, provided that we have accurate physical models for each phenomena. We could leverage the vast body of research in physical simulations to obtain these models. However, this extension would also require precise models for wind and other external forces (potentially time-varying) in order to match with video, which would increase the dimensionality of the search space.

User interaction. The inverse simulation framework presented in this thesis is a fully automatic technique for estimating simulation parameters from video. However, having a user in the loop could potentially improve the different blocks of the inverse simulation framework. For example, a user could guide the optimization by specifying higher weights for certain features over others (folds over silhouettes, for instance), or by specifying portions of the video where the match between simulation and video is acceptable and portions where it is not. The optimizer could then focus on those features or regions in subsequent iterations. The design of effective user interfaces for steering the optimization will be challenging but is essential for the success of this approach.

4.2.2 Video Editing

Flow lines are central to our video editing framework. Here, we discuss a few ideas to generalize the concept of flow lines to a broader class of natural phenomena. But

first, we present a few ideas for improving the performance of our existing particle model.

Model Improvements. Our current particle model uses texture from video without segmenting the foreground (fluid) layer from the background, which causes blending artifacts at interfaces between fluids. Segmenting video sequences of natural phenomena into foreground layers with alpha and a background layer might reduce this artifact (e.g., [CAC+02, Yu04]). Segmentation would improve the particle color estimates from video, and would allow us to correctly model the motion and appearance of translucent phenomena such as mist.

Automating the input flow lines. The particle system model presented in this thesis assumes that the flow lines in the input video are *nearly stationary*. The constant flow line assumption is not a fundamental limitation; it just simplifies the user interaction—our algorithm will easily scale to non-stationary phenomena as long as the time-varying flow lines can be specified in some way. Designing algorithms for automatically estimating flow lines, for example, using optical flow geared towards fluid-like phenomena [CEY02], is an interesting area of future research.

Extensions for visual effects (F/X) applications. An useful extension of our technique for F/X applications would be to combine flow lines from multiple input sequences to create an edited video. Such a technique might be useful for increasing the realism of *scale models*, that are commonly used for animating phenomena like waterfalls in a studio. For example, the mist caused due to water-rock collisions from a real waterfall such as Niagara falls could be used to increase the realism of a video from a small scale model where such effects are hard to create.

Physics-based flow lines. Our flow-based editing interface allows an artist to create dramatic edits to a scene by manually specifying the flow patterns in video. However, to synthesize scenes with temporally varying flow patterns, it might be advantageous to use simple physics models to animate the output flow lines. For example, to create the animation of a flame blowing in the wind, an artist might sketch the output flow line in one frame, and use a physics-based wind model to deform the flow line in subsequent frames in a controllable fashion.

Combining physics with our flow-based framework could also allow us to handle a larger class of natural phenomena. For example, waves crashing on a beach could be modelled using irregularly looping flow lines, whose magnitude (length) varies stochastically to account for waves of different sizes. Such models could perhaps be obtained from physical simulations—for example, a simple water simulator based

on Navier-Stokes could be used to generate the flow lines for waves by simulating the basic motion of water. An interesting direction for future research would be to use physical models that are more sophisticated than particle systems to describe the dynamics and use video to model the appearance.

4.3 Comparison of video-based animation methods

The two techniques presented in this thesis share a common theme of using video for creating realistic animations. This section compares the strengths and weaknesses of the two approaches, and discusses ideas for future research that combine the strengths of these two approaches.

An important distinction between the two techniques is the complexity of the model required to match the reference video. In inverse simulation we use a physical simulation to model the 3D dynamics of the phenomena observed in video, while in flow-based editing, we manually model the dynamics and appearance of video using a particle system. This difference arises because physical simulations of phenomena like rigid bodies and cloth are much easier to match with real video footage in comparison to natural phenomena. For example, waterfalls (and other complex natural phenomena) would be very hard to match with video because the simulation would require an accurate terrain model and wind fields. A water model with sufficient accuracy to model Niagara falls would also likely have more parameters than the cloth model we used. For such phenomena, flow-based video editing presents a simple and effective alternative to physical simulation.

Parameters obtained from calibration experiments using inverse simulation generalize to new, more complex simulation conditions. For example, parameters from the swatch test were used to create realistic animations of skirts of the same material. In contrast, the video editing framework models each video sequence without explicitly extracting much knowledge of the underlying physical phenomenon. Therefore, information from one video sequence may not generalize to new situations.

Finally, the ease of setting up and working with these two techniques differs significantly. Inverse simulation requires a custom simulator for each phenomenon. Additionally, it requires video footage from carefully designed experiments. Both steps are difficult to implement and require prior knowledge about the physics of the phenomenon. In contrast, flow-based video editing is relatively simple to set up. However, the video editing technique is not fully automatic—it requires input from the user to specify the input and edited flow lines. Therefore, users generally spent a little time training with the flow-based framework before successfully creating their own edits.

We hope that the techniques presented in this thesis will inspire new video-based

animation algorithms that combine the strengths of the two approaches. For example, combining Navier-Stokes fluid simulators with our particle framework for rendering from video might allow us to produce controllable animations of complex phenomenon such as explosions. The design of video-based algorithms for articulated systems (e.g., trees) and systems with active dynamics (e.g., humans, animals) might prove fruitful. Although several video-based techniques for animating humans and animals have been proposed (e.g., video sprites [SE02], video rewrite [BCS97]), these techniques are purely data-driven and do not use any knowledge of the physics of the system. In human animation, we might be able to apply inverse simulation techniques to improve the realism of existing human motion simulators [LP02, HWBO95]. We hope that video-based animation techniques will inspire the design of better simulation models for a broad range of phenomena, enabling a better match with reality.

Bibliography

[Ach90] D. J. Acheson. *Elementary Fluid Dynamics*. Oxford University Press, Oxford, 1990.

[AP98] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[BA96] M. J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, pages 75–104, 1996.

[Bar94] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 23–34, July 1994.

[BCS97] C. Bregler, M. Covell, and M. Slaney. Video rewrite: Driving visual speech with audio. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 353–360, August 1997.

[BFA02] R. Bridson, R. P. Fedkiw, and J. Anderson. Robust treatment of collisions, contact, and friction for cloth animation. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):594–603, July 2002.

[BHW94] D. E. Breen, D. H. House, and M. J. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 365–372, July 1994.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 28–36, July 2003.

[Bre97] C. Bregler. Learning and recognizing human dynamics in video sequences. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8–15, June 1997.

[BS99] B. E. Bishop and M. W. Spong. Vision-based control of an air hockey playing robot. *IEEE Control Systems*, pages 23–32, June 1999.

[BSK00] K. S. Bhat, M. Saptharishi, and P. K. Khosla. Motion detection and segmentation using image mosaics. *IEEE International Conference on Multimedia and Exposition*, pages 1577–1580, 2000.

[BW98] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 43–54, July 1998.

[BWK03] D. Baraff, A. Witkin, and M. Kass. Untangling cloth. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):862–870, July 2003.

[CAC+02] Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski. Video matting of complex scenes. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):243–248, July 2002.

[CC91] S. Chandrashekhar and R. Chellappa. Passive navigation in a partially known environment. *Proceedings of the IEEE Workshop on Visual Motion*, pages 2–7, 1991.

[CEY02] J. Chang, D. Edwards, and Y. Yu. Statistical estimation of fluid flow fields. *ECCV workshop on Statistical Methods in Video Processing*, pages 91–96, 2002.

[Che04] S. Chenney. Flow tiles. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 233–242, 2004.

[CK01] R. L. Carceroni and K. N. Kutulakos. Multi-view scene capture by surfel sampling: From video streams to non-rigid 3d motion, shape & reflectance. *Proceedings of International Conference on Computer Vision*, pages 60–67, 2001.

[CK02] K. Choi and H. Ko. Stable but responsive cloth. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):604–611, July 2002.

[CVMT95] M. Courshesnes, P. Volino, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects.

In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 137–144, August 1995.

[DCSW03] G. Doretto, A. Chiuso, S. Soatto, and Y. N. Wu. Dynamic textures. *International Journal of Computer Vision*, 51(2):91–109, February 2003.

[DF99] Q. Delamarre and O. Faugeras. 3d articulated models and multi-view tracking with silhouettes. *Proceedings of the International Conference on Computer Vision*, pages 716–721, 1999.

[DOF93] G. C. DeAngelis, I. Ohzawa, and R. D. Freeman. Spatiotemporal organization of simple-cell receptive fields in the cat's striate cortex. *Journal of Neurophysiology*, 69(4):1091–1117, 1993.

[DS03] G. Doretto and S. Soatto. Editable dynamic textures. *Proceedings of the IEEE Computer Vision and Pattern Recognition*, 2:137–142, 2003.

[EF01] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 341–346, August 2001.

[ERH03] U. Eysholdt, F. Rosanowski, and U. Hoppe. Vocal fold vibration irregularities caused by different types of laryngeal asymmetry. *Eur Arch Otorhinolaryngol*, 260(8):412–417, 2003.

[FA91] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991.

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 23–30, August 2001.

[Fie87] D. J. Field. Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America A*, 4(12):2379–2394, December 1987.

[Fie94] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 6(4):559–601, July 1994.

[FL04] R. Fattal and D. Lischinski. Target-driven smoke animation. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23:441–448, August 2004.

[Fle17] M. Fleischer. Method of producing moving picture cartoons. *US Patent no. 1242674*, 1917.

[FLP01] O. Faugeras, Q. T. Luong, and T. Papadopoulo. *The Geometry of Multiple Images*. MIT Press, 2001.

[Fro84] C. Frohlich. Aerodynamic drag crisis and its possible effect on the flight of baseballs. *American Journal of Physics*, 52(4):325–334, April 1984.

[FSJ01] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22, August 2001.

[GBF03] E. Guendelman, R. Bridson, and R. P. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):871–878, July 2003.

[GL95] B. K. Ghosh and E. P. Loucks. A perspective theory for motion and shape estimation in machine vision. *SIAM Journal of Control and Optimization*, 33(5):1530–1559, 1995.

[GMW82] P. E. Gill, W.r Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1982.

[Gra92] A. Grace. *Optimization Toolbox Users Guide*. MathWorks, 1992.

[Har92] C. Harris. Tracking with rigid models. In A. Blake and A. Yuille, editors, *Active Vision*, chapter 4, pages 59–73. The MIT Press, 1992.

[HB00] D. H. House and D. E. Breen, editors. *Cloth Modeling and Animation*. A.K. Peters, Ltd., Natick, Massachusetts, July 2000.

[HDB96] D. H. House, R. W. DeVaul, and D. E. Breen. Towards simulating cloth dynamics using interacting particles. *International Journal of Clothing Science and Technology*, 8(3):75–94, 1996.

[HF98] J. Haddon and D. A. Forsyth. Shading primitives: finding folds and shallow grooves. *Proceedings of International Conference on Computer Vision*, pages 236–41, 1998.

[HFP98] J. Haddon, D. Forsyth, and D. Parks. The appearance of clothing. http://http.cs.berkeley.edu/haddon/clothingshade.ps, 1998.

[HJO+01] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 327–340, August 2001.

[HK03] S. W. Hasinoff and K. N. Kutulakos. Photo-consistent 3d fire by flame-sheet decomposition. *Proceedings of the International Conference on Computer Vision*, pages 1184–1191, 2003.

[HMB01] S. Huh, D. Metaxas, and N. Badler. Collision resolutions in cloth simulation. In *Computer Animation*, pages 122–127. IEEE, 2001.

[HW68] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195(1):215–243, 1968.

[HWBO95] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 71–78, August 1995.

[HZ00] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[JH97] N. Jojic and T. S. Huang. Estimating cloth draping parameters from range data. In *International Workshop on Synthetic-Natural Hybrid Coding and 3-D Imaging*, pages 73–76, Rhodes, Greece, 1997.

[Kai79] T. Kailath. *Linear Systems*. Prentice Hall, 1979.

[Kaw80] S. Kawabata. The standardization and analysis of hand evaluation. *The Textile Machinery Society of Japan*, 1980.

[Kho88] P. K. Khosla. Estimation of robot dynamics parameters: Theory and application. *International Journal of Robotics and Automation*, 3(1):35–41, 1988.

[KSE+03] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):277–286, July 2003.

[Lah02] T. J. Lahey. Modeling hysteresis in the bending of fabrics. Master's thesis, University of Waterloo, 2002.

[LAM01] T. Larsson and T. Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics 2001*, pages 325–333, 2001.

[LF02] A. Lamorlette and N. Foster. Structural modeling of natural flames. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):729–735, July 2002.

[Lin00] L. Ling. Aerodynamic effects. In D. H. House and D. E. Breen, editors, *Cloth Modeling and Animation*. A.K. Peters, Ltd., Natick, Massachusetts, July 2000.

[LP02] K. Liu and Z. Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):408–416, July 2002.

[MC95] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. *Proceedings of 1995 Symposium on Interactive 3D Graphics*, pages 181–188, April 1995.

[Möl97] T. Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997.

[MT93] D. Metaxas and D. Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):580–591, 1993.

[MTPS04] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23:449–456, August 2004.

[Nel00] O. Nelles. *Nonlinear System Identification*. Springer Verlag, 2000.

[NFJ02] D. Q. Nguyen, R. P. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):721–728, July 2002.

[OZH00] J. F. O'Brien, V. B. Zordan, and J. K. Hodgins. Combining active and passive simulations for secondary motion. *IEEE Computer Graphics & Applications*, 20(4):86–96, July 2000.

[PDI01] PDI. Shrek: The story behind the screen. *In SIGGRAPH Course Notes CD-ROM, Course #19*, pages 59–66, 2001.

[Pei37] F. T. Peirce. The geometry of cloth structure. *Journal of the Textile Institute*, 28(T45–T97), 1937.

[Pes77] C. S. Peskin. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25:220–252, 1977.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*, chapter 10.9, pages 444–455. Cambridge University Press, Cambridge (UK) and New York, 2nd edition, 1992.

[PH91] A. Pentland and B. Horowitz. Recovery of nonrigid motion and structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):730–742, July 1991.

[Pro95] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface '95*, pages 147–154, May 1995.

[Pro97] X. Provot. Collision and self-collision handling in cloth model dedicated to design. In *Computer Animation and Simulation '97*, pages 177–190, September 1997.

[PSE+00] J. Popović, S. M. Seitz, M. Erdmann, Z. Popović, and A. P. Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 209–218, July 2000.

[PSE03] J. Popović, S. M. Seitz, and M. Erdmann. Motion sketching for control of rigid body simulations. *ACM Transactions on Graphics*, 22(4):1034–1054, 2003.

[RB85] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Proceedings of SIGGRAPH '85*, Computer Graphics Proceedings, Annual Conference Series, pages 313–322, July 1985.

[Ree83] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. In *Proceedings of SIGGRAPH '83*, Computer Graphics Proceedings, Annual Conference Series, pages 91–108, July 1983.

[Rey87] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH '87*, Computer Graphics Proceedings, Annual Conference Series, pages 25–34, July 1987.

[RK96] A. A. Rizzi and D. E. Koditschek. An active visual estimator for dexterous manipulation. *IEEE Transactions on Robotics and Automation*, 12(5):697–713, 1996.

[SD91] J. Schick and E. D. Dickmanns. Simultaneous estimation of 3d shape and motion of objects by computer vision. *Proceedings of the IEEE Workshop on Visual Motion*, pages 256–261, 1991.

[SE02] A. Schödl and I. Essa. Controlled animation of video sprites. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 121–128, July 2002.

[Sim90] K. Sims. Particle animation and rendering using data parallel computation. In *Proceedings of SIGGRAPH '90*, Computer Graphics Proceedings, Annual Conference Series, pages 405–413, August 1990.

[SP91] S. Sclaroff and A. Pentland. Generalized implicit functions for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 247–250, July 1991.

[SS02] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal on Computer Vision*, 47(1):7–42, 2002.

[SSSE00] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 489–498, July 2000.

[Sta99] J. Stam. Stable fluids. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 121–128, August 1999.

[Ste94] R. Stengel. *Optimal Control and Estimation*. Dover Books on Advanced Mathematics, 1994.

[Sym71] K. Symon. *Mechanics, Third Edition*. Addison-Wesley Publishing Company, Reading, Massachussetts, 1971.

[TJ84] F. Thomas and O. Johnston, editors. *Disney Animation: The Illusion of Life*. Abbeville Press, New York, May 1984.

[TK92] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography—a factorization method. *International Journal on Computer Vision*, 9(2):137–154, 1992.

[TMPS03] A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulations. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):716–723, July 2003.

[TPB+89] D. Terzopoulos, J. Platt, A. Barr, D. Zelter, A. Witkin, and J. Blinn. Physically based modelling: Past, present and future. *Computer Graphics*, 23(5), 1989.

[TPBF87] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 205–214, July 1987.

[TV98] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*, chapter 7, pages 146–148. Prentice-Hall, Inc, New Jersey, 1998.

[TWK88] D. Terzopoulos, A. Witkin, and M. Kass. Constraints on deformable models: Recovering 3d shape and nonrigid motion. *Proceedings of the American Association of Artificial Intelligence Conference*, pages 91–122, 1988.

[TYAB01] L. Torresani, D. Yang, G. Alexander, and C. Bregler. Tracking and modelling non-rigid objects with rank constraints. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 493–500, 2001.

[vHR98] J. H. van Hateren and D. L. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proceedings of the Royal Society of London B*, 265(1412):2315–2320, 1998.

[VMT94] P. Volino and N. Magnenat-Thalmann. Efficient self-collision detection on smoothly discretised surface animations using geometrical shape regularity. In *Computer Graphics Forum (Eurographics Proceedings)*, volume 13, pages 155–166, 1994.

[VMT00] P. Volino and N. Magnenat-Thalmann. Accurate collision response on polygonal meshes. In *Computer Animation 2000*, pages 154–163, May 2000.

[VYH82] R. L. De Valois, E. W. Yund, and N. Hepler. The orientation and direction selectivity of cells in macaque visual cortex. *Vision Research*, 22(5):531–544, 1982.

[WH91] J. Wejchert and D. Haumann. Animation aerodynamics. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 19–22, July 1991.

[WK88] A. Witkin and M. Kass. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 159–168, August 1988.

[WL00] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 479–488, July 2000.

[Wre00] C. Wren. *Understanding Expressive Action*. Ph.D Thesis, Massachusetts Institute of Technology, March 2000.

[WZ02] Y. Wang and S. C. Zhu. A generative model for textured motion: Analysis and synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 582–598, 2002.

[Yu04] S. X. Yu. Segmentation using multiscale cues. *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 247–254, 2004.

[ZCS02] L. Zhang, B. Curless, and S. M. Seitz. Rapid shape acquisition using color structured light and multi-pass dynamic programming. In *Proceedings of Symposium on 3D Data Processing Visualization and Transmission (3DPVT)*, pages 24–36, 2002.