Image Based Spatio-Temporal Modeling and View Interpolation of Dynamic Events

Sundar Vedula

CMU-RI-TR-01-37

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

September 2001

Thesis Committee:

Takeo Kanade, Chair
Simon Baker
Martial Hebert
Frank Crow, nVidia Corporation
Steve Seitz, University of Washington

Keywords: image based modeling and rendering, dynamic scenes, non-rigid motion, spatio-temporal view interpolation

Abstract

Digital photographs and video are exciting inventions that let us capture the visual experience of events around us in a computer and re-live the experience, although in a restrictive manner. Photographs only capture snapshots of a dynamic event, and while video does capture motion, it is recorded from pre-determined positions and consists of images discretely sampled in time, so the timing cannot be changed.

This thesis presents an approach for re-rendering a dynamic event from an arbitrary viewpoint with any timing, using images captured from multiple video cameras. The event is modeled as a non-rigidly varying dynamic scene captured by many images from different viewpoints, at discretely sampled times. First, the spatio-temporal geometric properties (shape and instantaneous motion) are computed. Scene flow is introduced as a measure of non-rigid motion and algorithms to compute it, with the scene shape. The novel view synthesis problem is posed as one of recovering corresponding points in the original images, using the shape and scene flow. A reverse mapping algorithm, ray-casting across space and time, is developed to compute a novel image from any viewpoint in the 4D space of position and time. Results are shown on real-world events captured in the CMU 3D Room, by creating synthetic renderings of the event from novel, arbitrary positions in space and time. Multiple such re-created renderings can be put together to create re-timed fly-by movies of the event, with the resulting visual experience richer than that of a regular video clip, or simply switching between frames from multiple cameras.

Acknowledgements

I would like to dedicate this thesis to my parents, for their endless encouragement, love, and support throughout my education.

My advisor Takeo Kanade has been a great source of inspiration. His immense enthusiasm, high standards for excellence, and trust in me have made my years in graduate school fun, challenging, and memorable. I'd also like to thank my thesis committee - Simon Baker, Frank Crow, Martial Hebert, and Steve Seitz for their useful feedback and comments.

I'm especially grateful to Simon Baker. He has almost been like an advisor - helping out when nailing down a thesis topic seemed like a dead-end, and being a great source of ideas and feedback on most of this work, in addition to being a mentor and friend. Pete Rander was a great help during my early days with the Virtualized Reality project, as I was trying to figure out how a lot of things worked. Thanks to Steve Seitz for many useful discussions (some on 6D spaces) and detailed comments, and particularly for flying in for my thesis defense barely a week after the terrible events of September 11, 2001.

Thanks to German Cheung, Hideo Saito, P.J. Narayanan, Makoto Kimura, and Shigeyuki Baba for all the help as members of the Virtualized Reality project - without your assistance, the PCs would not have worked as smoothly, the cameras would not have behaved themselves as well, and I wouldn't have had the data for this thesis.

The Vision and Autonomous Systems Center (VASC) at CMU has been an extraordinary environment to work in. Kiran Bhat, Louise Ditmore, Ralph Gross, Daniel Huber, Carolyn Ludwig, Iain Matthews, Pragyan Mishra, Daniel Morris, Bart Nabbe, Raju Patil, David LaRose, Stephanie Riso, and Jianbo Shi - you've all been great colleagues and friends.

Finally, life as a graduate student would never have been as much fun without my friends. Deepak Bapna, Nitin Mehta, Murali Krishna, Sameer Shah, Madhavi Vuppalapati, Vipul Jain, Rajesh Shenoy, Sita Iyer, Karthik Kannan, and Aneesh Koorapaty - thanks for the companionship over the years.

Contents

1 Introduction **1**

1.1 Modeling Dynamic Events from Images 2

1.2 Problem Definition 3

1.3 Overview of the Approach 5

1.4 Related Work 6

1.5 Thesis Outline 9

2 Dynamic Scene Properties **11**

2.1 Scene Flow: The Instantaneous Non-Rigid Motion of a Scene 12

2.2 Representing Scene Shape and Flow 13

2.2.1 The N dimensional Space of Scene Shape and Flow 13

2.2.2 Two Neighboring Time Instants: The 6D Space 13

2.2.3 Non-Coupled Representations of Shape and Flow 16

2.3 Related Work on Shape and Motion Estimation 16

2.3.1 Shape Estimation 16

2.3.2 Motion Estimation 18

3 Estimating Shape and Scene Flow **21**

3.1 Assumptions 21

3.2 6D Photo-Consistency Constraints 22

3.3 Computing 6D Hexel Occupancy 24

3.4 Review: Space Carving in 3D 25
3.5 A 6D Hexel Carving Algorithm 26
3.5.1 Visibility Within the Slab 29
3.5.2 Properties of the Flow Field 30
3.6 Experimental Results 32

4 Computing Scene Flow Given Shape **37**
4.1 Image Formation Preliminaries 38
4.1.1 Relative Camera and Surface Geometry 39
4.1.2 Illumination and Surface Photometrics 40
4.2 How are Scene Flow and Optical Flow Related? 41
4.2.1 Optical Flow 42
4.2.2 Three-Dimensional Scene Flow 43
4.3 Single Camera Case 45
4.3.1 Computing Scene Flow 46
4.3.2 Difficulty with Estimating Scene Flow from a Single Camera 47
4.4 Computing Scene Flow: Multi-Camera Case 47
4.5 Results 49
4.5.1 Performance 51
4.6 Three-Dimensional Normal Flow Constraint 52
4.7 Scene Flow: Which Algorithm to Use? 55

5 Spatio-Temporal View Interpolation **57**
5.1 High-Level Overview of the Algorithm 59
5.2 Flowing the Voxel Models 60
5.2.1 Shape Interpolation Using Scene Flow 60
5.2.2 Desired Properties of the Scene Flow for Voxel Models . . . 61
5.2.3 Enforcing the Desired Properties of Scene Flow 62
5.2.4 Results 63

5.3 Ray-Casting Across Space and Time 65
5.4 Ray-Casting to a Smooth Surface 68
5.5 Optimization Using Graphics Hardware 71
5.5.1 Intersection of Ray with Voxel Model 71
5.5.2 Determining Visibility of Cameras to Point on Model 72
5.6 Experimental Results and Discussion 73
5.6.1 Sequence 1: Paso Doble Dance Sequence 74
5.6.2 Sequence 2: Player Bouncing a Basketball 75
5.6.3 Performance 76

6 Conclusions **85**
6.1 Contributions 85
6.2 Future Work 88

A Image Acquisition Facility: 3D Room **91**

Bibliography **93**

List of Figures

1.1 Spatio-temporal view interpolation example 3
1.2 Overview of the approach 5

2.1 The 6D space of hexels 14

3.1 6-D Photo-consistency 23
3.2 Illustration of the 6D shape and motion carving algorithm 27
3.3 Multiple input frames from one camera, showing scene motion . . . 32
3.4 Recovered shapes using 6D carving 33
3.5 Recovered scene flow 34
3.6 Overlaid view of shape and scene flow 35
3.7 Colors used for photo-consistency of each voxel 36

4.1 Camera and surface geometry 38
4.2 Relationship between Scene Flow and Optical Flow 42
4.3 Computing $\frac{\partial \mathbf{x}}{\partial t}\Big|_{\mathbf{u}_i}$ 45
4.4 Multiple input frames from one camera, showing scene motion . . . 50
4.5 3-D model used for scene flow 51
4.6 Refined Scene Flow Vectors 52
4.7 Flow overlaid on shape 53
4.8 Magnitude of Scene Flow 53

5.1 4 input images and a novel view 58

5.2 Shape interpolated between two time instants 63
5.3 Effect of duplicate voxels 64
5.4 Schematic showing the ray-casting algorithm 66
5.5 Fitting a smooth surface to a voxel grid 68
5.6 Approximating a smooth surface through voxel centers 69
5.7 Rendering with and without surface fitting 70
5.8 Inputs in s-t space for dance sequence 78
5.9 Computed shapes for dance sequence 79
5.10 Computed flows for dance sequence 80
5.11 Collection of frames from dancer movie 81
5.12 Inputs for basketball sequence 82
5.13 Computed shapes and flows for basketball sequence 83
5.14 Collection of snapshots from basketball movie 84

A.1 The CMU 3D Room 91

Chapter 1

Introduction

The world around us consists of a large number of complex events occurring at any time. For example, a basketball game can be considered as an event that occurs within some area over a certain period. Similarly, a musical concert, a visit to a museum, or even a walk or daily chore are all examples of the millions of events that take place everyday. Clearly, most events around us are dynamic, meaning that things move in interesting ways, whether they are people, cars, animals, or other objects or natural phenomena that involve movement.

Photography and later, video, were exciting inventions in the 19th and 20th centuries. These let us capture and permanently archive the visual appearance of events around us, and let us re-live the moment at which they were captured, although in a restricted manner. A photographs only give us a snapshot from one position. Video does capture the motion in a dynamic event, but again, the position is either fixed or pre-determined.

Over the last 25 years, we have learned how to digitize photographs and video into a computer. However, these give us only 2-D representations of an event, and only capture the dynamic nature of an event in a limited way. A very useful capability would be if we could capture and digitize a dynamic event in its entirety. That way, we could re-render it from any viewpoint, similar to how it is possible to

view synthetic graphics models from an arbitrary position. In addition, we would also be able to play back any temporal segment of this re-rendered view, even at a speed different from the original (similar to slow-motion or fast-forward on normal video).

This ability to view an archived dynamic event, with complete control over spatial viewpoint, and the time and speed of occurrence of the event would enable a great immersive experience. Imagine a viewer being able to virtually fly around a basketball court, watching the game from any angle he chooses, even speeding up or slowing down the action as he views it! Similarly, such virtual renderings can be used in training and simulation applications, or even as dynamic scene content for video games. Camera angles for shots can be decided after the actual recording process in a movie production house.

1.1 Modeling Dynamic Events from Images

Synthetic 3-D computer models are used extensively in video games to enhance the visual immersive experience, and in computer animation for spectacular visual effects in movies. However, there is usually a distinct synthetic feel to such imagery; on careful inspection, it is not hard to tell the lack of visualism normally seen in images of the real world.

For the last 20-30 years, there has been extensive research in computer vision techniques looking into automatic creation of richer, three dimensional representations of the real world. More recently, there has been renewed interest in the problem of *image based* modeling and rendering, meaning that photographs are the basic input to an algorithm that attempts to re-create the geometry of the world, and/or re-render it from arbitrary new viewpoints.

Most recent image based modeling approaches have focused on modeling static scenes. We propose an image based approach to modeling dynamic events, by capturing images of the event from many different viewpoints simultaneously. By

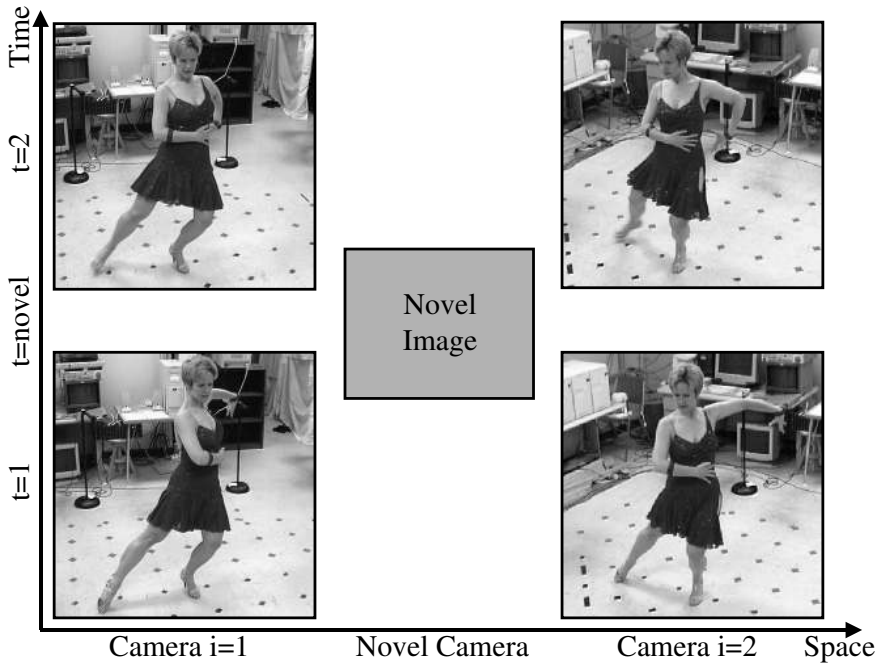


Figure 1.1: Spatio-temporal view interpolation consists of taking a collection of images of an event captured with multiple cameras at different times and re-rendering the event at an arbitrary viewpoint and time. In this illustrative figure, the 2 images on the left are captured with the same camera at 2 different times, and the 2 images on the right with a different camera at the same 2 time instants. The novel image and time are shown as halfway between the cameras and time instants but are really arbitrary.

creating a true dynamic model of the event, we can then re-create its appearance from any arbitrary position in space, at an arbitrary time during the occurrence of the event (irrespective of when and where the images are sampled from).

1.2 Problem Definition

We are interested in physical events that can be modeled as general time-varying scenes. For particular domain applications, it is sometimes possible to make simplifying assumptions about the scene, such as conformity to a parametrized or articulated model. Apart from physical limitations on workspace, we make no use of

domain knowledge - for our purposes, the event itself is equivalent to a scene with a few free-form surfaces arbitrarily changing shape in time. In addition, we wish to avoid interfering with the event; hence only video cameras, rather than any active imaging method are used to capture the action.

The problem of re-rendering a dynamic event with arbitrary control over viewer position and event speed comes down to one of being able to create a novel image with the desired viewing parameters by suitably combining the original images. Figure 1.1 presents an illustrative example of this novel view generation task which we call *spatio-temporal view interpolation.* The figure contains 4 images captured by 2 cameras at 2 different time instants. The images on the left are captured by camera C_1, those on the right by camera C_2. The bottom 2 images are captured at the first time instant and the top 2 at the second. Spatio-temporal view interpolation consists of combining these 4 views into a novel image of the event at an arbitrary viewpoint and time. Although we have described this in terms of 2 images taken at 2 time instants, our problem generalizes to an arbitrary number of images taken from an arbitrary collection of cameras spread over an extended period of time.

Before we can do spatio-temporal view interpolation, we need a way to establish corresponding points between the various input images, spread apart in space and in time. Thus, there is also the problem of computing a dynamic model of the event as a time-varying scene. Our problem can therefore be summarized as:

Compute a true dynamic model of a physical event, and reconstruct its appearance from an arbitrary point of view, at any arbitrary time within the occurrence of the event, by interpolating image data from input images captured from different positions and at different times.

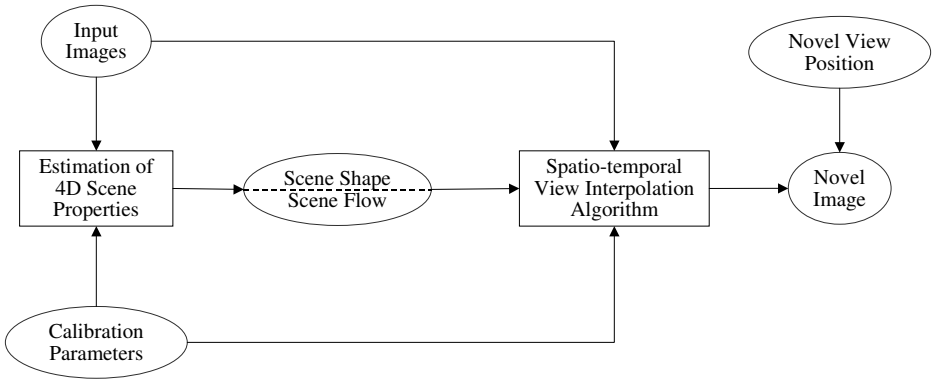


Figure 1.2: An overview of the various steps in our approach.

1.3 Overview of the Approach

Our approach for spatio-temporal view interpolation is based on the explicit recovery of scene properties. For a dynamic event, we need to understand the time-varying geometry of the scene and objects that comprise the event. We model the scene as consisting of one or more opaque objects, each of which may be changing shape in a non-rigid way, which we refer to as *motion* of the scene. We compute this motion by measuring *scene flow*, which we define as a first order measure of the instantaneous non-rigid motion of all objects in the scene.

Figure 1.2 shows the steps in our approach to render the appearance of the scene at any time. First, the event is imaged by multiple time-synchronized video cameras, each of which gives us a sequence of images at known time instants. Also, the calibration parameters for each of these cameras (both extrinsic and intrinsic) are estimated using the algorithm of [Tsai, 1986].

Using the input images and calibration parameters, we compute the scene shape and scene flow, to get a complete 4D model of the time-varying geometry and instantaneous motion of the scene. This is done at all time instants at which original images are captured. The 4D model is different from just a sequence of shapes, since the availability of shape and instantaneous motion at the sampled time instants

allows us to compute geometric information as a continuous function of time.

Then, the 4D scene properties, along with the input images and calibration parameters, become inputs to the spatio-temporal view interpolation algorithm. For any requested position (in space and time) of the novel view, the algorithm first estimates the interpolated scene shape at the desired time by flowing the computed shapes at the neighboring times using the scene flow. The points on this shape that correspond to each pixel in the novel image are determined, and then the corresponding points on the computed models at the sampled time instants are found. The known geometry of the scene at those times, with the camera calibration parameters is then used to project these corresponding points into the input images. The input images are sampled at the appropriate locations and the estimates combined to generate the novel image at the intermediate space and time, one pixel at a time.

1.4 Related Work

The work on Image-Based Visual Hulls [Matusik *et al.*, 2000] is closest in spirit to our approach. Using views from multiple cameras, their system reconstructs the Visual Hull [Laurentini, 1994], which is an approximation to the shape, of a dynamic scene in real time. This shape is then shaded efficiently using the original images, so that novel views of the scene can be rendered from arbitrary viewpoints. Both the construction of the visual hulls and the rendering occur in real time, however, there is no attempt to model motion of the scene or compute novel images between sampled time instants.

Almost all other related work on the synthesis of images from new viewpoints has been in the area of image based modeling and rendering of static scenes. In this section, we review some of these approaches. In order to get corresponding pixels across space and time, Our method explicitly recovers complete scene shape and motion, and various approaches to shape and motion recovery are discussed in

Chapter 2.

- **Image interpolation methods:** Movie maps [Lippman, 1980] was arguably the first image based rendering system. It used video-discs to store a number of pre-acquired video sequences, which are interactively played back to the user as they drive through a city, while controlling the route, angle, and speed. One of the first algorithms to actually synthesize new views of a scene by interpolating between existing images was View Interpolation [Chen and Williams, 1993], which used images and depth maps of a synthetic scene from two viewpoints. This image data was then forward mapped to the novel image using the depth to determine correspondences. View morphing [Seitz and Dyer, 1996] extended this approach with a pre-warping and post-warping step to ensure physically correct reconstruction when the novel camera and two input cameras were in a straight line, even under strong perspective projection. Even on real data, the results were impressive; it was clear that image based rendering could produce photo-realistic results.

- **Reprojection based algorithms:** [Avidan *et al.*, 1997] showed that for physically correct reconstruction, the novel camera wasn't limited to the line between two input views like in view morphing, if the trilinear tensor was used to specify the novel camera position in terms of the two input views. The interpolation of the views therefore happened in projective space. In a similar spirit, [Laveau and Faugeras, 1994a] represented a scene as a collection of images and fundamental matrices. The novel viewpoint would have to be specified by choosing the positions of image control points, which would then compute the fundamental matrices between the novel view and existing images. While the results are impressive, camera control is awkward with these systems. Another re-projection based system is Layered Depth Images [Gortler *et al.*, 1997], where multiple intensity images are warped and combined into a single image coordinate system, with layers for different depths.

Information about occluded surfaces is needed as the viewpoint moves away, and since these are stored at each pixel, the novel view can be rendered.

- **Plenoptic function based algorithms:** A different set of approaches focussed on recovering the plenoptic function [Adelson and Bergen, 1991], which is essentially the pencil of rays visible from any point in space. Plenoptic modeling [McMillan and Bishop, 1995] captured cylindrical panoramas from two nearby positions and after disparities were estimated, they could be warped to novel views. Lightfield rendering [Levoy and Hanrahan, 1996] and Lumigraph [Gortler *et al.*, 1996] adopt a different standpoint. Using a large number of input images from arbitrary positions, they reconstruct the complete lightfield or lumigraph, which is just the plenoptic function as a function of position and orientation in regions free of occlusions. The problem of generating novel views now comes down to querying the appropriate samples of the plenoptic function. If approximate scene geometry is known, the lumigraph produces better results, with fewer input images required. [Buehler *et al.*, 2001] is a newer approach that generalizes the lumigraph algorithm and uses 3D information more explicitly.

- **Use of explicit 3-D geometry:** A few approaches explicitly recovered 3-D geometry. [Sato *et al.*, 1997] used laser scanned range images and color images to create a high quality model of shape and reflectance, which allows for easy synthesis of novel views using standard graphics hardware. Later, [Nishino *et al.*, 1998] used eigenspace compression on input images to perform this interpolation more efficiently. [Debevec *et al.*, 1996a] developed a model based stereo system for computing architectural shapes and presented a view-dependent texture mapping method for rendering them. [Rander *et al.*, 1997] obtained dynamic texture mapped models of human-sized objects. These were computed by stereo reconstruction and volumetric merging from a number of widely separated views.

1.5 Thesis Outline

The outline of the remainder of this thesis is as follows. In Chapter 2, we discuss what the dynamic properties of a scene are, and formally define scene flow as a measure of the instantaneous motion of the scene. We describe various possible representations of scene shape and flow, and also related work in recovery of scene shape and motion.

In Chapter 3, we introduce the notion of photo-consistency across space and time, and describe an algorithm for unified recovery of shape and scene flow. Results of the algorithm run on real data are presented.

Chapter 4 extends the theory of scene flow to relate to the well-known optical flow, and describes an alternate algorithm for computing scene flow using smoothness constraints from optical flow, if the shape is already known. The merits and demerits of this and the unified algorithm are discussed.

An algorithm for spatio-temporal view interpolation using the computed shape and flow is presented in Chapter 5. First, an algorithm for shape interpolation using computed scene flow, and then the actual ray-casting algorithm for computing the novel image are presented. Results of creating novel views are shown for 3 real world dynamic events.

Finally, contributions of the thesis and ideas for future research are discussed.

Chapter 2

Dynamic Scene Properties

The first step towards modeling dynamic events is to understand the geometric properties of a dynamic scene. The scene can consist of many objects, each of which can be fixed or moving. A fundamental property of the scene is the shape of all of the objects in it. In addition, since the scene is dynamic, the motion of each object is typically some combination of rigid motion (such as rotation and translation), and non-rigid motion (bending, warping). Recall that we use *scene flow* as a measure of the instantaneous motion (rigid and non-rigid combined) of all parts of the scene. So while shape is a static (or zeroth order) description of the geometry of the scene, scene flow is a measure of the first order of motion (or velocity) of the scene. Since our approach for spatio-temporal view interpolation is based on explicit recovery of scene properties, we first formally define scene flow and then look into various representations for shape and flow. We also review related work in recovery of scene structure and motion.

2.1 Scene Flow: The Instantaneous Non-Rigid Motion of a Scene

Since the surfaces of all objects in the scene (or simply, the surface of the scene) are what are actually observed in the cameras, we define scene flow in terms of points on a surface. Let S^t be the surface of the scene at any given time instant t. Then, the scene flow F^t simply is

$$F^t = \frac{dS^t}{dt} \tag{2.1}$$

Assume that the surface S^t is somehow discretized so that there are V^t points on it. We represent it as a collection of points $\mathbf{X}_i^t$, so that

$$S^t = \{\mathbf{X}_i^t \mid i = 1, \ldots, V^t\} \tag{2.2}$$

where $\mathbf{X}_i^t = (x_i^t, y_i^t, z_i^t)$ is one of the V^t points on the surface at time t. Then, we can define the Scene Flow as the collection of the instantaneous motion of all of these points:

$$F^t = \{\frac{d\mathbf{X}_i^t}{dt} \mid i = 1, \ldots, V^t\} \tag{2.3}$$

Therefore, the scene flow is a representation of the scene motion, and is a dense three-dimensional vector field defined for every point on every surface in the scene. Knowledge of motion data such as scene flow has numerous potential applications ranging from motion analysis tasks, to motion capture for character animation. In addition, integrating the recovery of shape and motion into one procedure can potentially produce superior shape estimates compared to models obtained from images taken at a single time instant, because of the extra constraints provided by the flow.

2.2 Representing Scene Shape and Flow

2.2.1 The N dimensional Space of Scene Shape and Flow

The shape of the scene at any given time instant is 3-dimensional. As the scene changes in time, each point on the objects comprising the scene moves through some trajectory in space. Suppose we have N sampled time instants. While it takes 3 coordinates to describe the position of such a point at one time instant, it takes 3N such coordinates to completely describe the motion of such a point (in the absence of a higher level parametric motion model, such as a spline). Thus, the space of all possible shapes and flows over N sampled time instants, is actually 3N dimensional.

Because of this magnitude of increase in dimensionality of the space, we restrict ourselves to the case of two neighboring time instants, so the space of possible shapes and flows is now 6D. Note that this is more information than simply a pair of 3D shapes at two time instants. There is also the scene flow, that describes a correspondence from each 3D point at one time to a 3D point at the other time instant.

2.2.2 Two Neighboring Time Instants: The 6D Space

Consider the scene shape at two neighboring time instants, each of which is represented as a 3D grid of voxels. Now, the space of all possible shapes at two time instants, and the scene flow that relates them is 6D. Each 3D voxel at the first time could correspond to any point in an entire 3D space at the second time. We define a *hexel* as a point in this 6D space, characterized by the beginning and ending position of a particular voxel, representing a point on the surface of the object.

If a scene point moves from $\mathbf{x}^1 = (x^1, y^1, z^1)$ at $t = 1$ to $\mathbf{x}^2 = (x^2, y^2, z^2) = (x^1 + \Delta x^1, y^1 + \Delta y^1, z^1 + \Delta z^1)$ at $t = 2$, there are two natural ways of representing the space of possible scene shapes and flows. One possibility is the asymmetric 6D space of all 6-tuples:

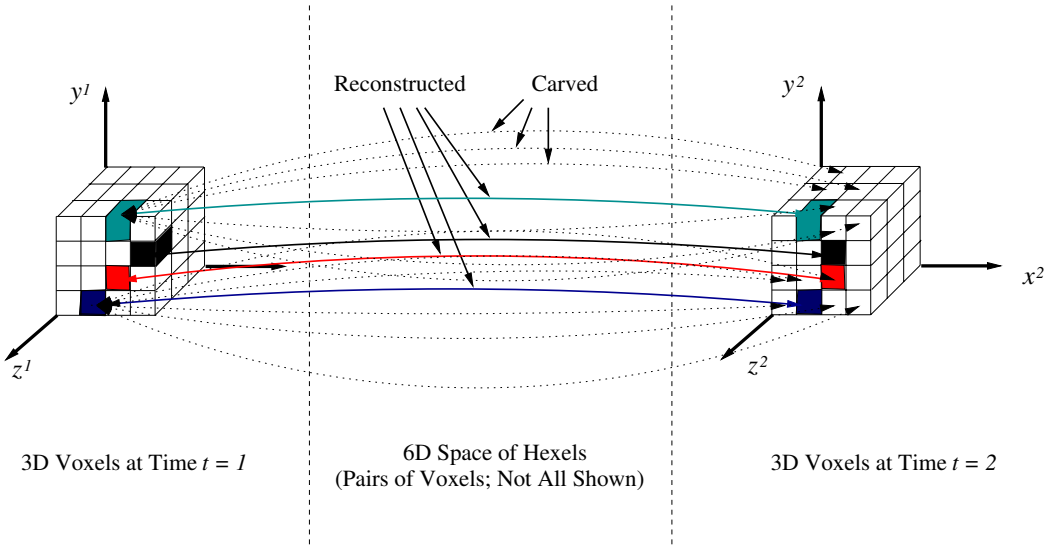


Figure 2.1: An illustration of the 6D space of hexels. A hexel can be regarded as a pair of corresponding 3D voxels, one at each time instant. The 6D hexel $(x^1, y^1, z^1, x^2, y^2, z^2)$ defines the two voxels: (x^1, y^1, z^1) at $t = 1$ and (x^2, y^2, z^2) at $t = 2$. It also defines the scene flow $(\Delta x^1, \Delta y^1, \Delta z^1) = (x^2 - x^1, y^2 - y^1, z^2 - z^1)$ that relates them. Only a subset of all possible hexels are valid; the problem of recovering shape and scene flow boils down to determining which ones are valid and which ones are not.

$$(x^1, y^1, z^1, \Delta x^1, \Delta y^1, \Delta z^1). \tag{2.4}$$

Here, the shape (x^1, y^1, z^1) at time $t = 1$ and the scene flow $(\Delta x^1, \Delta y^1, \Delta z^1)$ uniquely determine the shape at time $t = 2$, as above. A second possible representation is the symmetric 6D space of 6-tuples:

$$(x^1, y^1, z^1, x^2, y^2, z^2). \tag{2.5}$$

Here, the shapes at the two times uniquely determine the flow

$$(\Delta x^1, \Delta y^1, \Delta z^1) = (x^2 - x^1, y^2 - y^1, z^2 - z^1) \tag{2.6}$$

We choose to treat both time instants equally and work in the symmetric space,

mainly because it leads to a more natural algorithm for shape interpolation and novel view generation. One advantage of the asymmetric definition, however, is that the space can be kept smaller, since $\Delta\mathbf{x}$ usually has a much smaller range of magnitudes than $\mathbf{x}$. The asymmetric representation may therefore be more natural for some other algorithms, particularly ones that are extrapolating the shape computed at $t = 1$ as a means of predicting that at $t = 2$.

Therefore, a hexel is a 6D vector of the form in Equation (2.5), analogous to a voxel in 3D. Figure 2.1 contains an illustration of the 6D hexel space. The hexel space is the Cartesian product of the two voxel spaces; i.e., it is the set of all pairs of voxels, one at each time instant. It is a single entity that defines one voxel at each time, and the scene flow that relates them. Therefore, it does not exist at either time, but simultaneously at both.

If a hexel $(x^1, y^1, z^1, x^2, y^2, z^2)$ is occupied, then the voxel (x^1, y^1, z^1) lies on the surface of the scene at $t = 1$, and has a single matching voxel (x^2, y^2, z^2) at $t = 2$, that also lies on the surface. If the hexel is not occupied, then it means that there is no valid correspondence between the same two points.

Thus, we see that determining which hexels are occupied and which ones are not amounts to the reconstruction of the shape of the scene, and a valid scene flow. In reality, the surface is a 2D manifold at one time. As will be shown in the next chapter, we formulate the problem of simultaneously computing shape and motion as one of determining which points in the 6D space lie on this 2D space-time manifold. Our algorithm operates by *carving away* hexels that do not lie on the manifold. In particular, a hexel is carved if it corresponds to points at the two time instants that are not photo-consistent, i.e., whose projections do not agree in all images at each time instant, and also between time instants.

2.2.3 Non-Coupled Representations of Shape and Flow

The representation of scene shape and flow in a higher dimensional space is best suited for an algorithm that recovers both simultaneously in such a volumetric space. However, there are alternate representations possible, where the shape and flow are independently recovered.

The surface of the scene may be defined analytically, using either an implicit surface function or as a combination of splines. Then, the scene flow is simply a 3D vector field defined on the 2D surface manifold. This vector field defines the instantaneous motion at any point on the manifold, and therefore is a complete representation of the shape and first order non-rigid motion of the scene.

In practice, image-based shape reconstruction is often easier when the shape is represented as an occupancy grid of voxels. At any time instant t, if there are V^t voxels identified as surface voxels, the shape S^t is defined as:

$$S^t = \{\mathbf{X}_i^t \mid i = 1, 2, ...V^t\} \tag{2.7}$$

where $\mathbf{X}_i^t$ is simply the co-ordinate of the center of each voxel. Every such voxel $\mathbf{X}_i^t$ has a unique scene flow $\mathbf{F}_i^t$ associated with it, which is just a 3×1 vector for each of the three components of the scene flow. The set S^t and F^t, where $F^t = \{\mathbf{X}_i^t \mid i = 1, 2, ...V^t\}$ represent the dynamic properties of the scene at time t.

2.3 Related Work on Shape and Motion Estimation

2.3.1 Shape Estimation

There have been many different varieties of algorithms proposed for estimating the shape of a scene at a single time instant. We briefly review a few representative algorithms for shape estimation.

The classic stereo problem of recovering scene depth from two cameras was

proposed by [Marr and Poggio, 1979]. Since then, there have been many different approaches and improvements to correspondence-based dense stereo. [Ohta and Kanade, 1985] use dynamic programming to determine correspondences between edge features. [Fua, 1993] described a correlation based multi resolution algorithm to compute dense depth maps. [Okutomi and Kanade, 1993] developed a multi-baseline stereo algorithm that can uses multiple baselines between different pairs of cameras. In [Kang and Szeliski, 1997], stereo matching was done from multiple panoramas of a 3D scene. A more comprehensive survey of stereo algorithms can be found in [Szeliski and Zabih, 1999].

In recent years, multi-camera stereo approaches have become increasingly popular. Shape from silhouette, first proposed by [Baumgart, 1974], is the simplest volumetric shape reconstruction algorithm if object silhouettes are available from multiple viewpoints. It reconstructs the visual hull which is a superset of the true shape, but with a large enough number of views, often produces reasonable results. [Collins, 1996] uses a plane sweep algorithm, which counts the number of image features back-projected to a voxel to determine occupancy. Similarly, Space carving [Seitz and Dyer, 1999] volumetrically reconstructs the scene by keeping voxels that project to consistent colors in the various images. [Narayanan *et al.*, 1998] used a multi-baseline stereo algorithm with volumetric merging to compute 3D shape. In [Zitnick and Kanade, 1998], shape is recovered using a stereo algorithm iteratively in volumetric space by using a likelihood function that weights uniqueness and continuity. A different algorithm is proposed by [Debevec *et al.*, 1996b] which describes model-based stereo, a human assisted system for producing high quality models from multiple images. In addition, there is a large body of work on recovering shape from range images [Curless and Levoy, 1996, Sato *et al.*, 1997, Hilton *et al.*, 1996, Wheeler *et al.*, 1998].

Structure from Motion (SFM) is a sparse feature based shape recovery method. Multiple images of a static object are captured from different viewpoints, and a sparse set of common features are tracked across frames, and used to recover both

the shape and relative motion of the camera. The factorization algorithm [Tomasi and Kanade, 1992] is a popular approach to the SFM problem, and although initially an orthographic camera was assumed, the approach has been extended to perspective projection as well [Christy and Horaud, 1996].

2.3.2 Motion Estimation

The term *motion* has been used in many different ways, sometimes referring to the motion of the camera with respect to objects in the scene, or rigid/non-rigid motion of one or more of the objects themselves.

In the case of rigid motion, the image varies in a highly constrained manner as either the object or camera moves. This simple observation has led to a large body of techniques for reconstructing rigid scenes from multiple image sequences. (See, for example, [Waxman and Duncan, 1986, Young and Chellappa, 1999, Zhang and Faugeras, 1992].)

The problem of modeling non-rigid scenes from image sequences is far less well-understood, an unfortunate fact given that much of the real world moves non-rigidly. A few approaches have attempted to recover 3-D non-rigid motion from a single camera. [Pentland and Horowitz, 1991] and [Metaxas and Terzopoulos, 1993] assume that the scene can be represented as a deformable model. [Ullman, 1984] assumes that the motion minimizes the deviation from a rigid body motion. In both of these approaches (and in general), it is not possible to compute non-rigid motion from a single camera without the use of *a priori* assumptions about the scene. See [Penna, 1994] for a survey of monocular non-rigid motion estimation.

A major difficulty with modeling non-rigid motion is the lack of general-purpose constraints that govern it. In order to keep the problem tractable, previous research has therefore focused on specific objects and motions such as cardiac motion [Pentland and Horowitz, 1991], articulated figures [Bregler and Malik, 1998], faces [Guenter *et al.*, 1998], and curves [Carceroni and Kutulakos, 1999]. The last

approach has since been generalized in [Carceroni and Kutulakos, 2001] to use surface elements - recovering the shape, motion, and reflectance of a number of surface patches without explicit model constraints.

Another common approach to recovering three-dimensional motion is to use multiple cameras and combine stereo and motion in an approach known as *motion-stereo*. Nearly all motion-stereo algorithms assume that the scene is rigid. See, for example, [Waxman and Duncan, 1986], [Young and Chellappa, 1999], and [Zhang and Faugeras, 1992]. A few motion-stereo papers do consider non-rigid motion, including [Liao *et al.*, 1997] and [Malassiotis and Strintzis, 1997]. The former uses a relaxation-based algorithm to co-operatively match features in both the temporal and spatial domains. It therefore does not provide dense motion. The latter uses a grid which acts as a deformable model in a generalization of the monocular approaches mentioned above.

Chapter 3

Estimating Shape and Scene Flow

We now consider the problem of computing the dynamic scene properties - scene shape and flow, at any single time instant. As mentioned in the previous chapter, the dimensionality of the space of all possible shapes and flows grows linearly with more time instants considered. We therefore focus on the 6D space of two neighboring time instants, and develop a unified algorithm to simultaneously recover both shapes, and the scene flow connecting them. The hexel representation leads naturally to such an algorithm; we first review some of the assumptions made regarding the motion.

3.1 Assumptions

The scene is assumed to consist of an arbitrary number of opaque objects, each of which can move in a non-rigid manner. We do not have any higher level domain knowledge about the nature of the motion in the scene. Rather than making any strong assumptions, we propose instead to use very general constraints on the motion of the scene:

First, we assume that a particular point on an object projects to pixels of approximately the same color in all images at consecutive time instants. This constraint

is an extension of the notion of photo-consistency, introduced in [Seitz and Dyer, 1999], to time-varying scenes. It is assumed that the surface is lambertian, so that any point in the scene appears to be the same color from any viewing direction. In addition, the constraint also assumes brightness constancy, so that a point in the scene has the same color over time. For small motions of the scene and camera, both of these are reasonable assumptions.

Next, we assume that the motion between frames is finite. This constraint imposes a very weak form of regularization which improves reconstruction accuracy without penalizing complex shapes or motions. In practice, this weak regularization is enforced by restricting the maximum amount of scene flow between two neighboring time instants. This also reduces the search space, making the algorithm more efficient.

We assume that the cameras are calibrated, so that both the intrinsic parameters (focal length, center of projection, skew, lens distortion) and extrinsic parameters (position, orientation) are known. Finally, we assume that the cameras are placed so that every part of the scene is imaged at each time instant by at least two cameras, so that we can recover dense shape and flow.

3.2 6D Photo-Consistency Constraints

Recall that the 6D space of all hexels is descriptive of all possible shapes at two neighboring time instants - and of all possible flows connecting them. We now formulate the extended photo-consistency constraint in the 6D hexel space.

Suppose that the scene is imaged by the cameras $\mathbf{P}_i$. The image projection $\mathbf{u}_i = (u_i, v_i)$ of a scene point $\mathbf{x} = (x, y, z)$ by camera $\mathbf{P}_i$ is expressed by the relation

$$u_i = \frac{[\mathbf{P}_i]_1 (x, y, z, 1)^{\mathrm{T}}}{[\mathbf{P}_i]_3 (x, y, z, 1)^{\mathrm{T}}} \tag{3.1}$$

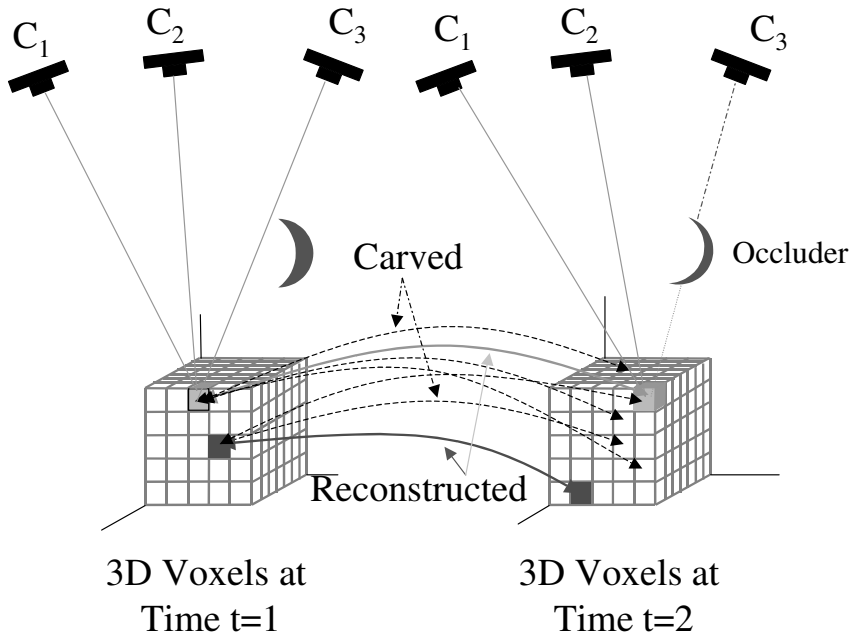


Figure 3.1: The 6-D photo-consistency shown for two neighboring time instants.

$$v_i \quad = \quad \frac{[\mathbf{P}_i]_2 \, (x, y, z, 1)^{\mathrm{T}}}{[\mathbf{P}_i]_3 \, (x, y, z, 1)^{\mathrm{T}}} \tag{3.2}$$

where $[\mathbf{P}_i]_j$ is the j^{th} row of $\mathbf{P}_i$. The images captured by the i^{th} camera at $t = 1$ and $t = 2$ are denoted $I_i^1(\mathbf{u}_i)$ and $I_i^2(\mathbf{u}_i)$ respectively.

A hexel $(x^1, y^1, z^1, x^2, y^2, z^2)$ is said to be *photo-consistent* if (x^1, y^1, z^1) and (x^2, y^2, z^2) project to pixels of approximately the same color in all of the cameras that see these two points. Note that this definition of photo-consistency is stronger than that introduced in [Seitz and Dyer, 1997] because it requires that the points have the same color: (1) from all viewpoints and (2) at both instants in time. These are the same as the assumptions of lambertian surface model and brightness constancy. Figure 3.1 shows a sample hexel space; the photo-consistent hexels are shown with solid lines, these are the ones that project to similar colors in all the cameras that they can be seen from. The hexels shown by a dotted line are ones that

do not project to a single color, and are therefore not photo-consistent.

Algebraically, we define hexel photo-consistency using the following measure:

$$\mathrm{Var}\left(\bigcup_{\mathrm{Vis}^1(\mathbf{x}^1)}\left\{I_i^1(\mathbf{P}_i(\mathbf{x}^1))\right\} \cup \bigcup_{\mathrm{Vis}^2(\mathbf{x}^2)}\left\{I_i^2(\mathbf{P}_i(\mathbf{x}^2))\right\}\right). \tag{3.3}$$

Here, $\mathrm{Var}(\cdot)$ is the variance of a set of numbers and $\mathrm{Vis}^t(\mathbf{x})$ is the set of cameras $\mathbf{P}_i$ for which $\mathbf{x}$ is visible at time t. Clearly, a smaller variance implies a greater likelihood that the hexel is photo-consistent.

Many other photo-consistency functions could be used instead, including robust measures. One particularly nice property of the variance, however, is that the 6D space-time photo-consistency function can be expressed as a combination of two 3D spatial photo-consistency functions. If we store the number of cameras n^t as the size of the set $\mathrm{Vis}^t(\mathbf{x}^t)$, the sum of the intensities $S^t = \Sigma_i I_i^t(\mathbf{P}_i(\mathbf{x}^t)$, and the sum of the squares of the intensities $SS^t = \Sigma_i[I_i^t(\mathbf{P}_i(\mathbf{x}^t)]^2$ for the two spaces (x^t, y^t, z^t), $t = 1, 2$, then the photo-consistency of the hexel $(x^1, y^1, z^1, x^2, y^2, z^2)$ is:

$$\frac{SS^1 + SS^2 - (S^1 + S^2) * (S^1 + S^2)}{n^1 + n^2}. \tag{3.4}$$

If we store S^1, S^2, SS^1, SS^2 (each of which is a 3D function), then the 6D photo-consistency measure in Equation (3.3) can be quickly computed from the stored representation using Equation (3.4). This is far less memory intensive than attempting to store the complete 6D photo-consistency, with a very small computational overhead.

3.3 Computing 6D Hexel Occupancy

The next step is to develop an algorithm to compute scene shape and flow from the photo-consistency function. After discretizing the volume at both time instants into volumetric grids, we need to determine which hexels (see Figure 3.1) represent

valid shape and flow; i.e., for which tuples $(x^1, y^1, z^1, x^2, y^2, z^2)$ there is a point (x^1, y^1, z^1) at time $t = 1$ that flows to (x^2, y^2, z^2) at time $t = 2$. Thus, the reconstruction problem can be posed as determining a binary hexel occupancy function in the discretized 6D space.

For each hexel there are therefore two possibilities; either it is reconstructed or it is carved. If it is reconstructed, the two voxels that it corresponds to are both reconstructed and the scene flow between them is determined. In this case, the hexel can be thought of as having a color; i.e., the color of the two voxels (which must be roughly the same if they are photo-consistent across time). If a hexel is carved however, it does not imply anything with regard to the occupancy of the two voxels; it just says that this particular match between the two is a bad one. For a voxel to be carved away, an entire 3D subspace of hexels (which corresponds to all possible flows for this voxel) has to be searched, and no valid match found.

We estimate the hexel occupancy function via a 6D carving algorithm; i.e., we initially assume that all of the hexels are occupied, meaning that no shape and flow hypotheses have been eliminated. We then remove any hexels that we can conclude are not part of the reconstruction. The result gives us an estimate of the scene shape at both time instants and the scene flow relating them.

3.4 Review: Space Carving in 3D

Our algorithm is closely related to the 3D voxel coloring algorithm of [Seitz and Dyer, 1997] and subsequently, Space Carving [Kutulakos and Seitz, 1999]. Therefore, we first briefly review the voxel coloring approach before describing our algorithm.

Voxel coloring is a simple algorithm to compute the shape of a static scene from a number of spatially separated images. The locations of the cameras are assumed to be known, as are the intrinsic calibration parameters of the cameras. It operates as follows. First, the volume of interest is partitioned into a voxel grid. Then, a plane

is swept across the volume, one layer of voxels at a time. Within each plane, voxels are sequentially evaluated for photo-consistency. Each voxel is first projected into the set of visible images. Then, the decision whether to retain or carve a voxel is simple; a voxel is carved if its photo-consistency is above a threshold, otherwise the voxel is retained. The measure of photo-consistency is simply the variance of the color values from all the visible cameras.

Note that this determination of the set of visible cameras is critical, but non-trivial. As can be seen from Equation (3.3), photo-consistency cannot be computed until the set of cameras that view the voxel is known. This visibility, in turn, depends on whether other voxels are carved or not. In voxel coloring, decisions for the voxels are ordered in a way such that the decisions for all potential occluders are made before the decision for any voxel they may occlude.

A special case is when the scene and the cameras are separated by a plane. Then the carving decisions can be made in the correct order by sweeping the plane through the scene in the direction of the normal to the plane. To keep track of visibility, a collection of 1-bit masks are used, one for each input image. The masks keep track of which pixels are accounted for in the reconstruction. For each voxel considered, the color from a particular camera is only used if the mask at the pixel corresponding to the projection of the voxel has not already been set, implying that no other voxel along the line of sight is occupied. If the voxel is reconstructed (not carved) the pixels that it projects to are masked out in all of the cameras.

3.5 A 6D Hexel Carving Algorithm

We now generalize the 3D plane sweep algorithm to the 6D hexel space. In order to estimate the scene shape at two neighboring time instants and the flow between them, we need to carve hexels in 6D. The algorithm proceeds by starting with the complete 6D hexel space, and carving away hexels that are not consistent with images at both times. It operates by sweeping a *slab* (a thickened plane) through

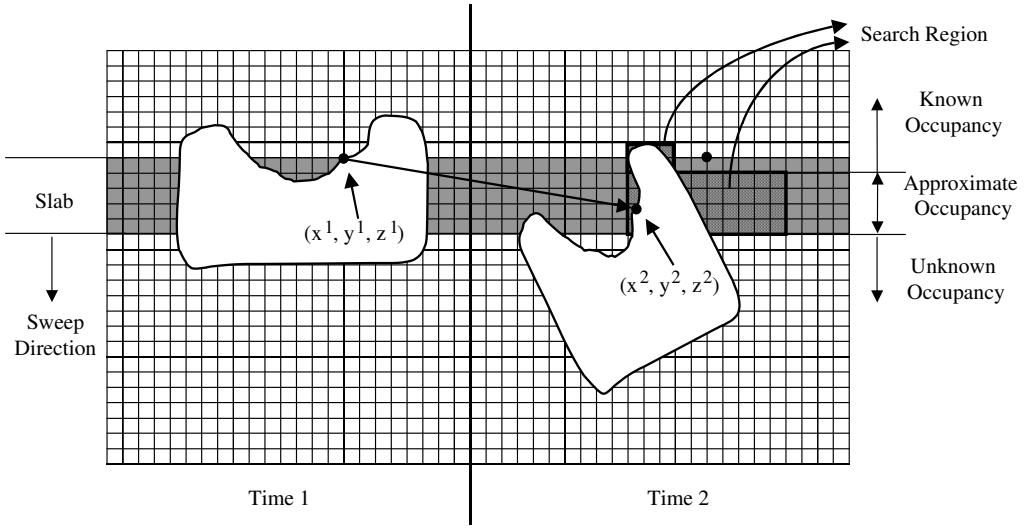


Figure 3.2: An illustration of the 6D Slab Sweeping Algorithm. A thick plane, or *slab*, is swept through the scene simultaneously for both $t = 1$ and $t = 2$. Occupancy decisions are made for each voxel on the top layer of the slab by searching the other time instant to see whether a matching voxel can be found. While the set of cameras visible to any voxel above the slab is known, the visibility of voxels in the slab (except for the top layer) needs to be approximated. See Section 3.5.1 for more details.

the volume of the scene. The slab is swept through the space simultaneously for both $t = 1$ and $t = 2$, as is illustrated in Figure 3.2. (In the figure, the slab sweeps from top to bottom.) For each position of the slab, all of the voxels on the top layer are considered and a decision made whether to carve them or not. We assume that the cameras and scene can be separated by a plane, so the shape and motion are simultaneously recovered at both times in a single pass sweeping the slab through the scene.

We also assume that there is an upper bound on the magnitude of the scene flow, and set the thickness of the slab to be this value. At any particular voxel, we therefore only need to consider hexels for which the other endpoint is either in the slab, or above the slab by a distance less than the width of the slab. This saves us

from having to consider the entire 3D subspace of hexels that have this voxel as an endpoint. We describe the algorithm for the voxel $\mathbf{x}^1 = (x^1, y^1, z^1)$ at $t = 1$, as is shown in the figure. The steps for $t = 2$ are of course the same, switching the roles of $t = 1$ and $t = 2$. For each position of the slab, we perform the following steps for each voxel in the top layer.

1. Compute the visibility and color statistics: The visibility of $\mathbf{x}^1$ is computed by projecting it into the cameras. If the pixel it projects to is already masked out (as in 3D space carving), $\mathbf{x}^1$ is occluded in that camera. Otherwise, the color of the pixel is incorporated into the color statistics n^1, S^1, and SS^1 needed to compute the photo-consistency using Equation 3.4.

2. Determine the search region: The search region is initialized to be a cube at the other time, centered on the corresponding voxel. The length, width, and height of the search region are all set equal to twice the maximum flow magnitude. This cube is then intersected with the known surface above the slab. (See Figure 3.2.) The search region defines a set of hexels to be searched, each of which corresponds to a possible hypothesis for the scene flow of $\mathbf{x}^1$.

3. Compute all the hexel photo-consistencies: The combination of $\mathbf{x}^1$ and each voxel in the search region corresponds to a hexel. For each such hexel, the visibilities and color statistics of the voxel being searched are computed and combined with those for $\mathbf{x}^1$ using Equation 3.4 to give the photo-consistency value for this hexel. (Computing the visibility for the voxel in the search region is non-trivial: if the voxel is above the slab, the visibility is known and was already computed when the voxel was in the top layer. But if the voxel is in the slab, it is impossible to compute the visibility without carving further into the scene. This step is involved and is described in Section 3.5.1.)

4. Hexel carving decision: The hexel that has the best photo-consistency value is found. If the photo-consistency measure (variance) is above a threshold, all of

the hexels in the search region are carved. This also means that the voxel $\mathbf{x}^1$ is carved. Otherwise, the best matching hexel is reconstructed, which means that $\mathbf{x}^1$ and the corresponding voxel at the other time are reconstructed and the scene flow between them computed. All of the other hexels are carved (a step which is equivalent to eliminating all of the other flow possibilities for $\mathbf{x}^1$.)

5. Update the visibility masks: If $\mathbf{x}^1$ was reconstructed (not carved) it is projected into all of the cameras. The masks at the projection locations are set. (The masks in the cameras for which $\mathbf{x}^1$ is not visible will already be set and so are not changed by this operation.)

3.5.1 Visibility Within the Slab

It is easy to show that it is impossible to determine the visibility below the top layer in the slab without first carving further into the slab.

Hypothesis: *Apart from the cameras which are occluded by the structure that has already been reconstructed above the slab, it is impossible to determine whether* **any** *of the other cameras are visible or not for voxels below the top layer (and that are not on the sides of the slab) until some occupancies in the top layer or below are known.*

Proof: For any voxel below the top layer, if all of the voxels on the top layer and the sides of the slab turn out to be occupied, none of the cameras will be visible. On the other hand, if all of the voxels below the top layer turn out not to be occupied, all of the cameras that are not already occluded will be visible. □

Without making assumptions, such as that the visibility of a point does not change between its initial and its flowed position, we therefore have to carve into the slab to get an approximation of the visibility. We do this by performing a space

carving in the slab, but with a high photo-consistency threshold. This gives us a thickened estimate of the surface because of the high threshold. A thickened surface will give an under-estimate of the visibility (at least on the surface) which is preferable[1] to an over-estimate of the visibility, which might arise if we chose a lower threshold and mistakenly carved away voxels on the true surface.

Space carving in the slab only defines the visibility on or above the thickened surface. To obtain estimates of the visibility below the thickened surface, we propagate the visibility down below the surface assuming that there are no local occlusions between the thickened surface and the true surface. This assumption is reasonable, either if the thickening of the surface caused by the high threshold carving is not too great, or if the range of motions (i.e., the size of the search region) is small relative to the size of local variations in the shape of the surface.

3.5.2 Properties of the Flow Field

The flow field produced by our algorithm will not, in general, be bijective (one-to-one). This bijectivity, however, is not desirable since it means that the shapes at the two times must contain the same number of voxels. Contractions and expansions of the surface may cause the "correct" discrete surfaces to have different numbers of voxels. Hence, bijectivity should not be enforced.

Ideally we want the flow field to be well defined for each visible surface voxel at both time instants. That is, the flow should be to a visible surface voxel at the other time. The algorithm described above does not guarantee that the flow field is well defined in this sense. It is possible for a voxel that is reconstructed in Step 4 to later appear in the top layer, but not be visible in any of the cameras. Such a voxel is not a surface voxel and should not have a flow.

To make sure that flow field is only defined between visible voxels on the two

[1] An under-estimate of the visibility is preferable to an over-estimate because under-estimates cannot lead to false carving decisions. On the other hand, mistakenly using a camera could lead to the addition of an outlier color and the carving of a hexel in the correct reconstruction.

surfaces, we add a second pass to our algorithm. We perform the following two operations on any voxels that were reconstructed by Step 4 and which later turned out not to be visible in any of the cameras when they appeared in the top layer:

6. Carve the hexel: Since this interior voxel is one endpoint of a reconstructed hexel, that hexel (and this voxel) are carved.

7. Find the next best hexel: Find the corresponding voxel at the other time (i.e., the one at the other end of the hexel just carved) using the flow stored in this voxel. Repeat Steps 2–4 for that voxel to find the next best hexel (even if the photo-consistency of that hexel is above the threshold). Since the slab has passed through the entire scene once, the search region can be limited to voxels known to be on the surface.

With this second pass, our algorithm does guarantee the property that the flow field is only defined for surface voxels. Note that guaranteeing this property is only possible in an efficient manner by assuming that if the best matching hexel in Step 4 turns out to have an endpoint that is not on the surface, then another hexel can be found in Step 7 that is also photo-consistent (and for which the other endpoint is a surface voxel). This assumption is reasonable since the other endpoint of the best matching hexel will likely be close to the surface to start with, and because the photo-consistency function should be spatially continuous.

This two-pass procedure may also be used to model any inherent ambiguities in the motion field, such as the *aperture problem*. For example, the aperture problem manifests itself in the form of multiple hexels passing the photo-consistency threshold test in Step 4 of the algorithm. Since all of these hexels yield photo-consistent hypotheses for the flow, the flow cannot be estimated without motion (smoothness) constraints. In our carving algorithm we choose the most photo-consistent hexel to yield a unique flow at every reconstructed voxel, followed by simple averaging over a small $3 \times 3 \times 3$ spatial volume.

For the specific application of shape interpolation/morphing, a bijective flow field is necessary. We discuss this issue in detail in Chapter 5.

3.6 Experimental Results

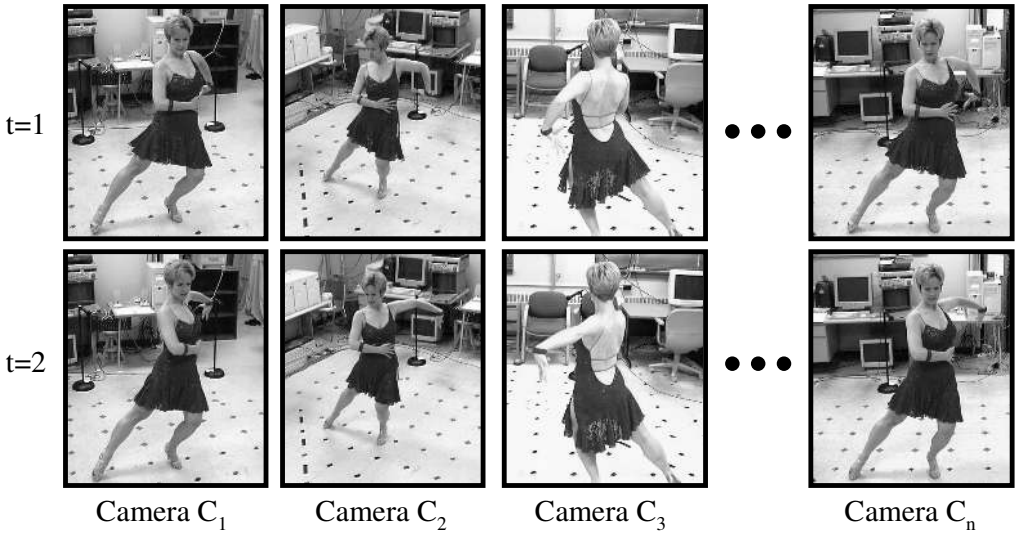


Figure 3.3: A sequence of images that show the scene motion from different cameras, at two consecutive time instants.

We demonstrate results of our 6D carving algorithm on a collection of 14 image pairs (one set at t = 1, the other at t = 2). The image pairs are taken from different cameras viewing a dynamic event consisting of a dancer performing a flamenco dance piece. The input image pairs from a few of the 14 cameras are shown for both time instants in Figure 3.3. The image sequences are all captured using the CMU 3D Room system, described in Appendix A.

At the moment the images at t = 1 were captured, the dancer is twisting her torso to the left, as she stretches out her right arm. The hexel grid was modeled as the Cartesian product of two 3D voxel grids of size $80 \times 80 \times 120$. It therefore contains 80^4x$120^2 \approx 10^{11}$ hexels. The maximum flow was set to be 8 voxels (corresponding

(a) t=1 (b) t=2

Figure 3.4: The reconstructed shapes at time $t = 1$ and $t = 2$ using the algorithm.

to 20cm in the scene). Therefore the search region in Step 4 of the algorithm had a maximum size of 16x16x16 voxels. This reduces the number of hexels considered to $80^2 \times 120 \times 16^3 \approx 10^9$.

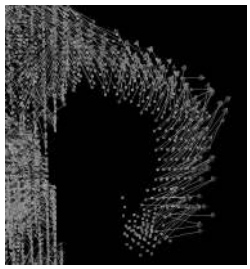
Figure 3.4 shows two views of the shapes obtained by the 6D carving algorithm at the two time instants. To give a rough idea of the size of the voxels, voxels are displayed with different random grey levels. The shapes are obtained simultaneously after a 6D slab sweep, and a second pass to enforce the surface flow properties described in the previous section. The reconstruction consists of approximately 60000 hexels, and the algorithm takes approximately 4 minutes to run on an R10000 SGI O2, using 250 MB of memory. The sequence was imaged by 18 cameras, which were all used for the reconstruction (The computation time is linear in the number of images used).

Figure 3.5: The recovered scene flow shown as a 3-D needlemap. The flow recovered is dense, so every surface voxel on the shapes shown in Figure 3.5 has a flow vector. In this view, the flow is shown for all voxels (including ones on hidden surfaces).

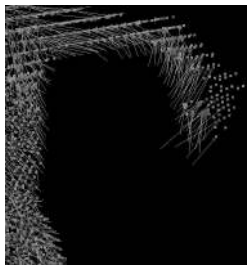
Figure 3.5 shows the computed scene flow over the entire volume. The scene flow vectors are displayed as needle maps with an arrowhead drawn at the $t = 2$ endpoint of each vector. Figure 3.6 shows the same scene flow overlaid with the shape. The two close-ups on the right show the instantaneous outward motion of the dancer's left arm. At $t = 1$ and $t = 2$, it is seen that the voxels are at either end of the arrows showing the scene flow, thus verifying that the computed scene flow is indeed a reasonably accurate measurement of instantaneous motion of the scene.

Figure 3.7 shows the colors that are computed for each voxel, and used for com-

(a) Shape + Flow



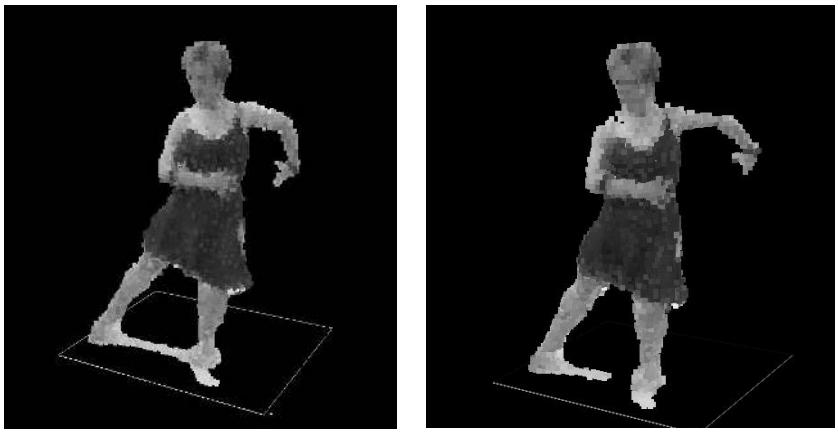
(b) t=1



(c) t=2

Figure 3.6: Figure (a) shows the scene flow overlaid on the computed shape. (b) shows a close-up of the shape at t = 1with the scene flow vectors, while (c) shows the same vectors with the shape at t = 2. Notice the motion of the voxels from the base to the tip of the arrows.

puting the photo-consistency in equation 3.4. The colors shown for the voxels at each time instant are obtained by computing the average of the color contributed by each camera that sees a particular voxel. During the carving algorithm, the most consistent hexel is chosen, which gives us a pair of surface voxels with similar colors, one at each time instant. In fact, since visibility is resolved during the sweep, all voxels at either time instant can be classified as carved, surface, or interior voxels. This image shows only the surface voxels (note that there are no significant holes produced, which can often be a problem with voxel coloring style approaches [Seitz and Dyer, 1999]).



(a) t=1 (b) t=2

Figure 3.7: At both time instants, the color that is computed for each voxel and used for the 6-D photo-consistency test is shown.

Chapter 4

Computing Scene Flow Given Shape

In the previous chapter, we described a shape and motion carving algorithm to simultaneously compute shape at two time instants, and the flow between them. The algorithm uses the photo-consistency function to carve away hexels that are not consistent with the various images. The computation of the scene flow is completely independent for neighboring voxels on either shape. Because no local smoothness is assumed, the flow is often quite noisy even within small local regions.

Since motion in the real world is inherently smooth, introducing such prior information into the computation of scene flow seems like a natural choice. Optical flow (used to compute 2-D displacements between images) uses regularization as such a smoothness prior [Barron *et al.*, 1994]. Because of the close connection between 2-D optical flow and 3-D scene flow, we study this relationship in more detail. We then present an algorithm for computing smooth scene flow given shape, using the inherent smoothness in optical flow fields. We also compare this algorithm with the unified shape and flow approach presented in the previous chapter. First, we look at some of the basics of the geometry behind image formation and develop the notation used for the scene flow algorithm.

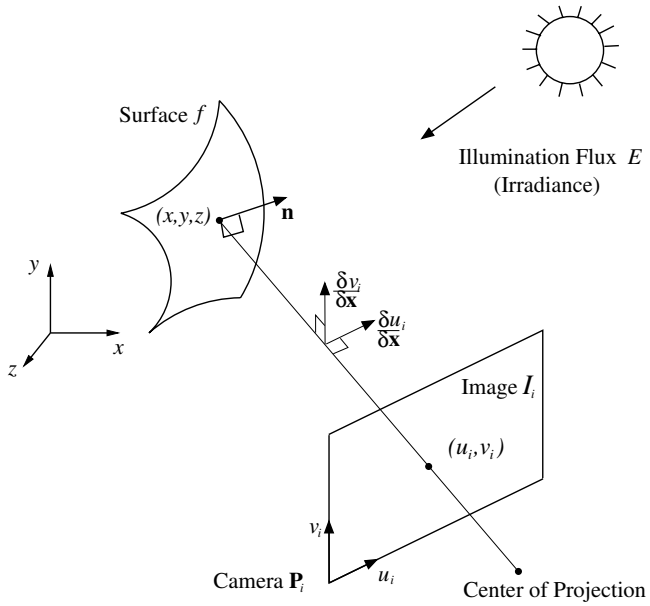


Figure 4.1: A non-rigid surface $S^t = \{\mathbf{X}_i^t\}$ is moving with respect to a fixed world coordinate system (x, y, z). The normal to the surface is $\mathbf{n}^t = \mathbf{n}^t(x, y, z)$. The surface is assumed to be Lambertian with albedo $\rho^t = \rho^t(x, y, z)$ and the illumination flux (irradiance) is E. The i^{th} camera is fixed in space, has a coordinate frame (u_i, v_i), is represented by the 3×4 camera matrix $\mathbf{P}_i$, and captures the image sequence $I_i^t = I_i^t(u_i, v_i)$.

4.1 Image Formation Preliminaries

Consider a non-rigidly moving surface $S^t = \{\mathbf{X}_i^t\}$ imaged by a fixed camera i, with 3×4 projection matrix $\mathbf{P}_i$, as illustrated in Figure 4.1. There are two aspects to the formation of the image sequence $I_i^t = I_i^t(u_i, v_i)$ captured by camera i: (1) the relative camera and surface geometry, and (2) the illumination and surface photometrics.

4.1.1 Relative Camera and Surface Geometry

The relationship between a point (x, y, z) on the surface and its image coordinates (u_i, v_i) in camera i is given by:

$$u_i = \frac{[\mathbf{P}_i]_1 (x, y, z, 1)^{\mathrm{T}}}{[\mathbf{P}_i]_3 (x, y, z, 1)^{\mathrm{T}}} \tag{4.1}$$

$$v_i = \frac{[\mathbf{P}_i]_2 (x, y, z, 1)^{\mathrm{T}}}{[\mathbf{P}_i]_3 (x, y, z, 1)^{\mathrm{T}}} \tag{4.2}$$

where $[\mathbf{P}_i]_j$ is the j^{th} row of $\mathbf{P}_i$. Equations (4.1) and (4.2) describe the mapping from a point $\mathbf{x} = (x, y, z)$ on the surface to its image $\mathbf{u}_i = (u_i, v_i)$ in camera i. Without knowledge of the surface, these equations are not invertible. Given the surface S^t, they can be inverted, but the inversion requires intersecting a ray in space with S^t.

At any fixed time t, the differential relationships between $\mathbf{x}$ and $\mathbf{u}_i$ can be represented by a 2×3 Jacobian matrix $\frac{\partial \mathbf{u}_i}{\partial \mathbf{x}}$. The 3 columns of the Jacobian matrix store the differential change in projected image coordinates per unit change in x, y, and z. A closed-form expression for $\frac{\partial \mathbf{u}_i}{\partial \mathbf{x}}$ as a function of $\mathbf{x}$ can be derived by differentiating Equations (4.1) and (4.2) symbolically. The Jacobian $\frac{\partial \mathbf{u}_i}{\partial \mathbf{x}}$ describes the relationship between a small change in the point on the surface and its image in camera i via $\Delta \mathbf{u}_i \approx \frac{\partial \mathbf{u}_i}{\partial \mathbf{x}} \Delta \mathbf{x}$. Similarly, the inverse Jacobian $\frac{\partial \mathbf{x}}{\partial \mathbf{u}_i}$ describes the relationship between a small change in a point in the image of camera i and the point it is imaging in the scene via $\Delta \mathbf{x} \approx \frac{\partial \mathbf{x}}{\partial \mathbf{u}_i} \Delta \mathbf{u}_i$.

Since image coordinates do not map uniquely to scene coordinates, the inverse Jacobian cannot be computed without knowledge of the surface. If we know the surface (and its gradient), the inverse Jacobian can be estimated as the solution of the following two sets of linear equations:

$$\frac{\partial \mathbf{u}_i}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{u}_i} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4.3}$$

$$\frac{\partial S^t}{\partial \mathbf{u}_i} \;=\; \frac{\partial S^t}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{u}_i} \;=\; \boldsymbol{\nabla} S^t \frac{\partial \mathbf{x}}{\partial \mathbf{u}_i} \;=\; (\,0\;\;0\,)\,. \tag{4.4}$$

Equation (4.3) expresses the constraint that a small change in $\mathbf{u}_i$ must lead to a small change in $\mathbf{x}$ which when projected back into the image gives the original change in $\mathbf{u}_i$. Equation (4.4) expresses the constraint that a small change in $\mathbf{u}_i$ does not lead to a change in S^t since the corresponding point in the world should still lie on the surface. This is because S^t itself is defined as the set of all points on the surface. S^t can also be thought of as an implicit function whose value is zero on the surface and non-zero elsewhere, and so $S^t(\mathbf{x}) = 0$.

The 6 linear equations in Equations (4.3) and (4.4) can be decoupled into 3 for $\frac{\partial \mathbf{x}}{\partial u_i}$ and 3 for $\frac{\partial \mathbf{x}}{\partial v_i}$. Unique solutions exist for both $\frac{\partial \mathbf{x}}{\partial u_i}$ and $\frac{\partial \mathbf{x}}{\partial v_i}$ if and only if:

$$\left(\frac{\partial u_i}{\partial \mathbf{x}} \times \frac{\partial v_i}{\partial \mathbf{x}}\right) \cdot \boldsymbol{\nabla} S^t \;\neq\; 0\,. \tag{4.5}$$

Since $\boldsymbol{\nabla} S^t$ is parallel to the surface normal $\mathbf{n}$, the equations are degenerate if and only if the ray joining the camera center of projection and $\mathbf{x}$ is tangent to the surface.

4.1.2 Illumination and Surface Photometrics

At a point $\mathbf{x}$ in the scene, the irradiance or illumination flux at time t measured in the direction $\mathbf{m}$ can be represented by $E = E(\mathbf{m}; \mathbf{x})$ [Horn, 1986] (We drop the t superscript for clarity). This 6D irradiance function E is similar to what is described as the *plenoptic function* in [Adelson and Bergen, 1991] (they have a seventh variable for wavelength of light).

We denote the net directional irradiance of light at the point (x, y, z) on the surface by $\mathbf{r} = \mathbf{r}(x, y, z)$. The net directional irradiance $\mathbf{r}$ is a vector quantity and is given by the (vector) surface integral of the irradiance E over the visible hemisphere

of possible directions:

$$\mathbf{r}(x,y,z) \;=\; \int_{H(\mathbf{n})} E(\mathbf{m};x,y,z)\,\mathrm{d}\mathbf{m} \tag{4.6}$$

where $H(\mathbf{n}) = \{\mathbf{m} \;:\; \|\mathbf{m}\| = 1 \text{ and } \mathbf{m}\cdot\mathbf{n} \leq 0\}$ is the hemisphere of directions from which light can fall on a surface patch with surface normal $\mathbf{n}$.

We assume that the surface is Lambertian with albedo $\rho = \rho(\mathbf{x})$. Then, assuming that the point $\mathbf{x} = (x,y,z)$ is visible in the i^{th} camera, and that the intensity registered in image I_i is proportional to the radiance of the point that it is the image of (i.e. image irradiance is proportional to scene radiance [Horn, 1986]), we have:

$$I_i(\mathbf{u}_i;t) \;=\; -C\cdot\rho(\mathbf{x};t)\,[\mathbf{n}(\mathbf{x};t)\cdot\mathbf{r}(\mathbf{x};t)] \tag{4.7}$$

where C is a constant that only depends upon the diameter of the lens and the distance between the lens and the image plane. The image pixel $\mathbf{u}_i = (u_i, v_i)$ and the surface point $\mathbf{x} = (x,y,z)$ are related by Equations (4.1) and (4.2).

4.2 How are Scene Flow and Optical Flow Related?

Recall that optical flow is a two-dimensional motion field in the image plane, with each pixel representing the 2-D projection of the displacement of a certain point in the scene. Similarly, the collection of the 3D displacements of all points in the scene is simply the scene flow. As seen in Figure 4.2, when a point $\mathbf{x}$ is displaced by its scene flow $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, the optical flow $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$ in any camera is just the 2-D projection of the scene flow, as long as we ensure that there isn't another part of the scene that comes between point $\mathbf{x}$ and the camera. We first review the basics of optical flow in this differential framework, and then formulate the relationship mathematically.

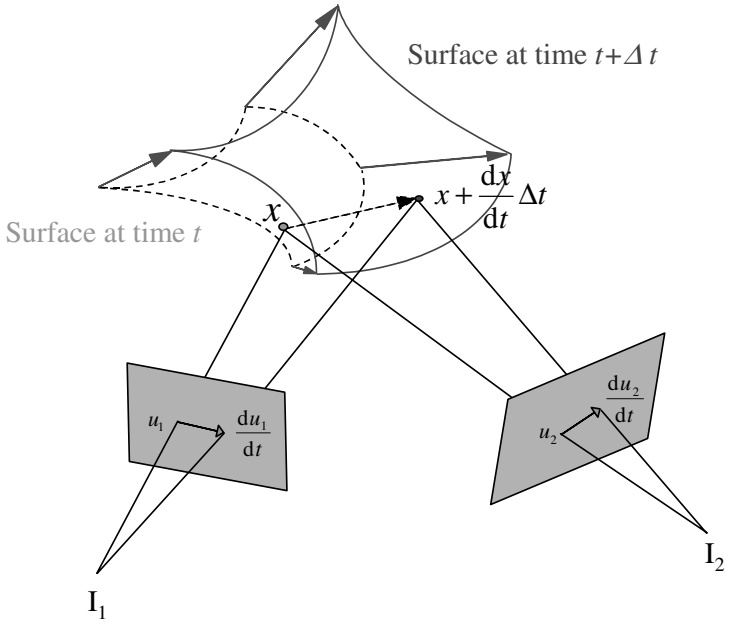


Figure 4.2: The optical flow in any image is the field of instantaneous displacement of all pixels. Therefore, optical Flow is just a 2-D projection of scene flow.

4.2.1 Optical Flow

Suppose $\mathbf{x}^t$ is the 3D path of a point on the surface and the image of this point in camera i is $\mathbf{u}_i(t)$. The 3D motion of this point is $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$ and the 2D image motion of its projection is $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$. The 2D flow field $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$ is known as optical flow. As the point $\mathbf{x}(t)$ moves on the surface, we assume that its albedo $\rho = \rho(\mathbf{x}(t))$ remains constant; i.e. we assume that

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} \;=\; 0. \tag{4.8}$$

The basis for optical flow algorithms is the time-derivative of Equation 4.7:

$$\frac{\mathrm{d}I_i}{\mathrm{d}t} \;=\; \boldsymbol{\nabla} I_i \cdot \frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} + \frac{\partial I_i}{\partial t} \;=\; -C \cdot \rho(\mathbf{x})\, \frac{\mathrm{d}}{\mathrm{d}t}[\mathbf{n} \cdot \mathbf{r}] \tag{4.9}$$

where $\boldsymbol{\nabla} I_i$ is the spatial gradient of the image, $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$ is the optical flow, and $\frac{\partial I_i}{\partial t}$ is the instantaneous rate of change of the image intensity $I_i^t = I_i^t(\mathbf{u}_i)$.

The term $\mathbf{n} \cdot \mathbf{r}$ depends upon both the shape of the surface ($\mathbf{n}$) and the illumination ($\mathbf{r}$). To avoid explicit dependence upon the structure of the three-dimensional scene, it is often assumed that:

$$\mathbf{n} \cdot \mathbf{r} \;=\; \int_{H(\mathbf{n})} E(\mathbf{m}; \mathbf{x})\, \mathbf{n} \cdot \mathrm{d}\mathbf{m} \tag{4.10}$$

is constant ($\frac{\mathrm{d}}{\mathrm{d}t}[\mathbf{n} \cdot \mathbf{r}] = 0$). With uniform illumination or a surface normal that does not change rapidly, this assumption holds well (at least for Lambertian surfaces). In either scenario $\frac{\mathrm{d}I_i}{\mathrm{d}t}$ goes to zero, and we arrive at the *Normal Flow* or *Gradient Constraint* Equation, used by "differential" optical flow algorithms [Barron *et al.*, 1994]:

$$\boldsymbol{\nabla} I_i \cdot \frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} + \frac{\partial I_i}{\partial t} \;=\; 0. \tag{4.11}$$

Using this constraint, a large number of algorithms have been proposed for estimating the optical flow $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$. See [Barron *et al.*, 1994] for a survey.

4.2.2 Three-Dimensional Scene Flow

In the same way that optical flow describes an instantaneous motion field in an image, we can think of scene flow as a three-dimensional flow field $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$ describing the motion at every point in the scene. The analysis in Section 4.1.1 was only for a fixed time t. Now suppose there is a point $\mathbf{x} = \mathbf{x}(t)$ moving in the scene. The image of this point in camera i is $\mathbf{u}_i = \mathbf{u}_i(t)$. If the camera is fixed, the rate of change of $\mathbf{u}_i$ is uniquely determined as:

$$\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} \;=\; \frac{\partial \mathbf{u}_i}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}. \tag{4.12}$$

Inverting this relationship is impossible without knowledge of the surface S^t. To invert it, note that $\mathbf{x}$ depends not only on $\mathbf{u}_i$, but also on the time, indirectly through the surface S^t. That is $\mathbf{x} = \mathbf{x}(\mathbf{u}_i(t); t)$. Differentiating this expression with

respect to time gives:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \;=\; \frac{\partial \mathbf{x}}{\partial \mathbf{u}_i}\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} + \left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i} . \tag{4.13}$$

This equation says that the motion of a point in the world is made up of two components. The first is the projection of the scene flow on the plane tangent to the surface and passing through $\mathbf{x}$. This is obtained by taking the instantaneous motion on the image plane (the optical flow $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$), and projecting it out into the scene using the inverse Jacobian $\frac{\partial \mathbf{x}}{\partial \mathbf{u}_i}$.

The second term is the contribution to scene flow arising from the 3-D motion of the point in the scene imaged by a fixed pixel. It is the instantaneous motion of $\mathbf{x}$ along the ray corresponding to $\mathbf{u}_i$. The magnitude of $\left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i}$ is (proportional to) the rate of change of the depth of the surface S^t along this ray. We now derive an expression for $\left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i}$.

Computing $\left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i}$

The term $\left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i}$ is the 3D motion of the point in the scene imaged by the pixel $\mathbf{u}_i$. Suppose the depth of the surface measured from the i^{th} camera is $d_i = d_i(\mathbf{u}_i)$. Then, the point $\mathbf{x}$ can be written as a function of $\mathbf{P}_i$, $\mathbf{u}_i$, and d_i as follows. The 3×4 camera matrix $\mathbf{P}_i$ can be written as:

$$\mathbf{P}_i \;=\; [\,\mathbf{R}_i\;\mathbf{T}_i\,] \tag{4.14}$$

where $\mathbf{R}_i$ is a 3×3 matrix and $\mathbf{T}_i$ is a 3×1 vector. The center of projection of the camera is $-\mathbf{R}_i^{-1}\mathbf{T}_i$, the direction of the ray through the pixel $\mathbf{u}_i$ is $\mathbf{r}_i(\mathbf{u}_i) = \mathbf{R}_i^{-1}(u_i, v_i, 1)^{\mathrm{T}}$, and the direction of the camera z-axis is $\mathbf{r}_i(\mathbf{0}) = \mathbf{R}_i^{-1}(0, 0, 1)^{\mathrm{T}}$. Using simple geometry, (see Figure 4.3) we therefore have:

$$\mathbf{x} \;=\; -\mathbf{R}_i^{-1}\mathbf{T}_i + d_i \left[\frac{\|\mathbf{r}_i(\mathbf{0})\|\,\mathbf{r}_i(\mathbf{u}_i)}{\mathbf{r}_i(\mathbf{0}) \cdot \mathbf{r}_i(\mathbf{u}_i)}\right] . \tag{4.15}$$

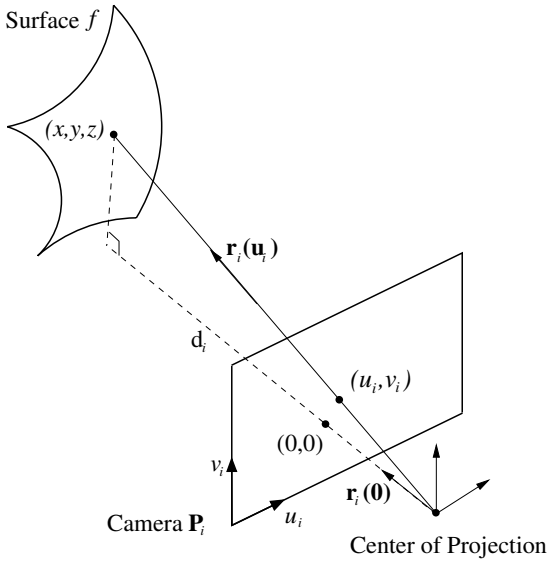


Figure 4.3: Given the camera matrix $\mathbf{P}_i$ and the distance d_i to the surface, the direction of the ray through the pixel $\mathbf{u}_i$ and the direction of the z-axis of the camera can be used to derive an expression for the point $\mathbf{x}$. This expression can be symbolically differentiated to give $\frac{\partial \mathbf{x}}{\partial t}\Big|_{\mathbf{u}_i}$ as a function of $\mathbf{x}$, $\mathbf{P}_i$, and $\frac{\partial d_i}{\partial t}$.

(Care must be taken to choose the sign of $\mathbf{P}_i$ correctly so that the vector $\mathbf{r}_i(\mathbf{u}_i)$ points out into the scene.) If camera $\mathbf{P}_i$ is fixed, we have:

$$\left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i} = \left[\frac{\|\mathbf{r}_i(\mathbf{0})\|\,\mathbf{r}_i(\mathbf{u}_i)}{\mathbf{r}_i(\mathbf{0})\cdot\mathbf{r}_i(\mathbf{u}_i)}\right]\frac{\partial d_i}{\partial t}. \tag{4.16}$$

So, the magnitude of $\frac{\partial \mathbf{x}}{\partial t}\Big|_{\mathbf{u}_i}$ is proportional to the rate of change of the depth map and the direction is along the ray joining $\mathbf{x}$ and the center of projection of the camera.

4.3 Single Camera Case

We first consider the case of how to compute scene flow using optical flow from a single camera, using Equation 4.13. This is applicable only when the geometry of the surface of the scene is completely known at one time instant. This includes

surface normals at all points on the surface, the depth map from the point of view of the camera, and the temporal rate of change of this depth map.

4.3.1 Computing Scene Flow

If the surface S^t is accurately known, the surface gradient ∇S^t can be computed at every point. The inverse Jacobian $\frac{\partial \mathbf{x}}{\partial \mathbf{u}_i}$ can then be estimated by solving the set of 6 linear equations in Equations (4.3) and (4.4). Given the inverse Jacobian, the scene flow can be estimated from the optical flow $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$ using Equation (4.13):

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \frac{\partial \mathbf{x}}{\partial \mathbf{u}_i}\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} + \left.\frac{\partial \mathbf{x}}{\partial t}\right|_{\mathbf{u}_i}. \tag{4.17}$$

Complete knowledge of the scene structure thus enables us to compute scene flow from one optical flow, (and the rate of change of the depth map corresponding to this image.) These two pieces of information correspond to the two components of the scene flow; the optical flow is projected onto the tangent plane passing through $\mathbf{x}$, and the rate of change of depth map is mapped onto a component along the ray passing through the scene point $\mathbf{x}$ and the center of projection of the camera.

Note that we assume that the surface is locally planar when computing the inverse Jacobian. Since the surface is known, it is possible to project the "flowed" point in the image and intersect this ray with the surface. We currently do not perform this to save an expensive ray-surface intersection for every pixel.

If only one optical flow is used, the scene flow can be computed only for those points in the scene that are visible in that image. It is possible to use multiple optical flows in multiple cameras for better visibility, and for greater robustness. Also, flow is recovered only when the change in depth map is valid - that is, when an individual pixel sees neighboring parts of the surface as time changes. If the motion of a surface is large relative to its size, then a pixel views different surfaces, and flow cannot be computed.

4.3.2 Difficulty with Estimating Scene Flow from a Single Camera

In practice, knowing the rate of change of depth map (contributing to the second term in Equation 4.13) is difficult. In addition, the algorithm only works for pixels where this change in depth map is continuous, which happens only if the pixel projects to the same surface at both time instants. At occluding boundaries, the same pixel in an image will usually project to surfaces that are at two different distances from the image. With the continuity assumption not satisfied, this unfortunately means that the scene flow is invalid at occluding boundaries, where the motion of the scene is usually the most interesting. This turns out to be a big limitation in practice giving poor results, and therefore, we do not use this algorithm. Instead, we use the scene flow algorithm that uses multiple cameras, which does not depend on the rate of change of depth map.

4.4 Computing Scene Flow: Multi-Camera Case

In our calibrated setting, once we have computed the shape of the scene, we can easily project the shape into any camera to get a depth map. However, these depth maps are often too noisy to estimate surface normals and temporal rates of change. This situation of having a shape as a voxel model without an accurate representation of complete surface properties is common to all volumetric shape reconstruction algorithms. While algorithms such as Marching Cubes [Lorenson and Cline, 1987] can recover a surface from volumetric data, these surfaces are sub-voxel accurate only if the distance to the surface (the 3-D level set) at every voxel is known.

If we have two or more cameras viewing a particular point in the scene, then Equation 4.13 can be solved without knowledge of the complete surface properties. It seems intuitive that less knowledge of scene structure requires the use of more optical flows, and indeed this result does follow from the degeneracy in the linear

equation used to compute scene flow. Note that the first term in the set of Equations 4.13 provide 2 constraints on 3 variables. However, each image that views the same 3D provides two such constraints, so with two or more cameras, we will be able to solve for the scene flow without the need for the rate of change of depth maps.

With the 3-D voxel models computed using our 6-D carving algorithm, the surface properties are not completely known. Hence, it is not possible to solve for $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{t}}$ directly from one camera. Instead, consider the implicit linear relation between the scene and the optical flow, as described by the Jacobian in Equation (4.13).

This set of equations provides two linear constraints on $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$. Let $\mathrm{Vis}(\mathrm{x})$ be the set of all cameras in which x is visible, and let N be the number of cameras in this set. If we have $N > 2$, we can solve for $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, by setting up the system of equations $\mathbf{B}\mathbf{x} = \mathbf{U}$, where:

$$\mathbf{B} = \begin{bmatrix} & & \\ \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} & \frac{\partial u_1}{\partial z} \\ \frac{\partial v_1}{\partial x} & \frac{\partial v_1}{\partial y} & \frac{\partial v_1}{\partial z} \\ \cdot & \cdot & \cdot \\ & & \\ \cdot & \cdot & \cdot \\ \frac{\partial u_N}{\partial x} & \frac{\partial u_N}{\partial y} & \frac{\partial u_N}{\partial z} \\ \frac{\partial v_N}{\partial x} & \frac{\partial v_N}{\partial y} & \frac{\partial v_N}{\partial z} \\ & & \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \\ \frac{\partial u_1}{\partial t} \\ \frac{\partial v_1}{\partial t} \\ \cdot \\ \\ \cdot \\ \frac{\partial u_N}{\partial t} \\ \frac{\partial v_N}{\partial t} \\ \end{bmatrix} \tag{4.18}$$

This gives us $2N$ equations in 3 unknowns, and so for $N \geq 2$ we have an over-constrained system and can find an estimate of the scene flow. (This system of equations is degenerate if and only if the point $\mathbf{x}$ and the N camera centers are collinear.) A singular value decomposition of $\mathbf{B}$ gives the solution that minimizes the sum of least squares of the error obtained by re-projecting the scene flow onto each of the optical flows.

Note that it is critical that only the cameras in the set $\mathrm{Vis}(\mathrm{x})$ are used in comput-

ing the scene flow. Since we are computing the flow for a voxel model that has been recovered already, we simply set $\mathbf{x}$ to be the center of each voxel as we loop over the entire set of surface voxels in the scene. In our implementation, the visibility of the cameras has already been determined during the 6-D carving algorithm, as described in Step 1 of Section 3.5, so this same visibility is used for the computation in the scene flow equation.

4.5 Results

We have implemented the scene flow algorithm on real data. We show results of applying the algorithm to the same dataset that was used in the previous chapter. The scene shape used is that computed from the 6-D carving algorithm. Image sequences from 14 cameras are captured in the CMU 3D Room (see Appendix A). Optical flows are first computed between each combination of neighboring images in time, for each camera using a hierarchical version of [Lucas and Kanade, 1981]. These optical flows are used as inputs to the scene flow algorithm, with the shape and calibration.

Figure 3.3 shows the input images at two different time instants from different camera positions. This data sequence is the same as that used for the simultaneous shape and motion algorithm. For recomputing the scene flow, we use 18 different cameras. This large number of cameras ensures (a) Robust computation in the presence of outliers in one or more cameras, and that (b) Every part of the scene is seen by at least 2 cameras, so that scene flow can be computed. The instantaneous motion of the dancer is such that she is turning her torso to the left, while also raising and stretching out her left arm.

Figure 4.4 shows the motion from a single camera position, and the 2-D optical flow computed. A hierarchical version of the [Lucas and Kanade, 1981] algorithm is used for determining the optical flow, and the figure shows the computed horizontal and vertical flow fields. Darker pixels indicate a greater motion to the right and

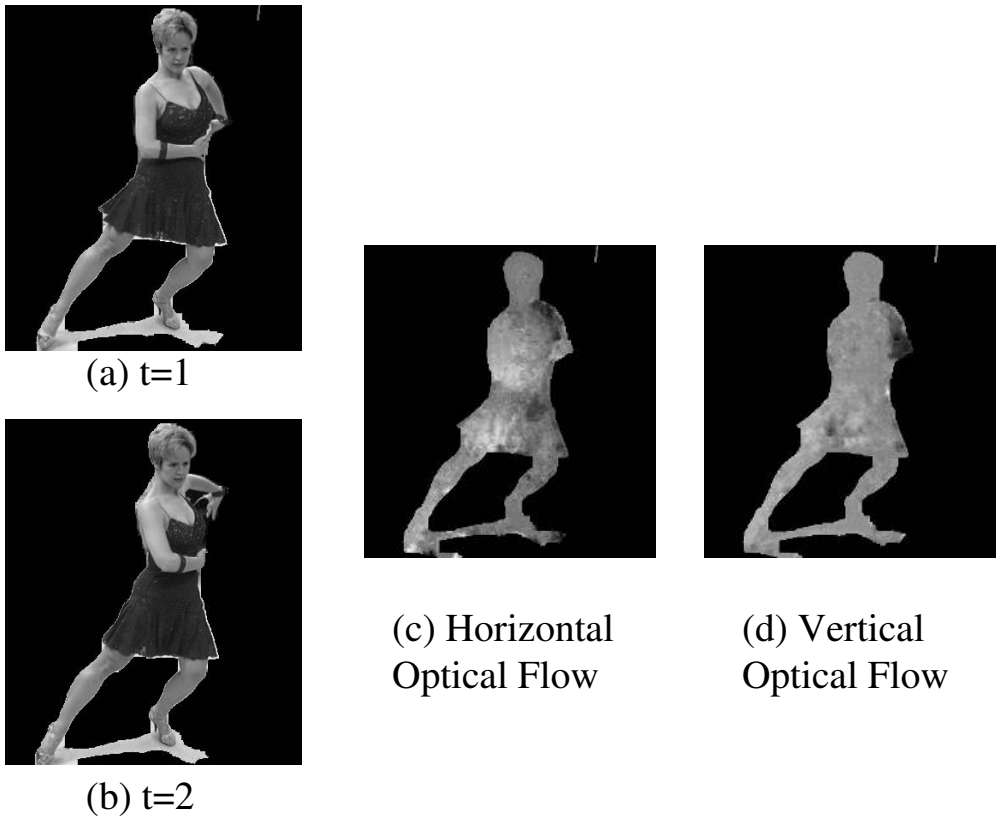


Figure 4.4: Two input images at consecutive time instants (shown after background subtraction) and the horizontal and vertical 2-D optical flow fields, computed using a hierarchical version of the algorithm of [Lucas and Kanade, 1981]. Darker pixels indicate larger flow to the right and bottom in the horizontal and vertical flow fields respectively.

bottom, while lighter pixels indicate motion to the left and upwards, respectively. The other input needed for scene flow computation is the shape itself, computed by 6-D shape and motion carving and shown in Figure 4.5.

Figure 4.6 shows two snapshots of the computed scene flow. Scene flow is computed as a dense flow field, so there is a motion vector computed for every single voxel on the shape. The close-up snapshot shows the motion of the voxels as the dancer raises and stretches out her arm. Figure 4.7 shows the same flow vectors overlaid on the shape. Compared to the 6D carving algorithm, the scene flow results

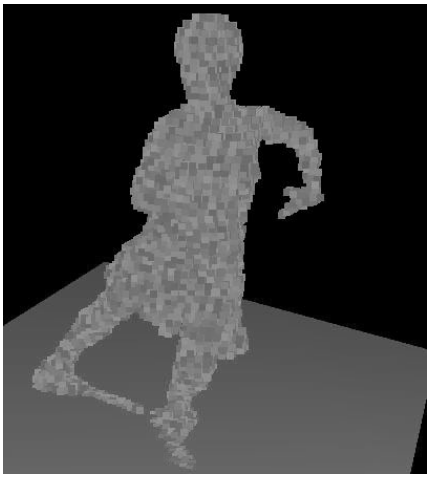


Figure 4.5: An example voxel model, showing the geometry on which the 3-D scene flow is computed.

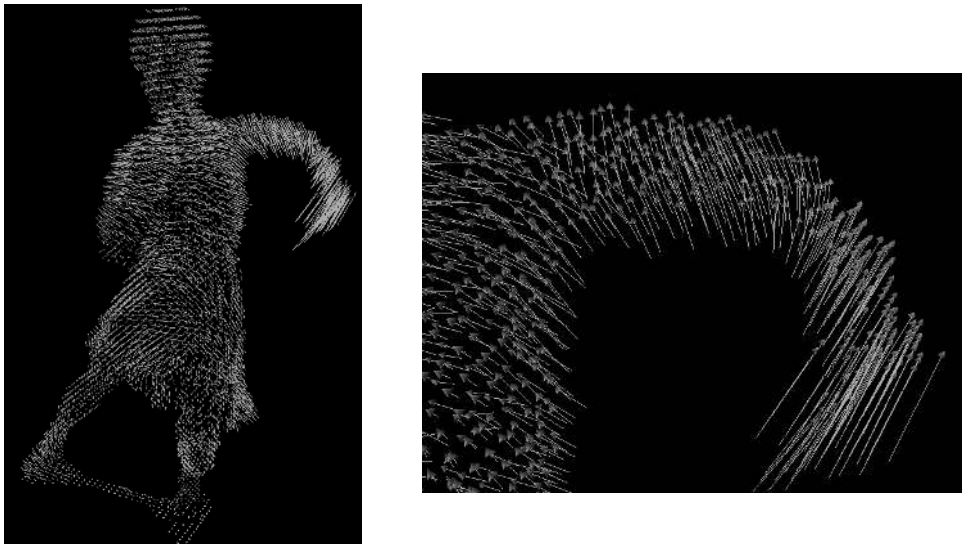
are far less noisy, because because scene flow is now computed using smoothness constraints, which are implicitly imposed through the 2-D optical flow.

Figure 4.8 shows the magnitude of the computed scene flow. The absolute values of the computed flows in all 3 directions are averaged and displayed as the intensity for each point. Darker pixels are those with larger scene flow magnitudes; the rotation about her torso results in both her arms having a large flow, in addition to motion of her skirt.

4.5.1 Performance

The computational cost of the scene flow algorithm is linear in the number of cameras, and in the number of surface voxels for which flow needs to be computed. In our example, we compute the flow for approximately 6000 surface voxels, from 14 cameras. On an R10000 SGI O2, this takes about 10 seconds. Once the model and the input optical flows are loaded, memory required is trivial since the scene flow is computed one voxel at a time.

The main bottleneck, however, is the computation of the optical flow. The hier-



(a) (b)

Figure 4.6: The refined scene flow, computed using optical flows from multiple cameras, shown from two different viewpoints. Notice that the overall motion of the dancer is highly non-rigid.

archical Lucas-Kanade algorithm takes about 1.5 minutes to compute optical flow between a pair of frames on an SGI. Although, on today's fastest desktop processor (a 2GHz Pentium 4), this time should be down to about 10 seconds. For a long sequence with n frames and C cameras, we need to compute flow for $n-1$ pairs of frames, on each of the C cameras. For a sequence of 5 seconds (at 15 fps) from 14 cameras, this is about 25 hours of processing time on our system, which is easily divided among all available CPUs.

4.6 Three-Dimensional Normal Flow Constraint

It is natural to ask the question: since the normal component of the optical flow in an image can be computed without any regularization [Barron *et al.*, 1994], is it

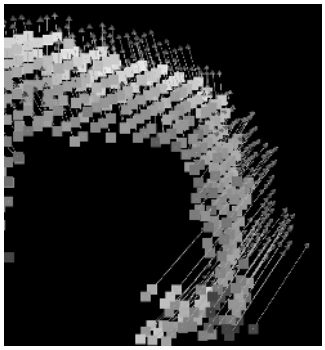


Figure 4.7: Scene flow shown together with the shape. Notice the upward motion of the shoulder and the outward motion of the arm.

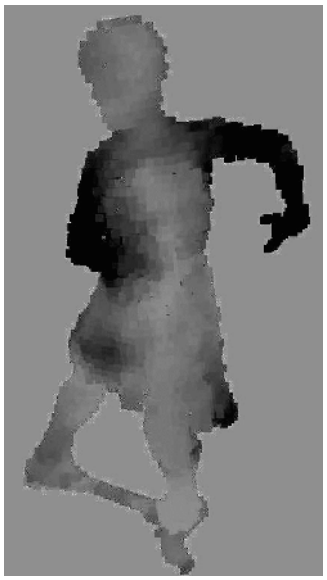


Figure 4.8: The shape intensity coded with the magnitude of the scene flow. Darker pixels are those with larger scene flow values. The motion on the right arm is mostly due to rotation about the torso, and that on the left arm is mostly because of shoulder and elbow movement.

possible to use these normal flows from many cameras to directly compute scene flow, without having to use any regularization?

Optical flow $\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}$ is a two dimensional vector field, and so is often divided into two components, the *normal flow* and the *tangent flow*. The normal flow is the

component in the direction of the image gradient $\boldsymbol{\nabla} I_i$, and the tangent flow is the component perpendicular to the normal flow. The magnitude of the normal flow can be estimated directly from Equation (4.11) as:

$$\frac{1}{|\boldsymbol{\nabla} I_i|} \boldsymbol{\nabla} I_i \cdot \frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} \;=\; -\frac{1}{|\boldsymbol{\nabla} I_i|} \frac{\partial I_i}{\partial t}. \tag{4.19}$$

Estimating the tangent flow is an ill-posed problem. Hence, some form of local smoothness is required to estimate the complete optical flow [Barron *et al.*, 1994]. Since the estimation of the tangent flow is the major difficulty in most algorithms, it is natural to ask whether the normal flows from several cameras can be used to estimate the 3D scene flow without having to use some form of regularization.

The Normal Flow Constraint Equation (4.11) can be rewritten as:

$$\boldsymbol{\nabla} I_i \cdot \left[\frac{\partial \mathbf{u}_i}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \right] + \frac{\partial I_i}{\partial t} \;=\; 0. \tag{4.20}$$

This is a scalar linear constraint on the components of the scene flow $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$. Therefore, at first glance it seems likely that it might be possible to estimate the scene flow directly from three such constraints. Unfortunately, differentiating Equation (4.7) with respect to $\mathbf{x}$ we see that:

$$\boldsymbol{\nabla} I_i \frac{\partial \mathbf{u}_i}{\partial \mathbf{x}} \;=\; -C \cdot \boldsymbol{\nabla} \left(\rho(\mathbf{x}; t) \left[\mathbf{n}(\mathbf{x}; t) \cdot \mathbf{r}(\mathbf{x}; t) \right] \right). \tag{4.21}$$

Since this expression is independent of the camera i, and instead only depends on properties of the scene (the surface albedo ρ, the scene structure $\mathbf{n}$, and the illumination $\mathbf{r}$), the coefficients of $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$ in Equation (4.20) should ideally always be the same. Hence, any number of copies of Equation (4.20) will be linearly dependent. In fact, if the equations turn out not to be linearly dependent, this fact can be used to deduce that $\mathbf{x}$ is not a point on the surface, as described in [Vedula *et al.*, 1999].

This result means that it is impossible to compute 3D scene flow independently

for each point on the object, without some form of regularization of the problem. We wish to emphasize, however, that this result does not mean that is it not possible to estimate other useful quantities directly from the normal flow, as for example is done in [Negahdaripour and Horn, 1987] and other "direct methods."

4.7 Scene Flow: Which Algorithm to Use?

In this chapter, we have developed some fundamental geometric properties of scene flow. If we know the complete scene geometry, optical flow from one camera may be used to compute the scene flow, although this algorithm is often not very practical. We have also described an algorithm to compute scene flow using known shape and optical flow from multiple cameras. The natural question is - is this preferable to the shape and motion carving algorithm that computes scene shape and flow simultaneously?

Each of the algorithms has its own advantages. The unified algorithm is more elegant in that the shape and flow, both fundamental properties of a dynamic scene, are computed at the same time. Also, it is possible to generalize it to use other decision functions, in conjunction with photo-consistency. If the motion of the scene provides extra constraints instead of just photo-consistency, it can arguably improve the quality of the shape. The algorithm is also not limited to the use of just two time instants, although the dimensionality of the problem grows so it becomes much harder to implement in practice.

On the other hand, the scene flow algorithm has the advantage that because of the smoothness prior built in (using optical flow, that has been time-tested), the results of the scene flow are better. The use of optical flow provides for a simple and efficient algorithm, with far less computation compared to the 6D algorithm, which involves searches in a higher dimensional space. Because the computations of shape and flow are not coupled, the shape can be estimated using more accurate techniques (or even active ones such as laser range scanning). With a more accurate

shape, the visibility computation is also more accurate, and this in turn improves the quality of the flow by ensuring that only cameras that are visible at any point contribute to the calculation of the scene flow. An interesting direction for future work would be to investigate how to utilize these smoothness priors effectively in our 6D carving algorithm.

For our end goal of creating views from novel positions in space and time, we use scene flow to interpolate shape in time. Because of this essential role of the scene flow, it is important that we use the most accurate flow available, and therefore use the latter algorithm. The main disadvantage is that it is not clear how to use scene flow with voxel models; we address this issue in the context of shape interpolation in Section 5.2 in the next chapter.

Chapter 5

Spatio-Temporal View Interpolation

So far, we have discussed how to create a dynamic model of the time-varying motion of the scene. The geometry at any time instant, together with the scene flow constitutes a complete time-varying model of the dynamic event. With this, we can trace the motion of any point in the scene, or even a pixel in any image, as a continuous function of time. This representation is far richer than simply modeling the event as a sequence of 3D shapes, as in [Narayanan *et al.*, 1998]. There, although the shape is available at discrete time instants, there are no temporal correspondences between shapes at neighboring times, and therefore one cannot estimate shape at any time but at the original time instants when the images were captured.

We now have all the intermediates necessary to address the spatio-temporal view interpolation problem. Recall that we seek to re-create the appearance of the dynamic event from an arbitrary camera position at an arbitrary time. We choose to generate this appearance pixel by pixel, by finding corresponding pixels in other input images and then blending them suitably. Consider the simple case shown in Figure 5.1, where we have images from two cameras at two different time instants. Recall that the subscript represents spatial position, the superscript represents time. Let a novel image in this space be I^*_+. Now for any pixel (x, y) in this image p^*_+, we have

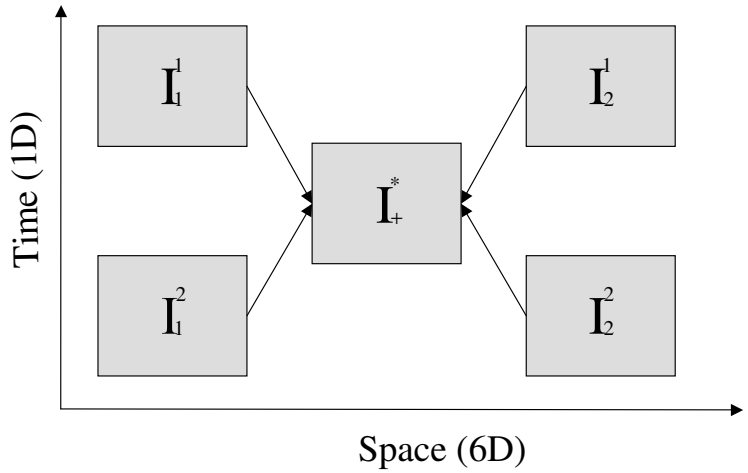


Figure 5.1: An example of images from 2 cameras at each at 2 time instants, shown along with a novel image. We create the novel view by combining the appropriate corresponding pixels from each of these 4 images.

$$I^*_+(x,y) = f[I^1_1(x^1_1, y^1_1), I^1_2(x^1_2, y^1_2), I^2_1(x^2_1, y^2_1), I^2_2(x^2_2, y^2_2)] \tag{5.1}$$

where the (x^i_j, y^i_j) are the pixels in each of the 4 input images that view the same point in the scene as (x, y), and f is some function that weights each of these contributions in a suitable manner. Finding these corresponding pixels (x^i_j, y^i_j) across the different images is the most difficult part. There is no simple closed-form representation for determining the corresponding points, since the correspondence relationship depends on the shape of the scene, the non-rigid motion of the scene, and the visibility relationship that cameras have with different parts of the scene. The function f depends on the position of the novel camera with respect to each of the input cameras (where position refers to the general position in both the spatial and temporal domains).

The example above is simplistic in that it illustrates the case of creating a novel image using sampled images from just two cameras at two time instants. In reality, we need to combine samples from many more cameras, to account for difficulties

with calibration errors, and the fact that just two cameras almost never have completely overlapping areas. In fact, the more complicated the scene is, the more cameras it usually takes to ensure that every part of the scene is visible by at least 2 cameras (the minimum required to recover shape). The spatio-temporal view interpolation algorithm that we describe uses all images that contain pixels corresponding to a particular pixel in the novel image, and the weighting function also generalizes to an arbitrary number of images.

5.1 High-Level Overview of the Algorithm

Suppose we want to generate a novel image I_+^* from virtual camera C_+ at time t^*, where $t \leq t^* \leq t+1$. Recall that we have computed voxel models S^t and S^{t+1} at time t and time $t+1$ respectively. The first step is to "flow" S^t and S^{t+1} using the scene flow to estimate an interpolated voxel model S^* at time t^*. The second step consists of fitting a smooth surface to the flowed voxel model S^*. The third step consists of computing the novel image by reverse-mapping or ray-casting from each pixel across space and time. For each pixel (u, v) in I_+^* a ray is cast into the scene and intersected with the interpolated scene shape (the smooth surface). The scene flow is then followed forwards and backwards in time to the neighboring time instants, t and $t+1$. The corresponding points at those times are projected into the input images, the images sampled at the appropriate locations, and the results blended to give the novel image pixel $I_+^*(u, v)$. Spatio-temporal view interpolation can therefore be summarized as:

1. Flow the voxel models to estimate S^*.
2. Fit a smooth surface to S^*.
3. Compute the novel image by reverse mapping, casting rays from each pixel to S^* and then across time, before projecting into the input images.

We now describe these 3 steps in detail starting with Step 1. Since Step 3 is the most important step and can be explained more easily without the complications of surface fitting, we describe Step 3 before explaining how intersecting with a surface rather than a set of voxels (Step 2) modifies the algorithm.

5.2 Flowing the Voxel Models

In order to generate the novel image I^*_+, the first step is to compute the shape S^* at time t^*, along with the mapping from each point on S^* to corresponding points on S^t and S^{t+1}. The computed scene flow does implicitly give us this correspondence information, but requirements such as discretization for a voxel model, continuity, and the need for inverse flows make the problem non-trivial. In this section, we describe how we use the scene flow to interpolate the shape between time instants, the various ideal properties of the scene flow, and how the effects of incorrectly computed scene flow manifest themselves in the rendered images.

5.2.1 Shape Interpolation Using Scene Flow

The scene shape is described by the voxels S^t at time t and the voxels S^{t+1} at time $t+1$. The motion of the scene is defined by the scene flow F^t_i for each voxel X^t_i in S^t. We now describe how to interpolate the shapes S^t and S^{t+1} using the scene flow. By comparison, previous work on shape interpolation [Turk and O'Brien, 1999, Alexa *et al.*, 2000] is based solely on the shapes themselves rather than on a flow field connecting them. We assume that the voxels move at constant speed in straight lines and so flow the voxels with the appropriate multiple of the scene flow. If t^* is an intermediate time ($t \leq t^* \leq t+1$), we interpolate the shape of the scene at time t^* as:

$$S^* \;=\; \{X^t_i + (t^* - t) \times F^t_i \mid i = 1, \ldots, V^t\} \tag{5.2}$$

i.e. we start with the voxel model S^t at time t, and then flow the voxels proportionately forwards using the scene flow F^t, which measures the complete flow from time t to time $t+1$.

5.2.2 Desired Properties of the Scene Flow for Voxel Models

Equation 5.2 defines S^* in an asymmetric way; the voxel model at time $t+1$ is not even used. Clearly, a symmetric algorithm is preferable; we now discuss symmetry and other desirable properties of the scene flow. A related question is whether the interpolated shape is continuous as $t^* \to t+1$; i.e. in this limit, does S^* tend to S^{t+1}? Ideally we want this property to hold, but how do we enforce it?

One suggestion might be that the scene flow F^t should map *one-to-one* from S^t to S^{t+1}. Then, the interpolated scene shape will definitely be continuous. The problem with this requirement, however, is that it implies that the voxel models must contain the same number of voxels at times t and $t+1$. It is therefore too restrictive to be useful. For example, it outlaws motions that cause the shape to expand or contract. The properties that we really need are:

Inclusion: Every voxel at time t should flow to a voxel at time $t+1$: i.e. $\forall t, i\ X_i^t + F_i^t \in S^{t+1}$.

Onto: Every voxel at time $t+1$ should have a voxel at time t that flows to it: $\forall t, i,\ \exists j \text{ s.t. } X_j^t + F_j^t = X_i^{t+1}$.

These properties immediately imply that the voxel model at time t flowed forward to time $t+1$ is *exactly* the voxel model at time $t+1$:

$$\{X_i^t + F_i^t \mid i = 1, \ldots, V^t\} \ = \ S^{t+1}. \tag{5.3}$$

This means that the scene shape will be continuous at $t+1$ as we flow the voxel model forwards using Equation (5.2).

5.2.3 Enforcing the Desired Properties of Scene Flow

Satisfaction of the inclusion property is relatively easy. While the scene flow is calculated as a continuous quantity, the fact that we represent shapes as discrete voxel models requires that the scene flow also be discretized. So we just have to ensure that the endpoints of the flow vectors are discretized to the nearest voxel.

The onto property is a little more complicated. The natural question that arises is, is it possible to enforce both inclusion and onto conditions without making the scene flow be one-to-one? It may seem impossible because the second condition seems to imply that the number of voxels has to remain the same at all times. It is possible to satisfy both properties, however, if we introduce what we call *duplicate voxels*. Duplicate voxels are additional voxels at time t which flow to different points in the model at $t + 1$; i.e. we allow two voxels X_i^t and X_j^t $(i \neq j)$ where $(x_i^t, y_i^t, z_i^t) = (x_j^t, y_j^t, z_j^t)$ but yet $F_i^t \neq F_j^t$. We can then still think of a voxel model as just a set of voxels and satisfy the two desirable properties above. There may just be a number of duplicate voxels with different scene flows.

Duplicate voxels also make the formulation more symmetric. If the two properties *inclusion* and *onto* hold, the flow can be inverted in the following way. For each voxel at the second time instant there are a number of voxels at the first time instant that flow to it. For each such voxel we can add a duplicate voxel at the second time instant with the inverse of the flow. Since there is always at least one such voxel (onto) and every voxel flows to some voxel at the second time (inclusion), when the flow is inverted in this way the two properties hold for the inverse flow as well. So, given forwards scene flow where inclusion and onto hold, we can invert it using duplicate voxels to get a backwards scene flow for which the properties hold also. Moreover, the result of flowing the voxel model forwards from time t to t^* with the forwards flow field is the same as flowing the voxel model at time $t + 1$ backwards with the inverse flow. We can then formulate shape interpolation symmetrically as flowing either forwards and backwards, with identical results. Thus, our scene flow

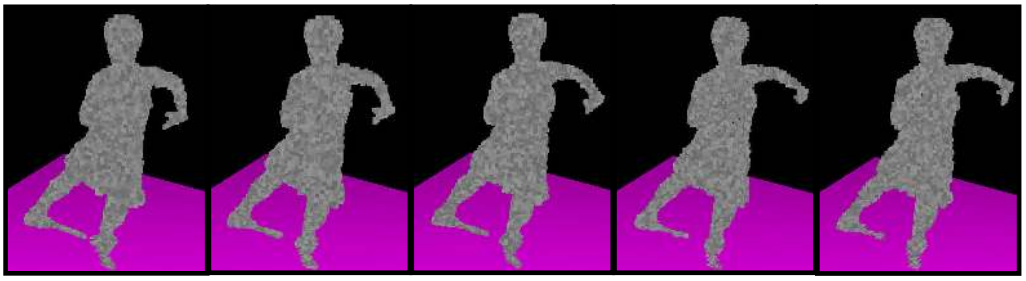


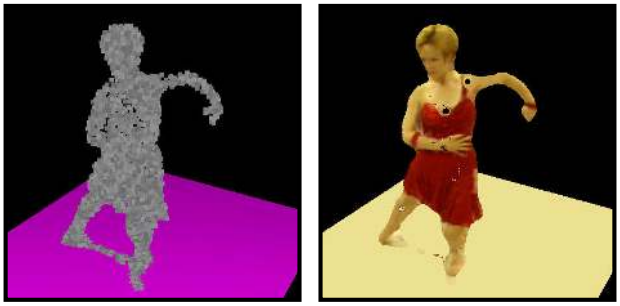
Figure 5.2: The scene shape between neighboring time instants can be interpolated by flowing the voxels at time t forwards. Note how the arm of the dancer flows smoothly from $t = 1$ to $t = 2$.

field now has the desirable properties of smoothness, symmetry, and compatibility with our discretized voxel grids.

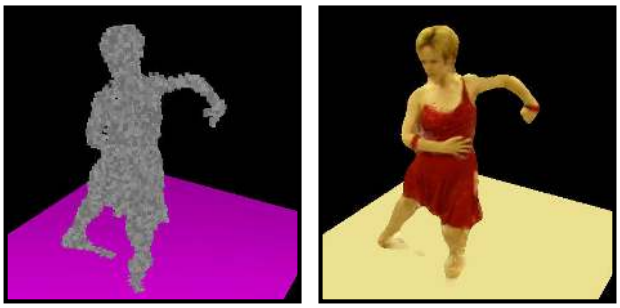
We implement the inclusion and onto properties as follows: To ensure inclusion, the set of all endpoints of the flow vectors ($X_i^t + F_i^t$) has to be a part of the shape S_i^{t+1}. To satisfy this, we simply discretize F_i^t so that the endpoints of the flow vectors lie on the same voxel grid as S_i^{t+1}, which in effect resets each flow vector to flow to the nearest voxel at time $t + 1$. Each of these voxels at time $t + 1$ that have been flowed to, are assigned the inverse flow backwards. Now, all voxels X_i^{t+1} at time $t + 1$ that do not have a voxel at time t flowing to them are the ones that do not satisfy the onto property, and therefore need a duplicate voxel at time t that would flow to them. First, an inverse flow is computed for each such voxel (by simply averaging the inverse flow available for its neighbors). Corresponding to this inverse flow, a duplicate voxel at time t that flows to it is added.

5.2.4 Results

Figure 5.2 shows the results of interpolating the shape using the computed scene flow between two time instants. The voxel model shown is that of a dancer stretch-



(a) Without Duplicate Voxels



(b) With Duplicate Voxels

Figure 5.3: A rendered view at an intermediate time, with and without duplicate voxels. Without the duplicate voxels, the model at the first time does not flow *onto* the model at the second time. Holes appear where the missing voxels should be. The artifacts disappear when the duplicate voxels are added.

ing out her arm as she moves it upwards. The shapes on the left and right are computed using the images at times $t = 1$ and $t = 2$ respectively, whereas the in-between voxel models are computed by flowing these models as described above. Notice how the motion is smooth, without any asymmetry near either $t = 1$ or $t = 2$.

The importance of the duplicate voxels used to enforce the into/onto properties and computed using the algorithm described above, is illustrated in Figure 5.3. This figure contains two views at an intermediate time rendered using the algorithm in Section 5.3, one with duplicate voxels and one without. Without the duplicate voxels the model at the first time instant does not flow *onto* the model at the second time. When the shape is flowed forwards holes appear in the voxel model (left) and

in the rendered view (right). With the duplicate voxels the voxel model at the first time does flow *onto* the model at the second time and the artifacts disappear. In addition, the discontinuity in the flow without the duplicate voxels shows up dramatically when a series of interpolated novel views are played back as a movie. Without the duplicate voxels, there is always a discontinuity at each of the original sampled time instants since the flowed shape differs slightly from the computed shape at each time instant. With the duplicate voxels, the voxel models flow smoothly into each other.

5.3 Ray-Casting Across Space and Time

Once we have interpolated the scene shape we can ray-cast across space and time to generate the novel image I_+^*. We use a reverse mapping approach similar in concept to that in [Laveau and Faugeras, 1994b], but adapted to our problem of pixel interpolation across both space and time. As illustrated in Figure 5.3, we shoot a ray out into the scene for each pixel (u, v) in I_+^* at time t^* using the known geometry of camera C_+. We find the intersection of this ray with the flowed voxel model. Suppose for now that the first voxel intersected is $X_i^{t^*} = X_i^t + (t^* - t) \times F_i^t$. (Note that we will describe a refinement of this step in Section 5.4.)

We need to find a color for the novel pixel $I_+^*(u, v)$. We cannot project the voxel $X_i^{t^*}$ directly into an image because there are no images at time t^*. We can find the corresponding voxels X_i^t at time t and $X_j^{t+1} = X_i^t + F_i^t$ at time $t + 1$, however. We take these voxels and project them into the images at time t and $t + 1$ respectively (using the known geometry of the cameras C_i) to get multiple estimates of the color of $I_+^*(u, v)$. This projection must respect the visibility of the voxels X_i^t at time t and X_j^{t+1} at time $t + 1$ with respect to the cameras at the respective times.

Once the multiple estimates of $I_+^*(u, v)$ have been obtained, they are *blended*. We just have to decide how to weight the samples in the blend. Ideally we would like the weighting function to satisfy the property that if the novel camera C_+ is

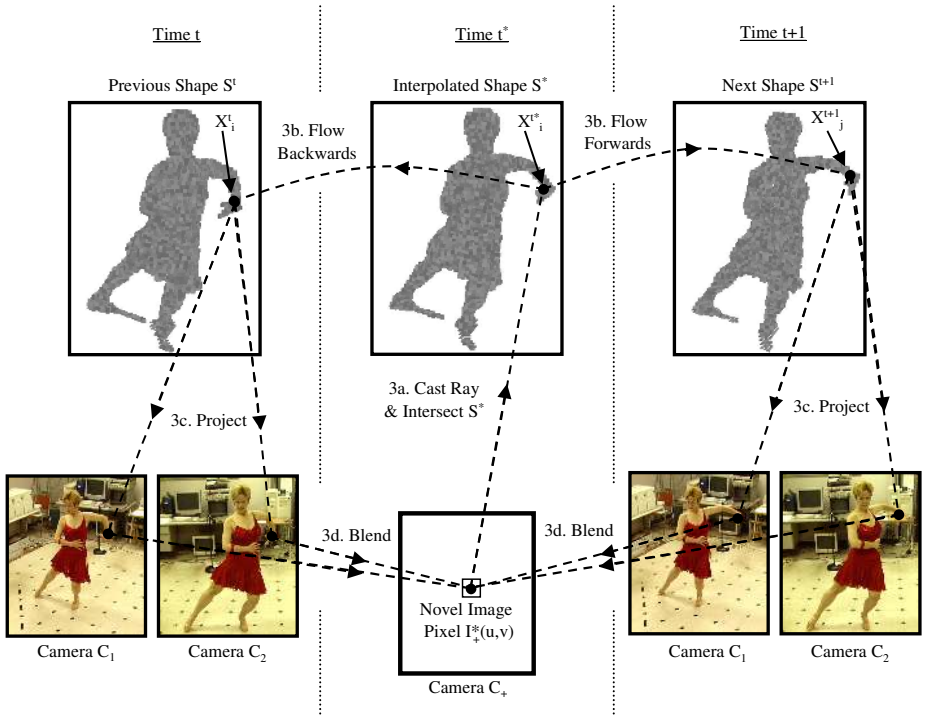


Figure 5.4: Ray-casting across space and time. 3a. A ray is shot out into the scene at time $t = t^*$ and intersected with the flowed voxel model. (In Section 5.4 we generalize this to an intersection with a smooth surface fit to the flowed voxels.) 3b. The scene flow is then followed forwards and backwards in time to the neighboring time instants. 3c. The voxels at these time instants are then projected into the images and the images sub-sampled at the appropriate locations. 3d. The resulting samples are finally blended to give $I^*_+(u, v)$.

one of the input cameras C_i and the time is one of the time instants $t^* = t$, the algorithm should generate the input image I^t_i, *exactly.* We refer to this requirement as the *same-view-same-image* principle.

There are 2 components in the weighting function, space and time. The temporal aspect is the simpler case. We just have to ensure that when $t^* = t$ the weight of the pixels at time t is 1 and the weight at time $t + 1$ is 0. We weight the pixels at time t by $(t + 1) - t^*$ and those at time $t + 1$ so that the total weight is 1; i.e. we weight the later time $t^* - t$.

The spatial component is slightly more complex because there may be an ar-

bitrary number of cameras. We could use a simple weighting function where the weight for each camera is inversely proportional to the angle subtended between it and the novel camera, at some point on the surface [Debevec *et al.*, 1996b]. However, the major requirement to satisfy the same-view-same-image principle, is that when $C_+ = C_i$ the weight of the other cameras is zero. This can be achieved for time t as follows. Let $\theta_i(u, v)$ be the angle between the rays from C_+ and C_i to the flowed voxel $X_i^{t^*}$ at time t^*. The weight of pixel (u, v) for camera C_i is then:

$$\frac{1/(1 - \cos\theta_i(u, v))}{\sum_{j=1}^{\mathrm{Vis}(t,u,v)} 1/(1 - \cos\theta_j(u, v))} \tag{5.4}$$

where $\mathrm{Vis}(t, u, v)$ is the set of cameras for which the voxel X_i^t is visible at time t. This function ensures that the weight of the other cameras tends to zero as C_+ approaches one of the input cameras. It is also normalized correctly so that the total weight of all of the visible cameras is 1.0. An equivalent definition is used for the weights at time $t + 1$.

In summary (see also Figure 5.3), ray-casting across space and time consists of the following four steps:

3a. Intersect the (u, v) ray with S^{t^*} to get voxel $X_i^{t^*}$.

3b. Follow the flows to voxels X_i^t and X_j^{t+1}.

3c. Project X_i^t & X_j^{t+1} into the visible cameras at times t & $t + 1$.

3d. Blend the estimates as a weighted average.

For simplicity, the description of Steps 3a. and 3b. above is in terms of voxels. We now describe the details of these steps when we fit a smooth surface through these voxel models, and ray-cast onto it.

5.4 Ray-Casting to a Smooth Surface

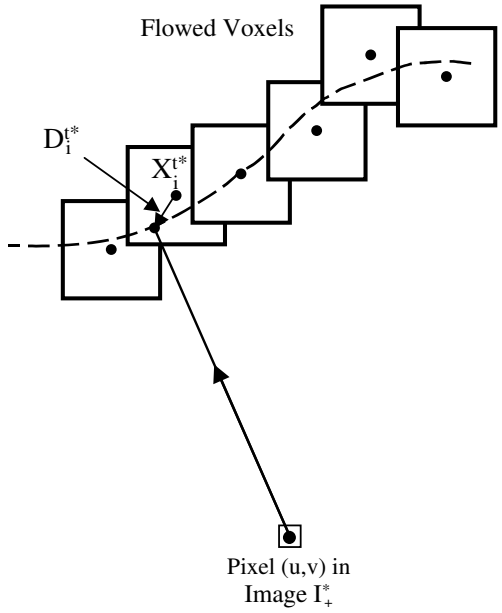


Figure 5.5: Ray-casting to a smooth surface. We intersect each cast ray with a smooth surface interpolated through the voxel centers (rather than requiring the intersection point to be one of the voxel centers, or boundaries.) Once the ray is intersected with the surface, the perturbation to the point of intersection D_i^{t*} can be transferred to the previous and subsequent time steps.

The ray-casting algorithm described above casts rays from the novel image onto the model at the novel time t^*, finds the corresponding voxels at time t and time $t+1$, and then projects those points into the images to find a color. However, the reality is that voxels are just point samples of an underlying smooth surface. If we just use voxel centers, we are bound to see cubic voxel artifacts in the final image, unless the voxels are extremely small.

The situation is illustrated in Figure 5.5. When a ray is cast from the pixel in the novel image, it intersects one of the voxels. The algorithm, as described above, simply takes this point of intersection to the be center of the voxel X_i^{t*}. If, instead, we fit a smooth surface to the voxel centers and intersect the cast ray with

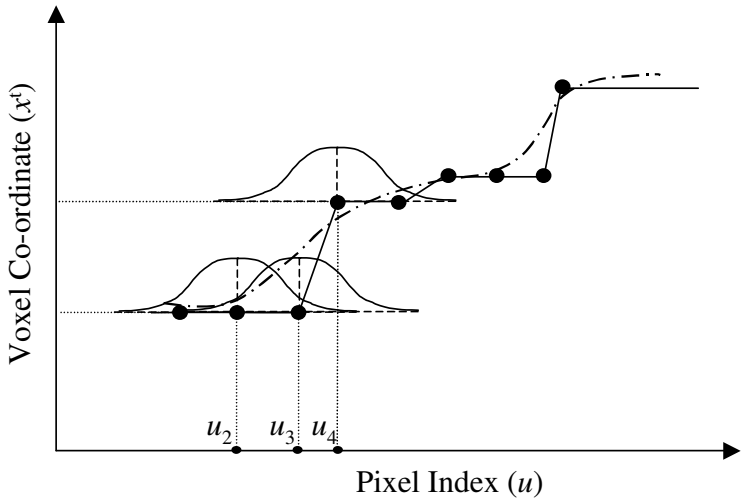


Figure 5.6: The voxel coordinate changes in an abrupt manner for each pixel in the novel image. Convolution with a simple gaussian kernel centered on each pixel changes its corresponding 3-D coordinate to approximate a smoothly fit surface.

that surface, we get a slightly perturbed point $X_i^{t*} + D_i^{t*}$. Assuming that the scene flow is constant within each voxel, the corresponding point at time t is $X_i^t + D_i^{t*}$. Similarly, the corresponding point at $t + 1$ is $X_j^{t+1} + D_i^{t*} = X_i^t + F_i^t + D_i^{t*}$. If we simply use the centers of the voxels as the intersection points rather than the modified points, a collection of rays shot from neighboring pixels will all end up projecting to the same points in the images, resulting in obvious box-like artifacts.

Fitting a surface through a set of voxel centers in 3-D is complicated. However, the main contribution of a fit surface in our case would be that it prevents the discrete jump while moving from one voxel to a neighbor. What is really important is that the interpolation between the coordinates of the voxels be smooth. Hence, we propose the following simple algorithm to approximate the true surface fit. For simplicity, we explain in terms of time t^* and time t, the same arguments hold for time $t + 1$.

For each pixel u_i in the novel image that intersects the voxel X^{t^*}, the coordinates of the corresponding voxel at time t, $X^t = (x^t, y^t, z^t)$ (which then get pro-

(a) Colored Voxel Model (b) Ray-Casting With Cubic Voxels (c) Ray-Casting With Surface Fit

Figure 5.7: The importance of fitting a smooth surface. (a) The voxel model rendered as a collection of voxels, where the color of each voxel is the average of the pixels that it projects to. (b) The result of ray-casting without surface fitting. showing that the voxel model is a coarse approximation. (c) The result of intersecting the cast ray with a surface fit through the voxel centers.

jected into the input images) are stored. We therefore have a 2-D array of (x, y, z) values. Figure 5.6 shows the typical variation of the x component of X^t with u_i. Because of the discrete nature of the voxels, this changes abruptly at the voxel centers, whereas, we really want it to vary smoothly like the dotted line. Therefore, we apply a simple gaussian operator centered at each pixel (shown for u_2, u_3, and u_4) to the function $x^t(u)$ to get a new value of $x^{'t}$ for each pixel u_i (and similarly for $y^t(u)$ and $z^t(u)$), that approximates the true fit surface. These perturbed values $X^{'t} = (x^{'t}, y^{'t}, z^{'t})$ for each pixel in the novel image are projected into the input images as described earlier. [Bloomenthal and Shoemake, 1991] suggest the use of convolution as a way to generate smooth potential surfaces from point skeletons, although their intent is more to generate a useful representation for solid modeling operations.

Figure 5.7 illustrates the importance of this surface fitting step. Figure 5.7(a) shows the voxel model rendered as a collection of voxels. The voxels are colored with the average of the colors of the pixels that they project to. Figure 5.7(b) shows the result of ray-casting by just using the voxel centers directly. Figure 5.7(c) shows

the result after intersecting the cast ray with the smooth surface. As can be seen, without the surface fitting step the rendered images contain substantial voxel artifacts.

5.5 Optimization Using Graphics Hardware

The algorithm described in the previous section involves two operations that can be very computationally expensive. However, it is possible to optimize these using standard graphics hardware, so that they are performed much more quickly. We now describe details of both of these optimizations.

5.5.1 Intersection of Ray with Voxel Model

Steps 3a. and 3b. of the ray-casting algorithm involve casting a ray out of pixel (u, v) in the novel camera at time t^*, finding the voxel $X_i^{t^*}$ that this ray intersects, and the corresponding voxels X_i^t and X_j^{t+1} at the two neighboring times. Finding the first point of intersection of a ray with such a voxel model is an expensive computation, since it typically involves an exhaustive search over all voxels (unless geometry-aware data structures are used). In addition, extra book-keeping is necessary to determine the corresponding voxels at the sampled time instants t and $t+1$, for each $X_i^{t^*}$.

Instead, we implement this as follows: Every voxel in the models S^t and S^{t+1} is given a unique ID, which is encoded as a unique (r, g, b) triplet. Then, both of these models are flowed as discussed in Section 5.2, to give us the voxel model S^* at time t^*. This model S^* at time t^* (see Equation 5.2) is rendered as a collection of little cubes, one for each voxel comprising it, colored with the unique ID that gives us back the corresponding voxels at time t and time $t+1$.

Then, the voxel model S^* is rendered from the viewpoint of the novel camera using standard OpenGL. Lighting is turned off (to retain the base color of the

cubes), and z-buffering turned on, to ensure that only the closest voxel along the ray corresponding to any pixel is visible. Immediately after the rendering, the color buffers are read and saved. Then, for any pixel (u, v), the (r, g, b) value at that pixel gives a unique index as to what voxel X_i^t or X_j^{t+1} this corresponds to.

This idea of using color to encode a unique ID for each geometric entity is similar in concept to that of the *item buffer*, introduced in [Weghorst *et al.*, 1984]. They use the item buffer as an aid for visibility computation in ray tracing, to find out which polygons are hit by an eye ray.

5.5.2 Determining Visibility of Cameras to Point on Model

Step 3c. of the algorithm involves projecting X_i^t (and similarly, X_j^{t+1}) into the images sampled at those times, to find out a suitable color. But, how do we know whether the point X_i^t was actually visible in any camera C_k?

Again, we use a z-buffer approach similar to the previous case, except this time, we don't need to encode any sort of information in the color buffers (that is, there are no voxel IDs to resolve). The occlusion test for Camera C_k runs as follows. Let $\mathbf{R}_k$ and $\mathbf{t}_k$ be the rotation matrix and translation vector for camera C_k relative to the world co-ordinate system. Then, X_i^t is first transformed to camera co-ordinates:

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{bmatrix} \mathbf{R}_k & \mathbf{t}_k \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{X}_i^t \tag{5.5}$$

The image co-ordinates of the projection $\mathbf{u} = (u, v)$ are obtained by multiplying by the 3×4 camera matrix P_k

$$\mathbf{u} = P_k \mathbf{x} \tag{5.6}$$

The voxel model is then rendered, with the camera transformation matrix set to be exactly that corresponding to the calibration parameters of camera C_k. After the rendering, the hardware z-buffer is read. This z-buffer now gives the depth to the nearest point on the shape for any particular pixel in the rendered image, and therefore any pixel in the real image as well, since the viewpoints are identical for both. (In reality, the value of the hardware z-buffer is between zero and one, since the true depth is transformed by the perspective projection that is defined by the near and far clipping planes of the viewing frustum. However, since these near and far clipping planes are user-specified, the transformation is easily invertible and the true depth-map can be recovered from the value of the z-buffer at any pixel).

Let (u, v) be the image co-ordinates in image I_k, as computed from Equation 5.6. The value of the z-buffer at that pixel, $z_k(u, v)$ is compared against the value of x_3 (which is the distance to the point $\mathbf{X}(i, j)$ from the camera). If $x_3 = z_k(u, v)$, then it means that it is the same point seen by the camera. Instead, if $x_3 > z_k(u, v)$, then it means that it is occluded by another part of the scene that is closer, and hence image I_k is occluded. Therefore, it should not be included in the list of cameras visible to the voxel X_i^t.

5.6 Experimental Results and Discussion

We have applied the spatio-temporal view interpolation algorithm to several real world dynamic events, which were captured within the CMU 3D Room (described in Appendix A). We show results for two different sequences: A short dance sequence, and a player bouncing a basketball. The input images, computed shapes and scene flows, and fly-by movies created using the spatio-temporal view interpolation algorithm may be seen at the URL:

`http://www.cs.cmu.edu/~srv/stvi/results.html`

The reconstruction results are shown in this chapter as a few snapshots from the fly-by movie. They are best viewed as movie clips from the website, and the material

there supplements the results shown in this chapter.

5.6.1 Sequence 1: Paso Doble Dance Sequence

The first event is a short Paso Doble dance routine. The image sequences captured are the same as those used in examples throughout the paper. In the sequence, the dancer turns as she uncrosses her legs, and raises her left arm.

The input to the algorithm consists of 15 frames from each of 17 cameras. The input frames for each of the cameras are captured 1/10 of a second apart, so the entire sequence is 1.5 seconds long. The 17 cameras are distributed all around the dancer, with 12 of the cameras on the sides, and 5 overhead. Figure 5.8 shows the input images at 5 time instants, from 3 different cameras.

Figure 5.9 shows the results of the computed voxel grids at the same 5 time instants as the inputs in Figure 5.8. The entire space is partitioned into $50 \times 50 \times 80$ voxels, and after the 6D carving algorithm, we are left with roughly 6500 surface voxels at each time instant.

Figure 5.10 shows the computed scene flows at those same time instants. At $t = 5$, the motion is mostly on her right leg as she unfolds it. At $t = 11$, she stretches out her arm, as seen from the computed scene flow vectors.

Figure 5.11 shows a collection of frames from a virtual fly-through of this dance sequence. The path of the camera is initially towards the scene, then rotates around the dancer and then moves away. Watch the floor (which is fixed) to get a good idea of the camera motion. We interpolate 9 times between each neighboring pair of input frames, and each of these is computed individually using the raycasting algorithm, after computing the scene shape and flow for the entire image sequence.

The movie dance_flyby.mpg shows the movie created by assembling all of the computed in-between images. dance_flyby_comparison.mpg shows the same movie, but compared with what would have been obtained, had we just switched between the closest sampled input images, both in space and time. Note that the the inputs

look like a collection of snap-shots, whereas the fly-by is a much smoother and natural looking re-rendering of the dance sequence.

Looking at the fly-by movie, some artifacts are visible, such as blurring, and occasional discontinuities. The blurring is because the shape estimation is imperfect, and therefore the corresponding points from neighboring cameras used to compute the output pixels are slightly misaligned. The discontinuities are because of imperfect computation of the scene flow - a few voxels have flows that are wrongly computed, and cannot therefore find corresponding voxels to flow to at the next time instant.

5.6.2 Sequence 2: Player Bouncing a Basketball

The second dynamic event we model is that of a player bouncing a basketball. In the sequence of images used, the player throws the ball to the ground as he starts to bend, and then catches it at a lower height than when it was first thrown down. The input images for this example consist of 20 frames ($t = 25$ to $t = 44$) from each of 13 different cameras, placed all around the player. The sequence is captured at 30 frames per second, therefore the actual length of the sequence we model is $\frac{2}{3}$ of a second long. Figure 5.12 shows the input images from 3 of the 13 cameras used. These 3 cameras are actually adjacent to each other, so they are indicative of the average spacing between cameras. 5 representative time instants are selected. The input images at $t = 31$ show some motion blur on the ball, because of its relatively high speed (the camera shutter speed was $\frac{1}{60}$ second).

The shape was computed at every time instant using the 6D carving algorithm. Figure 5.13 shows these voxel modes at 5 sample times. The reconstruction algorithm was run on a $60 \times 40 \times 80$ voxel grid, and consists of an average of around 8000 surface voxels, and 15000 interior voxels. The results of the scene flow computed using the algorithm described in Chapter 4 are shown at 3 of these time instants (the flow at the other two is close to zero). At $t = 31$, the player's left arm and the ball

are moving downwards. At $t = 34$, the arm is stationary, while the ball has a large instantaneous backwards motion just after it bounces from the floor. At $t = 39$, the ball is still moving upwards, but with a reduced upward velocity. Thus, we see that the scene flow is able to capture the motion of the ball, even at $t = 34$, when the shape appears like it is attached to the floor.

Figure 5.14 shows a collection of frames from a fly-by movie (bball_flyby.mpg), created by spatio-temporal view interpolation. The novel camera pans from left to right from $t = 25$ to $t = 31$, moves upwards till $t = 39$, and then moves back again till $t = 44$. This movie is re-timed by adding in 9 extra frames between each pair of original images, and therefore, we see that as it is played, the motion appears much smoother than the input images. Also, the novel viewpoint changes continuously. The color artifacts seen are because the different cameras are not color calibrated. We set the raycasting algorithm to use only the 3 closest cameras to avoid over blending between different camera textures, and as the novel viewpoint moves, one of the cameras that is far away as replaced by a closer one, causing the change in color. Also noticeable as the ball bounces back up, is that the ball gets very close to the player, leading to imperfect shape and scene flow estimation (some voxels on the ball end up flowing onto the leg and vice versa). This is what causes blended texture near the boundary of the ball and the player's leg.

5.6.3 Performance

The computation time of the spatio-temporal view interpolation algorithm per frame is linear in the number of pixels in the output image, irrespective of the complexity of the model (as are most image based rendering algorithms). Also, it is linear in the number of input images that are used to contribute to each pixel of the output. In our examples, we compute a 640×480 novel image, using the six closest images (3 closest cameras at each of 2 time instants). The algorithm takes about 5 seconds to run for each output frame, with the use of graphics hardware

for computing the ray-model intersection and visibility determination. This is after the shapes and scene flows have been estimated. However, the real bottleneck is the time taken to load all the images and to compute the scene flow, as described in Section 4.5.1. This computation is strongly dependent on the size of the voxel grid used to compute the model and the flow.

Of course, the quality of the output mainly depends on how far away from each other the input images are, both in space and time. In general, we have found that a temporal sampling rate of 15 frames per second is sufficient for normal human motions. For faster sequences (like the bouncing of the ball above), 30 frames per second are needed. If the amount of motion between frames is too large, the flow is incorrect, and this shows up as significant artifacts in the rendered images. In addition, the assumption that motion is linear between frames is less true, and interpolation of shape by linearly moving voxels starts to produce significant distortions in the intermediate shapes. Spatially, we have used between 15 and 20 cameras around an event with the geometry of a person. It appears, from experiments, that there should be a noticeable improvement in quality upto about 30 cameras for this order of complexity of the scene geometry.

Figure 5.8: A sample subset of the captured input images used for the dance sequence. The images are captured from various viewpoints, at multiple time instants as the sequence progresses.

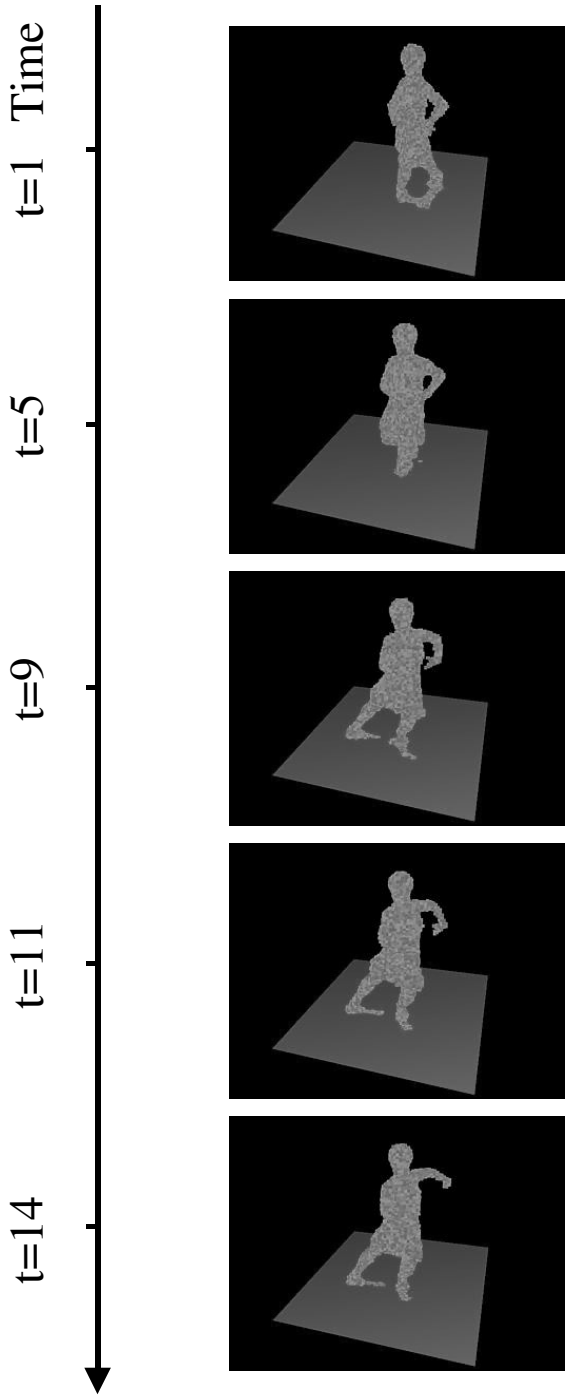


Figure 5.9: The shapes shown as voxel grids that are computed for each time instant. The time samples are the same as that in the previous figure.

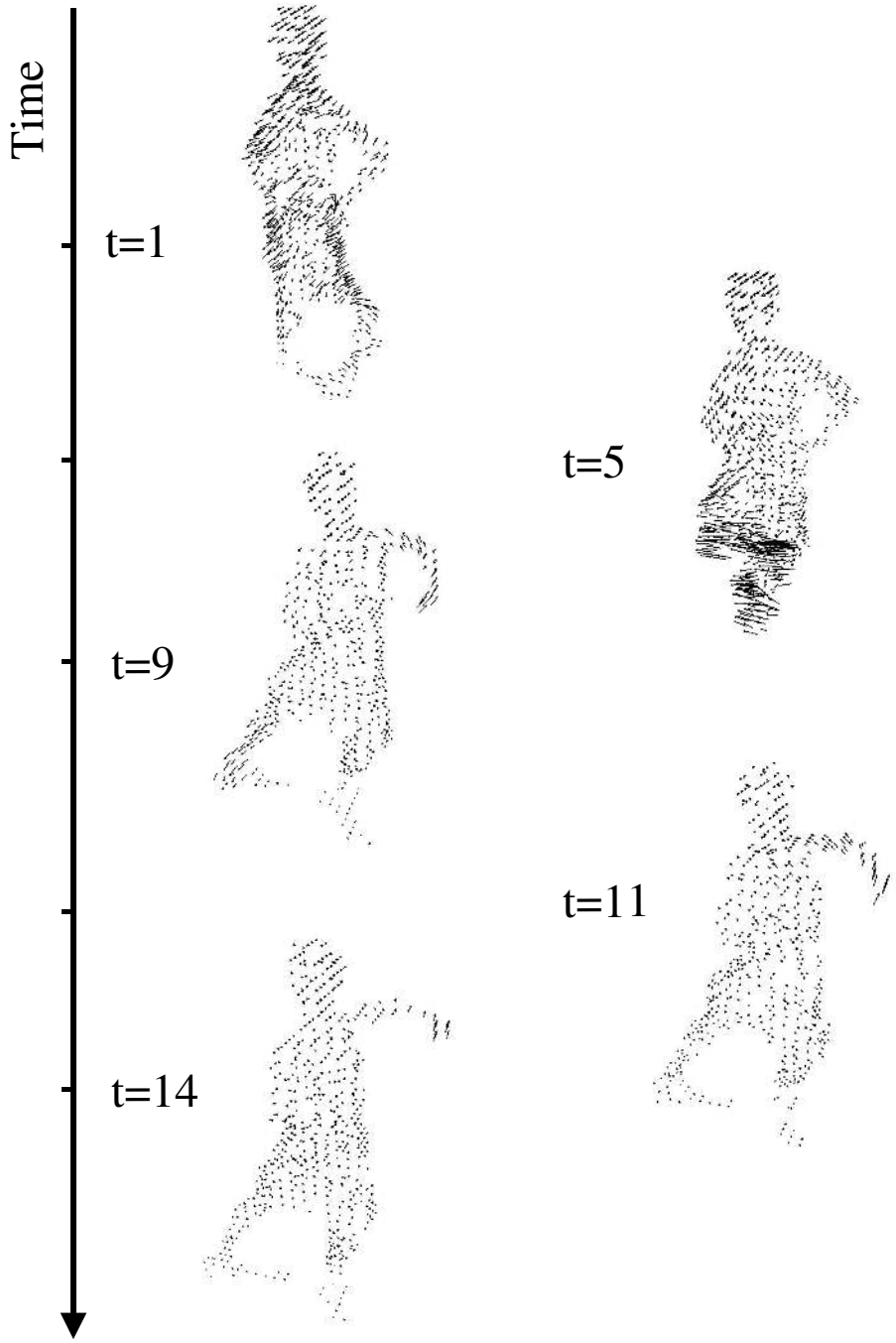


Figure 5.10: The computed scene flows at each of the same five time instants, for the dance sequence. The scene flow fields are shown as 3-D needle-maps.

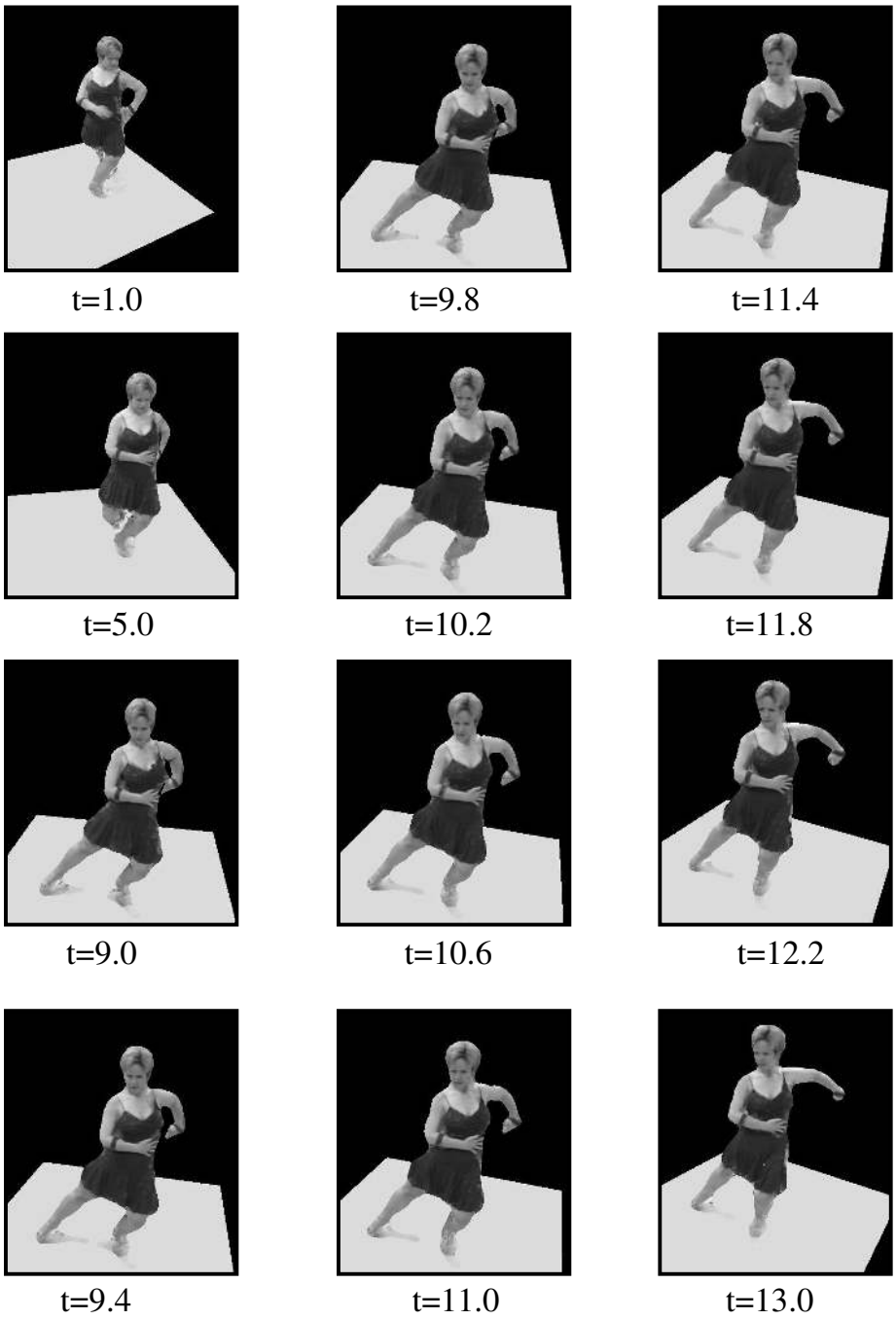


Figure 5.11: A collection of frames from a slow motion fly by movie (dance_flyby.mpg) of the dance sequence. Some of the inputs are shown in Figure 5.8. The novel camera moves along a path that first takes it towards the scene, then rotates it around the scene, and the takes it away from the dancer. The new sequence is also re-timed to be 10 times slower than the original camera speed.

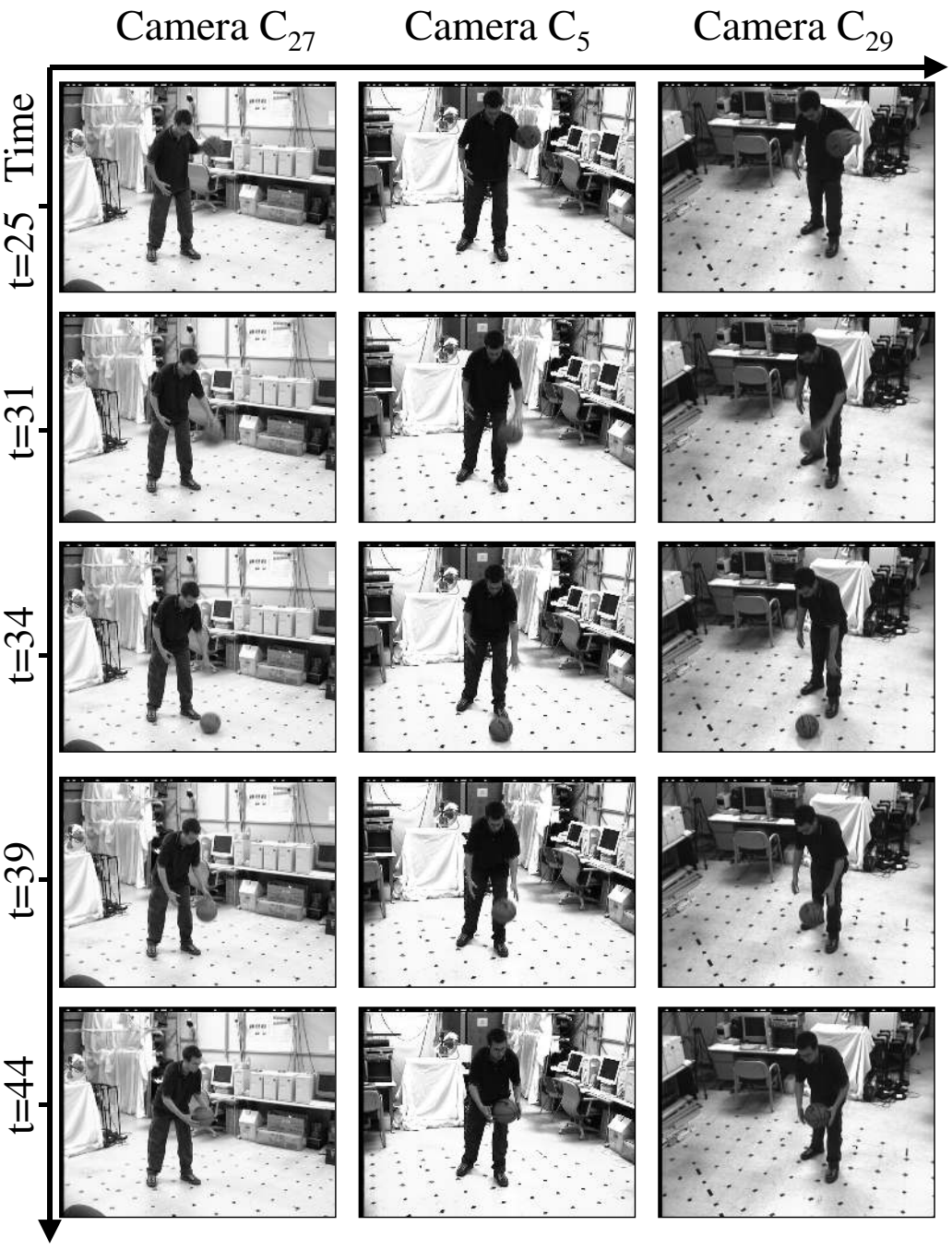


Figure 5.12: A few of the input images used for the basketball sequence. Images from 3 of the 13 input cameras, at 5 of the 20 time instants are shown here.

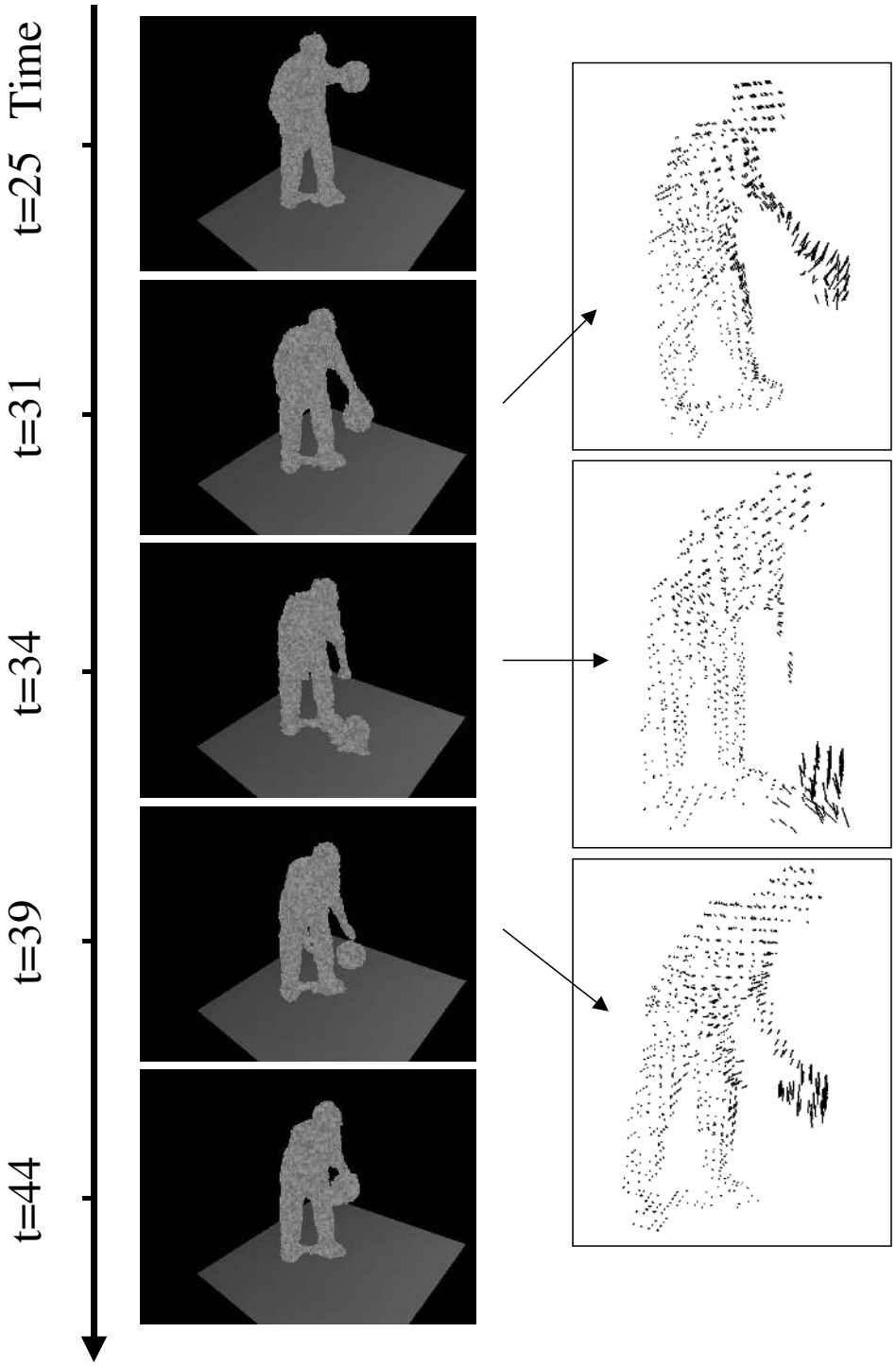


Figure 5.13: The shapes shown as voxel grids that are computed for each time instant, along with the flows at 3 times. The time samples are the same as those in the previous figure.

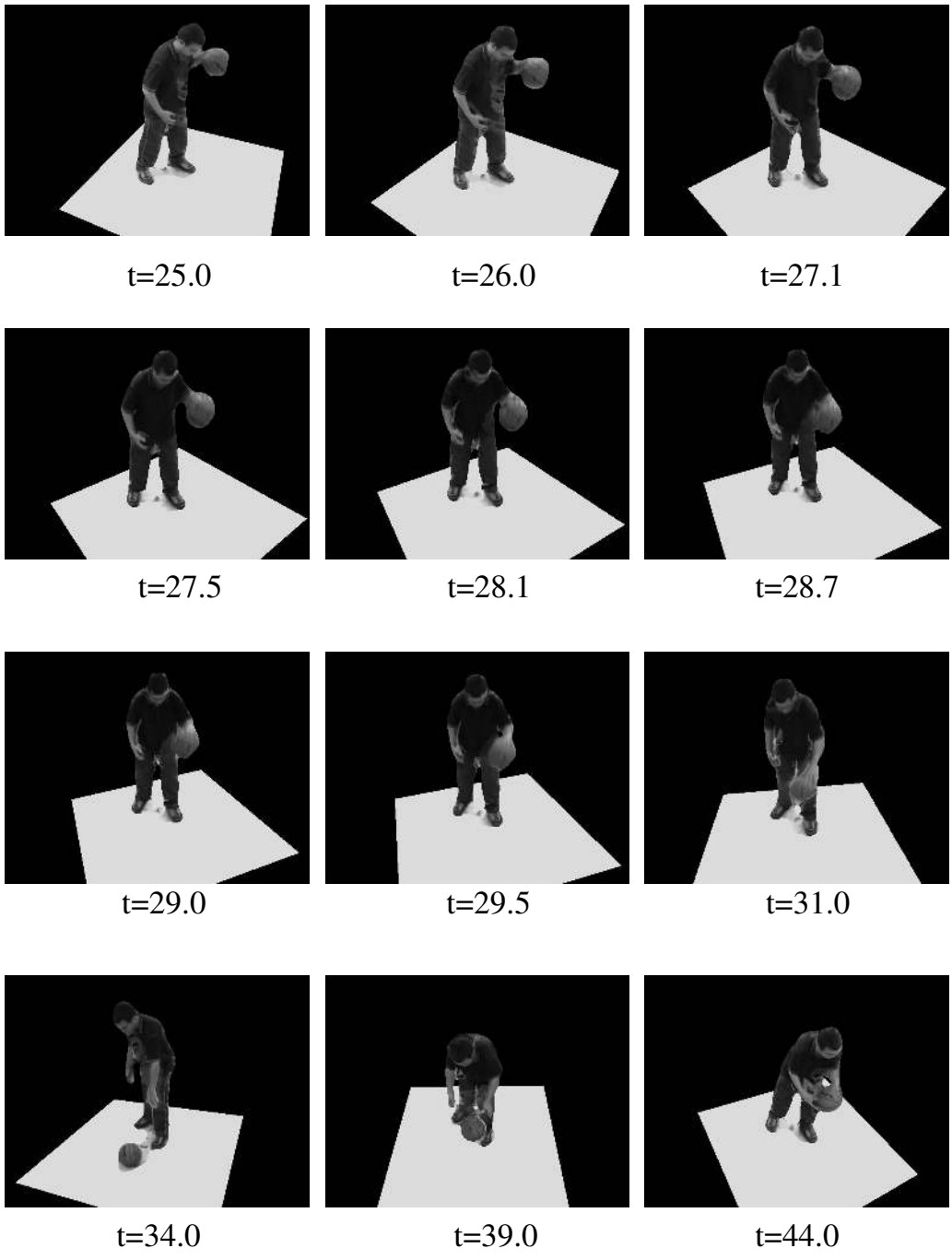


Figure 5.14: A collection of snapshots from the slow motion fly by movie (bball_flyby.mpg) of the basketball sequence. Some of the inputs are shown in Figure 5.12. The novel camera first pans from left to right, moves upwards, and then pans back to the left.

Chapter 6

Conclusions

In this thesis, we have defined and addressed the problem of spatio-temporal view interpolation, which involves re-rendering a dynamic, non-rigid scene using images captured from different positions in space and time. The sampled images are combined to synthesize the appearance of time-varying real world events from novel viewpoints at previously unsampled time instants. In doing so, we have also developed algorithms to compute a continuous 4D model of the scene (geometry and instantaneous motion) without using any higher level domain knowledge of the event. We shown results of the view interpolation algorithm by generating movie clips that show visual fly-bys of the event taking place. They are also re-timed, so that extra frames that show the event from new positions at originally unsampled time instants are added in, thus giving us smooth slow motions replays with a dynamic viewpoint.

6.1 Contributions

Most image based modeling work to date has only focused on the modeling of static objects. As a whole, the primary contribution of this thesis is the definition of the spatio-temporal view interpolation problem, and a 3-D image based modeling

and rendering algorithm specifically developed for non-rigidly varying dynamic scenes. The overall spatio-temporal view interpolation algorithm encompasses the areas of image based rendering, shape estimation, and non-rigid motion analysis. A representation for building a continuous 4D model of the shape and motion of the scene has also been developed. Just as an image based rendering algorithm for a static scene requires corresponding points between various images (either through a 3D model or by manual specification of correspondences), the spatio-temporal view interpolation algorithm needs corresponding points across images captured from different positions *and* at different time instants. We compute and use this model of shape and motion as a way to recover these correspondences that can be input to our spatio-temporal view interpolation algorithm.

We have introduced the concept of scene flow to characterize non-rigid scene motion and developed a theory of how it relates to scene geometry and optical flow. The various scene constraints that can be used to solve for the inherent ambiguity in scene flow have been studied. In addition, we have addressed a number of issues that arise in using scene flow with voxel models for shape interpolation and spatio-temporal view interpolation, such as enforcing flow to surface voxels only, bijectivity, relationship with optical flow, smoothness, normal flow constraints, and single and multiple camera cases. Then, we presented an algorithm to compute scene flow using optical flow as a means to enforce smoothness constraints.

The joint space of scene shape and non-rigid flow has been studied, and we have presented hexels as a representation of one element in the 6D space of two neighboring shapes and the scene flow between them. This is a first step towards the complete understanding of the temporal aspects of shape variation from multiple sets of images viewing a dynamic scene.

Using the hexel representation, we have developed an algorithm to simultaneously recover the shape and non-rigid motion of the scene. The shape and motion carving algorithm developed is an extension to the voxel coloring method [Seitz and Dyer, 1999] that uses the notion of photo-consistency across multiple time in-

stants. However, in addition to just the scene shape, the algorithm simultaneously estimates two neighboring shapes and the scene flow between them. We now have a 4D model of the scene (geometry and instantaneous motion) that can be used to temporally interpolate scene structure using the point-to-point flow between them, rather than using purely geometric constraints such as maximizing rigidity [Alexa *et al.*, 2000]. This model can also be used to determine corresponding pixels across images that have been captured at different positions, and at different times.

The actual spatio-temporal view interpolation algorithm developed (that uses scene shape and flow as inputs) provides an effective way to generate novel views. For any novel time specified, the voxel models are interpolated in a manner consistent with the scene flow and preserving the inclusion/onto properties. An important contribution is the result showing that using just the centers of the voxels as a shape model leads to obvious artifacts, but that there is no need for a complex surface fitting step in order to eliminate these. Instead, a simple algorithm is proposed that approximates this surface fit, satisfying the main requirement that the interpolation between the co-ordinates of the voxel centers be smooth. A ray-casting algorithm is developed, which for any pixel in the novel image, uses the voxel and flow representation to find the corresponding pixels in the various images that have useful color information. This is integrated with the surface fitting approximation so that artifacts of using a discrete voxel grid are eliminated in the final rendered image.

We have also demonstrated that it is possible to use the algorithm to create retimed fly-by renderings of a dynamic scene. This continuous control over both the position and the timing of any one shot has great potential in changing how movies are typically made - the shot can be taken just once, and all the camera control now becomes a post-shooting exercise.

6.2 Future Work

This work has been a first attempt at an approach for spatio-temporal modeling and view interpolation of dynamic real-world events from images, and lends itself to a number of possible extensions and new research directions.

While the basic ideas behind scene flow have been developed in this work, the algorithm to compute it is by no means an end in itself. The problem is more complicated and ill-posed than optical flow, which is still an active area of research after over 20 years of work. Combining smoothness and photo-consistency constraints is an interesting area worth exploring. In addition, computing scene flow for using voxel models as a shape representation produces a new set of challenges, since the space of flows is now discrete, and the into/onto properties described in Section 5.2.2 are desirable. While the voxel representation was an obvious choice in this work, it is unclear that voxels are the best representation of shape for scene flow. It is possible that implicit representations [Turk and O'Brien, 1999] or surface representations such as oriented particles [Szeliski and Tonnesen, 1992] may lend themselves to more robust algorithms. [Carceroni and Kutulakos, 2001] is a promising approach to using a surface element representation to recover shape and flow together.

With the use of constraints on motion, it is arguably possible to improve the accuracy of the shape that is computed using images at one time alone. Photo-consistency across space and time is a fundamental idea that can lead to many different algorithms for shape estimation using motion constraints.

To simplify the analysis in this work, several assumptions were made. The scene was always assumed to be lambertian, which does not take into effect specularities and reflection effects. In computing the flow, it was assumed that the amount of relative motion between the frames is not very large. Developing more robust scene flow algorithms that can handle large amounts of motion is an area for future research, as the brightness constancy assumption will no longer be valid.

It was also assumed that no higher level information about scene geometry was known. If for example, we know that the scene only involves humans, articulated models of shape and/or knowledge of human behavioral patterns can constrain the search space significantly, while also providing implicit smoothness priors. On the other hand, the domain knowledge needs to be good - errors due to calibration and real-world noise can result in the system not converging to any solution, while attempting to fit the data to a model that is too stiff. Clearly, this is an area wide open for further investigation.

An interesting application for spatio-temporal view interpolation is the modeling of fast moving scenes, where the normal temporal sampling rate is inadequate. This will give us a unique 3D re-timing capability, so that fly-through reconstructions can be performed in slow motion, without the familiar jerkiness associated with the limited number of frames in regular slow-motion replays. In fact, with the continuous control of timing and spatial position of the novel viewpoint, special effects such as motion blur, time-lapsed fades, and other effects so far found only in video games (that use purely synthetic data) can now be extended to use visual models of real world events, and thereby substantially enhance the immersive experience.

Appendix A

Image Acquisition Facility: 3D Room

Figure A.1: The CMU 3D Room used for capturing the multi-viewpoint time-varying image data.

The CMU 3D Room [Kanade *et al.*, 1998] is an image acquisition facility that was built during the course of this thesis, as part of the Virtualized Reality project (See `http://www.cs.cmu.edu/~virtualized-reality` for more information). This is a facility for recording image sequences of dynamic, life-sized events from many different angles using multiple video cameras, and was used to capture all the datasets used in this thesis.

The room consists of 49 cameras mounted along the four walls and ceiling, as shown in Figure A.1. The cameras are all pointed roughly towards the center

of the room, giving about a $2\mathrm{m} \times 2\mathrm{m} \times 2\mathrm{m}$ workspace that can be seen by all cameras. Of these, 14 are high-quality 3CCD progressive scan cameras, while the remaining 35 are commercial grade security cameras (in this work, only the 14 3CCD cameras were used). All of the cameras are externally synchronized using a single genlock signal. The video streams from each of these cameras are then time-coded (details are explained in [Narayanan *et al.*, 1995]), and then sent to frame-grabbers in PCs, which digitize these video streams into image sequences. With three cameras connected to each PC, there are 17 PCs to digitize all the video streams. Image data is first streamed into memory as the event takes place, and then saved to disk off-line. Note that all of this data is captured and digitized in real time passively, without obstructing the event itself in any manner.

To collect a dataset, the cameras are first calibrated (for both extrinsic and intrinsic parameters). We sweep a large calibration grid through the volume, and use the well-known Tsai calibration algorithm [Tsai, 1986] to compute the camera parameters. When the user is ready to collect data, the PCs are all instructed to begin recording at a designated time, shortly in the future. Then, as the event takes place, it is recorded and digitized in real-time by all the cameras and PCs in a synchronous manner. Images are saved to memory in real time and once the memory is full, they are all saved to disk off-line. Currently, this imposes a limit of about 7 seconds on the length of the sequence, since each camera streams NTSC video (640×486) at 30 frames per second, which translates to roughly 18 Megabytes per second per channel when digitized at 16 bits per pixel. As they are saved, the time-code on each image is also extracted, which lets us uniquely extract the set of images captured at any given time from all the different cameras.

This recording system has been used successfully by the Virtualized Reality project [Narayanan *et al.*, 1998, Rander, 1998, Kanade *et al.*, 1999], and also for real-time shape reconstruction [Cheung *et al.*, 2000]. It was an outgrowth of the earlier *3D Dome* [Rander *et al.*, 1997] system, which recorded each of the video streams on analog VCRs and was thus limited to off-line applications.

Bibliography

[Adelson and Bergen, 1991] E. Adelson and J. Bergen. The plenoptic function and the elements of early vision. In Landy and Movshon, editors, *Computational Models of Visual Processing*. MIT Press, 1991.

[Alexa *et al.*, 2000] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 157–164, 2000.

[Avidan *et al.*, 1997] S. Avidan, T. Evgeniou, A. Shasua, and T. Poggio. Image-based view synthesis by combining trilinear tensors and learning techniques. In *VRST '97*, pages 103–110, September 1997.

[Barron *et al.*, 1994] J.L. Barron, D.J. Fleet, and S.S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[Baumgart, 1974] B.G. Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University, Palo Alto, 1974.

[Bloomenthal and Shoemake, 1991] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 251–256, 1991.

[Bregler and Malik, 1998] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *CVPR '98*, pages 8–15, 1998.

[Buehler *et al.*, 2001] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, 2001.

[Carceroni and Kutulakos, 1999] R.L. Carceroni and K.N. Kutulakos. Multi-view 3D shape and motion recovery on the spatio-temporal curve manifold. In *7th ICCV*, 1999.

[Carceroni and Kutulakos, 2001] Rodrigo Carceroni and Kiriakos Kutulakos. Multi-view scene capture by surfel sampling: From vdeo streams to non-rigid 3d motion, shape and reflectance. In *Proceedings of the IEEE International Conference on Computer Vision*, 2001.

[Chen and Williams, 1993] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '93)*, pages 279–288, 1993.

[Cheung *et al.*, 2000] G.K.M. Cheung, T. Kanade, J.Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages II:714–720, 2000.

[Christy and Horaud, 1996] S. Christy and R. Horaud. Euclidean shape and motion from multiple perspective views by affine iterations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11):1098–1104, November 1996.

[Collins, 1996] Robert Collins. A space sweep approach to true multi-image matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 358–363, June 1996.

[Curless and Levoy, 1996] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *Computer Graphics*, 30(Annual Conference Series):303–312, 1996.

[Debevec *et al.*, 1996a] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based

approach. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH ' 96)*, pages 11–20, 1996.

[Debevec *et al.*, 1996b] P.E. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96*, pages 11–20, 1996.

[Fua, 1993] P. Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6:35–49, 1993.

[Gortler *et al.*, 1996] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '96)*, pages 43–54, 1996.

[Gortler *et al.*, 1997] Steven J. Gortler, Li-Wei He, and Michael F. Cohen. Rendering layered depth images. Technical Report MSTR-TR-97-09, Microsoft Research Advanced Technology Division, Redmond, WA, March 19 1997.

[Guenter *et al.*, 1998] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin. Making faces. *SIGGRAPH 98*, pages 55–66, 1998.

[Hilton *et al.*, 1996] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. *Lecture Notes in Computer Science*, 1064:117–??, 1996.

[Horn, 1986] B.K.P. Horn. *Robot Vision*. McGraw Hill, 1986.

[Kanade *et al.*, 1998] Takeo Kanade, Hideo Saito, and Sundar Vedula. The 3d room: Digitizing time-varying 3d events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1998.

[Kanade *et al.*, 1999] T. Kanade, P. Rander, S. Vedula, and H. Saito. Virtualized reality: Digitizing a 3d time-varying event as is and in real time. In Yuichi Ohta and Hideyuki Tamura, editors, *Mixed Reality: Merging Real and Virtual Worlds*, chapter 3. Springer-Verlag, 1999.

[Kang and Szeliski, 1997] S.B. Kang and R. Szeliski. 3-d scene data recovery using omnidirectional multi-baseline stereo. *ijcv*, 25(2):167–183, November 1997.

[Kutulakos and Seitz, 1999] K. Kutulakos and S. Seitz. A theory of shape by space carving. In *7th ICCV*, 1999.

[Laurentini, 1994] A. Laurentini. The visual hull concept for silhouette-based image understanding. *PAMI*, 16(2):150–162, February 1994.

[Laveau and Faugeras, 1994a] Stephane Laveau and Olivier Faugeras. 3-d scene representation as a collection of images. Technical Report Technical Report RR-2205, INRIA - The French Institute for Research in Computer Science and Control, February 1994. Available from http:www.inria.fr.

[Laveau and Faugeras, 1994b] Stephane Laveau and Olivier Faugeras. 3-d scene representation as a collection of images and fundamental matrices. Technical Report No-2205, INRIA Sophia-Antipolis, February 1994. Available from http://www.inria.fr.

[Levoy and Hanrahan, 1996] Marc Levoy and Pat Hanrahan. Light field rendering. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '96)*, pages 31–42, 1996.

[Liao *et al.*, 1997] W.-H. Liao, S.J. Aggrawal, and J.K. Aggrawal. The reconstruction of dynamic 3D structure of biological objects using stereo microscope images. *Machine Vision and Applications*, 9:166–178, 1997.

[Lippman, 1980] A. Lippman. Movie-maps: An application of the optical videodisc to computer graphics. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 32–42, 1980.

[Lorenson and Cline, 1987] W. Lorenson and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 163–169, 1987.

[Lucas and Kanade, 1981] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981.

[Malassiotis and Strintzis, 1997] S. Malassiotis and M.G. Strintzis. Model-based joint motion and structure estimation from stereo images. *CVIU*, 65(1):79–94, 1997.

[Marr and Poggio, 1979] D. C. Marr and T. Poggio. A computational theory of human stereo vision. *Proc. of the Royal Soc. of London*, B 204:301–328, 1979.

[Matusik *et al.*, 2000] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image-based visual hulls. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 369–374, 2000.

[McMillan and Bishop, 1995] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 39–46, 1995.

[Metaxas and Terzopoulos, 1993] D. Metaxas and D. Terzopoulos. Shape and nonrigid motion estimation thr-ough physics-based synthesis. *PAMI*, 15(6), 1993.

[Narayanan *et al.*, 1995] P.J. Narayanan, P.W. Rander, and T. Kanade. Synchronizing and capturing every frame from multiple cameras. Technical Report CMU-RI-TR-95-25, Robotics Institute, Carnegie Mellon University, 1995.

[Narayanan *et al.*, 1998] P.J Narayanan, P.W. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *Proc. of the Sixth ICCV*, pages 3–10, 1998.

[Negahdaripour and Horn, 1987] S. Negahdaripour and B.K.P. Horn. Direct passive navigation. *PAMI*, 9(1):168–176, 1987.

[Nishino *et al.*, 1998] Ko Nishino, Yoichi Sato, and Katsushi Ikeuchi. Eigen texture method - appearance compression based on 3d model. Technical Report IIS-CVL-98-102, Institute of Industrial Science, The University of Tokyo, July 1998.

[Ohta and Kanade, 1985] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(2):139–154, March 1985.

[Okutomi and Kanade, 1993] M. Okutomi and T. Kanade. A multiple-baseline stereo. *pami*, 15(4):353–363, 1993.

[Penna, 1994] M.A. Penna. The incremental approximation of nonrigid motion. *CVGIP*, 60(2):141–156, 1994.

[Pentland and Horowitz, 1991] A. Pentland and B. Horowitz. Recovery of nonrigid motion and structure. *PAMI*, 13(7):730–742, 1991.

[Rander *et al.*, 1996] P.W. Rander, P.J Narayanan, and T. Kanade. Recovery of dynamic scene structure from multiple image sequences. In *Proc. of the 1996 Intl. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, pages 305–312, 1996.

[Rander *et al.*, 1997] P.W. Rander, P.J Narayanan, and T. Kanade. Virtualized reality: Constructing time-varying virtual worlds from real world events. In *Proceedings of IEEE Visualization 1997*, pages 277–283, Phoenix, AZ, October 1997.

[Rander, 1998] Pater Rander. A multi-camera method for 3d digitization of dynamic, real-world events. In *Ph. D. thesis, Robotics Institute, Carnegie Mellon University*, june 1998.

[Sato *et al.*, 1997] Yoichi Sato, Mark Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, 1997.

[Seitz and Dyer, 1996] S. M. Seitz and C. R. Dyer. View morphing. In *Computer Graphics Proceedings, Annual Conference Series (Proc. SIGGRAPH '96)*, pages 21–30, 1996. Available from ftp.cs.wisc.edu.

[Seitz and Dyer, 1997] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1067–1073, 1997.

[Seitz and Dyer, 1999] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *IJCV*, 35(2):1–23, 1999.

[Szeliski and Tonnesen, 1992] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 185–194, 1992.

[Szeliski and Zabih, 1999] R. Szeliski and R. Zabih. An experimental comparison of stereo algorithms. In *IEEE Workshop on Vision Algorithms, pages 59–66*, 1999.

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

[Tsai, 1986] Roger Tsai. An efficient and accurate camera claibration technique for 3d machine vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 364–374, June 1986.

[Turk and O'Brien, 1999] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, 1999.

[Ullman, 1984] S. Ullman. Maximizing the rigidity: The incremental recovery of 3-D shape and nonrigid motion. *Perception*, 13:255–274, 1984.

[Vedula *et al.*, 1998] S. Vedula, P. Rander, H. Saito, and T. Kanade. Modeling, combining, and rendering dynamic real-world events from image sequences. In *Proceedings of Fourth International Conference on Virtual Systems and Multimedia*, pages 326–332, Gifu, Japan, November 1998.

[Vedula *et al.*, 1999] S. Vedula, S. Baker, P. Rander, R Collins, and T. Kanade. Three dimensional scene flow. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, September 1999.

[Vedula *et al.*, 2000] S. Vedula, S. Baker, S. Seitz, and T. Kanade. Shape and motion carving in 6d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume II, pages 592–598, 2000.

[Vedula *et al.*, 2001] Sundar Vedula, Simon Baker, and Takeo Kanade. Spatio-temporal view interpolation. Technical Report CMU-RI-TR-01-35, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, November 2001.

[Waxman and Duncan, 1986] A. Waxman and J. Duncan. Binocular image flows: Steps toward stereo-motion fusion. *PAMI*, 8(6):715–729, 1986.

[Weghorst *et al.*, 1984] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.

[Wheeler *et al.*, 1998] M. Wheeler, Y. Sato, and K. Ikeuchi. Consensus surfaces for modeling 3d objects from multiple range images. In *Proceedings of 6th ICCV, Bombay*, 1998.

[Young and Chellappa, 1999] G.S. Young and R. Chellappa. 3-D motion estimation using a sequence of noisy stereo images: Models, estimation, and uniqueness. *PAMI*, 12(8):735–759, 1999.

[Zhang and Faugeras, 1992] Z. Zhang and O. Faugeras. Estimation of displacements from two 3-D frames obtained from stereo. *PAMI*, 14(12):1141–1156, 1992.

[Zitnick and Kanade, 1998] C.L. Zitnick and T. Kanade. A volumetric iterative approach to stereo matching and occlusion detection. Technical Report CMU-RI-TR-98-30, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1998.