Surface Tracking from Multiview Video

by

Salih Cihan Bilir

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science
in
Electrical and Computer Engineering

Koc University

March 2009

Koc University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Salih Cihan Bilir

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Yucel Yemez, Assistant Professor (Advisor)

_____

T. Metin Sezgin, Assistant Professsor

_____

Cagatay Basdogan , Assistant Professsor

Date: _____

ABSTRACT

We present a fast and efficient surface tracking method for modeling dynamic objects from multiview video streams. Starting from an initial mesh representation, the surface of a dynamic object is tracked over time, both in geometry and connectivity, based on multiview silhouette and 3D scene flow information. The mesh representation of each frame is obtained by deforming the mesh representation of the previous frame towards the optimal surface defined by the time-varying multiview silhouette information, using mesh restructuring operations and vertex displacements assisted by 3D scene flow vectors. The whole time-varying surface is then represented as a mesh sequence that can efficiently be encoded in terms of restructuring operations and small-scale vertex displacements along with the initial model. Our reconstruction method hence yields a compact time-varying mesh representation of the dynamic object, which is smooth both in time and space. The proposed method is not only fast and produces storage efficient mesh representations, but it also has the ability to deal with dynamic objects that may undergo nonrigid transformation. The time-varying mesh structure of such nonrigid surfaces, which is not necessarily of fixed connectivity, can also successfully be tracked thanks to the restructuring operations employed in our deformation scheme. We demonstrate the performance of the proposed method both on real and synthetic sequences.

ÖZETÇE

Zamanla değişen nesnelerin çok-bakışlı video dizilerinden 3B geriçatımı için hızlı ve verimli bir yüzey izleme yöntemi tanıtılmaktadır. Dinamik bir nesne örgü modelinin geometrisi ve bağlanırlığı, çok-bakışlı silüet ve 3B sahne akış bilgisine dayalı bir yöntemle, bir ilk gösterimden yola çıkarak, zaman içinde izlenir. Her çerçeveye ait örgü gösterimi, bir önceki çerçevenin örgü gösterimini silüet bilgisi ile belirlenen optimal yüzeye doğru deforme ederek elde edilir. Bu deformasyon süreci, örgü yeniden-yapılandırma işlemleri ve 3B sahne akış bilgisi ile desteklenir. Elde edilen uzay-zamanda pürüzsüz örgü dizisi, örgü yeniden-yapılandırma işlemleri ve tepe noktalarının küçük ölçekli yerdeğiştirme vektörleri cinsinden, ilk çerçevenin örgü gösterimi ile birlikte, verimli bir şekilde kodlanabilir. Önerilen yöntemin hızlı olması ve gösterim maliyeti düşük örgü modelleri üretmesinin yanısıra, bir diğer avantajı da devinimi katı olmayan dinamik nesnelerin modellenmesi için kullanılabilmesidir. Devinimi katı olmayan bir nesneyi temsil eden örgü gösteriminin zamanla değişebilen bağlanırlığı, deformasyon şırasında kullanılan örgü yeniden-yapılandırma işlemleri sayesinde başarı ile izlenebilmektedir. Yöntemin başarımı, hem gerçek hem de sentetik video dizileri üzerinde sınanmıştır.

ACKNOWLEDGEMENTS

I would like to thank Assist. Prof. Yücel Yemez who has guided, inspired, and encouraged me with his profound knowledge and genuine support throughout this journey. His vision contributed to my perspective in life as much as it contributed to this thesis.

I would also like to thank Assist. Prof. Tevfik Metin Sezgin and Assist. Prof. Çağatay Başdoğan for taking part in my thesis committee.

I am also grateful to Prof. Murat Tekalp, Assoc. Prof Alper Erdoğan, and Assist. Prof. Engin Erzin for their support and guidance during my graduate studies. Each of my colleagues in the graduate school in these years played a major role in keeping me motivated to my goal, special thanks goes to all my colleagues.

My final gratitude is to the people who always make me feel special by their love and care, my dear family and my precious girlfriend.

This work has been supported by TUBITAK under the project EEEAG-105E143 and by the European FP6 Network of Excellence 3DTV.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

t	time/frame index
$M^{(t)}$	mesh representation of frame t
T	mesh transformation
$S^{(t)}$	object surface at frame t
e	error/distance of an object surface and its mesh representation
Dist	Euclidian distance of a vertex to object surface
$\mathbf{d}_{\text{sil}}$	silhouette based displacement vector of a vertex
$\mathbf{d}_{\text{flow}}$	scene flow based displacement vector of a vertex
δ	magnitude of a displacement vector
ε_{min}	minimum edge length constraint
ε_{max}	maximum edge length constraint
Δ	smoothing component of a vertex
L	Laplacian displacement for a vertex
F	fairing displacement for a vertex
$\mathbf{N}$	surface normal
$\mathbf{p}$	vector representation of a vertex
ζ	threshold for the collision margin
$f(\cdot)$	isolevel function
mü	
$\text{Proj}_I(\mathbf{v})$	projection of a vertex to an image plane
G	Bilinearly interpolated value of a sub-pixelic point.
$\mathbf{w}$	scene flow vector at a vertex
$\hat{\mathbf{v}}$	position of a vertex at the target surface

$[\mathbf{P_c}]$	projection matrix of a camera
Vis($\mathbf{v}$)	set of camera viewpoints that the vertex $\mathbf{v}$ is visible to
C	camera plane
$\mathbf{R}$	rotation matrix of the pose registration parameters
$\mathbf{t}$	translation vector of the pose registration parameters
α	weight coefficient of the silhouette based displacement
β	weight coefficient of the scene flow based displacement
τ	time coefficient for the displacement weights
$\hat{K}$	expected number of iterations to converge for a vertex
B	bit-load of a mesh sequence through the proposed approach
B_0	bit-load of a mesh sequence through the classical approach
P	amount of bits to represent a floating point coordinate

Chapter 1

INTRODUCTION

3D modeling of dynamic real scenes is an emerging research field with applications in various domains such as 3D television, free viewpoint video, virtual reality and computer animation [1], [2]. Unlike optical motion capture systems which are widely used in computer animation applications [3], 3D video methods aim to recover the complete shape of a dynamic object, not only its motion. Most of the techniques addressing the dynamic object modeling problem adhere to passive surface reconstruction methods exploiting silhouette and/or stereo information acquired from multicamera video sequences [4], [5], [6], [7], [8], [9], due to the limitations of active reconstruction methods in temporal axis [10].

The goal of dynamic scene modeling schemes is usually to generate a sequence of meshes each of which represents the geometry of a dynamic object at the corresponding video frame. There are three major challenges involved in achieving this goal. The first two of these challenges concern efficiency: computational complexity of the reconstruction method and the resulting representation load. A time-varying scene sampled at a standard rate of 30 frames per second would yield enormous 3D model data for representation and a considerable amount of time for reconstruction if no particular care is shown to exploit redundancies between consecutive time frames. In this respect, time-varying mesh representations with fixed connectivity, but with changing vertex positions, would certainly provide efficiency both for storage and processing. The third challenge concerns generality of the proposed solutions, that is, their applicability to modeling general dynamic scenes with arbitrary shape and motion. Existing methods often aim at fixed connectivity

representations and/or make use of object-specific prior models. Hence they consider primarily rigid and/or articulated motion, and may not handle the reconstruction problem when the object of interest undergoes an arbitrary nonrigid motion or deformation.

In this thesis, we present an efficient surface tracking method for modeling dynamic objects based on multiview silhouette and 3D scene flow information. Here the term "surface tracking", in the way we use it, refers to reconstruction of the surface geometry of a dynamic object at time $t + 1$ based on the reconstruction at time t, starting from an initial representation at $t = 0$. There exist actually very few methods in the literature, which are surface tracking in this sense and which can build complete shape models of dynamic objects [4], [5], [7], [8]. The main distinction of the method that we present in this thesis, as compared to previous work, is in the way we represent time-varying geometry. We relax the fixed connectivity requirement and encode time-varying geometry in terms of both connectivity changes and vertex displacements. Relaxing the fixed connectivity constraint has two major impacts. First, in this way objects with arbitrary shape and motion can easily be handled. Second the reconstruction problem is reduced to an energy minimization problem which can be solved by a fast snake-based deformation scheme. Unlike existing surface tracking methods, our scheme does not require any object-specific mesh representation, or 3D models separately reconstructed for all frames of the sequence prior to the tracking process. Starting from an initial mesh representation, the surface of the dynamic object is tracked over time, both in geometry and connectivity, based on mesh deformation. We assume that the topology of the object in the scene remains unchanged over time, as it actually does in real scenes. Nevertheless, we address the self-collision problem, which is disregarded in most surface tracking methods via a very efficient collision handling strategy.

The mesh representation of each frame is obtained by evolving the mesh of the previous frame towards the optimal surface defined by the time-varying multiview

silhouette information, using mesh restructuring operations and vertex displacements. These mesh operations and small-scale displacements along with the initial mesh representation yield a compact and spatiotemporally coherent representation of the whole time-varying surface. Our deformable model is based mainly on the dynamic triangle meshes scheme which was proposed in [11] for mesh editing purposes. This scheme enables us to control parametrization, smoothness and uniformity of the dynamic mesh model for a robust mesh evolution across time.

1.1 Related Work

There is a vast and quite mature literature on 3D reconstruction of static objects. In general, reconstruction techniques for static scenes can be collected under two groups: active and passive. Active techniques make use of calibrated light sources such as lasers and coded light [10]. Most of the active scene capture technologies become inapplicable in the dynamic case since currently it is very difficult to scan the whole surface of an object at a standard rate of 30 Hz. There exist though several attempts to achieve scanning at standard rates such as in [12], [13] by projecting coded light patterns on the object. The methods proposed in these works however have severe limitations on resolution, object's surface properties and its motion, and are capable of producing only depth images, not full surface representations. On the other hand, passive reconstruction techniques, which are based on solely image cues such as multiview stereo [14] and/or silhouettes [15], are mostly free of these limitations and hence they currently seem to be a more viable option for the dynamic object modeling problem.

Most of the methods in the literature proposed for dynamic object modeling require as a first step that the object shape, which is usually represented as a surface mesh, be reconstructed from scratch, separately for each time instance [6], [7], [8], [9], [16]. The

resulting sequence of meshes can then be matched so as to obtain a time consistent representation with fixed connectivity. In order to achieve temporal coherence in this sense, Starck et al [6] use spherical reparametrization of the resulting mesh sequence whereas other methods basically cast the reconstruction problem to a surface tracking problem: Starting from an initial mesh, the time-varying geometry is tracked over time by preserving the connectivity and exploiting the temporal redundancies between consecutive frames [7], [8], [16]. Hence the problem becomes finding a suitable transformation that maps the vertices of a mesh at time t onto the surface represented by another mesh at $t + 1$.

Two other recent and closely related works [4], [5] follow a very particular approach to capture human performances from multiview video. Prior to video recording, they first take a static full-body scan of the subject using a laser scanner and construct a detailed complete 3D mesh model. This mesh model representing the shape of the human actor in the first frame is then tracked over time by preserving the connectivity based on multiview image cues. In particular, the method in [4] presents very high quality reconstructions but the method requires some user interaction and an extensive computation time which is reported as about 10 minutes per frame on a standard computer. Moreover the method, which aims at a fixed connectivity representation, has no mechanism to handle arbitrary nonrigid motion and self-collisions.

Methods for reconstruction of dynamic objects rely mainly on multiview silhouette information [17]. The strength of the shape from silhouette technique lies in its simplicity, efficiency and robustness especially when applied to convex shapes. The main drawback of this technique is that it fails to capture hidden concavities. Multiview stereo information on the other hand can be incorporated into reconstruction schemes in several different ways. It can be used for instance to enhance silhouette based reconstructions so as to capture finer surface concavities [4], or to impose additional constraints on the silhouette reconstruction process to avoid self-occlusion problems [6]. Another possibility is to compute 3D scene

flow vectors or image feature based 3D correspondences to incorporate into the mesh tracking process [5], [7], [8]. Relying too much on 3D scene flow vectors, which are very prone to errors, as in [5] for instance, may however fail the tracking process especially when the motion in the scene is very fast and complex. In our earlier work [18], we have shown that, given a sufficient number of multiview silhouette images at each frame, the time-varying geometry of an object with a relatively complex shape, such as a human actor, can be tracked based on solely silhouette information in a very fast manner using a snake-based deformable model. In this thesis, we basically follow the same framework but extend it with additional features to make it more efficient and robust such as incorporation of 3D scene flow into the deformation scheme.

Surface tracking methods usually resort in some way or other to mesh deformation methods, such as Laplacian deformation [19], which is a powerful tool for mesh morphing and editing, and which can be used to obtain animating mesh sequences with fixed connectivity [4]. However, with Laplacian deformation which is a differential but piecewise linear scheme, mesh connectivity cannot be altered, hence dynamic objects with arbitrary motion cannot be tracked. Another alternative [20] is based on volumetric level-set technique and builds a spatially and temporally smooth surface model. Level set based deformation is however computationally very demanding. Although it can implicitly handle topological changes in geometry, the topology control is often very difficult to achieve. Moreover, with the level set approach, the explicit connectivity information of the initial shape model is lost through the iterations between the initial state and its convergence. Thus the level set technique becomes inapplicable to track objects in motion and to build efficient time-varying representations. In this respect, snake-based deformable models, when coupled with restructuring operations as we do in this work, enable keeping track of the changes both in geometry and connectivity and hence they are more appropriate to

track surfaces with arbitrary motion and shape. The methods in [7], [8] also employ snake-based deformable models, but neither of them addresses the connectivity tracking problem.

1.2 Overview and Organization

The block diagram of the overall surface tracking scheme is given in Figure 1.1. The basic input data to reconstruct the surface representation at frame t+1 are:

i) the camera calibration/projection parameters,
ii) the object silhouettes at frame t+1,
iii) the 3D scene flow vectors from frame t to t+1,
iv) the pose registered mesh representation of frame t.

The main tasks to prepare these input are: multicamera video acquisition, initial reconstruction, silhouette extraction, 3D scene flow estimation and pose registration. The multicamera video acquisition block provides the camera calibration/projection parameters and the multiview images of each frame to be used in silhouette extraction and 3D scene flow estimation.

The raw input to the surface tracking scheme is the multiview video sequence of the dynamic scene captured with a calibrated multicamera system. The camera calibration/projection parameters are of use in several stages of the system. The images obtained are in use in two primary tasks i) to extract the silhouettes for each view at each frame, ii) to estimate the 3D scene flow vectors for each consecutive frame couple. The initial surface model of the first frame is reconstructed prior to the surface tracking/deformation task by using a shape from silhouette technique that produces a topologically correct shape model which is eligible for further deformation [21]. The overall time-varying surface representation of the dynamic scene is then reconstructed by successively estimating the surface representation of each time frame by deforming the

mesh representation of the previous frame based on the multiview silhouette information assisted by the 3D scene flow vectors. This surface tracking process produces a sequence of meshes, $M^{(0)}$, $M^{(1)}$, ..., $M^{(t)}$, ..., representing the time-varying geometry. For the surface evolution to successfully converge to the desired mesh representation, at each frame we first estimate the global rigid motion (translation and rotation) of the object from its 3D scene flow vectors so as to register the 3D pose of the starting mesh with reference to the target. This initial pose registration does not only improve the chances of the surface evolution to successfully converge to the desired surface, but it also speeds up the deformation process. Each mesh representation reconstructed at each frame t is then fed back to the tracker for 3D scene flow computation and pose registration at the next frame t+1.

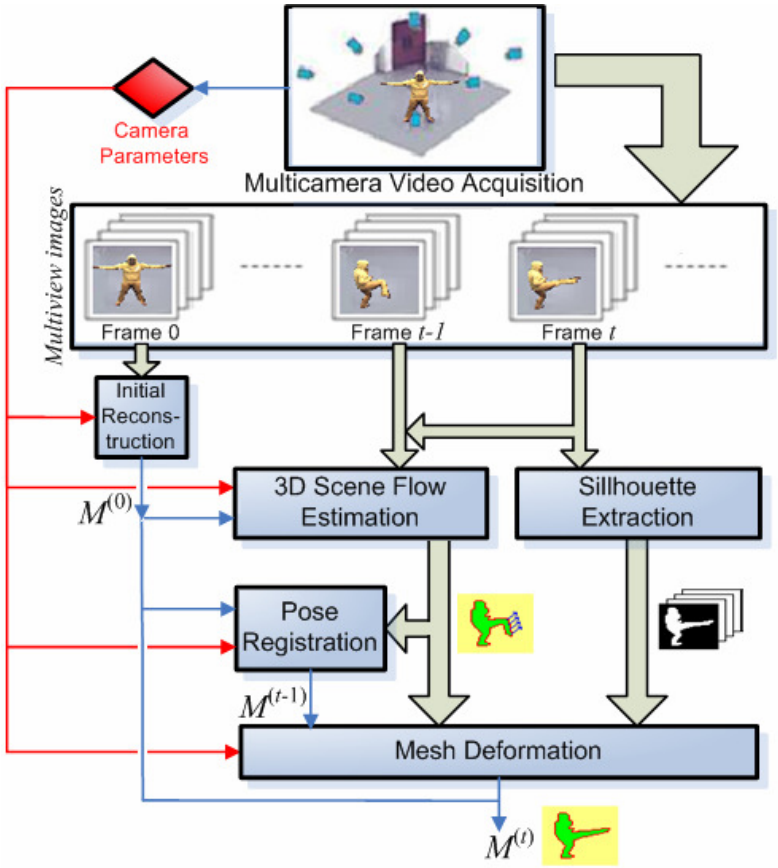

Figure 1.1: The block diagram of the overall surface tracking scheme.

The organization of the thesis is as follows. In Chapter 2 we describe the generic surface deformation framework that we employ for surface tracking. In Chapter 3, we explain how we drive this deformation scheme to track surface representations of dynamic objects using the time-varying silhouettes and the 3D flow vectors of the scene. In this chapter we also provide a pseudocode for the overall surface tracking algorithm, and estimate the bit-load of our dynamic connectivity representation scheme. Chapter 4 presents and discusses the experimental results, and finally Chapter 5 gives concluding remarks and some future research perspectives.

Chapter 2

SURFACE DEFORMATION SCHEME

Our deformation technique is based on the iterative use of an appropriate transformation T that deforms, at each frame t, an initial triangle mesh $M_0^{(t)}$ towards the object surface $S^{(t)}$ through the following surface evolution equation:

$$M_{k+1}^{(t)} = T(M_k^{(t)}) \tag{2.1}$$

The deformable model $M_k^{(t)}$ is required to remain as a smooth topologically correct mesh representation free of geometrical distortions during its evolution and to converge to an optimal mesh $M_{k'}^{(t)}$ that faithfully represents the object surface $S^{(t)}$ at the equilibrium state

$$M_{k^*}^{(t)} = T(M_{k^*}^{(t)}) \tag{2.2}$$

We define T as the composition of three transformations: $T = T_d \circ T_r \circ T_s$, which we will refer to as displacement, smoothing and restructuring operators, respectively. The displacement operator pushes the deformable mesh towards the object surface based while the smoothing operator regularizes the effect of this displacement and the restructuring operator modifies the mesh connectivity to eliminate any geometrical distortions that may appear during surface evolution. In this sense, the displacement operator corresponds to the external force whereas the other two correspond to the internal force of the classical snake formulation [23].

2.1 Vertex Displacement

The distance between the deformable mesh M_k and the object surface S, dropping the index t, can be approximated by the average distance from the vertex set of M_k to the surface:

$$e(M_k, S) = \frac{1}{N} \sum_{i=1}^{N_k} \text{Dist}(\mathbf{v}_{i,k}, S) \tag{2.3}$$

where $\mathbf{v}_{i,k}$ is the position vector of the i th vertex, N_k is the number of mesh vertices, and Dist($\mathbf{v}_{i,k}$, S) is the Euclidean distance of the vertex to the surface S. To reduce the distance $e(M_k, S)$, the operator $T_d(M_k)$ maps the deformable mesh M_k to M_k' by moving each vertex $\mathbf{v}_{i,k}$ with a displacement $\mathbf{d}(\mathbf{v}_{i,k})$

$$\mathbf{v}_{i,k}' = \mathbf{v}_{i,k} + \mathbf{d}(\mathbf{v}_{i,k}) \tag{2.4}$$

where $\{ \mathbf{v}_{i,k}' \}$ is the vertex set of the transformed mesh M_k' which has the same connectivity as M_k. The direction and the magnitude of this displacement vector $\mathbf{d}(\mathbf{v}_{i,k})$ is computed based on the signed distance from the vertex $\mathbf{v}_{i,k}$ to the target surface S at each iteration, as will later be explained in Chapter 3. Also note that the magnitude of the displacement has to be bounded above for a stable surface evolution.

The distance function defined in Eq. 2.3 is only a discrete approximation of the true distance. Moreover it does not take into account the distance from the surface S to M_k. Hence the optimality of the surface obtained at convergence heavily depends on two factors, the resolution and the location of the initial deformable mesh. If the initial mesh is of sufficiently high resolution and initially placed near the object surface S, the surface evolution is expected to converge to an optimal surface that accurately represents the target surface.

2.2 Mesh Restructuring

The restructuring operator, T_r, is the composition of three operators: $T_r = T_{split} \circ T_{col} \circ T_{flip}$, which are edge split, edge collapse and edge flip transformations (Figure 2.1) introduced in [26] for mesh optimization. We use these elementary transformations in the way [11] uses them for mesh editing. At the end of each iteration of the surface evolution, the operator T_{split} first splits all edges longer than ε_{max} at their midpoints. Then, the operator T_{col} successively eliminates all edges shorter than ε_{min} by edge collapses. Finally, the flip operator T_{flip} is applied to reduce the number of irregular vertices possibly created by the previous collapse and split operations. For the split operation to be compatible with the collapse operation, the threshold ε_{max} has to be chosen such that $\varepsilon_{max} \geq 2\varepsilon_{min}$ since otherwise split operations would create edges with length smaller than ε_{min}. We set $\varepsilon_{max} = 3\varepsilon_{min}$ to have uniformly sized triangles with small aspect ratios. Since the edge length ratio is then bounded by $\varepsilon_{max} / \varepsilon_{min} = 3$ and the valence distribution preserves its uniformity by flip operations, the deformable mesh maintains a high quality in terms of the aspect ratio of the triangles during surface evolution.

Thanks to the restructuring operation applied at each iteration of the surface evolution, the deformable mesh can adapt its shape to the object surface, avoiding geometrical distortions such as degenerate triangles and irregular vertices. Note that, with the restructuring operator as formulated above, the surface evolution results in an optimal surface M_{k^*} that has the same topology as the initial mesh M_0 unless explicit topology modifying operators for merging and/or splitting are incorporated.

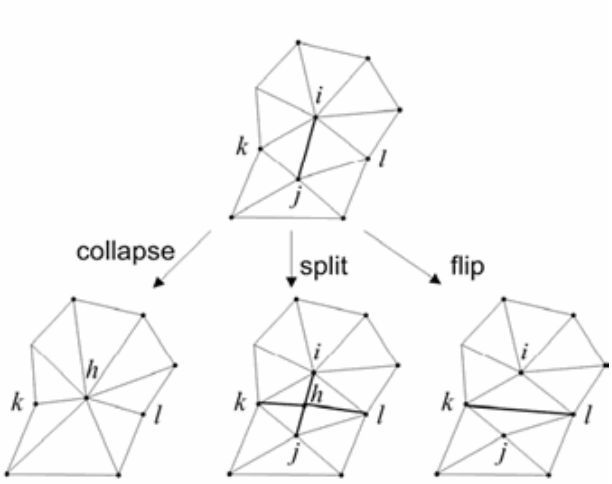

Figure 2.1: Edge operations: collapse, split and flip.

***Edge collapse*:** As the vertices are pulled towards the object boundary by the displacement operator, neighboring vertices may get too closer and cause degenerate edges. Thus we collapse an edge by merging their endpoints to the midpoint whenever its length falls below the threshold $\varepsilon_{\min}$. The merging point can actually be optimized according to the needs of the application; it can be for instance one of the endpoints, whichever is appropriate, or the optimal position on the edge if it is possible to define one. The edge collapse operation may occasionally cause a mesh triangle to fold over another and may create a non-manifold triangulation. As explained in [26], the collapse of an edge defined by two vertices P_i and P_j is legal in a closed manifold mesh if and only if for all vertices P_k adjacent to both P_i and P_j , $\{P_i , P_j , P_k \}$ is a face of the mesh. To strictly comply with the minimum edge length constraint, whenever an illegal collapse operation is encountered, we first detect those vertices P_k for which $\{P_i , P_j , P_k \}$ is not a face, remove them from the mesh structure, and then safely apply the collapse operation. This process of handling illegal edge collapses is depicted in Figure 2.2:

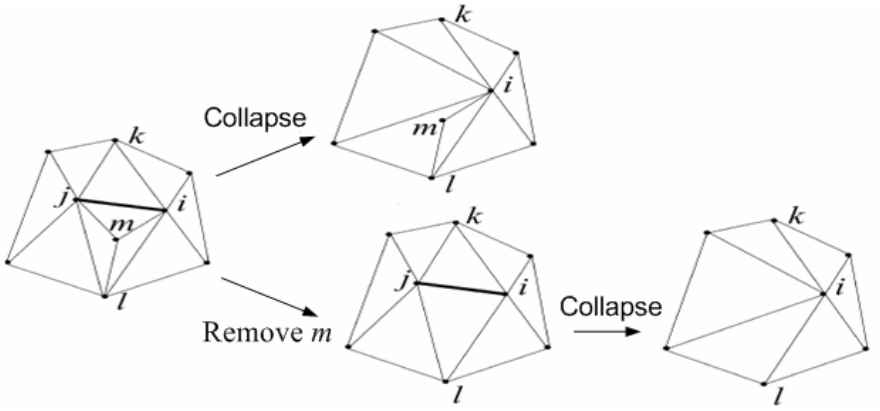

Figure 2.2: Handling illegal edge collapses. (Top row) The collapse of the edge $\{i,j\}$ produces a non-manifold triangulation. (Bottom row) The vertex m is removed from the mesh structure and the edge collapse operation becomes a legal move.

***Edge split*:** Similarly, as the deformable surface evolves, neighboring vertices may get further from each other and cause very long edges that degrade the regularity of the mesh. Moreover, parts of the deformable mesh where such long edges accumulate can not capture the details of the object shape. Thus whenever the length of an edge exceeds a certain threshold $\varepsilon_{max}= 3\varepsilon_{min}$, an additional vertex is inserted on the middle position of such an edge and the data structure is updated accordingly. For the split operation to be compatible with the collapse operation, the threshold ε_{max} has to be chosen such that $\varepsilon_{max} \geq 2\ \varepsilon_{min}$ since otherwise the split operation would create edges with length larger than ε_{min}. Note that the split operations must be applied in an appropriate order to avoid split operations causing new edges exceeding ε_{max}. All the edges of the mesh, that need to be split, are first arranged in descending order with respect to their lengths and then split in that order. The significance of taking the order into account while splitting edges is illustrated in Figure 2.3.

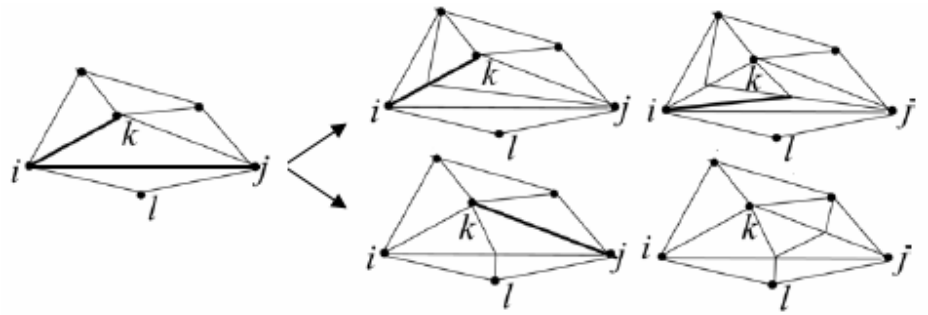

Figure 2.3: Significance of taking the order into account when splitting edges. Splitting first the longer edge *ij* (top row) yields a more plausible triangulation than splitting the shorter edge *ik* (bottom row) first.

***Edge flip*:** Edge collapse and split operations inevitably change the valence distribution of the mesh structure, that yield irregular vertices. To prevent this, during surface evolution, the common edge of any two neighboring triangles is swapped with the one joining the unshared vertices of the triangles, as long as this operation favors the existence of the vertices of valence close to 6. An edge flip is allowed if and only if the edge is adjacent to two triangles whose union is a convex quadrilateral. Figure 2.4 illustrates an edge split operation that results in a non-manifold triangulation.

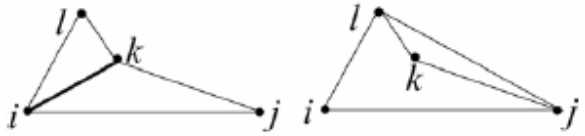

Figure 2.4: An illegal flip operation. Flipping the edge *ik* with *lj* creates a non-manifold triangulation; hence it is not allowed.

2.3 Mesh Smoothing

The smoothing operator, T_s, is necessary for a robust mesh evolution that is free of topological errors and to have eventually a visually pleasant fair surface representation. It should be easy to compute, yet must not yield any geometrical shrinkage and bias in the final surface estimate. To achieve this, at the end of each frame transition we employ a combination of the tangential Laplacian smoothing [25] and Taubin's surface fairing technique [24], and during the evolution, at each iteration, we employ the tangential component of the classical Laplacian smoothing such that the mesh representation of the surface preserves its volume without any shrinking while the mesh geometry is regularized. It is essential to avoid displacements along the surface normal while smoothing during the evolution so that the smoothing operator does not slow down the process of inflating towards a target surface. The operator $T_s(M)$ maps the deformable mesh M to M' by moving each vertex $\mathbf{v}$ to $\mathbf{v}'$ (dropping the vertex index i and the iteration index k), according to rather the evolution has converged or not.

$$\mathbf{v}' = \begin{cases} \mathbf{v} + \Delta\mathbf{v}_T & \text{during evolution} \\ \mathbf{v} + \Delta\mathbf{v}_T + \Delta\mathbf{v}_N & \text{at convergence} \end{cases} \tag{2.5}$$

where the displacements $\Delta\mathbf{v}_T$ and $\Delta\mathbf{v}_N$ correspond to smoothing along tangential and normal directions of the surface, respectively. We obtain the tangential component, $\Delta\mathbf{v}_T$, by tangential Laplacian smoothing:

$$\Delta\mathbf{v}_T = L(\mathbf{v}) - (L(\mathbf{v}) \cdot \mathbf{N})\mathbf{N} \tag{2.6}$$

where $L(\mathbf{v})$ denotes the Laplacian displacement that moves the vertex $\mathbf{v}$ to the centroid of the vertices in its one-ring neighborhood. The component $\Delta\mathbf{v}_N$, on the other hand, is obtained by fairing the surface along its normal direction:

$$\Delta\mathbf{v}_N = (F(\mathbf{v}) \cdot \mathbf{N})\mathbf{N} \tag{2.7}$$

where $F(\mathbf{v})$ denotes the displacement created by the non-shrinking surface fairing algorithm described in [24]. These two components of the smoothing operator are depicted on Figure 2.5 by successively applying them on a random noise added mesh structure.

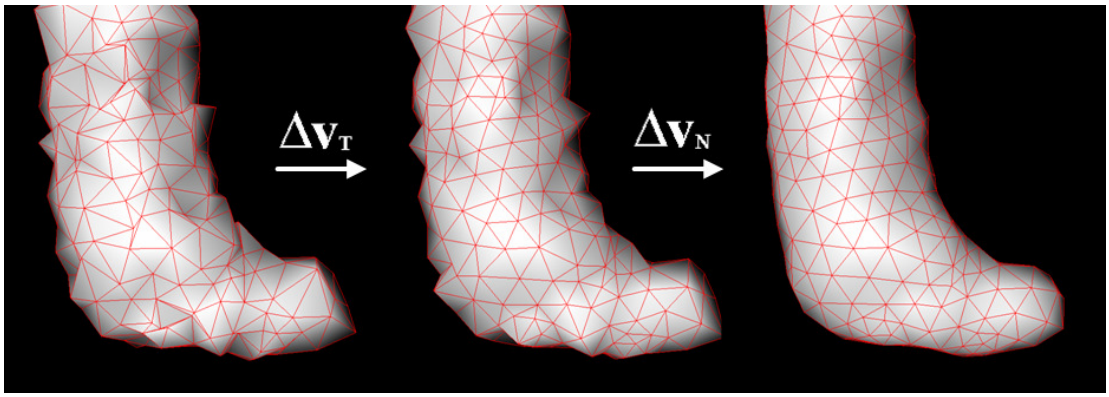

Figure 2.5: The components of the smoothing operator on a noisy mesh structure. Left to right: Random noise added model; Laplacian smoothing along surface tangential; Fairing along surface normal (applied only once at convergence).

2.4 Collision handling

A dynamic surface, as it moves or deforms, may collide with itself at various occasions (self-collision). These collisions must be detected and handled properly for a flawless surface evolution scheme as depicted in Figure 2.6. We have developed an efficient collision detection algorithm to handle such self-collisions.

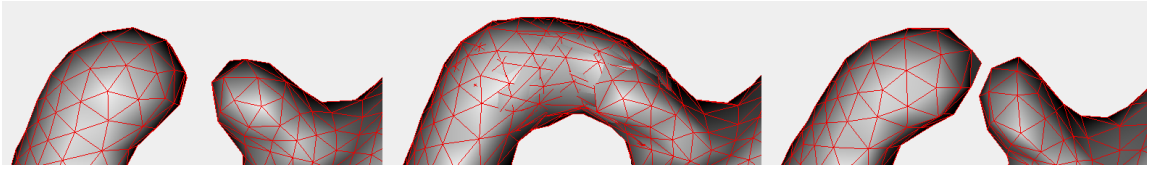

Figure 2.6: Illustration of collision handling. Left to right: The initial position of two segments that are about to collide; A few iterations later the segments evolve into each other if the collisions are not handled; The segments preserve a margin to avoid collision.

Our algorithm is based on the minimum and maximum edge length constraints, ε_{min} and ε_{max} respectively, imposed on the deformable mesh. Note that the intersection of neighbouring vertices during the surface evolution is avoided thanks to the regularization in the smoothing operator in each iteration and the minimum edge length constraint ε_{min}. Also recall that the magnitude of the displacement of each vertex in each iteration is bounded above by half of the minimum edge length constraint ($\varepsilon_{min}/2$). The basic idea in collision detection is to prevent non-neighboring vertices from approaching each other by more than some distance threshold ζ. We define neighboring vertices as those which are in the two-ring neighborhood of each other.

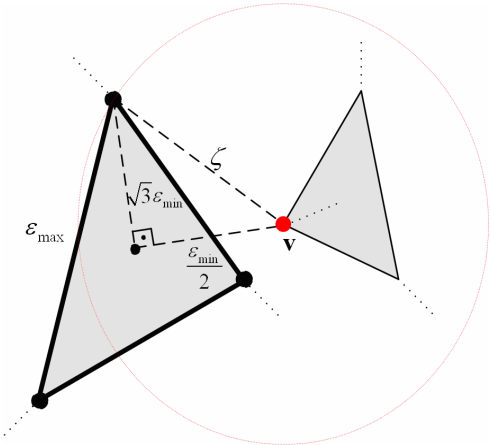

Figure 2.7: Worst case scenario for the decision of collision threshold ζ. The vertex v is checked against the largest possible triangle on the mesh, that is, an equilateral triangle of side ε_{max}. If the vertex approaches to any vertex of the mesh by more than the threshold ζ, a collision occurs.

The procedure that we use to handle collisions is as follows. Right after all the vertices of the deformable mesh are displaced by the displacement operator, each vertex is checked one by one against the non-neighboring vertices. If a vertex is found to have approached any other vertex by more than the collision detection threshold ζ, then a collision is detected and the vertex is moved back to its position before the displacement operator. The value of the collision threshold ζ depends on the maximum edge length parameter ε_{max} and the displacement bound $\varepsilon_{min}/2$. For the decision of this threshold ζ that guarantees that all collisions are detected, we consider the worst case scenario where the position of a vertex is checked against the largest possible triangle on the deformable mesh, which is an equilateral triangle of sides $\varepsilon_{max,}$ as visualized in Figure 2.7. The vertex must not approach to any point inside this triangle by more than the maximum possible displacement $\varepsilon_{min}/2$ since otherwise, at the next iteration, the triangle and the vertex may move at opposite directions, intersecting each other and falling apart by more than $\varepsilon_{min}/2$ distance. Note that the centroid of the triangle is the farthest interior point from all three corners of the triangle

with distance $3\varepsilon_{min} / \sqrt{3} = \sqrt{3}\,\varepsilon_{min}$ to each of the corners. The collision zone starts at the point whose vertical distance from the centroid is $\varepsilon_{min}/2$. The distance of this point to each of the triangle vertices is given by the diagonal length of the right triangle formed by this point, the centroid and a triangle vertex. Hence the collision threshold ζ must satisfy the following inequality,

$$\zeta > \sqrt{\left(\sqrt{3}\varepsilon_{min}\right)^2 + \left(\frac{\varepsilon_{min}}{2}\right)^2} \geq \frac{\sqrt{13}}{2}\varepsilon_{min} \tag{2.8}$$

By using a uniform partitioning where each vertex is associated with a voxel of a 3D grid and by checking each vertex only against those in its neighbouring voxels, the collision detection algorithm is implemented in an efficient manner with O($n$log$n$) complexity, where n is the number of vertices in the deformable mesh. The length of one side of a voxel is set to ζ, so that wherever the vertex of interest is located in its voxel, the sphere of radius ζ is covered by the search space, as depicted in Figure 2.8:

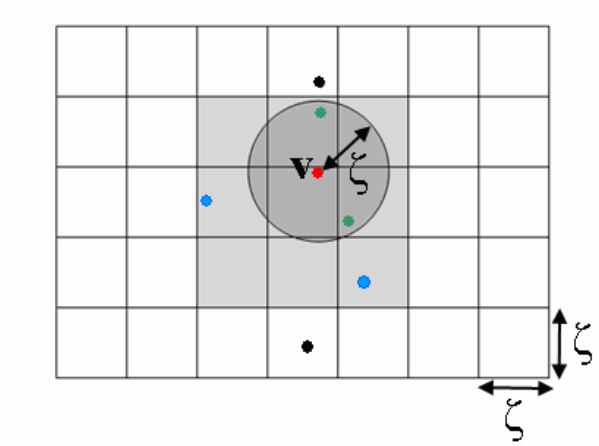

Figure 2.8: The collision zone and the search space for collision detection of a vertex. The vertex **v** is located towards an edge of its voxel. Green vertices are inside the collision zone. Blue vertices are within the search space (neighbouring voxels) whereas the black vertices are not.

Chapter 3

SURFACE TRACKING

The mesh representation $M_{k^*}^{(t)}$ of the object surface at each frame t is reconstructed by deforming the surface reconstructed at the previous frame, $M_0^{(t)} = M_{k^*}^{(t-1)}$. The deformation process is driven by the time-varying silhouette and the 3D scene flow information, as explained in detail in this chapter.

3.1 Shape From Silhouette

The surface deformation process is primarily driven by the time-varying silhouette information. Hence we need to extract these silhouettes for each frame of the multiview video prior to tracking. For this purpose we use the silhouette extraction method proposed in [31], which is based on statistical modeling of the background pixel colors with a training set of background images. We note that the performance of the surface tracking scheme is highly related to the performance of the silhouette extraction stage.

3.1.1 Silhouette Based Deformation

The displacement, $\mathbf{d}(\mathbf{v}_{i,k})$, at each vertex i of the mesh and at each iteration k of the surface evolution, can be computed based on the time-varying silhouette information. Let $\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k})$ denote the silhouette-based displacement. The direction of this displacement

(3.1)

vector is set to be in the direction of the surface normal $\mathbf{N}(\mathbf{v}_{i,k})$, inwards or outwards, depending on the positioning of the vertex with respect to the target surface.

$$\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k}) = \delta_{\text{sil}}(\mathbf{v}_{i,k})\mathbf{N}(\mathbf{v}_{i,k})$$

The magnitude of the displacement, $\delta_{\text{sil}}(\mathbf{v}_{i,k})$, is based on how far and in which direction (inside or outside) the vertex **v** is with respect to the silhouettes at that time instant. Thus the displacement scalar δ_{sil}, which may take negative values as well, is computed by projecting $\mathbf{v}_{i,k}$ onto the image planes and thereby estimating an isolevel value $f(\mathbf{v}_{i,k})$ via bilinear interpolation:

$$\delta_{\text{sil}}(\mathbf{v}_{i,k}) = \varepsilon_{\min} f(\mathbf{v}_{i,k}) = \varepsilon_{\min} \min\{G[\text{Proj}_{I_n}(\mathbf{v}_{i,k})] - 0.5\} \tag{3.2}$$

where $\text{Proj}_{I_n}(\mathbf{v}_{i,k})$ is the projection of the vertex $\mathbf{v}_{i,k}$ to I_n, the n'th binary silhouette image (0 for outside, 1 for inside) in the sequence. The function G, taking values between 0 and 1, is the bilinear interpolation of the sub-pixelic projection of the vertex $\mathbf{v}_{i,k}$. Thus, the isolevel function $f(\mathbf{v}_{i,k})$ takes on values between -0.5 and 0.5, and the zero crossing of this function reveals the isosurface. The isovalue of the vertex **v** is provided by the image of the silhouette that is farthest away from the point, or in other words, where the interpolation function G assumes its minimum value. So the absolute displacement scalar is bounded above by $\varepsilon_{\min}/2$, that is, the maximum displacement that a vertex is allowed to move in one iteration.

We distinguish the vertices of the deformable mesh under three categories with respect to their isovalues. A vertex **v** is labeled as IN if $f(\mathbf{v}_{i,k})$ is 0.5, OUT if -0.5 and ON if in between. According to this definition, ON vertices are those positioned within a narrow band around the boundary surface. By Equation 3.2, the displacement at each ON vertex varies within the interval $(-\varepsilon_{\min}/2, \varepsilon_{\min}/2)$. The vertices which are out of this band are

labeled as IN or OUT, depending on whether they are located inside or outside the silhouettes and they have displacement scalars $\varepsilon_{min}/2$ or $-\varepsilon_{min}/2$

3.1.2 Fine Tuning

During surface evolution, the state of a vertex, which is initially OUT, can switch between any two of the three categories. A vertex moves not only due to the displacement operator, but also due to the regularization effect of the smoothing operator that alters its positioning. Depending on the magnitude of the displacement vector, which is bounded above by $\varepsilon_{min}/2$, the state of a vertex can even switch from OUT to IN, or vice versa, at one single iteration. The vertices of the deformable mesh, when they get close to the boundary, may oscillate between IN and OUT states until convergence, that is, until they no longer move. Some vertices remain as OUT or IN even at convergence. To improve accuracy and to speed up convergence, we incorporate a fine-tuning procedure to the surface deformation process. We detect the instances when a vertex crosses the target boundary due to the effect of the displacement operator, that is, when its state changes from outside to inside, or vice versa. We then precisely locate the point where it crosses the boundary via dichotomic subdivision as described below. If the current resolution of deformation ($\varepsilon_{min}/2$) does not match the resolution of the silhouette images, it is even possible that an IN or OUT vertex may cross the boundary several times at one single iteration, jumping over and missing fine shape details. To prevent this, before moving an IN or OUT vertex $\mathbf{v}$ with the displacement operator, we sample its motion trajectory, which is of length $\varepsilon_{min}/2$, at the maximum available resolution. If a zero-crossing is detected at any of these sampled points, say $\mathbf{v}'$, the displacement of the vertex is refrained at that point and a dichotomic subdivision is carried out to search for the point where the isolevel function $f(\mathbf{v})$ is zero on the line segment joining the point $\mathbf{v}'$ and the initial position $\mathbf{v}$. For a sufficiently small threshold

value μ, $-\xi < f(\mathbf{v}) < \xi$, is used to determine how close the isolevel of a given point must be to assume it as being exactly on the boundary surface (see Figure 3.1). The vertex is then moved onto this location.

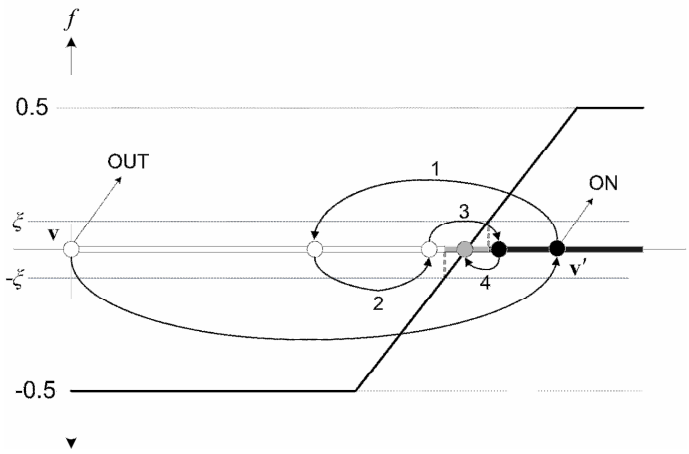

Figure 3.1: Dichotomic subdivision to accurately locate the position of a vertex. The vertex **v** is initially OUT. The boundary surface is assumed to pass through some point on the line segment joining **v** and **v'**, where the isolevel function $f(\mathbf{.})$ takes a value between $-\xi < f(\mathbf{v}) < \xi$. For this example, the binary search takes four steps to locate the boundary point.

Note that the a vertex may change its state (from OUT to IN, or vice versa), even if it is not close to the optimum coordinates where the vertex should be at the convergence of the evolution. This would occur if the vertex is passing some boundary other than the boundary at which it should ideally end to represent the target surface, that is, if its distance to the target surface point, or its 3D scene flow vector (as will be explained in Section 3.2), has a large magnitude. Thus to avoid fine tuning to be an obstacle for a smooth and fast evolution, we only activate fine tuning for a vertex when it is close to the target surface point.

3.2 3D Scene Flow

Silhouette-based surface evolution may fail to track the surface in case the local motion is too fast on small shape details that can be confused by the silhouette information. For the algorithm to work successfully, there is generally a compromise between the frame rate, the speed of the motion and the size of the shape details. To overcome the limitations of this compromise and to enhance the robustness and efficiency of the surface tracking process, we incorporate 3D Scene Flow (3DSF) information into silhouette-based deformation. Scene flow is defined by Vedula et al. in [28] as "Just as optical flow is the two-dimensional motion of points in an image, scene flow is the three-dimensional motion of points in the world". The various benefits of incorporating scene flow information is listed below:

• It eliminates unnecessary surface shrinkages and inflations during surface evolution by guiding the surface on a direct path towards the target surface. By this way, it reduces the number of necessary restructuring operations and thereby decreases the representation load and the reconstruction time.

• Pose registration parameters estimated from scene flow vectors, as will be described below, transforms the initial mesh, in a rigid manner (translates and rotates), towards the target surface so that the overall distance to the target surface is reduced. By this way, the maximum distance traveled by a vertex is shortened and hence the representation load and the reconstruction time are further decreased.

• It provides a smooth evolution by integrating a displacement direction other than the surface normal towards the target surface and avoids unnecessary decelerations around irrelevant surface boundaries due to small scale silhouette details and unnecessary fine tunings.

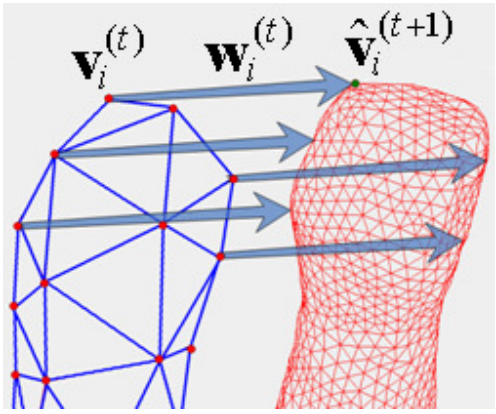

Figure 3.2: Illustration of 3D scene flow on an initial mesh (left) and its target surface

Obtaining the scene flow information necessary for our deformation scheme means to estimate a vector $\mathbf{w}_i^{(t)}$ for each vertex $\mathbf{v}_i^{(t)}$ of the mesh representation $M^{(t)}$ at frame t, that describes its position at the next frame t+1:

$$\hat{\mathbf{v}}_i^{(t+1)} = \mathbf{v}_i^{(t)} + \mathbf{w}_i^{(t)} \tag{3.3}$$

as depicted with an example in Figure 3.2. To achieve this estimation we have developed the method described in the sequel.

3.2.1 3DSF Estimation

The steps towards estimating the scene flow vector $\mathbf{w}_i^{(t)}$ for a vertex $\mathbf{v}_i^{(t)}$ of the mesh representation $M^{(t)}$ are as follows [28] (dropping the indices *i* and *t*):

i) Find the set of camera viewpoints that the vertex $\mathbf{v}$ is visible to, Vis($\mathbf{v}$)={C_1, C_2, …, C_K}, where *K* is the number of cameras that $\mathbf{v}$ is visible to. For this purpose we make use of the voxel grid generated for collision handling. We scan the voxels located along the line of sight from $\mathbf{v}$ to the optical center of a given camera C_k and check if any other vertex occludes its visibility from that camera.

ii) By using the camera projection matrix $\mathbf{P}_k$, project the vertex $\mathbf{v}$ = (*x*,*y*,*z*) onto each camera plane C_k in Vis($\mathbf{v}$):

$$u_k = \frac{[\mathbf{P}_k]_1 (x, y, z, 1)^{\mathrm{T}}}{[\mathbf{P}_k]_3 (x, y, z, 1)^{\mathrm{T}}}, \qquad v_k = \frac{[\mathbf{P}_k]_1 (x, y, z, 1)^{\mathrm{T}}}{[\mathbf{P}_k]_3 (x, y, z, 1)^{\mathrm{T}}}$$

where $\mathbf{u}_k = (u_k, v_k)$ is the projected point, and $[\mathbf{P}_k]_1$ and $[\mathbf{P}_k]_3$ denote the first and the third rows of the 3×4 projection matrix, respectively.

iii) Find the 2D optical flow vector, $\frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}t}$, at the projected point for each camera C_k. We employ the hierarchical Lucas-Kanade method [29] for this purpose.

iv) Estimate the 3D scene flow vector, $\mathbf{w} = \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}$, from the computed 2D motion vectors, $\left\{ \frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}t} \right\}_{k=1}^{K}$, by $\frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}t} = \frac{\partial \mathbf{u}_k}{\partial \mathbf{v}} \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}$. The Jacobian matrix $\frac{\partial \mathbf{u}_k}{\partial \mathbf{v}}$ can be computed explicitly for each *k* by symbolic differentiation of $\mathbf{u}_k$ with respect to *x, y,* and *z*, using the camera projection parameters. The 3D scene flow vector, $\mathbf{w} = \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}$, can then be estimated from the following overdetermined set of linear equations:

$$\begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} & \frac{\partial u_1}{\partial z} \\ \frac{\partial v_1}{\partial x} & \frac{\partial v_1}{\partial y} & \frac{\partial v_1}{\partial z} \\ . & . & . \\ . & . & . \\ \frac{\partial u_K}{\partial x} & \frac{\partial u_K}{\partial y} & \frac{\partial u_K}{\partial z} \\ \frac{\partial v_K}{\partial x} & \frac{\partial v_K}{\partial y} & \frac{\partial v_K}{\partial z} \end{bmatrix} \mathbf{w} = \begin{bmatrix} \frac{\partial u_1}{\partial t} \\ \frac{\partial v_1}{\partial t} \\ . \\ . \\ \frac{\partial u_K}{\partial t} \\ \frac{\partial v_K}{\partial t} \end{bmatrix} \tag{3.4}$$

The scene flow vector of a vertex **v** can be estimated from its 2D motion vectors only if the vertex is visible from at least 2 cameras, i. e., $K \geq 2$.. When the number of visible cameras is sufficient, the set of linear equations is solved via the least-squares method for **w** so as to minimize the sum of the errors obtained by reprojecting the scene flow vector onto the camera planes.

The estimated scene flow vectors theoretically represent the coordinates of each vertex at the mesh representation of the following frame. But this data is usually not complete and noisy, therefore it must be regularized before it can be utilized. We smoothen the estimated scene flow vectors by averaging each in their 3-link neighbourhood and extrapolate the missing ones during this stage, again by averaging in their neighbourhood. We employ the scene flow vectors for two purposes. Firstly for pose registration to apply once at each frame transition, secondly to assist the silhouette-based deformation through displacement iterations to render the process more robust and efficient.

3.2.2 Pose Registration

The purpose of pose registration is to adjust the position and orientation of the mesh representation $M^{(t)}$ so that the distance between $M^{(t)}$ and the target surface $S^{(t+1)}$ is reduced before surface evolution from frame t to t+1 gets started. This is depicted with an example in Figure 3.3. This initial transformation does not only improve the chances of the surface evolution to successfully converge to the desired surface, but it also speeds up the deformation process and reduces the maximum distance traveled by a vertex thereby decreasing the representation load.

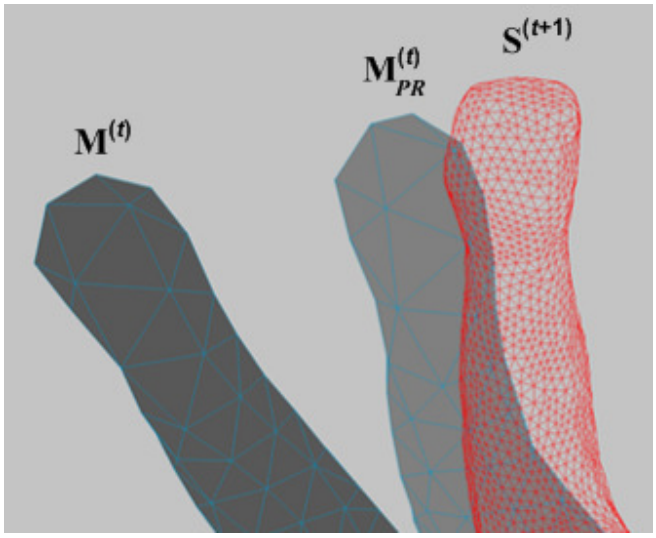

Figure 3.3: Illustration of pose registration. $\mathbf{M}^{(t)}$: Mesh representation at frame t, $\mathbf{M}^{(t)}_{PR}$: Pose registered mesh, $S^{(t+1)}$: Target surface at frame t+1.

The 3D scene flow vectors can be used to estimate the pose registration (rotation and translation) parameters. Let us denote the global rigid body motion parameters between the surfaces at frame t and t+1 by $\mathbf{R}^{(t)}$ the rotation matrix and $\mathbf{t}^{(t)}$ the translation vector. We represent the 3D coordinates of the vertices at these consecutive frames as $\mathbf{V}^{(t)}$ and $\hat{\mathbf{V}}^{(t+1)}$,

$$\mathbf{V}^{(t)} = \left[\mathbf{v}_1^{(t)}, \mathbf{v}_2^{(t)}, ..., \mathbf{v}_n^{(t)} \right] \qquad \hat{\mathbf{V}}^{(t+1)} = \left[\hat{\mathbf{v}}_1^{(t+1)}, \hat{\mathbf{v}}_2^{(t+1)}, ..., \hat{\mathbf{v}}_n^{(t+1)} \right] \tag{3.5}$$

where n is the number of vertices in $M^{(t)}$. Recall also that $\hat{\mathbf{v}}_i^{(t+1)} = \mathbf{v}_i^{(t)} + \mathbf{w}_i^{(t)}$ where $\mathbf{v}_i^{(t)}$ and $\mathbf{w}_i^{(t)}$ are column vectors consisting of 3D coordinates. In this case, the relationship between $\mathbf{V}^{(t)}$ and $\hat{\mathbf{V}}^{(t+1)}$ is given by:

$$\hat{\mathbf{V}}^{(t+1)} = \left[\mathbf{R}^{(t)} \ \mathbf{t}^{(t)} \right] \begin{bmatrix} \mathbf{V}^{(t)} \\ \mathbf{1}^{\mathrm{T}} \end{bmatrix} \tag{3.6}$$

$\mathbf{R}^{(t)}$ and $\mathbf{t}^{(t)}$ are estimated from this equation using a nonlinear unitary-constraint optimization technique as described in [30]. The rotation and translation parameters are estimated prior to each frame transition to apply pose registration. This process of pose registration provides a good initialization of the surface at each frame and hence improves stability of the surface tracking process. The benefits of pose registration will be demonstrated with experiments in Chapter 4.

3.2.3 Scene Flow Assisted Deformation

In addition to pose registration, 3D scene flow vectors are also used at the deformation stage to contribute to the computation of the displacement operator. It is possible to integrate scene flow vectors into the deformation stage in several different ways. Although in theory a scene flow vector gives the exact location of a vertex on the surface of the next frame, scene flow computation itself is usually a noisy and unstable process in practice and hence cannot alone be relied upon for a robust mesh evolution. Nevertheless, 3D scene flow can assist silhouettes in driving the deformation process by ensuring that majority of the vertices are correctly led towards the target surface. That is, the direction and magnitude of the displacement vector of a vertex at a given iteration will depend partially

on the silhouettes and partially on the scene flow vectors. The best way to achieve this in our deformation framework is to use a linear combination of these two information sources while calculating the displacement vector defined by Equation 2.4:

$$\mathbf{d}(\mathbf{v}_{i,k}) = \alpha_{i,k}\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k}) + \beta_{i,k}\mathbf{d}_{\text{flow}}(\mathbf{v}_{i,k}) \tag{3.7}$$

where $\alpha_{i,k}$ and $\beta_{i,k}$ are weighting coefficients varying for every vertex i and iteration k; they take values between [0,1] and their sum is always unity: $\alpha_{i,k} + \beta_{i,k} = 1$. The silhouette-based component $\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k})$ is calculated according to Equations 3.1 and 3.2,

$$\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k}) = \varepsilon_{\min} \min\{G[\text{Proj}_{I_n}(\mathbf{v}_{i,k})] - 0.5\}\mathbf{N}(\mathbf{v}_{i,k}) \tag{3.8}$$

where $\mathbf{N}(\mathbf{v}_{i,k})$ represents the unit surface normal at vertex $\mathbf{v}_{i,k}$. The scene flow based component $\mathbf{d}_{\text{flow}}(\mathbf{v}_{i,k})$ is calculated based on $\mathbf{w}_i$ such that its direction is always towards the initial target $\hat{\mathbf{v}}_i$ that the scene flow vector points to and its magnitude is bounded by $\varepsilon_{\min}/2$ just like the silhouette-based displacement component $\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k})$:

$$\mathbf{d}_{\text{flow}}(\mathbf{v}_{i,k}) = \begin{cases} \dfrac{\varepsilon_{\min}}{2} \dfrac{\hat{\mathbf{v}}_i - \mathbf{v}_{i,k}}{\left\| \hat{\mathbf{v}}_i - \mathbf{v}_{i,k} \right\|} & \text{if } \left\| \hat{\mathbf{v}}_i - \mathbf{v}_{i,k} \right\| > \dfrac{\varepsilon_{\min}}{2} \\ (\hat{\mathbf{v}}_i - \mathbf{v}_{i,k}) & \text{otherwise} \end{cases} \tag{3.9}$$

Recall from Equation 3.3 that $\hat{\mathbf{v}}_i$, representing the target point, is different for each vertex and remains fixed throughout the iterations of a frame transition from t to t+1.

The weights $\alpha_{i,k}$ and $\beta_{i,k}$ in Equation 3.7 vary with iteration counter k. The purpose here is to provide a deformation where the scene flow vectors dominate the silhouette information at the early iterations ($\alpha_{i,0} = 1, \beta_{i,0} = 0$). As iterations proceed, this favour should gradually be carried to silhouette information ($\alpha_{i,k^*} \cong 0, \beta_{i,k^*} \cong 1$). In this way, the scene flow information will smoothly lead the deformation towards the target surface on a

stable and short path while the final surface reconstructed will totally be determined by the silhouette information. We set these weights as a function of iteration counter *k* such that:

$$\alpha_{i,k} = e^{-\tau_i k}, \qquad \beta_{i,k} = 1 - \alpha_{i,k} \tag{3.10}$$

The choice of the time coefficient τ_i here is crucial because it determines how long the scene flow information will be effective. The optimal value of τ_i is determined separately for each vertex based on the scene flow magnitude at that vertex. Our strategy is as follows: Taking into account the fact that the maximum vertex displacement at one iteration is bounded above by $\varepsilon_{\min}/2$, the total number of iterations for a vertex $\mathbf{v}_i$ to reach its target point $\hat{\mathbf{v}}_i$ by using only the scene flow vector is expected to be $\hat{K} = 2 \| \mathbf{w}_i \| / \varepsilon_{\min}$ (see Eq. 3.9). The coefficient τ_i is then chosen so that the weights satisfy $\alpha_{i,k} = \beta_{i,k} = 0.5$ at the half of this iteration count:

$$\tau_i = -\ln(0.5) \cdot (\|\mathbf{w}_i\| / \varepsilon_{\min})^{-1} \tag{3.11}$$

To summarize, the displacement $\mathbf{d}(\mathbf{v}_{i,k})$ at the vertex $\mathbf{v}_{i,k}$ is calculated in a similar way as given in Equation 3.2, but this time using also the scene flow information via $\mathbf{d}_{\text{flow}}(\mathbf{v}_{i,k})$. When using scene flow vectors, the fine tuning procedure described in Chapter 3.1.2 is invoked only when the vertex is close to its target point $\hat{\mathbf{v}}_i$. In this way, unnecessary decelerations around irrelevant surface boundaries are avoided. Integrating the scene flow information into the deformation process not only improves the stability of the surface tracking but also considerably decreases the operation count and the reconstruction time by leading the surface smoothly towards the target surface without unnecessary restructuring operations during surface evolution.

3.3 Surface Tracking Algorithm

We now describe the overall algorithm that generates a sequence of meshes representing the time-varying geometry. The inputs are the multiview video, the silhouette images at every frame, the initial mesh model $M^{(0)}$ representing the surface at the first frame, and the projection parameters of the multi-camera system. The vertices of the initial deformable mesh, $M_0^{(t)}$, are all set to be *active* prior to deformation at each frame. The overall algorithm is then as follows:

Iterate on t

- Set $M_0^{(t)} = M^{(t-1)}$;
- Estimate 3D scene flow $\mathbf{w}^{(t)}$;
- Estimate the rotation matrix $\mathbf{R}^{(t)}$ and the translation vector $\mathbf{t}^{(t)}$;
- Pose register $M_0^{(t)}$ using $\mathbf{R}^{(t)}$ and $\mathbf{t}^{(t)}$;
- Iterate on k
  - Displace active vertices in $M_k^{(t)}$ by $T_d(M_k^{(t)})$;
  - Detect and handle collisions;
  - Smooth active vertices in $M_k^{(t)}$ by $T_s(M_k^{(t)})$;
  - Restructure active edges in $M_k^{(t)}$ by $T_r(M_k^{(t)})$;
  - Deactivate vertices that no longer move;
- Till convergence
- Set $M^{(t)} = M_{k*}^{(t)}$ as final mesh representation at frame t;

Till end of scene

Note that the displacement and smoothing operators are applied only to active vertices of the deformable mesh whereas the restructuring operator is invoked only for active edges,

that is, for edges with at least one active vertex. The vertices that are detected to no longer move through iterations of the deformation algorithm are deactivated. Thus as iterations proceed and as more and more vertices become inactive, the time spent at each iteration significantly reduces, yielding on overall a computationally efficient algorithm. The algorithm converges when the vertices of the deformable mesh no longer move, that is, when the equilibrium condition in Equation 2.2 is satisfied.

3.4 Representation Load

The resulting mesh sequence, $M^{(0)}$, $M^{(1)}$, … $M^{(t)}$, …, representing the time-varying geometry, can be efficiently encoded in terms of the small-scale vertex displacements and the restructuring operations along with the initial model and the pose registration parameters of each frame. We will show in the experimental results section that this encoding approach significantly improves space efficiency as compared to encoding each frame separately. To calculate the bit-load of a sequence reconstructed by our method we use the number of restructuring operations, the number of vertices, and the maximum vertex displacements in *x*, *y*, and *z* directions for each frame.

Let us first assume the vertex coordinates are encoded with *P*-bit precision. We denote the number of vertices at frame *t* by N_v^t, the number of restructuring operations at frame *t* by N_{r}^t, the ratio of the radius of the sphere bounding the surface at frame *t* to the maximum displacements in *x*, *y*, and *z* directions to frame *t*+1 by respectively s_x^t, s_y^t and s_z^t, and the total number of frames in a sequence by *T*. The bit-load *B* for a mesh sequence can then be calculated (omitting the bit-load for the initial mesh $M^{(0)}$, the pose registration parameters for each frame and the maximum displacements in each direction for each frame) as follows:

$$B = \sum_{t=1}^{T} N_v^t \left(3P - \left\lceil \log_2 s_x^t \right\rceil - \left\lceil \log_2 s_y^t \right\rceil - \left\lceil \log_2 s_z^t \right\rceil\right) + 2N_r^t \left\lceil \log_2 N_v^t \right\rceil \quad (3.12)$$

In this equation, the first term in the sum corresponds to the bit-load of the vertex displacements, and the second term corresponds to the bit-load of the restructuring operations. The bit-load of the *x*-component for each vertex displacement at frame *t* is given by $\left\lceil \log_2 s_x^t \right\rceil$ which is usually much less than the required bit precision *P*. The same argument holds for *y* and *z* directions as well, to compute the total bit-load of each vertex displacement. Assuming that the number of edges in a mesh representation is equal to the number of vertices N_v^t, and noting that a restructuring operation can be represented by an edge and an edge can be represented with two vertex indices, the bit-load of a restructuring operation is twice the bit load of a vertex $2\left\lceil \log_2 N_v^t \right\rceil$ at frame *t*.

If each mesh representation in a sequence were to be encoded separately using the classical vertex-triangle list, the bit-load B_0 of the whole sequence would be calculated as:

$$B_0 = \sum_{t=1}^{T} 3N_v^t P + 6N_v^t \left\lceil \log_2 N_v^t \right\rceil \quad (3.13)$$

In this equation the first term in the sum corresponds to the bit-load of the vertex coordinates and the second term corresponds to the bit-load of the triangles. Here we assume that a triangle is represented with three vertex indices and that the number of triangles is twice the number of vertices, and also *P*-bit precision is used for each of the *x*, *y*, and *z* coordinates.

We have compared the representation load efficiency of our method with the classical vertex-triangle list representation and observed that it provides an approximately 1-to-5 encoding efficiency without applying a statistical compression algorithm. The experimental results on encoding efficiency will be presented in the next chapter.

Chapter 4

EXPERIMENTS AND RESULTS

We have conducted experiments to demonstrate the performance of our surface tracking method on three different sequences, one synthetic and two real sequences.

The synthetic mesh sequence, *Jumping Man,* originally reconstructed from a real scene by the authors of [27], exhibits the realistic motion of the jumping act of a human actor at 30 fps with 220 frames. We have artificially created the time-varying multiview silhouette images, each of size 1280×1024, from the 3D models of this sequence, using a horizontal circular camera configuration consisting of 16 cameras modeled with perspective projection. In the synthetic case, the silhouettes, the scene flow vectors and the camera calibration parameters are all given and error-free. Hence we can assess the performance of our method in ideal conditions. Moreover since we have the ground-truth mesh sequence in hand, we can quantitatively measure the quality of the reconstructed mesh sequence with respect to the original.

We have recorded two real video sequences at 30 fps by using the multicamera system equipped with 8 cameras (1332×980) in MVGL Lab at Koç University[1]. Original images from the 8 cameras of a sample frame is given in Figure 4.1. We have calibrated the multicamera system by using the technique described in [33]. For silhouette extraction, we have used the method presented in [31], which is based on statistical modeling of the background pixel colors with a training set of background images. To improve the accuracy of the silhouette extraction process, we have employed an artificial black background.

[1] http://mvgl.ku.edu.tr/

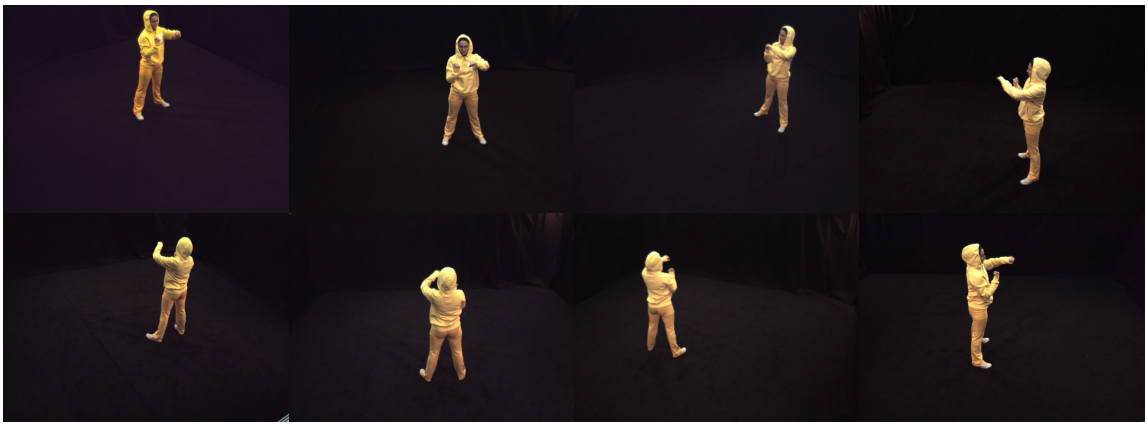

Figure 4.1: Sample original images. Views from cameras 1 to 8 at frame 1205

The first real sequence is a relatively long sequence (about 1260 frames) with various types of actions such as standing, walking, running, jumping, turning, stretching and kick-boxing. These actions have been chosen to test different types of motions varying in speed, locality and complexity (e.g., jumping is for slow local motion and fast global motion while kick-boxing is for fast local motion and slow global motion, and stretching is for self-collisions). The second real sequence has been chosen to test the performance of our deformation scheme in the case of highly non-rigid motion, e.g., total shrinking and inflation of a ball in the hands of a player while dribbling. The animation videos of the reconstructions of all three sequences together with a view from the original scenes may be reached from the public MVGL website[2].

4.1 Initial Reconstruction

Recall that the mesh representation of the initial frame, $M^{(0)}$, has to be reconstructed as the first step to be able to initiate the surface tracking process. Our reconstruction of the initial mesh is based on the silhouette-based static object reconstruction method described

[2] http://mvgl.ku.edu.tr/surftrack

in [32]. This method uses a deformation scheme that is similar to the one described in this thesis. In Figure 4.2, we provide views of the deformable model at various iterations of the surface deformation process as it evolves starting from the bounding sphere towards the object boundary at the first frame.. We decide on the resolution of the deformable mesh model, hence the value of ε_{min}, to model all sequences at the initial frame. The value of ε_{min} should be small enough to describe small shape details but as large as possible to reduce the total vertex count. In Table 4.1, we give the number of vertices and the reconstruction error for varying ε_{min}. The values of ε_{min} and the reconstruction error are both normalized with respect to the size of the object, e.g., the value of ε_{min} is given as the ratio of the minimum edge length to the radius of the bounding sphere. In Figure 4.2., we display the resulting meshes reconstructed at these varying resolutions. In Table 4.1, we detect a break point in the reconstruction error at ε_{min}= 0.025. For ε_{min} values larger than this breakpoint, the reconstruction error increases rapidly, and as also observed from Figure 4.3, the arms of the jumping man for instance start to get eroded and cannot be modeled properly. This optimal value, which corresponds 2.5% of the radius of the bounding sphere, can be applied to any sequence where the object in the scene is a human actor. Hence we will use this value for reconstruction of all other sequences, though we will also present results at higher resolutions.

ε_{min}	Vertex (#)	Average Error (10^{-3})	Maximum Error (10^{-3})
0.010	9052	2.85	99.46
0.015	3826	3.06	99.34
0.020	2203	3.47	100.24
0.025	1258	4.09	98.63
0.030	856	6.29	144.87
0.035	697	7.07	145.44

Table 4.1: Number of mesh vertices, average error, and maximum error for initial reconstruction of the *Jumping Man* with varying ε_{min} values.

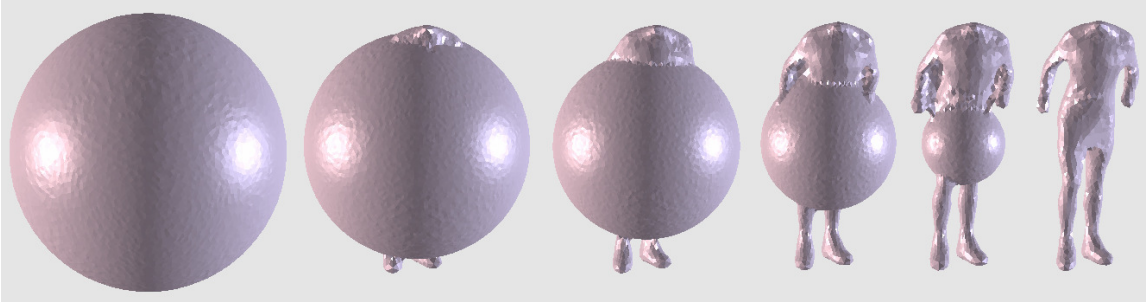

Figure 4.2: The deformable mesh at various iterations for initial reconstruction of the *Jumping Man* starting from the bounding sphere.

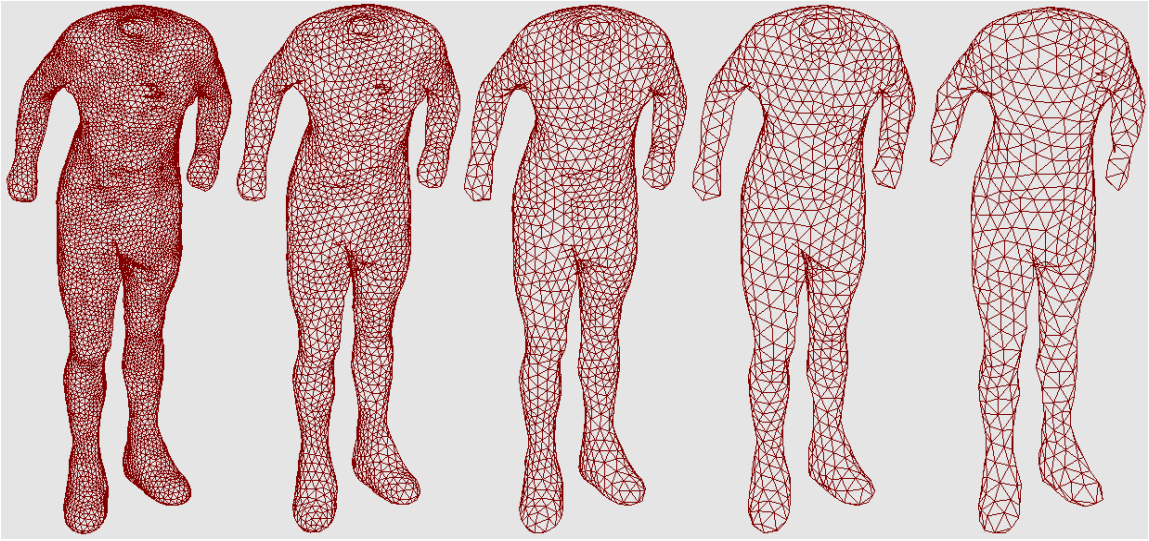

Figure 4.3: Initial meshes reconstructed at varying resolutions with $\varepsilon_{\min}$ values: (from left to right) 0.010, 0.015, 0.020, 0.025, 0.030.

4.2 Surface Tracking Results

We have tested the performance of our surface tracking method on three sequences. For each of them, we consider three distinct cases to demonstrate the contribution of each different component of our scheme. In the first case, the deformation is driven by only silhouette-based displacements without employing any pose registration or scene flow vectors. In the second case, we make use of silhouette information along with PR (pose

registration). In the third case, we consider the complete scheme that we have described in the previous chapter, i.e., we integrate the scene flow vectors to the deformation scheme to assist silhouette information and also for pose registration. We will refer these three distinct cases, respectively, as 1) Silhouette-only, 2) Silhouette-PR, and 3) Silhouette-PR-3DSF. In the sequel we present the results that we have obtained.

4.2.1 Sequence 1: Synthetic scene with ground truth data

In Figure 4.4, we display sample frames from the reconstructed mesh sequence side by side with the corresponding meshes from the original sequence. In this case, the mesh sequence has been reconstructed by employing the complete surface tracking scheme, that is, Silhouette-PR-3DSF. Although some discrepancies can be observed on the reconstructed mesh as compared to the original geometry, which are mainly due to the well known limitations of shape-from-silhouette approach, the geometry is recovered as smoothly and as faithfully to the original as possible.

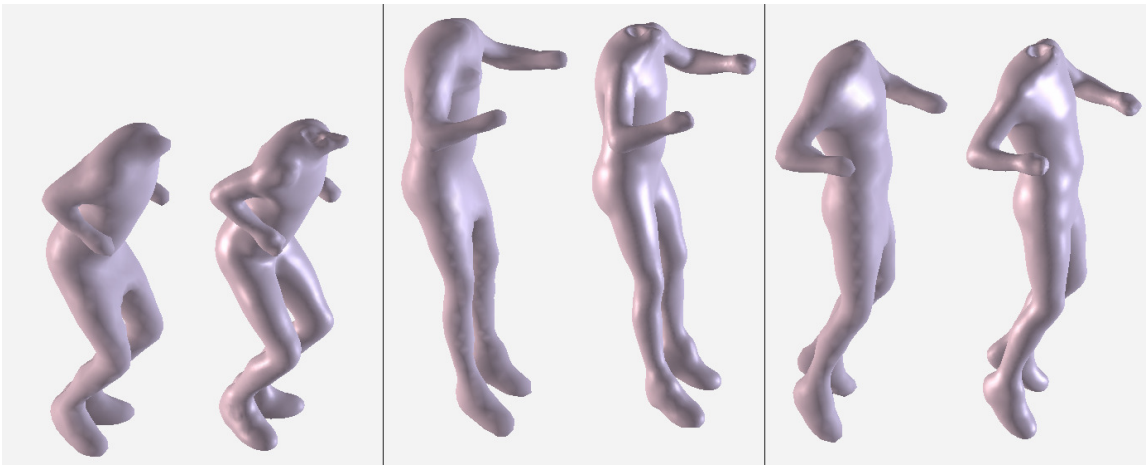

Figure 4.4: Sample reconstructions from *Jumping Man* sequence, along with the original models. From left to right, frames 19, 40 and 73.

In Table 4.2, we provide statistics per frame to quantitatively assess the performance of our method in three cases. In this table we give the average values per frame for the number of restructuring operations, reconstruction time, maximum vertex displacements and reconstruction error. Recall that the maximum vertex displacement of the deformable mesh within a given frame determines how much bit-load would be necessary to encode vertex displacements. Likewise, the number of restructuring operations also contributes to the representation load of each frame. It is observed from Table 4.2 that adding more and more components to the base scheme improves the performance of the surface tracking scheme by decreasing both reconstruction time and representation load. Also note that the average reconstruction error decreases only slightly as expected since inserting additional components into the deformation scheme rather aims to improve stability and efficiency but not the reconstruction quality. We would also like to note that the surface tracking process has failed twice over the whole sequence when Silhouette-only scheme is in use (case 1), whereas we have successfully tracked the whole time-varying geometry with additional features incorporated into the scheme, i.e., for both Silhouette-only and Silhouette-PR cases.

	Silhouette-only	**Silhouette-PR**	**Silhouette-PR-3DSF**
Total restructuring (#)	204.12	58.42	18.58
Split (#)	42.31	13.02	4.30
Collapse (#)	52.91	15.98	4.01
Flip (#)	108.90	29.42	10.26
Reconstruction time (sec)	14.16	7.38	5.93
Iterations (#)	26.60	22.21	19.13
Max. displacement (10^{-3})	132.88	78.25	62.75
Reconstruction error (10^{-3})	4.41	4.37	4.35

Table 4.2: Average statistics per frame for the *Jumping Man* sequence in three different cases.

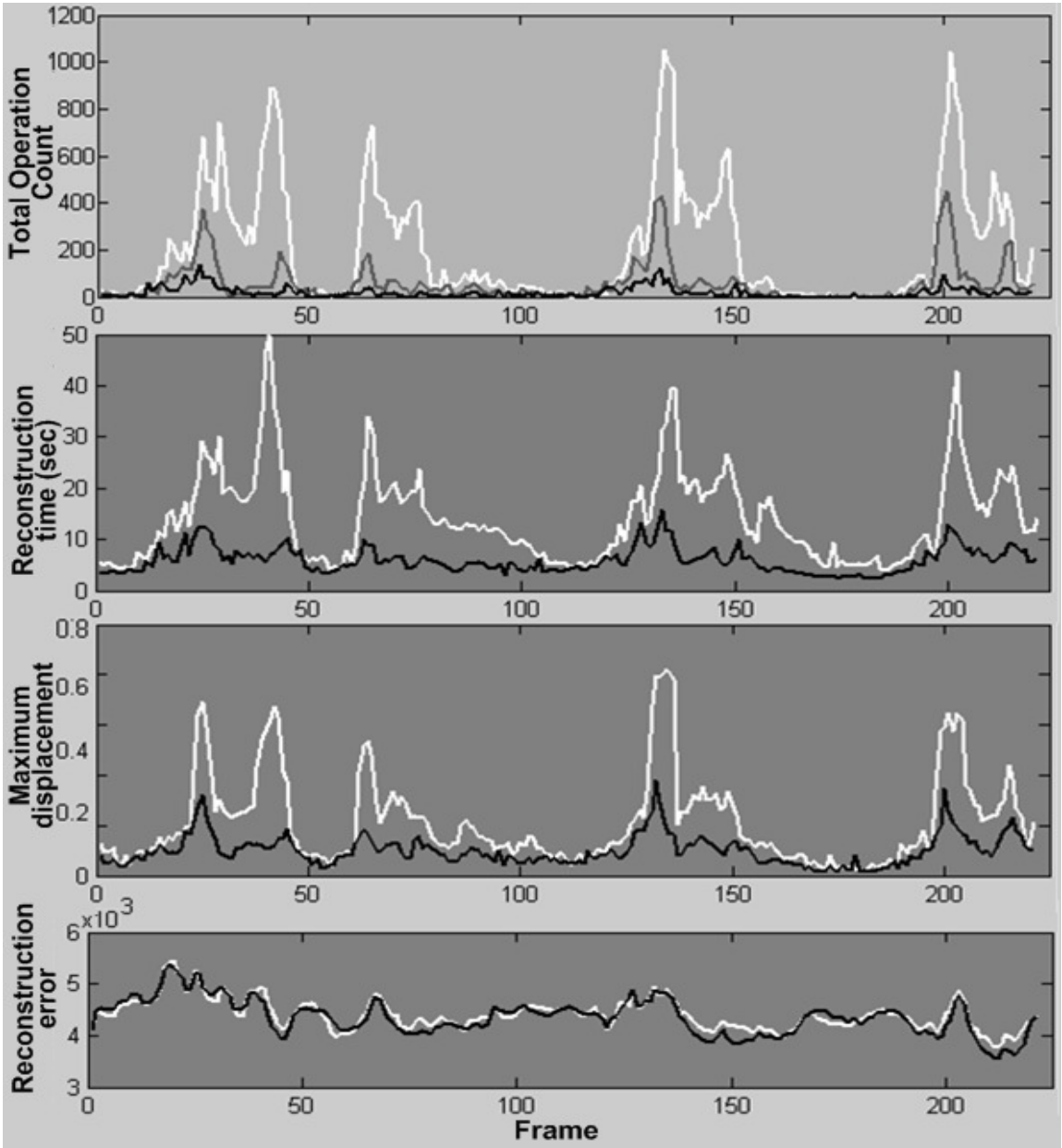

Figure 4.5: Total number of restructuring operations, reconstruction time, maximum vertex displacement and reconstruction error for each frame of the *Jumping Man* sequence in three cases: Silhouette-only (white), Silhouette-PR-3DSF (black), and only on the first plot, Silhouette-PR (gray).

In Figure 4.5, we plot per frame the number of restructuring operations, reconstruction time, reconstruction error and maximum vertex displacement. The four global peaks observed for the operation count, reconstruction time and maximum displacement plots correspond to the four jumping acts (respectively, forward and backward then again forward and backward) in the sequence. The local maxima within these peaks correspond to the fast motion of the arms during these jumps. The increases in operation count, which imply more changes in the mesh connectivity, are as expected where the motion of the object is faster. The increases in reconstruction time and maximum vertex displacements are correlated with the operation count, as also expected.

In Figure 4.6 we display the deformable mesh at various iterations within a frame transition for three different cases. We also zoom on a specific region, i.e., on one of the arms, to emphasize the differences between the cases. It is observed that pose registering the initial mesh so as to start from a closer location to the target surface provides some segments (e.g., arms) of the object to fall inside the target surface at the beginning; therefore eases the evolution process avoiding total shrinking and re-modeling of segments, and eventually reduces the number of mesh restructuring operations. However pose registration only partly avoids shrinking and re-modeling. When the 3D scene flow vectors are also incorporated, the deformable mesh evolves in a much smoother path towards the target surface without almost no shrinkage, as observed in Figure 4.6.

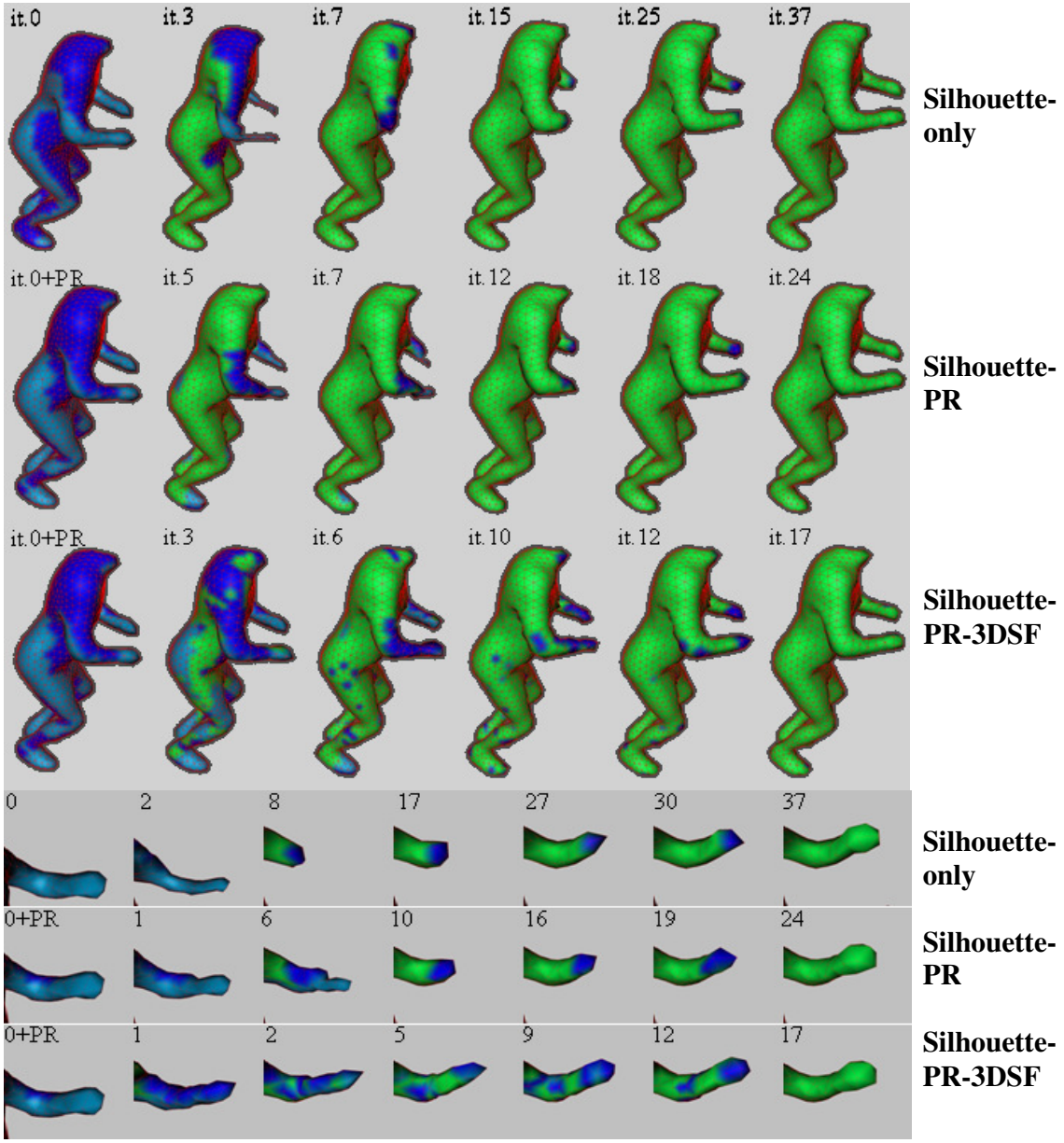

Figure 4.6: Deformable mesh at various iterations for transition from frame 131 to 132 in three different cases. (Below) Zoom on the left arm. Dark blue areas correspond to IN, light blue areas correspond to OUT, and green areas correspond to ON vertices. Total number of restructuring operations in three cases are respectively 760, 510 and 294.

4.2.2 Sequence 2: Real scene with various types of action

The first real sequence is a relatively long one, containing about 1260 frames, with various types of actions such as standing, walking, running, jumping, turning, stretching and kick-boxing. The manner how the motion pattern and intensity differ globally and locally at varying parts of the body for varying actions makes this long sequence a challenging experiment to test the performance of our deformation scheme. In Figure 4.7, we display sample frames from the reconstructed mesh sequence together with sample silhouette images from that frame.

In Table 4.3, we provide statistics per frame to quantitatively assess the performance of our method in three different cases. In this table we give the average values per frame for the number of restructuring operations, reconstruction time and maximum vertex displacements. The given reconstruction times do not include the time spent for scene flow estimation which takes about 8 sec per frame for this sequence at this resolution. Note also that we cannot establish a reconstruction error as we could in the synthetic case, since this time a ground-truth mesh sequence is not available. Similarly to the experiments on the first sequence, the surface tracking process has failed, a couple of times in thousands frames, when only the silhouette based deformation is used (case 1), though we have been able to track the time-varying geometry successfully when the additional system components, i.e., pose registration and scene flow vectors, are incorporated. We observe from the table again that adding more and more components to the base scheme improves the performance of the surface tracking scheme by decreasing both reconstruction time and representation load (through maximum displacement). Finally note that the 3D scene flow vectors were much more beneficial in the synthetic case when they were read from the ground-truth and clean but not estimated and noisy.

Figure 4.7: Samples from the reconstructed mesh sequence, one for each type of action, together with corresponding silhouette images.

	Silhouette-only	**Silhouette-PR**	**Silhouette-PR-3DSF**
Total restructuring (#)	216.24	119.14	97.92
Split (#)	43.23	23.44	18.98
Collapse (#)	52.13	28.30	21.71
Flip (#)	120.88	67.40	57.23
Reconstruction time (sec)	11.95	8.01	7.30
Iterations (#)	26.35	23.59	25.33
Max. displacement (10^{-3})	133.33	102.47	97.53

Table 4.3: Average statistics per frame for the reconstruction of the second (real) sequence in three cases.

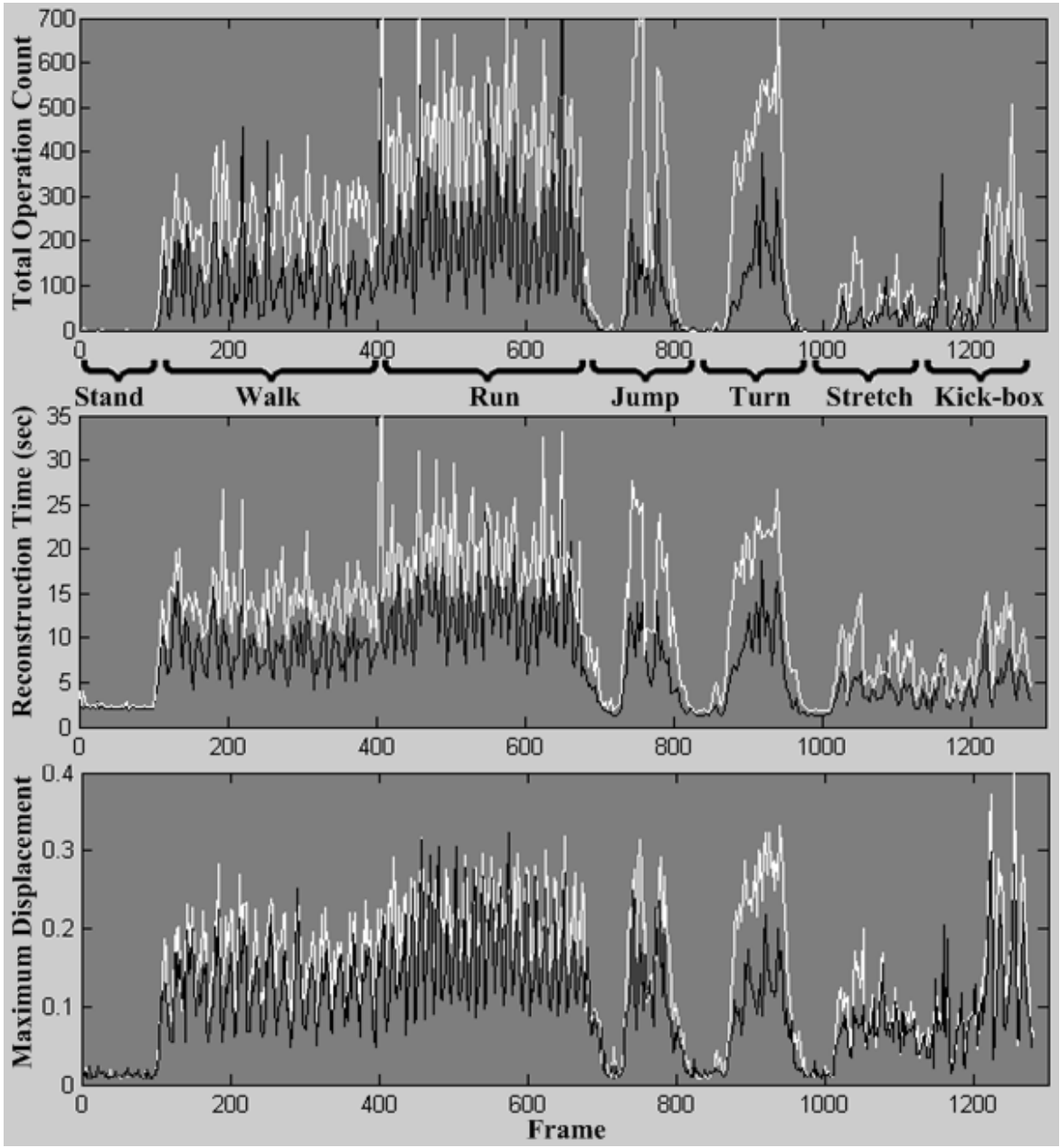

Figure 4.8: Total number of restructuring operations, reconstruction time and maximum vertex displacement for each frame of the second sequence in two different cases: Silhouette-only (white) and Silhouette-PR-3DSF (black)

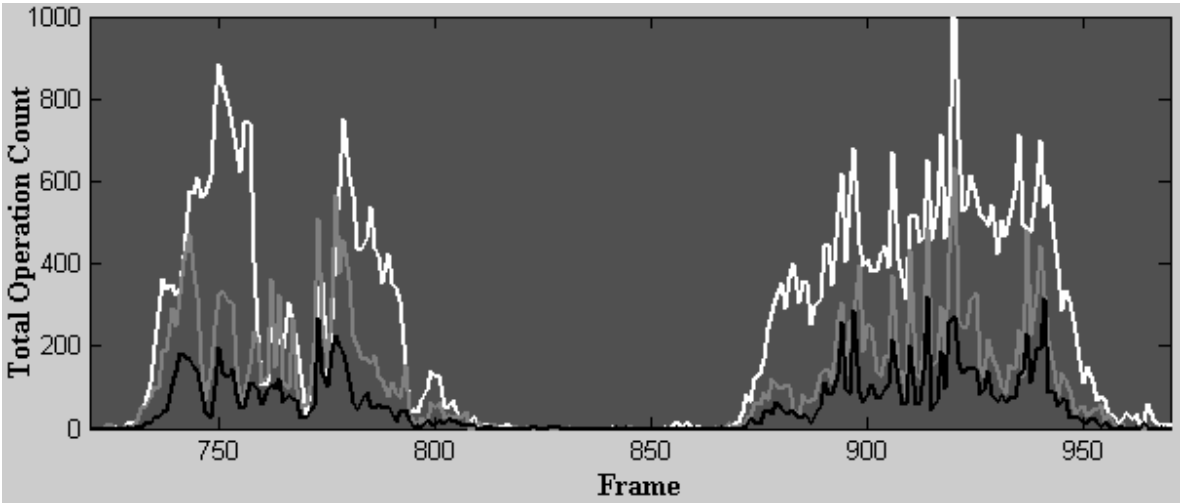

Figure 4.9: Total number of restructuring operations for each frame, when zoomed on jumping and turning (frames 720 to 970) of the second sequence in three cases: Silhouette-only (white), Silhouette-PR-3DSF (black), and Silhouette-PR (gray).

Figure 4.8 plots the total number of restructuring operations, reconstruction time, and maximum vertex displacement for each of the 1280 frames of the second sequence in two cases, Silhouette-only and Silhouette-PR-3DSF. In Figure 4.9, the restructuring operation plot is zoomed on the "jumping" and "turning" frames of the sequence. We observe that there is a strong correlation between the type of motion and the number of restructuring operations (due to change in connectivity), the time cost of the reconstruction (due to fast motion and/or high restructuring), and the maximum vertex displacement (due to fast local motion).

Standing: The initial frames of the sequence exhibit almost no motion, so the mesh connectivity is preserved and the restructuring operation count remains almost zero throughout these frames for all three cases.

Walking & Running: The oscillations observed in “walking” and “running” frames in all of the plots are due to two reasons: i) fast motion of the body parts while taking a step (rise) and the relatively slower motion while having the feet on the ground (fall), ii) mesh restructuring -shrinkage and inflation- between the legs when they are close and crossing each other and no such restructuring while apart. Here shrinkage and inflation occur mostly at the cavities between the legs, which suffer from severe self-occlusions. Note also that the frequency and the intensity of the oscillations at “Running” are almost twice the oscillations at walking as expected..

Jumping & Turning: The global motion is the strongest in these two actions when compared to other types of action available in the sequence. “Jumping” is the action with the strongest global translation while “Turning” is the action with the strongest global rotation. Due to these reasons, the benefit of pose registration is observed to be the highest for these two actions in Figure 4.9. The two peaks observed on the plots for the “Jumping” action correspond to forward and backward jumps, respectively. The frames in between these two have relatively slower motion and hence the benefit of pose registration at these frames decreases as observed. A similar decrease in benefit of pose registration for the “Turning” action occur while the legs are close and crossing each other where the major share of the reconstruction time and edge operation observed is due to mesh restructuring operations, similarly to “Walking” and “Running” actions.

Stretching & Kick-boxing: The “Kick-boxing” action has almost no global motion since the subject is standing at the same place. However the local motion is the strongest in this action with the two punches and two kicks (total of four moves). Therefore the benefit of scene flow assistance is observed to be the highest in these four moves. These moves and the benefit of scene flow assistance at these moves are observed at the peaks of the plots,

especially at the two highest peaks of maximum vertex displacement. The “Stretching” action is similar to “Kick-boxing” action as the model stands at the same place. The peculiarity of the “Stretching” action is that it contains intentional self-collisions of the arms with the body. Since the initial topology is preserved in these self-collisions and since our collision-detection algorithm handles these collisions properly, these frames exhibit no different behavior than the others in terms of restructuring operations as observed from the plots.

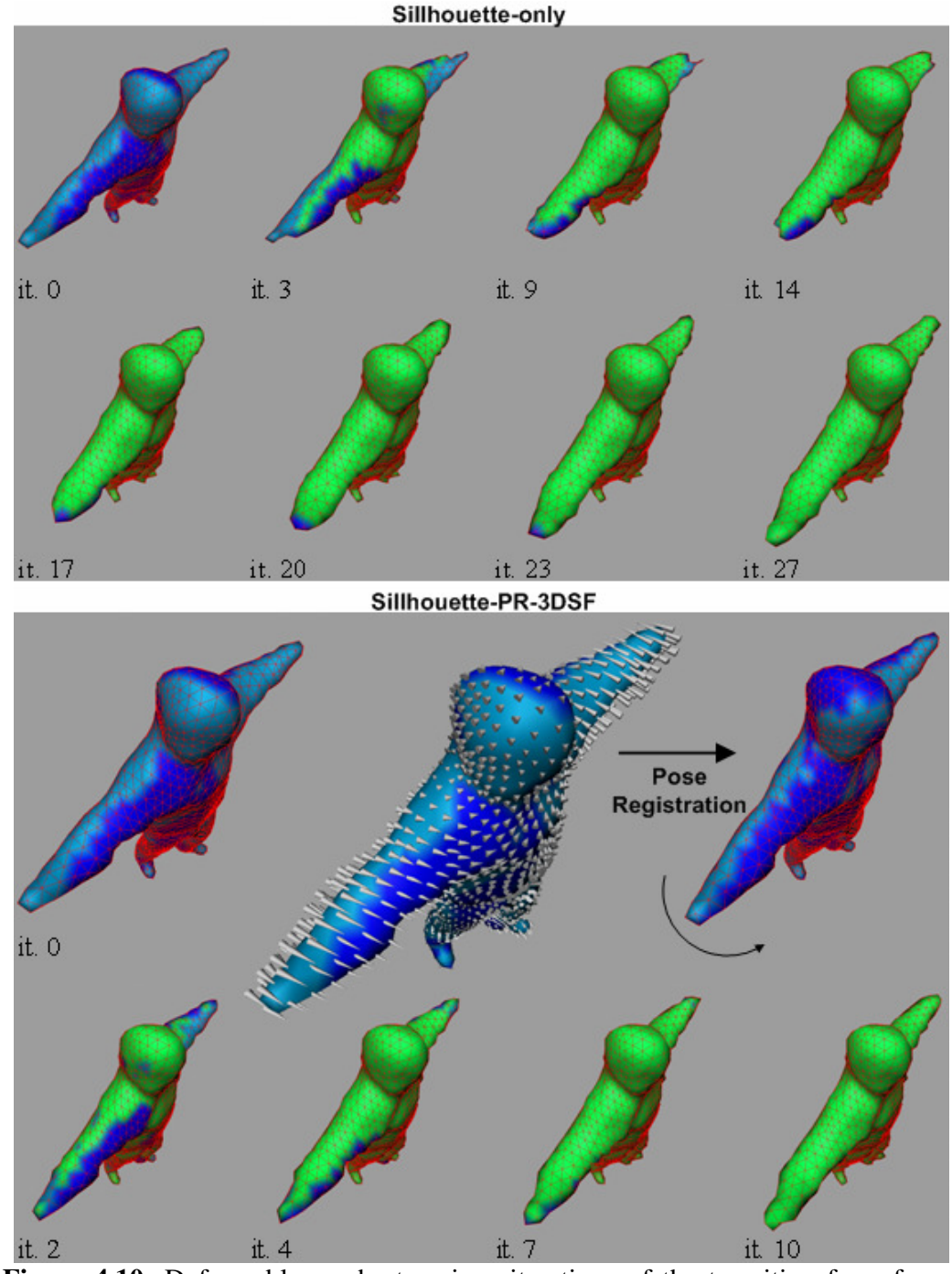

Figure 4.10: Deformable mesh at various iterations of the transition from frame t=878 to frame t=879 in two different cases: Silhouette-only and Silhouette-PR-3DSF. Dark blue areas correspond to IN, light blue areas correspond to OUT and green areas correspond to ON vertices.

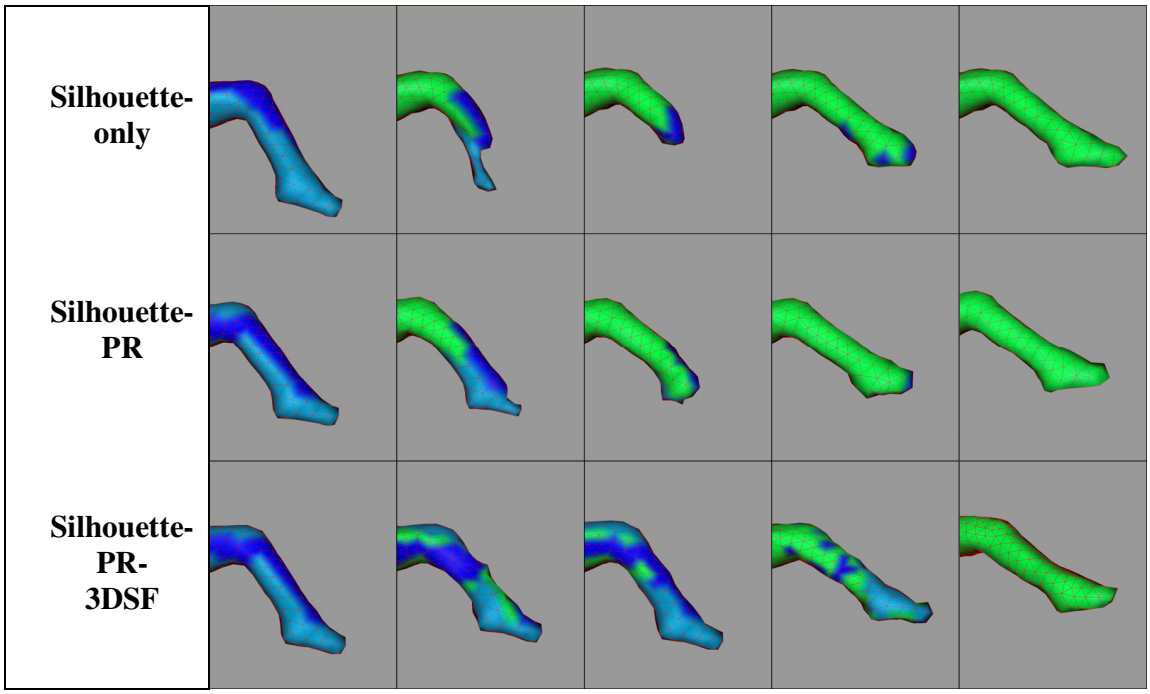

Figure 4.11: Zoom on the right leg of the deformable mesh at various iterations for transition from frame t=1255 to frame t=1256 for three different cases. Dark blue areas correspond to IN, light blue areas correspond to OUT and green areas correspond to ON vertices. The evolution converges at iterations 49, 35 and 25 respectively for the three cases.

In Figure 4.10 we display the deformable mesh at various iterations within a frame transition for two different cases. In Figure 4.11, we zoom on the right leg to better visualize the differences between the cases. As observed in Figures 4.10 and 4.11, the pose registration carries the initial surface to a position so that the mesh evolution does not need to re-model any surface segment from scratch. Although some small number of vertices may even get further away from their target destination after pose registration, the mesh evolution process is improved for majority of the vertices and on overall pose registration improves the algorithm efficiency. When the local motion at some small shape segment is too fast such that the pose registration parameters -estimated from the global motion of the

whole shape- are not sufficient to carry the position of the segment close enough to its target location, the 3D scene flow assistance serves to complete this task at the early iterations of the surface evolution by smoothly guiding that shape segment towards its target.

Figure 4.12: Sample frames with computed 2D optical flows. (Left) View from the 8th camera at frame t = 748 and (right) from the 2nd camera at frame t = 1040.

In Figure 4.12, we visualize the 2D optical flow vectors that we have computed on two sample frames t with respect to t+1 by using the hierarchical Lucas-Kanade technique. Note that 2D optical flow is estimated only for those pixels that are projections of the vertices of the mesh representation at that frame. We also eliminate the erroneous optical flows of the vertex projections near the silhouette boundaries, that cannot be estimated in a

robust manner due to covered and uncovered regions of the scene. The optical flow vectors of the remaining visible pixels, when combined for all available views, produce plausible 3D scene flow vectors, as visualized in Figure 4.13. Note that the 3D scene flow information is not available on some of the vertices due to self-occlusions. Finally we note that, in this case, the total time cost of computing the 2D optical flows with a Lucas-Kanade implementation of three-level hierarchy and a window size of 30×30 pixels for all visible vertex projections from 8 camera planes, and estimating the 3D scene flow vectors from these optical flows together with smoothing the estimated scene flow in a 3-link neighbourhood is about 8 seconds. The share of different tasks in this total time spent is approximately 1.25, 4.25 and 2.50 seconds, respectively for building the 3D voxel grid and computing visibilities, calculating the visible 2D optical flows and estimating the 3D scene flow.

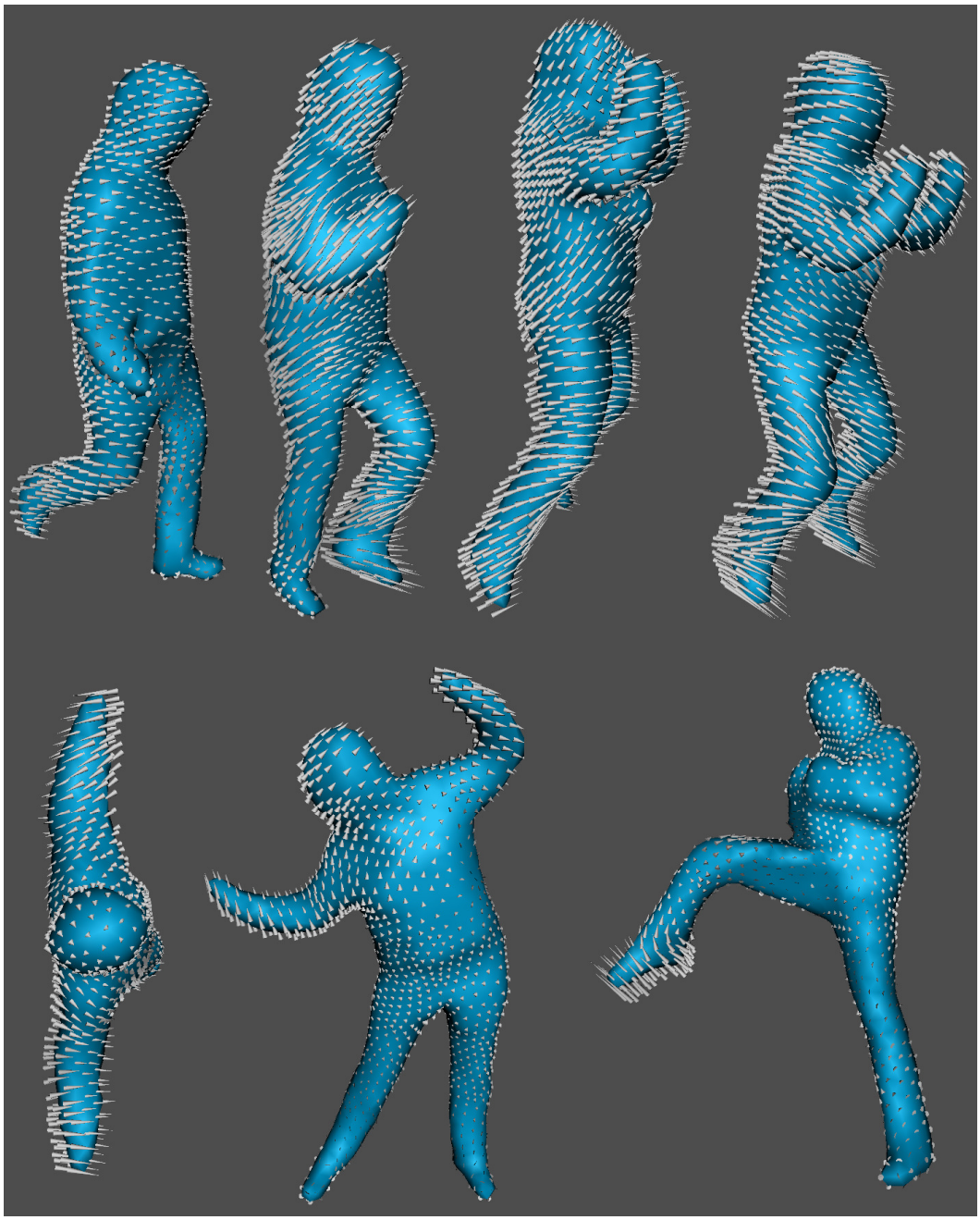

Figure 4.13: Visualization of the estimated 3D scene flow vectors, displayed on sample frames, corresponding to walking, running, jumping, falling, turning, stretching and kick-boxing, for frames 178, 493, 748, 751, 922, 1040 and 1222, respectively.

4.2.3 Sequence 3: Real Scene with severe non-rigid motion

Our third experiment aims to track the same subject of the second sequence while dribbling a ball so to challenge our surface tracking scheme in a video sequence with severe non-rigid motion. Our intention is to represent the body holding a ball as one whole surface instead of two adjacent surfaces. The purpose of this is to test the tracking scheme to model a surface inflation (when the ball arrives) and a surface shrinkage (when the ball departs) by utilizing the mesh restructuring operators so as to rearrange the connectivity of that region, namely the dribbling hand.

We have successfully tracked a multiview video sequence of 60 frames with 3 dribbles (3 receives and 3 throws) starting from an initial reconstruction. Figure 4.14 displays the reconstructed mesh representations of two sample consecutive frames of the sequence together with the corresponding original images. Figure 4.15 zooms on the dribbling hand with heavy restructuring operations for two different frame transitions (receiving and throwing).

Figure 4.14: The reconstructed mesh representations of two consecutive sample frames of the third sequence together with the corresponding original images.

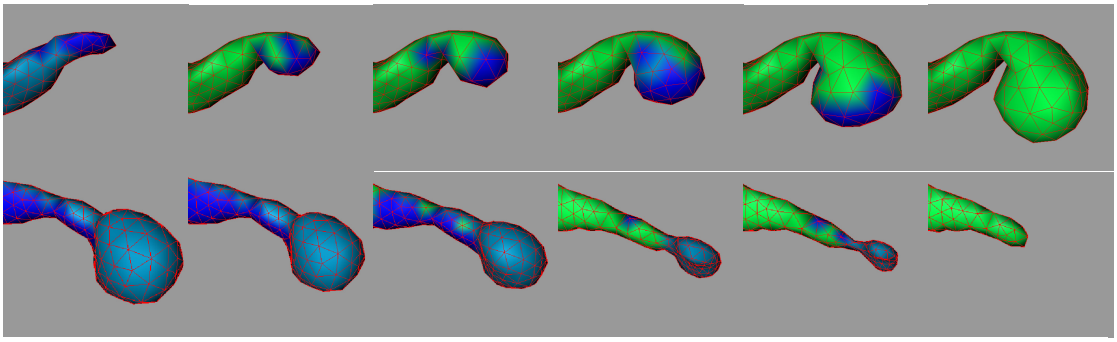

Figure 4.15: Zoom on the dribbling hand of the deformable mesh at various iterations of two different frame transitions. (Top row) Receiving the ball, and (bottom row) throwing the ball. Dark blue areas correspond to IN, light blue areas correspond to OUT and green areas correspond to ON vertices.

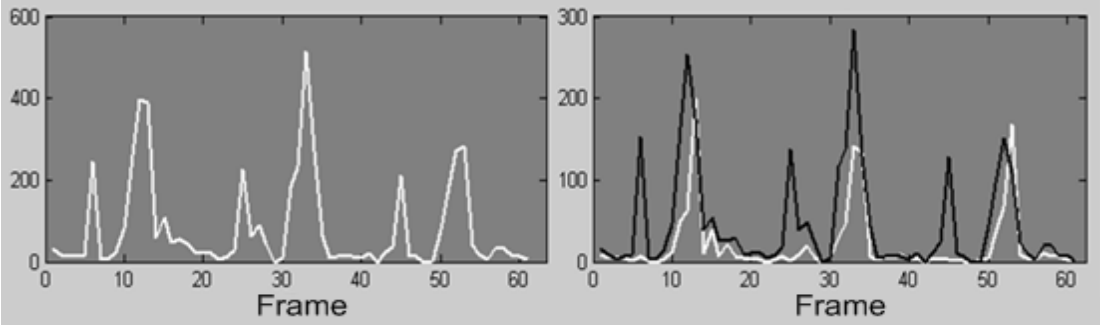

Figure 4.16: Restructuring operations for each frame of the third sequence. (Left) Total number. (Right) Split operations in white, collapse operations in black.

We observe from Figure 4.16 that inflation of the surface representation for the purpose of modeling a new segment requires mostly split operations. However the disconnection of a segment from the surface representation by shrinking that segment cannot be handled only by collapse operations but split operations are also necessary. Although it is counter-intuitive to observe edge splits while a mesh representation is shrinking, they arise due to the successive collapse and flip operations at each iteration which create edges longer then ε_{max}, The split operations remove these edges to maintain a smooth and topologically correct evolution.

4.3 Discussion

In the previous sections, we have presented the basic tracking results with optimum parameter setting. We now analyze our tracking scheme in several other aspects.

Regarding the representation load efficiency of our tracking algorithm, Table 4.4 provides the bit-load B of each sequence when encoded with the encoding scheme described in Chapter 3.4 (Equation 3.12) versus the bit-load B_0 when encoded with the standard vertex-triangle list approach (Equation 3.13), both with 12-bit precision. The results show that the representation load efficiency of the former strategy is about 5 times better than the classical approach. The Bit-load of each frame consists of three components, the bit-load of encoding the header containing the parameters (pose registration parameters and the amount of bits necessary to encode the maximum displacement in that frame transition), the bit-load of encoding the small-scale displacements B_{disp}, and the bit-load of encoding the restructuring operations B_{op}. We note that the storage cost of the initial mesh representation has not been included in the bit-loads given in the table.

	Frames	B_0 (MB)	B (MB)	B_{disp}(MB)	B_{op}(MB)
Sequence 1 (*Jumping Man*)	220	3.91	0.74	0.72	0.01
Sequence 2 (real)	1280	25.53	4.83	4.45	0.34
Sequence 3 (real)	60	1.33	0.23	0.21	0.01

Table 4.4: Bit-loads for the three sequences with different encoding strategies. The bit-load of the header of each frame in our strategy is fixed and it is 28.5 bytes per frame.

Recall from Chapter 3.2.3 that the scene flow information dominates the deformation process at the beginning of mesh evolution and its influence decreases gradually as iterations proceed. Recall also from Eq. 3.10 that the choice of the time coefficient τ determines how long the scene flow information will be effective on a vertex based on the expected number $\hat{K}$ of the total iterations until convergence. For all the experiments that we have presented so far, the coefficient τ was chosen so that $\alpha = \beta$ at the half of the

iterations, that is, at iteration $0.5\hat{K}$. In Table 4.5, we present the results that we have obtained on Sequence 2 by varying this equilibrium point. We observe that, for a fairly large interval around $0.5\hat{K}$, the reconstruction time and operation count change only very slightly, whereas outside of this interval, the efficiency of the tracking process starts to decrease significantly, especially at those frames where the local motion is very fast.

Time coefficient τ	$0.2\,\hat{K}$	$0.3\,\hat{K}$	$0.4\,\hat{K}$	$0.5\,\hat{K}$	$0.6\,\hat{K}$	$0.7\,\hat{K}$	$0.8\,\hat{K}$
Restructuring operation (#)	99.58	86.83	84.05	80.46	79.66	75.85	91.58
Reconstruction time (sec)	9.00	9.61	9.12	9.24	9.24	9.32	9.14

Table 4.5: Number of restructuring operations and the reconstruction time per frame for varying time coefficient τ over frames 720-820 of Sequence 2.

We have tested the performance of our surface tracking method also at a resolution higher than the optimum (ε_{min}= 0.025 *vs.* ε_{min}= 0.015). The results per frame, in terms of vertex number, operation count and reconstruction time, are given in Table 4.6, where the given reconstruction times do not include the time spent for scene flow estimation which takes about 8 sec per frame at low resolution and about 12 sec at high resolution. We observe that the efficiency decreases quadratically with the value of ε_{min} as expected. Therefore we do not recommend to exceed the optimal resolution unless the tradeoff between this quadratically increasing cost and the visual quality pays off. In Figure 4.17, we observe that the quality of the reconstruction does not significantly improve at higher resolutions since the number of cameras is limited with 8 in our case.

	Sequence 1		**Sequence 2**		**Sequence 3**	
	Optimum Resolution	**High Resolution**	**Optimum Resolution**	**High Resolution**	**Optimum Resolution**	**High Resolution**
Vertex (#)	1394	4216	1564	4364	1710	4295
Restruct. op. (#)	18.58	109.78	97.92	393.30	80.99	292.21
Recons. time (sec)	5.93	29.80	7.30	55.28	6.78	32.34

Table 4.6: Comparative statistics per frame for surface tracking at low and high resolutions. Minimum edge length constraint ε_{min} is 0.025 and 0.015 for the optimum and high resolutions, respectively.

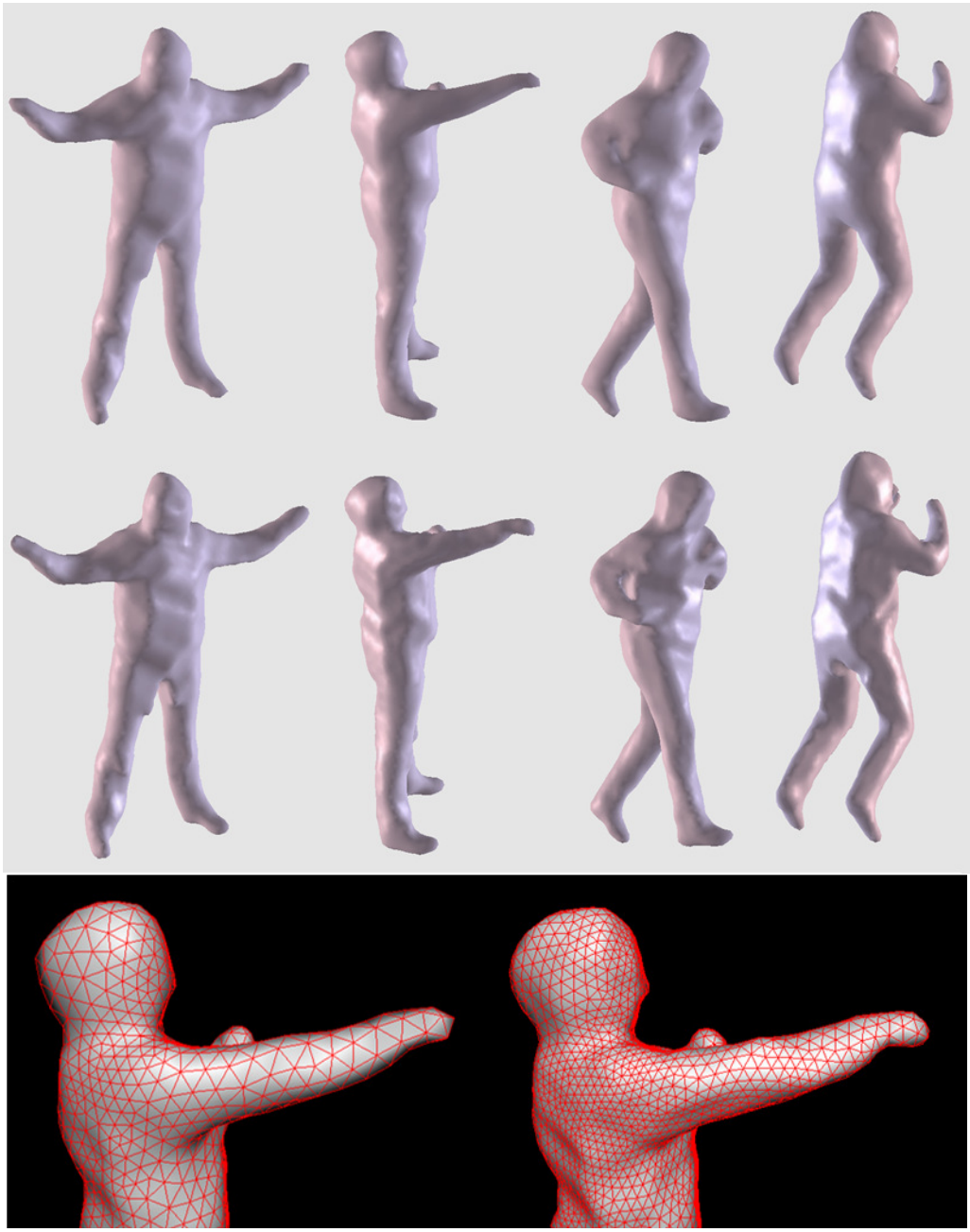

Figure 4.17: Comparing two resolutions. Top row: Mesh representations at optimum resolution (ε_{min}=0.025), Middle row: Corresponding high resolution reconstructions (ε_{min}=0.015). Bottom row: Zoom on a sample reconstruction at the two resolutions.

Regarding the quality of the reconstructed mesh sequences, there are noticeable geometrical discrepancies between the mesh representations and the original scene. These discrepancies are mostly related to the well-known limitation of the shape-from-silhouette technique, which is basically the inability to recover hidden cavities. This drawback does not fortunately lead to disturbing surface artifacts when the object to be tracked is mostly convex or composed of convex parts as in our case, and/or can be compensated by texture mapping.

There is however another source of geometrical discrepancy which may yield disturbing surface artifacts, especially when the number of cameras is limited. This is related to the self-occlusion problem. We display in Figure 4.18 the three extreme cases that have been encountered while tracking Sequence 2. These three problems are all due to self-occlusions and can be eliminated by using more cameras for multiview recording. In the synthetic case for instance, there were 16 cameras available and hence we did not encounter such extreme cases. In the sequel, we provide a more detailed analysis of these three cases.

Case a: A surface protrusion between the arms of the subject appears because there is no camera view to carve out the extra part growing out. When both the left arm and the right arm self-occlude the camera's visibility at the chest of the model, the isolevel function in Equation 3.2 returns positive values at that region and hence the vertices displace outwards in the direction of their surface normal. If there were one more camera, e.g., positioned above, then the corresponding isovalues would be negative and that region would be carved out.

Case b: The empty space between the legs cannot be carved out. Although the problematic region returns negative isolevel values, the colliding feet create a torus/cycle that imprisons the excessive surface between the legs, avoiding it to vanish out by restructuring operations. This case could be resolved either by applying a topology

modification operator, or if the reconstruction of this frame had started from a problem-free mesh structure where the excessive surface had already been carved out, prior to the collision of the feet, by using an extra camera viewpoint.

Case c: This has occurred because in the previous frames while the model was stretching towards her left, the problematic region was covered by the model herself, self-occluding all the available camera viewpoints and causing the collision region to move along the surface almost randomly. This case could be resolved again either by adding a camera viewpoint, for example positioned on the floor of the scene, or by implementing a collision-handling algorithm that keeps track of the positions of the vertices up to that frame transition.

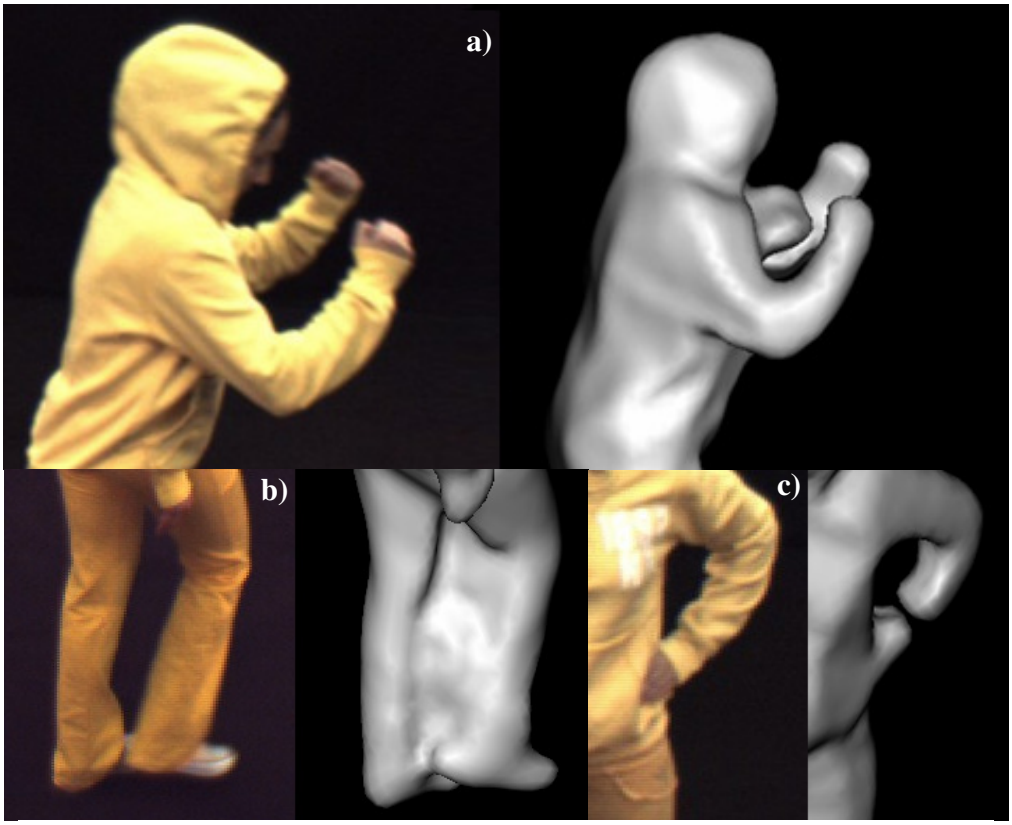

Figure 4.18: Three extreme cases, encountered while tracking Sequence 2, where the geometry cannot correctly be recovered due to self-occlusions.

Our final remark is on the robustness of the proposed surface tracking algorithm. Recall that the algorithm has failed to track the surface for all the three sequences we have tested, when the 3D scene flow information is not incorporated and only silhouette-based deformation is applied (case 1). The reason why the silhouette-only scheme crashes, which occurred approximately once for hundred frames in our experiments, is because the initial mesh geometry with which we start the evolution may sometimes be positioned such that a segment of the object may partially fall inside another segment of the object in the target surface that has not yet been evolved into. Usually this can be handled by mesh restructuring and silhouette-based displacements but there are extreme cases when the motion of a segment is too fast so that the segment may evolve into a totally separate part of the surface which would then rise the necessity of changing the mesh topology around those segments. We avoid such extreme cases by incorporating 3D scene flow assistance into silhouette-based deformation. We enforce each such segment of an object to position correctly according to the target surface thanks to pose registration, and if it does not, we rely on the scene flow vectors to smoothly carry the surface at the early iterations towards the target position.

Chapter 5

CONCLUSIONS AND FUTURE WORK

This work has shown that the silhouette geometry of a non-rigid dynamic object can efficiently be tracked from multiview video based on silhouette and scene flow information by using a fast snake-based deformation scheme coupled with mesh restructuring operations and a collision detection algorithm. The proposed surface tracking method exploits the temporal redundancies between consecutive frames, and as a result, the reconstruction time and the representation load of the time-varying geometry are significantly reduced when compared to reconstructing each frame from scratch. The mesh restructuring operations and small-scale vertex displacements can be used to efficiently encode the whole mesh sequence representing the smooth time-varying surface that has been tracked.

We have tested our method on relatively long and challenging real sequences exhibiting complex and diverse motion patterns. A remarkable advantage of the proposed method is its ability to track objects that may undergo an arbitrary non-rigid deformation since our scheme can track both geometry and connectivity of a dynamic surface mesh, in contrast to few other surface tracking methods available in the literature.

The only limitations to the presented framework are that the resulting surface representations lack the ability to model hidden cavities and that the quality of the reconstructions is restricted to the number of available camera views, which are both classical limitations of the shape-from-silhouette techniques. One can overcome the latter restriction, to some degree, simply by increasing the number of cameras used during multiview acquisition, however the complexity of the acquisition itself and the cost of the

image processing would linearly increase with the number of added viewpoints. We note that such multiview video recording systems, which employ 16 or even more cameras, are becoming more and more commonplace. As future work we plan to address both of these limitations. Currently we utilize the multiview texture information only to compute the 3D scene flow vectors. However the multistereo information, that could be extracted from multiview texture images, can be used to further enhance the produced silhouette-based reconstructions so as to capture finer surface concavities

BIBLIOGRAPHY

[1] A. Alatan, Y. Yemez, U. Gudukbay, X. Zabulis, K. Muller, C¸ . E. Erdem, C. Weigel, and A. Smolic, "Scene representation technologies for 3dtv a survey," *IEEE Trans. Circuits and Systems for Video Tech.*, vol. 17, no. 11, pp. 1587–1605, 2007.

[2] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G. B. Akar, G. A. Triantafyllidis, and A. Koz, "Coding algorithms for 3dtv a survey," *IEEE Trans. Circuits and Systems for Video Tech.*, vol. 17, no. 11, pp. 1606–1621, 2007.

[3] T. B. Moeslund, A. Hilton, and V. Kr¨uger, "A survey of advances in vision-based human motion capture and analysis," *Computer Vision and Image Understanding*, vol. 104, no. 2, pp. 90–126, 2006.

[4] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H. P. Seidel, and S. Thrun, "Performance capture from sparse multi-view video," *Proc. SIGGRAPH*, pp. 1– 10, 2008.

[5] E. de Aguiar, C. Theobalt, C. Stoll, and H. P. Seidel, "Marker-less deformable mesh tracking for human shape and motion capture," *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8, 2007.

[6] J. Starck and A. Hilton, "Surface capture for performance-based animation," *IEEE Computer Graphics and Applications*, vol. 27, no. 3, pp. 21–31, 2007.

[7] T. Matsuyama, X. Wu, T. Takai, and S. Nobuhara, "Real-time 3d shape reconstruction, dynamic 3d mesh deformation, and high fidelity visualization for 3d video," *Computer Vision and Image Understanding*, vol. 96, no. 3, pp. 393–434, 2004.

[8] K. Varanasi, A. Zaharescu, E. Boyer, and R. Horaud, "Temporal surface tracking using mesh evolution," *Proc. European Conference on Computer Vision (ECCV)*, pp. 30–43, 2008.

[9] S. W¨urmlin, E. Lamboray, O. G. Staadt, and M. H. Gross, “3d video recorder: a system for recording and playing free-viewpoint video,” *Computer Graphics Forum*, vol. 22, no. 2, p. 181193, 2003.

[10] B. Curless, “Overview of active vision techniques,” *Proc. SIGGRAPH Course on 3D Photography*, 1999.

[11] L. P. Kobbelt, T. Bareuther, and H. Seidel, “Multiresolution shape deformations for meshes with dynamic vertex connectivity,” *Computer Graphics Forum (Eurographics’ 00)*, vol. 19, 2000.

[12] O. Hall-Holt and S. Rusinkiewicz, “Stripe boundary codes for real-time structured-light range scanning of moving objects,” *IEEE Int. Conf. on Computer Vision (ICCV)*, pp. 359–366, 2001.

[13] L. Zhang, B. Curless, and S. M. Seitz, “Spacetime stereo: Shape recovery for dynamic scenes,” *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp. 367–374, 2003.

[14] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, “A comparison and evaluation of multi-view stereo reconstruction algorithms,” *Proc. Computer Vision and Pattern Recognition (CVPR)*, pp. 519–526, 2006.

[15] C. H. Esteban and F. Schmitt, “Silhouette and stereo fusion. for 3d object modeling,” *Computer Vision and Image Understanding*, vol. 96, no. 3, pp. 367–392, 2004.

[16] K. Mueller, A. Smolic, P. Merkle, M. Kautzner, and T. Wiegand, “Coding of 3d meshes and video textures for 3d video objects,” *Proc. Picture Coding Symposium*, 2004.

[17] K. M. Cheung, S. Baker, and T. Kanade, “Shapefrom- silhouette across time part i: Theory and algorithms,” *Int. Journal of Computer Vision*, vol. 63, no. 3, pp. 221–247, 2004.

[18] S. C. Bilir and Y. Yemez, “Time varying surface reconstruction from multiview video,” *IEEE Int. Conf. on Shape Modeling and Applications (SMI)*, pp. 47– 51, 2008.

[19] M. Botsch and O. Sorkine, “On linear variational surface deformation methods,” *IEEE Trans. Visualization and Comp. Graphics*, vol. 14, no. 1, pp. 213–230, 2008.

[20] M. A. Magnor and B. Goldlcke, “Spacetimecoherent geometry reconstruction from multiple video streams,” *Int. Symp. 3DPVT*, pp. 365–372, 2004.

[21] Y. Sahillioglu and Y. Yemez, “A surface deformation framework for 3d shape recovery,” *Workshop on Multimedia Content Representation, Classification and Security*, pp. 570–577, 2006.

[22] Y. Yemez and F. Schmitt, “3d reconstruction of real objects with high resolution shape and texture,” *Image and Vision Computing*, vol. 22, pp. 1137–1153, 2004.

[23] M. Kaas, A. Witkin, and D. Terzopoulos, “Snakes: Active contour models,” *Int. Journal of Computer Vision*, vol. 1, no. 4, pp. 321–332, 1988.

[24] G. Taubin, “A signal processing approach to fair surface design,” *Proc. SIGGRAPH*, pp. 315–358, 1995.

[25] Z. J. Wood, P. Schr¨oder, D. Breen, and M. Desbrun, “Semi-regular mesh extraction from volumes,” *Proc. Visualization’00*, pp. 275–282, 2000.

[26] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, “Mesh optimization,” *Proc. SIGGRAPH*, pp. 19–26, 1993.

[27] P. Sand, L. McMillan, and J. Popovic, “Continuous capture of skin deformation,” *Int. Conf. on Computer Graphics and Interactive Techniques*, 2003.

[28] S. Vedula, S. Baker, P. Rander, R. Collins, T. Kanade, “Three-Dimensional Scene Flow”, *IEEE Trans. PAMI*, pp. 475-480, 2005.

[29] B. D. Lucas, T. Kanade, “An iterative image registration technique with an application to stereo vision”, *International Joint Conference on Artificial Intelligence*, 1981.

[30] J. H. Manton, “Optimization algorithms exploiting unitary constraints”, *IEEE Transactions on Signal Processing*, pp. 635–650, 2002.

[31] T. Horprasert, D. Harwood, L. S. Davis, “A robust background subtraction and shadow detection”, *Asian Conference on Computer Vision,* 2000.
[32] Y. Sahillioglu and Y. Yemez, “Building Topologically Correct Mesh Models from Silhouette Images Using Mesh Deformation”, submitted to *Pattern Recognition Letters*, 2008.
[33] T. Svoboda, D. Martinec, T. Pajdla, “A convenient multi-camera self-calibration for virtual environments”, *PRESENCE: Teleoperators and Virtual Environments*, pp. 407-422, 2005.

VITA

Salih Cihan Bilir was born in Ankara, Turkey on May 05, 1985. He graduated from Gazi Anatolian High School, Ankara in 2002. He received his B.S. degree in Electrical and Electronics Engineering from Bilkent University, Ankara in 2006. In September 2006, he joined the M.S. Program in Electrical and Computer Engineering at Koç University, as a research and teaching assistant. Having received the M.S. degree in March 2009, he is now preparing to join the Turkish army to serve his duty.