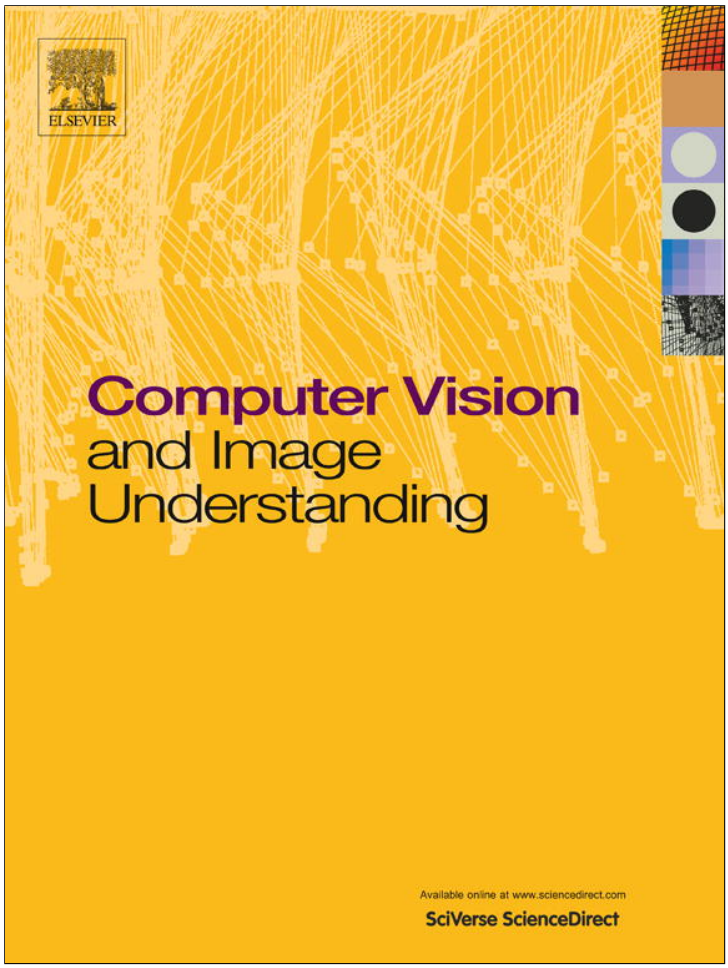

(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Computer Vision and Image Understanding 116 (2012) 1121–1134

Contents lists available at SciVerse ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Non-rigid 3D shape tracking from multiview video ☆

S.C. Bilir, Y. Yemez *

Multimedia, Vision and Graphics Laboratory, College of Engineering, Koç University, Sarıyer, Istanbul 34450, Turkey

ARTICLE INFO

Article history:
Received 15 November 2011
Accepted 17 July 2012
Available online 9 August 2012

Keywords:
3D shape tracking
Mesh deformation
3D scene flow
Shape from silhouette
3D video

ABSTRACT

We present a fast and efficient non-rigid shape tracking method for modeling dynamic 3D objects from multiview video. Starting from an initial mesh representation, the shape of a dynamic object is tracked over time, both in geometry and topology, 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 with the aid of 3D scene flow vectors. The whole time-varying shape is then represented as a mesh sequence which can efficiently be encoded in terms of restructuring and topological operations, and small-scale vertex displacements along with the initial model. The proposed method has the ability to deal with dynamic objects that may undergo non-rigid transformations and topological changes. The time-varying mesh representations of such non-rigid shapes, which are not necessarily of fixed connectivity, can successfully be tracked thanks to restructuring and topological operations employed in our deformation scheme. We demonstrate the performance of the proposed method both on real and synthetic sequences.

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, shading and/or stereo information acquired from multicamera video sequences [4–13], due to the limitations of active reconstruction methods in temporal axis [14].

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 for both 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 [11,15]. Hence they primarily consider rigid and/or articulated motion, and may not handle the reconstruction problem when the object of interest undergoes an arbitrary non-rigid motion or a topological transformation. As an example, we display in Fig. 1 three frames from a video sequence containing non-rigid motion that cannot be handled by fixed connectivity representations.

In this paper, we present an efficient *shape tracking* method for modeling dynamic objects based on multiview silhouette and 3D scene flow information. Here the term "shape tracking", in the way we use it, refers to reconstruction of the surface shape 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 few methods in the literature, which are shape tracking in this sense and which can build complete shape models of dynamic objects [4,5,7,8,10]. The main distinction of the method that we propose in this paper, as compared to previous work, is in the way we represent time-varying geometry. We track and encode time-varying geometry in terms of both connectivity changes and vertex displacements. In this way objects with arbitrary shape and motion can easily be handled, and the reconstruction problem is reduced to an energy minimization problem which can be solved by a fast snake-based deformation scheme, yielding a computationally very efficient shape tracking method. Unlike existing shape tracking

☆ This paper has been recommended for acceptance by R. Bergevin.

* Corresponding author. Fax: +90 212 3381548.
E-mail addresses: sbilir@ku.edu.tr (S.C. Bilir), yyemez@ku.edu.tr (Y. Yemez).

1077-3142/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.cviu.2012.07.001

Fig. 1. Three frames from a video sequence (in chronological order) of an actor while taking off his hat (zoomed on upper body). Our shape tracking scheme can handle this video sequence while methods based on fixed connectivity representations cannot.

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 shape of the dynamic object is tracked over time, both in geometry and connectivity, via mesh deformation based solely on image cues. We also address the self-collision problem, which is disregarded in most shape tracking methods, via a very efficient collision handling strategy coupled with topological split/merge operations.

The mesh representation of each frame is obtained by evolving the mesh of the previous frame towards the optimal surface defined by the silhouettes of that instance. This produces a sequence of meshes, $M^{(0)}, M^{(1)}, \ldots, M^{(t)}, \ldots$, representing the time-varying silhouette geometry, i.e., the time-varying visual hull of the dynamic object in the scene. Our snake-based deformable model is based upon the deformation scheme which was proposed in [16] for the classical static shape from silhouette. We extend this deformation scheme to the dynamic case so as to address the problem of shape tracking. This scheme enables us to control parametrization and topology of the dynamic mesh model for robust mesh evolution across time via local mesh transformation operations. These mesh operations and small-scale displacements along with the initial mesh representation yield a compact and spatio-temporally coherent representation of the whole time-varying shape. The block diagram of the overall shape tracking system is given in Fig. 2.

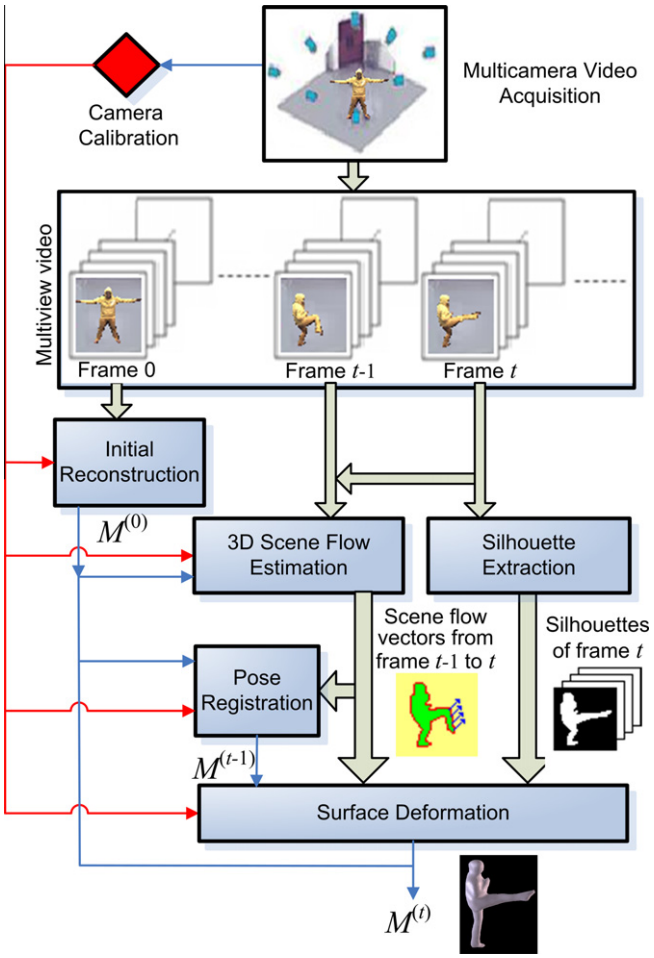

Fig. 2. Block diagram of the proposed shape tracking system.

The paper is organized as follows. First, in Section 2, we discuss the related work on time-varying shape modeling, and then proceed, in Section 3, with the description of the deformation framework that we employ in our tracking scheme. Our deformable model is coupled with efficient collision and topology handling procedures as described in Section 4. The overall shape tracking system, which makes use of multiview silhouette and 3D scene flow information, is described in Section 5. Section 6 provides and discusses the experimental results, and finally, Section 7 gives concluding remarks and some future research perspectives.

2. 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 [14]. 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 [17,18] 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 motion, and are capable of producing only depth images, not full surface representations. On the other hand, passive reconstruction techniques, which are based solely on image cues such as multiview stereo [19] and/or silhouettes [20], 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 [6–10,12]. The resulting sequence of meshes can then be matched so as to obtain a time-consistent representation. All these methods, with the exception of [8], impose fixed connectivity hence they cannot adapt to non-rigid deformations and topological changes. In order to achieve a fixed connectivity representation, Starck et al. [6] for instance use spherical reparametrization of the resulting mesh sequence

whereas other methods basically cast the reconstruction problem to a shape tracking framework: 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 [7,10]. 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 connectivity, based on multiview image cues. In particular, the method in [4] presents very high quality reconstructions but the method requires user interaction and an extensive computation time which is reported as about 10 min per frame on a standard computer with a 3 GHz CPU. Moreover the method, which aims at fixed connectivity, has no mechanism to handle arbitrary non-rigid motion and self-collisions.

There is actually very little work in shape tracking literature, that addresses the self-collision problem [8]. Self-collisions may occur during surface tracking due to two different reasons: (1) a misguided self-intersection of the deformable mesh, and (2) a real physical contact of different surface parts during motion. Varanasi et al. [8] treat these two different types of self-collisions in the same manner by making use of a topology adaptive self-intersection removal technique, namely TransforMesh [21], which is a costly algorithm that involves repeated self intersection tests, triangle validation and stitching procedures during mesh evolution. With this technique, each self-collision, whether a misguided self-intersection or a physical surface contact, implicitly creates a merge or split in topology. In contrast, the method that we present in this paper differentiates these two cases and avoids self-intersections based on the uniformity constraints imposed on the deformable mesh via the use of restructuring operators. Physical surface contacts, on the other hand, are detected explicitly and fixed only when necessary by using topological merge and split operations. In this way, redundant topological transformations are avoided and the resulting time-varying geometry can be encoded in terms of mesh restructuring operations and vertex displacements along with possibly a few explicit topology operations. We also note that the method presented in [8] requires reconstruction of the surface mesh from scratch, separately for each frame of a given sequence.

Existing methods for reconstruction of dynamic objects rely mainly on multiview silhouette information [22]. The strength of the shape from silhouette technique in general 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 in order 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 [23], 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 solely on silhouette information in a very fast manner using a snake-based deformable model. In this paper, we generalize our tracking framework to objects with arbitrary shape and topology, and make it more efficient and robust, by incorporating additional features such as 3D scene flow, collision detection and topological operations.

Shape tracking methods usually resort in some way or other to mesh deformation methods, such as Laplacian deformation [24], which is a powerful tool for mesh morphing and editing, and which can be used to obtain animating mesh sequences with fixed connectivity [4,10]. However, with Laplacian deformation which is a differential and piecewise linear scheme, mesh connectivity cannot be altered, hence dynamic objects with arbitrary motion cannot be tracked. Another alternative [25] is based on volumetric level-set technique, that 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 appropriate use of restructuring and topological 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 efficiently track and encode the temporal information of dynamic surfaces with arbitrary motion and shape. We finally note that our topology handling procedure adopts the topology split and merging operations of the deformation scheme which was proposed in [26] primarily for segmentation of 3D anatomical structures.

3. Deformable model

Our deformation technique is based on the iterative use of two transformations T and U that deform, at each frame t, an initial triangle mesh $M_{0,0}^{(t)}$ towards the object surface $S^{(t)}$ through the following surface evolution equations:

$$M_{k+1,l}^{(t)} = T\left(M_{k,l}^{(t)}\right), \tag{1}$$

$$M_{0,l+1}^{(t)} = U\left(M_{k^*,l}^{(t)}\right), \tag{2}$$

where k is the iteration counter for geometrical evolution of the surface and l for its topological evolution. For a given l, the deformable model $M_{k,l}^{(t)}$ is required to preserve its topological type during its geometrical evolution via Eq. (1) and to remain as a smooth manifold mesh representation free of geometrical distortions, and eventually to converge to an optimal mesh $M_{k^*,l}^{(t)}$ that represents the object surface $S^{(t)}$ as accurately as possible at the equilibrium state, so that the following equality is satisfied:

$$M_{k^*,l}^{(t)} = T\left(M_{k^*,l}^{(t)}\right). \tag{3}$$

The topology of the deformable model can then be modified at the convergence, only if necessary, by using the operator U via Eq. (2). The deformable model $M_{0,l+1}^{(t)}$ then becomes the initial mesh of the next reiteration of the geometrical evolution with the modified topology.

The operator U is defined as the composition of two operators involving topology merging and split transformations:

$$U = U_{\text{merge}} \circ U_{\text{split}}, \tag{4}$$

and applied at the convergence of each geometrical evolution process until no further topological operations are necessary, i.e., until

the topology of the deformable model matches that of the target surface, so that the following equality is satisfied:

$$M^{(t)}_{k^*,l^*} = U\left(M^{(t)}_{k^*,l^*}\right), \tag{5}$$

where $M^{(t)}_{k^*,l^*}$ denotes the mesh representation which is to be optimal both in topology and geometry. We will explain how we perform topological merge and split operations in more detail later in Section 4.2.

We define T as the composition of three transformations: $T = T_{\mathrm{d}} \circ T_{\mathrm{s}} \circ T_{\mathrm{r}}$, which we will refer to as *displacement*, *smoothing* and *restructuring* operators, respectively. The displacement operator pushes the deformable mesh towards the object surface 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 [27]. Note that the operator T does not modify the topology of the deformable mesh.

The restructuring operator, T_{r}, is the composition of three operators:

$$T_{\mathrm{r}} = T_{\mathrm{flip}} \circ T_{\mathrm{col}} \circ T_{\mathrm{split}}, \tag{6}$$

where T_{flip}, T_{col} and T_{split} are defined in terms of edge flip, edge collapse and edge split transformations which were first introduced by Hoppe et al. [28] for mesh optimization (see Fig. 3). We adopt these elementary transformations for our deformation process in the way Kobbelt et al. [29] use them for mesh editing. At the end of each iteration of the geometrical surface evolution, the operator T_{split} first splits all edges longer than $\varepsilon_{\max}$ at their midpoints. Then, the operator T_{col} successively eliminates all edges shorter than $\varepsilon_{\min}$ by edge collapses. Finally, the operator T_{flip} is applied to reduce the number of irregular vertices possibly created by the previous collapse and split operations: The common edge of any two neighboring triangles is swapped with the one joining the unshared vertices of the triangles, if this operation increases the number of vertices with valence close to 6. For the split operation to be compatible with the collapse operation, the threshold $\varepsilon_{\max}$ has to be chosen such that $\varepsilon_{\max} \geqslant 2\varepsilon_{\min}$ since otherwise split operations would create edges with length smaller than $\varepsilon_{\min}$. If we set $\varepsilon_{\max} = \kappa\varepsilon_{\min}$, then the edge length ratio is bounded by $\varepsilon_{\max}/\varepsilon_{\min} = \kappa$. To have uniformly sized triangles with small aspect ratios, one can choose κ as small as possible, i.e., $\kappa = 2$, which may however in turn redundantly increase the number of edge operations needed during surface evolution. Therefore in all our experiments we set κ as sufficiently larger than this minimum value: $\kappa = 3$. We note that the minimum and maximum edge length constraints are only soft requirements that may occasionally be violated during mesh evolution. This is described in detail in [16]. Nevertheless, the restructuring operator serves well to regularize the mesh connectivity and thereby provides a stable surface evolution.

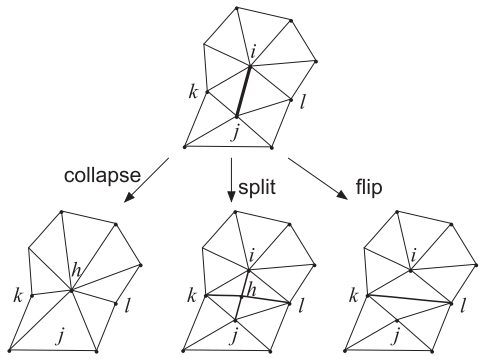

Fig. 3. Restructuring operations: edge collapse, edge split and edge flip.

The displacement operator $T_{\mathrm{d}}(M_k)$ maps the deformable mesh M_k (dropping the indices t and l) to M'_k by moving each vertex $\mathbf{v}_{i,k}$ with a displacement $\mathbf{d}(\mathbf{v}_{i,k})$, where $\mathbf{v}_{i,k}$ denotes the position vector of the ith vertex at iteration k (hence M'_k has the same connectivity as M_k). The displacement is set to be in the direction of the unit vector $\mathbf{D}(\mathbf{v}_{i,k})$ pointing from the vertex to the target surface:

$$\mathbf{d}(\mathbf{v}_{i,k}) = \delta(\mathbf{v}_{i,k}) \cdot \mathbf{D}(\mathbf{v}_{i,k}). \tag{7}$$

The displacement scalar $\delta(\mathbf{v}_{i,k})$ is computed based on the signed distance from vertex $\mathbf{v}_{i,k}$ to the target surface, as will later be explained in detail in Section 5. For the time being, we note that, for a stable surface evolution, the magnitude of the displacement vector in Eq. (7) is constrained by half of the minimum edge length, i.e., with $|\delta(\mathbf{v}_{i,k})| \leqslant \varepsilon_{\min}/2$, $\forall i, k$, so that neighboring vertices do not interfere with each other and yield self intersections.

The smoothing operator, T_{s}, should be easy to compute, yet must not yield any geometrical shrinkage and bias in the final surface estimate. To achieve this, we employ the tangential Laplacian smoothing [30]. The operator $T_{\mathrm{s}}(M_k)$ maps the deformable mesh M_k to M'_k by moving each vertex $\mathbf{v}_{i,k}$ to $\mathbf{v}'_{i,k}$ by a tangential displacement $\Delta\mathbf{v}_{i,k}$, which is obtained by

$$\Delta\mathbf{v}_{i,k} = \mathbf{L}(\mathbf{v}_{i,k}) - (\mathbf{L}(\mathbf{v}_{i,k}) \cdot \mathbf{N}(\mathbf{v}_{i,k}))\mathbf{N}(\mathbf{v}_{i,k}), \tag{8}$$

where $\mathbf{N}(\mathbf{v}_{i,k})$ is the surface normal at vertex $\mathbf{v}_{i,k}$ and $\mathbf{L}(\mathbf{v}_{i,k})$ stands for the Laplacian displacement that moves the vertex to the centroid of the vertices in its one-ring neighborhood. The smoothing operator not only regularizes the deformation process but also enforces mesh uniformity prior to restructuring operations.

4. Collision and topology handling

We deal with collisions (i) by avoiding any self-intersection of the deformable mesh during surface evolution iterations driven by Eq. (1), and (ii) by applying topological operations so as to handle any physical contact of different surface parts, that might be encountered during deformation, as given by Eq. (2).

4.1. Self-intersections

To avoid self-intersections, we employ the collision detection method proposed in [16]. The basic idea in this method is to prevent non-neighboring vertices from approaching each other by more than some distance threshold. This distance threshold, denoted by ζ, is based on the minimum and maximum edge length constraints imposed on the deformable mesh, and when $\varepsilon_{\max} = 3\text{-}\varepsilon_{\min}$ as in our case, it can be written as (following [16])

$$\zeta > \frac{\sqrt{13}}{2}\,\varepsilon_{\min}, \tag{9}$$

The basic procedure is as follows. At a given iteration of mesh evolution, all the vertices of the deformable mesh are first displaced by the displacement operator. Then each vertex is checked one by one against the vertices which are not its neighbors. If a vertex is found to have approached any other vertex by more than the collision detection threshold ζ, then the vertex is moved back to its original position. In [16], it is shown that when the edge length constraints are strictly met, the inequality given in (9) ensures that collision of non-neighboring vertices is avoided. Recall however from Section 3 that the maximum edge length requirement is not a hard constraint and can be violated in rare occasions. Thus some triangles with sides larger than $\varepsilon_{\max}$ may show up in the deformable mesh during surface evolution. These large triangles

are handled by sampling, that is, by virtually quadrisecting each triangle in a recursive manner until the maximum edge length requirement is met. The resulting virtual vertices are used only for collision detection purposes. The described collision detection algorithm can be implemented in an efficient manner with $O(N_v \log N_v)$ complexity, N_v being the number of vertices in the deformable mesh, by using an octree structure where each vertex is associated with a node (or voxel) and by checking each vertex only against those in its vicinity.

The collision handling procedure described above handles self-intersections of only non-neighboring vertices. Hence the success of the procedure relies upon the assumption that neighboring vertices never create self-intersections. This is a valid assumption due to the minimum edge length constraint which also bounds the displacement of a vertex at one iteration by half of the minimum edge length, $\varepsilon_{\min}/2$. The vertex neighborhood, in which vertices are allowed to move as free of the collision threshold, needs to be defined considerately. In order to have a flexible deformation, we define this area as the two-ring neighborhood of a vertex assuming that self-intersections are not likely to occur at very nearby vertices due to the regularizing effect of the smoothing operator as well as the minimum edge length constraint.

4.2. Surface contacts

Surface contacts are identified and handled only at the convergence of each mesh evolution process. A surface contact may manifest itself in two occasions, when two distant parts of a surface touch each other and/or when they break apart. We treat these two cases separately, in the first case by topology merging and in the second by topology split. To this effect, at the convergence of each surface evolution process, the operators U_{merge} and U_{split} are successively applied to the deformable mesh (see Eq. (4)). The order of these two operators usually does not matter.

The operator U_{merge} searches for pairs of two distant vertices both of which lie inside the convex hull of the object and which were detected to be colliding at the last iteration before convergence. If one is found (there may be more than one such pairs), the topology merging transformation of [26] is applied to only one of these pairs (whichever comes first and the others are left to be processed at the end of the next surface evolution process): The colliding vertices are first removed from the mesh structure. The vertices in their one-ring neighborhoods are then matched and connected so as to build a surface tunnel between the two separate parts of the deformable surface. In order to facilitate the process of vertex matching, before merging, we equalize the valences of the colliding vertices by applying collapse operation(s) at the vertex whichever has a valence higher than the other.

Following [26], the operator U_{split} searches for a triplet of adjacent vertices forming a triangle which is not a face of the deformable mesh. We additionally require this triangle to tend to vanish, i.e., at least one of its edges to be shorter than the minimum edge length constraint. There may exist more than one such triplets, but again only one of them is processed by cutting the mesh into two along the edges of the triangle and filling the created holes by two new faces. The surface evolution iterations and the topological operations at each convergence are repeated until the equilibrium state in Eq. (5) is reached (see the example given in Section 6.1.3).

5. Shape tracking

The mesh representation of the object surface at each frame t, $M^{(t)} = M^{(t)}_{k^*,l^*}$, is reconstructed by deforming the shape reconstructed at the previous frame, $M^{(t)}_{0,0} = M^{(t-1)}_{k^*,l^*}$. The deformation process is driven primarily by the silhouette information. In this section, we first describe how to determine the displacement operator based only on silhouettes and then explain how to incorporate 3D scene flow into the tracking/deformation process (see also Fig. 2).

5.1. Silhouette-based displacement

The displacement, $\mathbf{d}(\mathbf{v}_{i,k})$, at each vertex i of the deformable mesh and at each iteration k of the surface evolution, can be computed based on the time-varying silhouette information. We set the direction of the silhouette-based displacement, denoted by $\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k})$, so as to be perpendicular to the deformable surface (see also Fig. 4):

$$\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k}) = \delta_{\text{sil}}(\mathbf{v}_{i,k}) \cdot \mathbf{N}(\mathbf{v}_{i,k}). \tag{10}$$

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 $\mathbf{v}_{i,k}$ is with respect to the silhouettes at that iteration. 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:

$$\delta_{\text{sil}}(\mathbf{v}_{i,k}) = \varepsilon_{\min} f(\mathbf{v}_{i,k}) = \varepsilon_{\min} \min_n \{G[\text{Proj}_{I_n}(\mathbf{v}_{i,k})] - 0.5\}, \tag{11}$$

where the function G, taking values between 0 and 1, is the bilinear interpolation of the sub-pixelic projection $\text{Proj}_{I_n}(\mathbf{v}_{i,k})$ onto the camera plane of the binary silhouette image I_n (0 for outside, 1 for inside) in the sequence, and given by

$$\begin{aligned} G(u,v) = {} & (1-a)((1-b)I(\lfloor u \rfloor, \lfloor v \rfloor) + bI(\lfloor u \rfloor, \lfloor v \rfloor + 1)) \\ & + a((1-b)I(\lfloor u \rfloor + 1, \lfloor v \rfloor) + bI(\lfloor u \rfloor + 1, \lfloor v \rfloor + 1)), \end{aligned} \tag{12}$$

where $(\lfloor u \rfloor, \lfloor v \rfloor)$ denotes the integer part, and (a, b) the fractional part of the projection coordinate (u, v) on the binary silhouette image I. 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.

During surface evolution, the vertices of the deformable mesh can switch between three different states with respect to their isovalues: IN, OUT and ON. The state of a vertex $\mathbf{v}_{i,k}$ at a given iteration k is 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 Eq. (11), the displacement at each ON vertex takes a value 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 with displacement scalars $\varepsilon_{\min}/2$ or $-\varepsilon_{\min}/2$.

For a more accurate and faster convergence, we integrate a fine-tuning procedure to the surface evolution process as proposed in [16]. We detect the instances when a vertex $\mathbf{v}$ 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 by

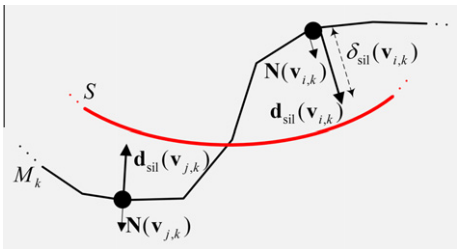

Fig. 4. Illustration of the silhouette-based displacement operator in 2D. The displacement $\mathbf{d}_{\text{sil}}(\mathbf{v}_{i,k})$ on each vertex $\mathbf{v}_{i,k}$ of the deformable mesh M_k at iteration k is set to be in the direction of the surface normal $\mathbf{N}(\mathbf{v}_{i,k})$ and computed based on the signed distance from the vertex to the target surface S.

searching for the point $\mathbf{v}^*$ where the isolevel function is (almost) zero on the line segment joining the positions of the vertex $\mathbf{v}$ before and after displacement. The vertex is then moved to this location. Note that the vertex is not deactivated at this point and can still move due to the smoothing operator at the next iterations until it finds its optimal placement within the narrow band around the object boundary.

5.2. Scene flow estimation

To enhance efficiency of the shape tracking process as well as its stability, we incorporate 3D scene flow (3DSF) information into the deformation scheme. To this effect, we associate a scene flow vector $\mathbf{w}_i^{(t)}$ to each vertex $\mathbf{v}_i^{(t)}$ of the mesh representation $M^{(t)}$ at frame t, which points to its position estimate, $\hat{\mathbf{v}}_i^{(t+1)}$, at the next frame $t+1$:

$$\hat{\mathbf{v}}_i^{(t+1)} = \mathbf{v}_i^{(t)} + \mathbf{w}_i^{(t)}, \tag{13}$$

To estimate the scene flow vector $\mathbf{w}_i^{(t)}$, we adopt the method presented in [31] (by dropping the indices i and t):

1. Find the set of camera viewpoints that the vertex $\mathbf{v}$ is visible to, $\mathrm{Vis}(\mathbf{v}) = \{C_1, C_2, \ldots, C_{N_{\mathrm{vis}}}\}$, where N_{vis} is the number of these viewpoints. For this purpose we make use of the voxel grid which is already generated for collision handling. We scan the voxels along the line of sight from $\mathbf{v}$ to the optical center of a given camera C_n and check if any other vertex occludes its visibility from that camera.
2. By using the camera projection matrix $\mathbf{P}_n$, project the vertex $\mathbf{v} = (x, y, z)$ onto each camera plane C_n in $\mathrm{Vis}(\mathbf{v})$:

$$u_n = \frac{[\mathbf{P}_n]_1 (x,y,z,1)^{\mathrm{T}}}{[\mathbf{P}_n]_3 (x,y,z,1)^{\mathrm{T}}}, \quad v_n = \frac{[\mathbf{P}_n]_2 (x,y,z,1)^{\mathrm{T}}}{[\mathbf{P}_n]_3 (x,y,z,1)^{\mathrm{T}}}, \tag{14}$$

where $\mathbf{u}_n = (u_n, v_n)$ is the projected point, and $[\mathbf{P}_n]_1$, $[\mathbf{P}_n]_2$, $[\mathbf{P}_n]_3$ denote the first, second and third row of the 3×4 projection matrix, respectively.

3. Find the 2D optical flow vector, $\frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t}$, at the projected point for each camera C_n, by using the hierarchical Lucas–Kanade method [32].
4. Estimate the 3D scene flow vector, $\mathbf{w} = \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}$, from the computed 2D motion vectors, $\{\frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t}\}_{n=1}^{N_{\mathrm{vis}}}$, by solving

$$\frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t} = \frac{\partial \mathbf{u}_n}{\partial \mathbf{v}} \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}. \tag{15}$$

The Jacobian matrix $\frac{\partial \mathbf{u}_n}{\partial \mathbf{v}}$ can be computed explicitly for each n by symbolic differentiation of $\mathbf{u}_n$ 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 solved from the overdetermined system of linear equations defined by Eq. (15) via the least-squares method. We note that the scene flow vector of a vertex $\mathbf{v}$ can be estimated from its 2D motion vectors only if the vertex is visible from at least two cameras, i.e., $N_{\mathrm{vis}} \geqslant 2$. Otherwise, we set the scene flow vector to zero. We finally smoothen the estimated scene flow vectors by averaging each over their 3-link neighborhood.

We employ the scene flow vectors for two purposes: First, for pose registration to apply once at each frame transition, and second, to assist the silhouette-based deformation through displacement iterations to render the process more robust and efficient.

5.3. 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 prior to surface evolution from frame t to $t+1$. This initial transformation not only improves 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.

We estimate the pose registration parameters (rotation and translation) from the 3D scene flow vectors. 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 by $\mathbf{V}^{(t)}$ and $\widehat{\mathbf{V}}^{(t+1)}$,

$$\mathbf{V}^{(t)} = [\mathbf{v}_1^{(t)}, \mathbf{v}_2^{(t)}, \ldots, \mathbf{v}_{N_v}^{(t)}],$$
$$\widehat{\mathbf{V}}^{(t+1)} = [\hat{\mathbf{v}}_1^{(t+1)}, \hat{\mathbf{v}}_2^{(t+1)}, \ldots, \hat{\mathbf{v}}_{N_v}^{(t+1)}],$$

where N_v 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 $\widehat{\mathbf{V}}^{(t+1)}$ is given by:

$$\widehat{\mathbf{V}}^{(t+1)} = \begin{bmatrix} \mathbf{R}^{(t)} & \mathbf{t}^{(t)} \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(t)} \\ \mathbf{1}^{\mathrm{T}} \end{bmatrix}, \tag{16}$$

The parameters $\mathbf{R}^{(t)}$ and $\mathbf{t}^{(t)}$ are estimated from this equation using a nonlinear unitary-constraint optimization technique as described in [33].

5.4. Scene flow assisted deformation

Although in theory a scene flow vector gives the 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 process. 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 Eq. (7):

$$\mathbf{d}(\mathbf{v}_{i,k}) = \alpha_{i,k} \mathbf{d}_{\mathrm{sil}}(\mathbf{v}_{i,k}) + \beta_{i,k} \mathbf{d}_{\mathrm{flow}}(\mathbf{v}_{i,k}), \tag{17}$$

where $\alpha_{i,k}$ and $\beta_{i,k}$ are weighting coefficients varying with vertex i and iteration k, taking values in $[0,1]$ such that $\alpha_{i,k} + \beta_{i,k} = 1$. The silhouette-based component $\mathbf{d}_{\mathrm{sil}}(\mathbf{v}_{i,k})$ is calculated according to Eqs. (10) and (11). The scene flow based component $\mathbf{d}_{\mathrm{flow}}(\mathbf{v}_{i,k})$ is computed 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 either zero or $\varepsilon_{\min}/2$:

$$\mathbf{d}_{\mathrm{flow}}(\mathbf{v}_{i,k}) = \begin{cases} \frac{\varepsilon_{\min}}{2} & \text{if } \|\hat{\mathbf{v}}_i - \mathbf{v}_{i,k}\| \geqslant \frac{\varepsilon_{\min}}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

We note that the target point $\hat{\mathbf{v}}_i$ for each vertex remains fixed throughout the iterations for a given frame transition from t to $t+1$, whereas the weights $\alpha_{i,k}$ and $\beta_{i,k}$ in Eq. (17) 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 such that $\alpha_{i,0} = 0$ and $\beta_{i,0} = 1$. As iterations proceed, this favor is gradually carried to silhouette information so that eventually $\alpha_{i,k^*} \approx 1$ and $\beta_{i,k^*} \approx 0$. In this way, the scene flow information smoothly leads the deformation towards the target surface on a stable and short path while the final surface reconstructed is totally

determined by the silhouette information. We set these weights as a function of iteration counter k such that:

$$\alpha_{i,k} = 1 - e^{-\tau_i k}, \quad \beta_{i,k} = 1 - \alpha_{i,k}. \tag{19}$$

The choice of the time coefficient τ_i 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: Recalling 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 information is expected to be at least $\widetilde{K} = 2\|\mathbf{w}_i\|/\varepsilon_{\min}$ (see Eq. (18)). The coefficient τ_i can then be chosen such that the weights satisfy $\alpha_{i,k} = \beta_{i,k} = 0.5$ at a certain fraction of this expected iteration count, i.e., at iteration $\gamma\widetilde{K}$, $\gamma \geqslant 0$:

$$\tau_i = -\frac{\ln(0.5)\varepsilon_{\min}}{2\gamma\|\mathbf{w}_i\|}. \tag{20}$$

In our experiments we have set the parameter $\gamma = 1.5$. We have also experimented with different settings of γ as will later be presented in Section 6. We also note that, when using scene flow vectors, the fine tuning procedure described in Section 5.1 is invoked at an iteration k only when the weight $\alpha_{i,k}$ is close to 1 (e.g., $\alpha_{i,k} > 0.95$) in order to avoid unnecessary decelerations around irrelevant surface boundaries.

Integrating the scene flow information into the deformation process not only improves the stability of the tracking process 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. We illustrate the benefit of integrating 3D scene flow information in Fig. 5, where we display three possible positioning of the deformable mesh with respect to the target surface prior to mesh evolution. In case (a), the deformable mesh can smoothly evolve to the target surface using only silhouette-based displacements without any surface shrinkage. In case (b), the deformable mesh can track the surface based only on silhouette information, but by first undergoing some surface shrinkage and then remodeling the shrunken segment. In this case, shrinkage and remodeling can be avoided by incorporating 3D scene flow. In case (c), the tracking process, when based only on silhouette information, fails since the two boundaries at the mid-part of the surface segment wrongly tends to shrink onto the same location whereas the end-parts correctly evolve to two different boundaries. In this case, the shrunken part could be handled, and the tracking process could resume, by first applying a topology split operation and then a topology merging. However, if 3D scene flow is integrated to the deformation process, the deformable mesh can track the surface without need for any such redundant topological operations. These three cases will be demonstrated on several examples in the experimental results section.

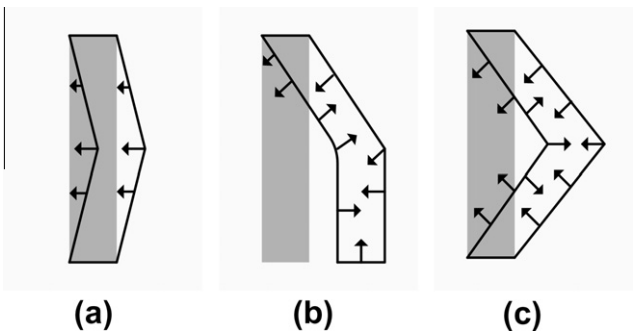

Fig. 5. Three distinct cases for positioning of the deformable mesh with respect to the target surface prior to mesh evolution. The arrows show the direction of silhouette-based mesh evolution.

5.5. Tracking algorithm

We now give the overall shape tracking algorithm which is initialized by $M^{(0)}$ representing the surface at the first frame of the video (recall that the deformable mesh at each frame t, at each iteration k of geometric evolution and at each iteration l of topological evolution is denoted by $M^{(t)}_{k,l}$):

```
Iterate on t
  Set M^(t)_{0,0} = M^(t−1);
  Extract silhouettes of frame t;
  Estimate 3D scene flow w_i^(t) for every vertex i;
  Estimate rotation matrix R^(t) and translation t^(t);
  Pose register M^(t)_{0,0} using R^(t) and t^(t);
  Iterate on l (topology)
    Activate all vertices
    Iterate on k (geometry)
      Restructure active edges in M^(t)_{k,l} by T_r;
      Displace active vertices in M^(t)_{k,l} by T_d;
      Detect and avoid collisions;
      Smooth active vertices in M^(t)_{k,l} by T_s;
      Deactivate vertices that no longer move;
    Till convergence M^(t)_{k*,l} = T(M^(t)_{k*,l}) (Eq. (3))
  Till convergence M^(t)_{k*,l*} = U(M^(t)_{k*,l*}) (Eq. (5))
  Set M^(t) = M^(t)_{k*,l*} as mesh representation of frame t;
Till end of sequence
```

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.

5.6. Representation load

The resulting mesh sequence, $M^{(0)}, M^{(1)}, \ldots, M^{(t)}, \ldots$, representing the time-varying shape, can be efficiently encoded in terms of small-scale vertex displacements and mesh operations along with the initial model and the pose registration parameters of each frame. We assume that the vertex coordinates are encoded using P-bit precision. We denote the total number of frames in the sequence by T, the number of vertices at frame t by N^t_v, the number of restructuring operations by N^t_r, the number of newly appeared vertices due to edge splits by $\widehat{N}^t_v$, and the ratio of the size of the box bounding the time-varying surface to the maximum displacement at frame t, respectively for x, y, and z directions, by s^t_x, s^t_y and s^t_z. The bit-load B for the mesh sequence can then be calculated (omitting the bit-load for the initial mesh $M^{(0)}$ and the topological operations (which are usually very few), and the parameter header for each frame) as follows:

$$B = \sum_{t=1}^{T}\left(N^t_v - \widehat{N}^t_v\right)\left(3P - \lceil\log_2 s^t_x\rceil - \lceil\log_2 s^t_y\rceil - \lceil\log_2 s^t_z\rceil\right) + 2N^t_r\lceil\log_2 N^t_v\rceil + 3\widehat{N}^t_v P, \tag{21}$$

where the first term of the summation corresponds to the bit-load of the vertex displacements, the second term to the bit-load of the restructuring operations, and the third term corresponds to encoding of the newly appeared vertices with precision P. The bit-load of the x-component for each vertex displacement at frame t is given by $P - \lceil \log_2 s_x^t \rceil$ which is usually significantly less than the required bit precision P. The same argument holds for y and z directions as well. 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 index, $2\lceil \log_2 N_v^t \rceil$, for every frame t. Note also that the number of newly appeared vertices, $\widehat{N}_v^t$, is at most equal to the number of edge split operations.

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

$$B_0 = \sum_{t=1}^{T} 3N_v^t P + 6N_v^t \lceil \log_2 N_v^t \rceil, \tag{22}$$

where the first term in the summation corresponds to the bit-load of the vertex coordinates and the second term corresponds to the bit-load of the triangles. We assume that a triangle is represented with three vertex indices and that the number of triangles is twice the number of vertices. We have compared the representation efficiency of our method with the classical vertex-triangle list and observed that it provides at least 5 times encoding efficiency without applying any statistical compression, as will be presented in the experimental results section. Recall that we encode each vertex displacement by allocating a fixed bit budget based on the maximum displacement recorded at the current frame. Hence we note that there still remains significant statistical redundancy, especially for geometry encoding which dominates the overall bitload, that could be exploited using statistical compression techniques.

6. Experimental results

We have conducted experiments to demonstrate the performance of our shape tracking method on three different sequences, one synthetic and two real sequences. The synthetic mesh sequence, *Jumping Man*, originally reconstructed from a real scene [34], exhibits the realistic motion of the jumping act of a human actor in a skin-tight suit 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 a priori, and hence we can assess the performance of our method in ideal conditions.

We have recorded two real video sequences at 30 fps by using a multicamera system equipped with 8 cameras (1332×980). We have calibrated the multicamera system by using the technique described in [35]. For silhouette extraction, we have used the method presented in [36], 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. The first real sequence is a relatively long sequence (1280 frames) with various types of actions such as standing, walking, running, jumping, turning, stretching and kick-boxing. In this sequence the human actor wears a loose clothing, hence her motion contains some small amount of non-rigidity (see Fig. 6). On the other hand, the second real sequence contains the highly non-rigid motion of an actor while taking off and putting on his hat (see Fig. 1).

We reconstruct the initial mesh $M^{(0)}$ using the silhouette-based static object reconstruction method described in [16]. This method employs a deformation scheme that is similar to the one described in this paper. The resolution of the deformable mesh model, hence the value of $\varepsilon_{\min}$, is chosen small enough to describe small shape details for accuracy but as large as possible to reduce the total vertex count for efficiency. We have experimented with different values of $\varepsilon_{\min}$ on the first frame of the Jumping Man and observed a breakpoint in the reconstruction error at $\varepsilon_{\min} = 0.025$. We have used this value, which corresponds 2.5% of the radius of the bounding sphere at the first frame, as a lower bound for reconstruction of all our sequences as they all similarly contain a human actor in the scene.

6.1. Tracking results

For each of the sequences, we consider three different tracking schemes. In the first one, the deformation is driven only by silhouette-based displacements without employing any pose registration or scene flow vectors. In the second, we additionally employ pose registration (PR) and in the third one, we consider the complete scheme, i.e., we also integrate the 3D scene flow vectors to the deformation process. We will refer to these three schemes, respec-

Fig. 6. Multiview images of the first frame of Real Sequence 1.

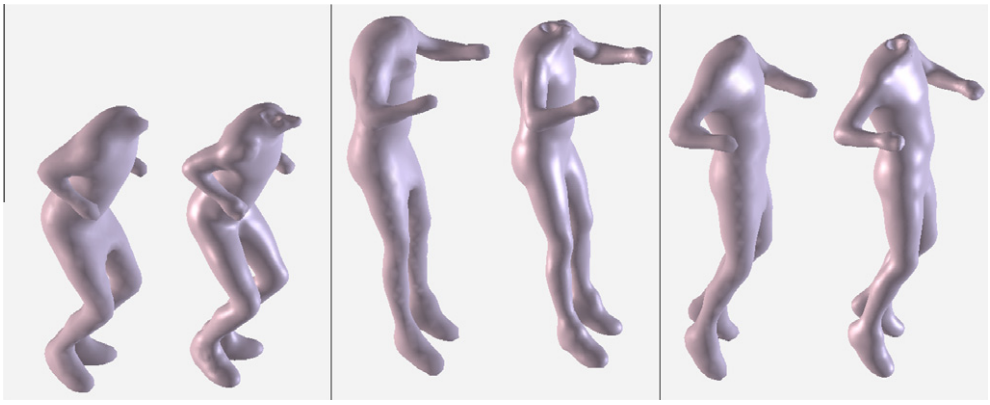

Fig. 7. Sample reconstructions from *Jumping Man* sequence (left), along with the original models (right).

tively, as (1) Silhouette-only, (2) Silhouette-PR, and (3) Silhouette-PR-3DSF.

6.1.1. Synthetic Jumping Man

In Fig. 7, we display sample frames from the mesh sequence reconstructed with $\varepsilon_{\min} = 0.025$ by employing the complete shape tracking scheme, that is, Silhouette-PR-3DSF. Although some geometric discrepancies can be observed on the reconstructed meshes as compared to the original geometry, which are mainly due to the well known limitations of shape-from-silhouette approach, the geometry is recovered from the available 16 multiview silhouettes as smoothly and as faithfully to the original as possible.

In Table 1, we provide average statistics per frame to quantitatively assess the performance of our method under three different schemes. The maximum vertex displacement magnitudes and reconstruction errors given in this table are normalized with respect to the radius of the bounding sphere. Recall that the maximum vertex displacement of the deformable mesh within a given frame transition determines how many bits would be necessary to encode a vertex displacement. Likewise, the number of restructuring operations also contributes to the representation load of each frame. We observe that adding more and more components to the base scheme improves the performance by decreasing

Table 1
Average statistics per frame for the Jumping Man.

	Sil-only	Sil-PR	Sil-PR-3DSF
Split (#)	38.6	14.2	2.8
Collapse (#)	45.2	13.0	2.2
Flip (#)	86.8	28.6	6.6
Time (s)	37.4	16.3	9.7
Iteration (#)	54.4	34.2	21.4
Max disp. ($\times 10^{-3}$)	107	63	49
Reconst. error ($\times 10^{-3}$)	4.41	4.37	4.35

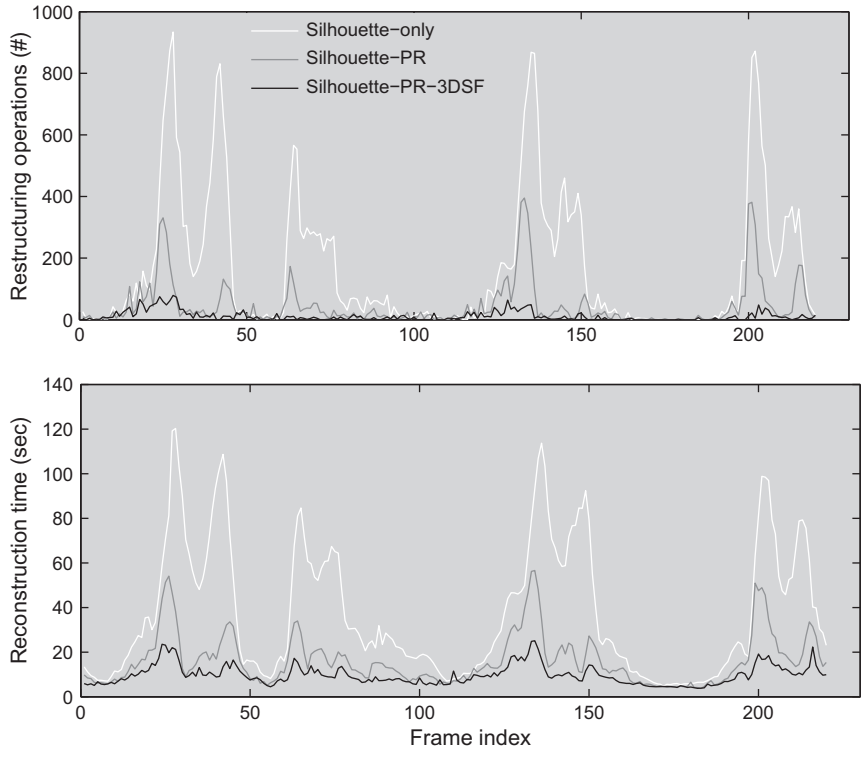

Fig. 8. Number of restructuring operations (top) and reconstruction time (bottom) for individual frames of the Jumping Man.

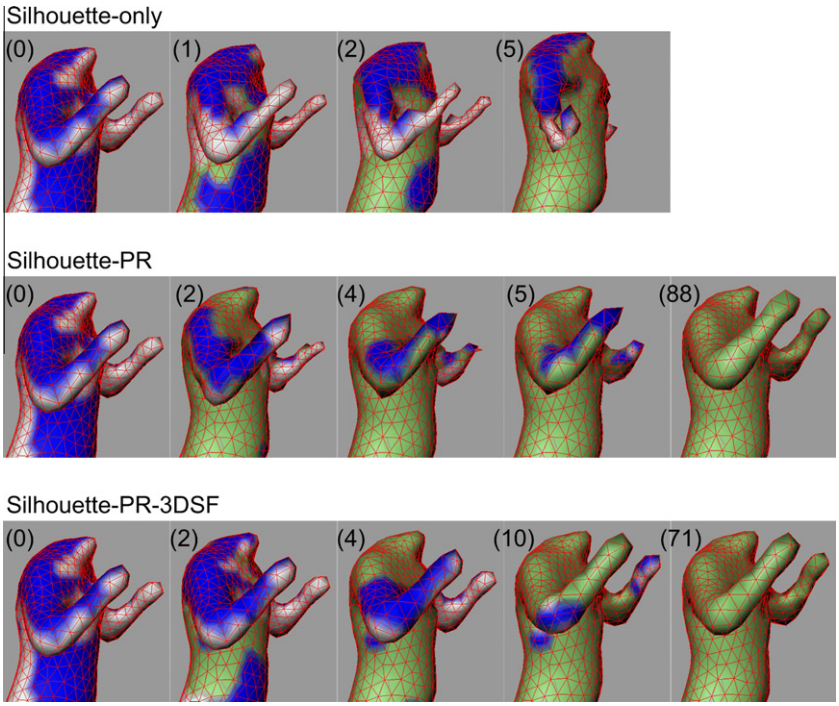

Fig. 9. Deformable mesh at various iterations for transition from frame 26 to 27. Blue (dark), white and green areas correspond to IN, OUT, and ON vertices, respectively.

both reconstruction time and representation load (the execution times given have been measured on a 2.2 GHz AMD Athlon 4200 + dual core processor). 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, not the reconstruction quality. We finally note that the tracking process fails several times over the whole sequence when Silhouette-only or Silhouette-PR scheme is in use, whereas we have successfully tracked the whole time-varying geometry with Silhouette-PR-3DSF. For the frames of failure, we have resumed tracking by reconstructing the model from scratch and assumed that the statistics of each such frame is the same as the previous frame (note that in our experiments we have not incorporated any topological operations that would handle the case (c) of Fig. 5). In Fig. 8, we plot the total number of restructuring operations and reconstruction time for individual frames. As expected, we observe increases in the operation count, which implies more changes in mesh connectivity, as well as in reconstruction time, along the frames where the motion of the object is faster. The four global peaks observed on the operation count and reconstruction time plots correspond to the four jumping acts (two times forward and backward) in the sequence. The local maximum within each of these peaks is mainly due to the fast local motion of the arms while jumping. Note also that no topological operation is needed for reconstruction of this sequence since the Jumping Man contains no self-collision.

In Fig. 9, we display the deformable mesh at various iterations within a frame transition for three different tracking schemes. With the Silhouette-only scheme, the deformable mesh cannot converge to the desired surface during frame transition, and hence the tracking process fails (due to case (c) in Fig. 5). When the Silhouette-PR scheme is employed, pose registration carries some surface segments (e.g., arms) of the object inside the target surface at the beginning of the deformation, hence surface tracking becomes possible (case (b) in Fig. 5). However we observe severe shrinking and re-modeling of the arms. When 3D scene flow vectors are incorporated to deformation, the deformable mesh evolves in a much smoother path towards the target surface with almost no shrinkage and also with less number of mesh restructuring operations.

Table 2
Average statistics at low and high resolutions for the Jumping Man.

$\varepsilon_{\min}$	0.025	0.015
Triangle (#)	2790	8454
Operation (#)	11.6	72.1
Iteration (#)	21.4	48.1
Recons. time (s)	9.7	54.7

We have tested the performance of our shape tracking method also at a higher resolution ($\varepsilon_{\min} = 0.025$ vs. $\varepsilon_{\min} = 0.015$). The average statistics per frame in terms of triangle number, operation count, iteration number and reconstruction time, are given in Table 2. We observe that the efficiency decreases as the value of $\varepsilon_{\min}$ decreases while the quality of the reconstruction does not significantly improve. Note also that the computation time mainly depends on the size of the mesh and the number of iterations. While the dependence of the overall algorithmic complexity on the mesh size is $O(N_v \log N_v)$ as dictated by the collision detection algorithm, the dependence on the number of iterations is lower than linear complexity since as iterations proceed and as more and more vertices become inactive, the time spent at each iteration significantly reduces.

6.1.2. Real Sequence 1

In Fig. 10, we display sample frames from the mesh sequence reconstructed with $\varepsilon_{\min} = 0.025$ along with sample images from the eight available silhouettes. In Table 3 we provide average statistics per frame to quantitatively assess the performance of our method for three different schemes with different settings of the parameter γ. Recall from Section 5.4 that the parameter γ determines how long the scene flow information will be effective on the deformation process. The reconstruction times given in the

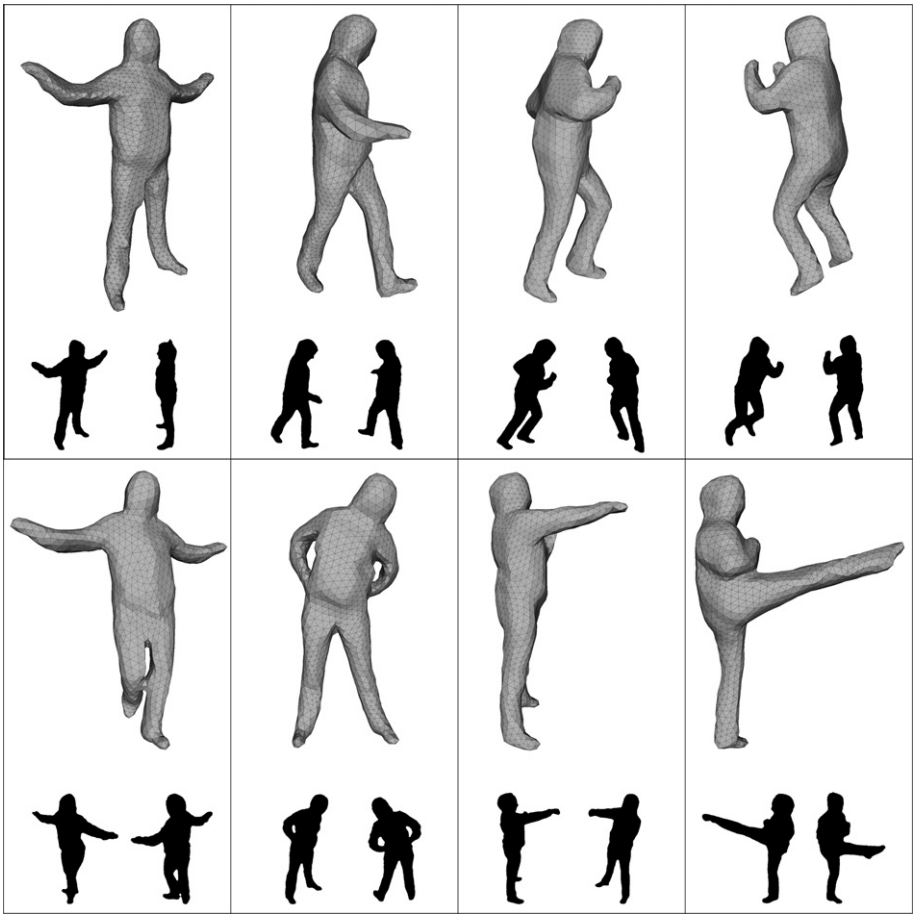

Fig. 10. Samples from the reconstructed mesh sequence, one for each type of action, displayed together with two of the corresponding silhouette images from Real Sequence 1.

Table 3
Average statistics per frame for Real Sequence 1.

	Sil-only	Sil-PR	Sil-PR-3DSF			
			$\gamma = 0.5$	$\gamma = 1.0$	$\gamma = 1.5$	$\gamma = 2.0$
Split (#)	41.5	20.5	16.7	16.5	16.8	17.3
Collapse (#)	49.1	23.8	18.1	17.2	17.4	18.0
Flip (#)	104.4	53.4	46.2	46.3	47.9	49.9
Time (s)	32.9	21.1	12.7	15.8	19.1	21.7
Iteration (#)	69.2	54.9	30.1	41.0	53.0	62.8
Max disp. ($\times 10^{-3}$)	111	81	76	77	78	78

table do not include the time spent for scene flow estimation which takes about 4.5 s/frame. We have been able to successfully track the time-varying geometry using the Silhouette-PR-3DSF scheme, whereas shape tracking has failed on several occasions when Silhouette-only or Silhouette-PR schemes is in use, especially at certain frames that exhibit We observe from Table 3 that adding more and more components to the base scheme improves the efficiency, though the 3D scene flow vectors, when read from the ground-truth, were more beneficial in the synthetic case.

Fig. 11 plots the number of restructuring operations and reconstruction time for individual frames. We observe that 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 schemes. The oscillations observed throughout "walking" and "running" frames are mainly due to the fast motion of the body parts while taking a step (rise) and relatively slower motion while having the feet on the ground (fall). Note also that the frequency and the intensity of the oscillations at "running" are almost twice the oscillations at "walking" as expected. "Jumping" is the action with the strongest global translation while "turning" is the action with the strongest global rotation. Hence, the benefit of pose registration is observed to be the highest for these two actions. The local motion is the strongest in "kick-boxing" (with two punches and two kicks). Therefore the benefit of scene flow assistance is observed to be the highest in this action. In Fig. 12, we display the deformable mesh at various iterations within a frame transition for the Silhouette-PR and Silhouette-PR-3DSF schemes using two different settings of γ. We clearly observe the benefit of scene flow integration which smoothly guides the leg towards its target, especially with the setting $\gamma = 1.5$, i.e., when the scene flow remains effective on the deformation process for a sufficiently long duration. As observed from Table 3, the number of iterations and the convergence time per frame increase only slightly as the value of γ increases. We also note that a total of 13 topological split and 13 merge operations are needed for reconstruction of the sequence to handle surface contacts which are mostly due to the collisions between the legs as well as those between the arms and the torso.

In Fig. 13, we visualize the estimated 3D scene flow vectors on sample reconstructions. The time share of different tasks in scene flow computation is approximately 0.4, 3.1 and 1.0 s, respectively for building the 3D voxel grid and visibility, calculating the 2D optical flows, and estimating the 3D scene flow. To compute the 2D optical flow vectors, we have used a Lucas–Kanade implementation of three-level hierarchy and a window size of 30×30 pixels.

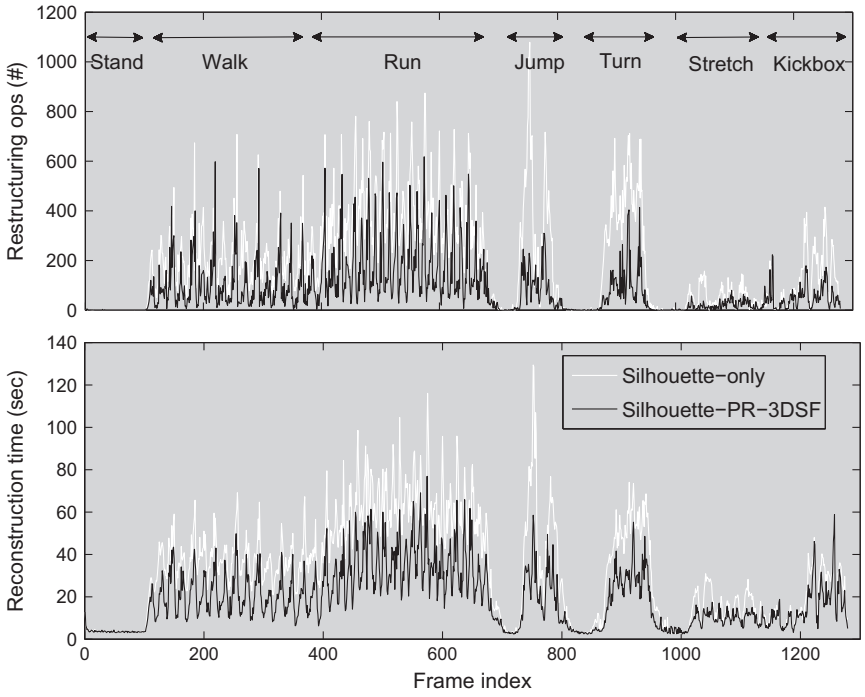

Fig. 11. Number of restructuring operations (top) and reconstruction time (bottom) for individual frames of Real Sequence 1.

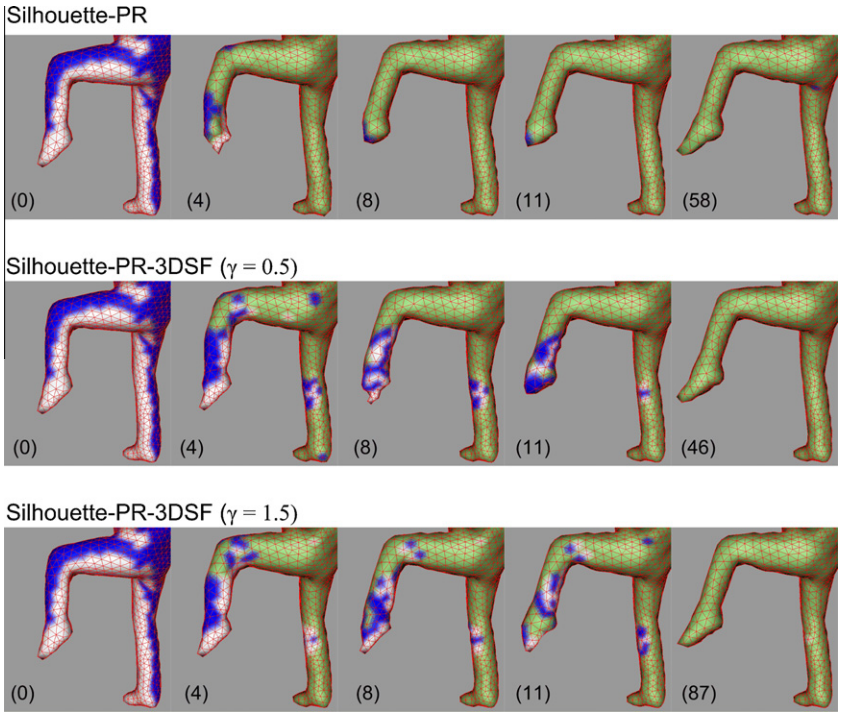

Fig. 12. Zoom on the deformable mesh at various iterations for transition from frame 1255 to frame 1256.

6.1.3. Real Sequence 2

Our third experiment aims to track the shape of an actor wearing a hat so as to challenge our shape tracking scheme in a video sequence with severe non-rigid motion that cannot be handled using fixed-connectivity methods (see Fig. 1). The particularity of this video sequence is that the topology of the shape changes due to the motion of the hat which moves between the actor's hand and his head, also yielding covered and uncovered surface parts. By utilizing two topological split and two merge operations in total along with a number of restructuring operations, we have successfully tracked a multiview video sequence of 165 frames, which includes taking off the hat and then putting on it back, starting from an initial reconstruction and using the Silhouette-PR-3DSF scheme with $\varepsilon_{\min} = 0.018$. Fig. 14 displays the reconstruction process at two different instances to demonstrate how topology operators are incorporated into the deformation scheme. In this experiment, the reconstruction of the mesh sequence, which contains approximately 7.1 K triangles on average, has required about 50 restructuring operations and 32.7 s of computation time per frame.

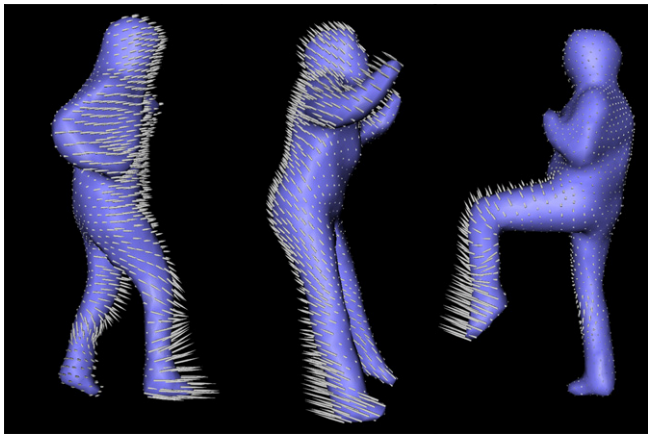

Fig. 13. Estimated 3D scene flow vectors displayed on sample frames.

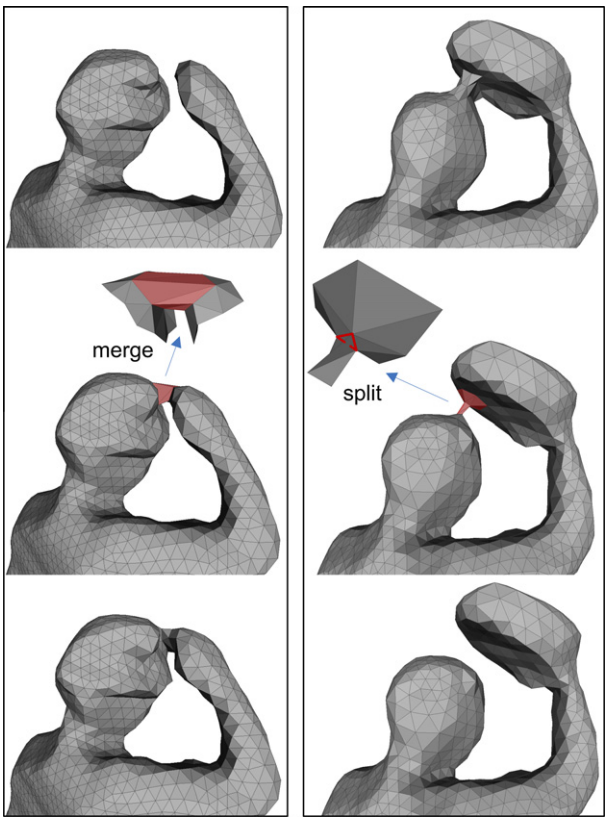

Fig. 14. (Left column, top) The initial mesh, zoomed on upper body, for transition from frame 52 to 53, (middle) topology merging at the convergence of the first mesh evolution process by creating a surface tunnel (marked in red) between the colliding surface parts, and (bottom) the final mesh at the convergence of the next mesh evolution. (Right column, top) The initial mesh for transition from frame 86 to 87, (middle) topology is split at the marked location at the convergence of the first mesh evolution process, and (bottom) the final mesh at the convergence of the next mesh evolution. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.2. Representation efficiency

Table 4 provides the bit-load B of each sequence when encoded with the encoding scheme described in Section 5.6 as compared to the bit-load B_0 when encoded with the standard vertex-triangle list approach, both using 12-bit geometric precision. The results show that the representation load efficiency of the former strategy is at least 5 times better than the classical approach. We also observe that the contribution of restructuring operations to the total bit-load (including the cost of the newly appeared vertices), that we denote by B_{op}, is only marginal. We note that the storage costs of the initial mesh representation, the parameter header and the topological operations, which are all negligible, have not been included in the bit-loads given in the table.

Table 4
Bit-loads (in megabytes) for the three sequences.

	Frames (#)	ε_{min}	B_0	B	B_{op}
Jumping Man	220	0.025	3.73	0.56	0.009
Real Sequence 1	1280	0.025	24.01	4.19	0.38
Real Sequence 2	165	0.018	7.54	1.35	0.029

7. Conclusion

We have presented a deformation-based technique to track and reconstruct the time-varying shape of a dynamic object from its multiview images. Our findings can be summarized as follows:

- The time-varying shape can successfully be tracked from the multiview images of a moving object using mesh deformation. This is mostly thanks to the robustness of the mesh evolution process coupled with restructuring and topological operations as well as an efficient collision detection method.
- Since the mesh representation at each frame is reconstructed by evolving the mesh obtained at the previous frame, the overall reconstruction of the time-varying geometry is obtained in a fast manner.
- The time-varying shape can be encoded in terms of restructuring operations and small-scale vertex displacements possibly along with a very few number of topological operations, hence the resulting representation is space efficient.
- Both the connectivity and the geometry of the object can be tracked, hence our method is topology-adaptive and applicable to objects exhibiting non-rigid motion.
- The resulting mesh representation is as smooth as possible with the available data, both in time and space.
- Since our method does not assume any prior object model, it can be applied to any shape, that however makes in turn the quality of the reconstruction heavily dependent on the quality of the extracted silhouettes.

The main limitations of the presented method are that the resulting surface representations lack the ability to model hidden cavities and that the quality of the reconstructions is restricted with 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 video acquisition. 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.

Acknowledgment

This work has been supported by TUBITAK under the Project EEEAG-105E143.

Appendix A. Supplementary material

Shape tracking animation videos. The animation videos of the reconstructions of all three sequences can be accessed from the public web site http://mvgl.ku.edu.tr/shapetracking/, and also as supplementary content through the publisher's web site. The videos include wireframe animations of the reconstructed mesh sequences for Jumping Man, Real Sequence 1 and Real Sequence 2, as well as the shaded version of the Real Sequence 1 reconstruction, each with a view from the original sequence displayed at the upper-left corner. While the minimum edge length parameter $\varepsilon_{\min}$ used for reconstruction of Jumping Man and Real Sequence 1 is 0.025, this value is 0.018 for Real Sequence 2. For all cases, the Silhouette-PR-3DSF scheme has been used and the parameters are set to be as $\kappa = 3$ and $\gamma = 1.5$. All videos are downsampled to 15 fps. Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.cviu.2012.07.001.

References

[1] A. Alatan, Y. Yemez, U. Gudukbay, X. Zabulis, K. Muller, Ç.E. Erdem, C. Weigel, A. Smolic, Scene representation technologies for 3DTV – a survey, IEEE Trans. Circuits Syst. Video Technol. 17 (11) (2007) 1587–1605.

[2] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G.B. Akar, G.A. Triantafyllidis, A. Koz, Coding algorithms for 3DTV – a survey, IEEE Trans. Circuits Syst. Video Technol. 17 (11) (2007) 1606–1621.

[3] T.B. Moeslund, A. Hilton, V. Krüger, A survey of advances in vision-based human motion capture and analysis, Comput. Vision Image Understand. 104 (2) (2006) 90–126.

[4] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, S. Thrun, Performance capture from sparse multi-view video, in: Proc. SIGGRAPH, 2008..

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

[6] J. Starck, A. Hilton, Surface capture for performance-based animation, IEEE Comput. Graph. Appl. 27 (3) (2007) 21–31.

[7] T. Matsuyama, X. Wu, T. Takai, S. Nobuhara, Real-time 3D shape reconstruction, dynamic 3D mesh deformation, and high fidelity visualization for 3D video, Comput. Vis. Image Understand. 96 (3) (2004) 393–434.

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

[9] S. Würmlin, E. Lamboray, O.G. Staadt, M.H. Gross, 3D video recorder: a system for recording and playing free-viewpoint video, Comput. Graph. Forum 22 (2) (2003) 181–193.

[10] N. Ahmed, C. Theobalt, C. Roessl, S. Thrun, H.-P. Seidel, Dense correspondence finding for parametrization-free animation reconstruction from video, in: Proc. Computer Vision and Pattern Recognition (CVPR), 2008.

[11] D. Vlasic, I. Baran, W. Matusiky, J. Popović, Articulated mesh animation from multi-view silhouettes, ACM Trans. Graph. 27 (3) (2008) 1–9.

[12] C. Cagniart, E. Boyer, S. Ilic, Probabilistic deformable surface tracking from multiple videos, in: Proc. European Conference on Computer Vision (ECCV), 2010, pp. 326–339.

[13] C. Wu, K. Varanasi, Y. Liu, H.-P. Seidel, C. Theobalt, Shading-based dynamic shape refinement from multi-view video under general illumination, in: IEEE Int. Conf. on Computer Vision (ICCV), 2011.

[14] B. Curless, Overview of active vision techniques, in: Proc. SIGGRAPH Course on 3D Photography, 1999.

[15] G. Pons-Moll, L. Leal-Taix, T. Truong, B. Rosenhahn, Efficient and robust shape matching for model based human motion capture, in: Proc. of the 33rd Int. Conf. on Pattern Recognition (DAGM), 2011.

[16] Y. Yemez, Y. Sahillioğlu, Shape from silhouette using topology-adaptive mesh deformation, Pattern Recognit. Lett. 30 (2009) 1198–1207.

[17] O.H. Holt, S. Rusinkiewicz, Stripe boundary codes for real-time structured-light range scanning of moving objects, in: Int. Conf. on Computer Vision (ICCV), 2001, pp. 359–366.

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

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

[20] C.H. Esteban, F. Schmitt, Silhouette and stereo fusion for 3D object modeling, Comput. Vis. Image Understand. 96 (3) (2004) 367–392.

[21] A. Zaharescu, E. Boyer, R. Horaud, Transformesh: a topology-adaptive mesh based method for surface reconstruction, in: Proc. Asian Conference on Computer Vision (ACCV), 2007, pp. 166–175.

[22] K.M. Cheung, S. Baker, T. Kanade, Shape-from-silhouette across time part I: theory and algorithms, Int. J. Comput. Vis. 63 (3) (2004) 221–247.

[23] S.C. Bilir, Y. Yemez, Time varying surface reconstruction from multiview video, in: Int. Conf. on Shape Modeling and Applications (SMI), 2008, pp. 47–51.

[24] M. Botsch, O. Sorkine, On linear variational surface deformation methods, IEEE Trans. Visual. Comput. Graph. 14 (1) (2008) 213–230.

[25] M.A. Magnor, B. Goldlücke, Spacetime-coherent geometry reconstruction from multiple video streams, in: Int. Symp. 3DPVT, 2004, pp. 365–372..

[26] J.-O. Lachaud, B. Taton, Deformable model with adaptive mesh and automated topology changes, in: Int. Conf. on 3D Digital Imaging and Modeling (3DIM), 2003, pp. 12–19.

[27] M. Kass, A. Witkin, D. Terzopoulos, Snakes: active contour models, Int. J. Comput. Vis. 1 (4) (1988) 321–332.

[28] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Mesh optimization, in: Proc. SIGGRAPH, 1993, pp. 19–26.

[29] L.P. Kobbelt, T. Bareuther, H.-P. Seidel, Multiresolution shape deformations for meshes with dynamic vertex connectivity, in: Proc. Eurographics, vol. 19, 2000.

[30] Z.J. Wood, P. Schröder, D. Breen, M. Desbrun, Semi-regular mesh extraction from volumes, in: Proc. Visualization, 2000, pp. 275–282.

[31] S. Vedula, S. Baker, P. Rander, R.T. Collins, T. Kanade, Three-dimensional scene flow, IEEE Trans. PAMI 27 (3) (2005) 475–480.

[32] B.D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: Int. Joint Conference on Artificial Intelligence, 1981, pp. 674–679.

[33] M.E. Sargın, Y. Yemez, E. Erzin, A.M. Tekalp, Analysis of head gesture and prosody patterns for prosody-driven head-gesture animation, IEEE Trans. PAMI 30 (8) (2008) 1330–1345.

[34] P. Sand, L. McMillan, J. Popović, Continuous capture of skin deformation, in: Int. Conf. on Computer Graphics and Interactive Techniques, 2003.

[35] T. Svoboda, D. Martinec, T. Pajdla, A convenient multi-camera self-calibration for virtual environments, PRESENCE: Teleoperators Virtual Environ. 14 (4) (2005) 407–422.

[36] T. Horprasert, D. Harwood, L.S. Davis, A robust background subtraction and shadow detection, in: Proc. Asian Conference on Computer Vision (ACCV), 2000.