Eurographics Symposium on Geometry Processing 2008
Pierre Alliez and Szymon Rusinkiewicz
(Guest Editors)

Volume 27 (2008), Number 5

Reconstructing Animated Meshes from Time-Varying Point Clouds

Jochen Süßmuth and Marco Winter and Günther Greiner

Computer Graphics Group, University Erlangen-Nuremberg, Germany

Abstract

In this paper, we describe a novel approach for the reconstruction of animated meshes from a series of time-deforming point clouds. Given a set of unordered point clouds that have been captured by a fast 3-D scanner, our algorithm is able to compute coherent meshes which approximate the input data at arbitrary time instances.
Our method is based on the computation of an implicit function in $\mathbb{R}^4$ that approximates the time-space surface of the time-varying point cloud. We then use the four-dimensional implicit function to reconstruct a polygonal model for the first time-step. By sliding this template mesh along the time-space surface in an as-rigid-as-possible manner, we obtain reconstructions for further time-steps which have the same connectivity as the previously extracted mesh while recovering rigid motion exactly.
The resulting animated meshes allow accurate motion tracking of arbitrary points and are well suited for animation compression. We demonstrate the qualities of the proposed method by applying it to several data sets acquired by real-time 3-D scanners.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

Surface reconstruction from unordered point clouds has been one of the most intensely researched problems in computational geometry and computer graphics. An important field of application for surface reconstruction is the generation of digital 3-D models for movies and computer games which have been scanned from real-life objects. 3-D scanning devices typically output a dense set of unorganized points, each sampled from the recorded object, which are corrupted by measurement noise and contain holes in areas that were occluded during the scanning process. The reconstruction of a polygonal model of the acquired object that approaches the sampled points is usually referred to as surface reconstruction.

Recent advances in 3-D scanning hardware, such as active stereo- and fast structured light scanners [WLG07, RHHL02, ZH06] or time-of-flight (*TOF*) cameras [LS01], allow to scan three-dimensional objects at interactive frame rates. Using one of the aforementioned devices, it is now possible to capture high quality 3-D models and their animation at once.

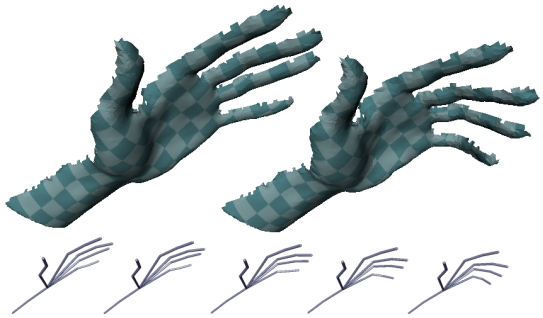

Figure 1: *Animated mesh and virtual motion tracking.*

A common approach to handle such scanner data is to reconstruct a 3-D model for each of the acquired time-frames independently. However, when handling all time-frames separately, data from preceding and succeeding frames, which contains additional information that can be used to close holes, to detect outliers and to increase the signal-to-noise

ratio, is disregarded. Moreover, the so obtained meshes are independent of each other, which means that further data processing (e.g. non-rigid registration) is required in order to obtain correspondences between the single meshes. Such correspondences are important for spatio-temporal post processing and texturing. For these reasons, it is highly preferable to find a consistent representation which contains both the 3-D geometry and the deformation that the model is undergoing.

A coherent representation for deforming geometry are animated meshes, i.e. sequences of meshes with a fixed connectivity whose vertex positions change over time. Animated meshes are widely used in computer graphics for modelling dynamic scene geometry. Since all meshes of an animation share the same connectivity and do only differ in their geometric embedding, which is given by the mesh's vertices, they can be used for coherent texturing and spatio-temporal shape manipulation. By simply tracking the vertex positions, trajectories can be computed which allow the analysis of the objects animation. Moreover, animated meshes serve as input data for many graphics algorithms such as animation compression [ASS06], deformation transfer [SP04] and object analysis (e.g. skeletonization [SY07]). It therefore is desirable to extract such animated meshes directly from a sequence of captured 3-D point clouds.

The central contribution of this paper is a new method for the completely automatic reconstruction of spatial and temporal coherent animated meshes from raw real-time high-resolution scanner data. The animated meshes computed by our method are well suited as input data for a wide range of mesh processing techniques, including the examples listed above (cf. Figure 1).

1.1. Related Work

In the past, most of the work on surface reconstruction has focused on the generation of one three-dimensional model from a single 3-D point cloud, and very little work has been done in directly reconstructing coherent dynamic geometry from time-deforming point clouds captured by real-time 3-D scanning techniques. A widely used approach to produce coherent animated meshes from a sequence of measured point clouds is reconstructing each point cloud independently and then using a template fitting approach [KHYS02, SKR*06, ACP02, SP04] to obtain a consistent meshing for all time-frames. These methods work well in practice, yet they are usually based on additional markers or landmarks which have to be specified by the user.

The first approach to directly reconstructing coherent deforming mesh models from a sequence of recorded point clouds was proposed by Shinya in [Shi04]. Shinya's method is similar to ours concerning the computation of a template mesh for the first time-frame which is then fitted to the remaining time-frames. However, the mesh fitting in [Shi04] is based on the optimization of an energy functional and requires the estimation of corresponding points between two different poses.

Wand et al. [WJH*07] address the problem of reconstructing point clouds in a spatio-temporal setting and propose a method which is based on statistical optimization. Therefore, they represent the input point clouds as a graph of oriented particles that move over time. Surface- and temporal smoothness is achieved by penalizing non-rigid movement and spatial discontinuities. Using a numerical optimization method, they find the most likely 4-D shape (which is a set of trajectories) for the input point clouds. Finally, meshes are extracted for every time-step by using a simple three-dimensional reconstruction method which uses the traced surfels as input. Their algorithm is capable of creating sets of polygonal meshes with dense correspondences. However, the high computational costs for the statistical optimization restrict the method to handle only rather small data sets.

Recently, Ahmed et al. [ATR*08] presented an algorithm for reconstructing coherent deforming mesh models directly from multi-view video data. Therefore, they utilize texture data acquired by the video cameras to find salient features which establish a sparse set of correspondences between different time-steps. The features are then used to compute a dense set of correspondences which are in turn used to fit a template mesh to the varying poses. Their method works well for multi-view video, though it is limited to the specific configuration that reliable texture information is available.

1.2. Overview

Our algorithm consists of four steps which will be detailed in the next sections. In a preprocessing step, we use the input data to construct a four-dimensional point cloud $\mathcal{P}$. Then, we estimate an implicit function $f : \mathbb{R}^4 \to \mathbb{R}$ which approximates the point cloud $\mathcal{P}$. We then compute a template mesh by extracting the zero-set of the implicit function f at the initial time-step. This template mesh is then moved along the time-space surface $f = 0$ in an as-rigid-as-possible manner [SA07] yielding the final deforming mesh model. The whole algorithm is depicted in Figure 2 using a 2-D example data set.

2. Preprocessing

The input data to our algorithm is a set of consecutive point clouds with normals, like they are provided by most real-time 3-D scanners. If the scanner does only generate an unstructured set of points for each time-frame without providing any surface normals, we construct these normals on-the-fly for each point cloud. Since we are dealing with range-scan data, a triangulation of a single time-frame can be easily obtained by constructing a Delaunay-triangulation of the points after they have been projected onto a plane perpendicular to the viewing direction of the scanner (usually the

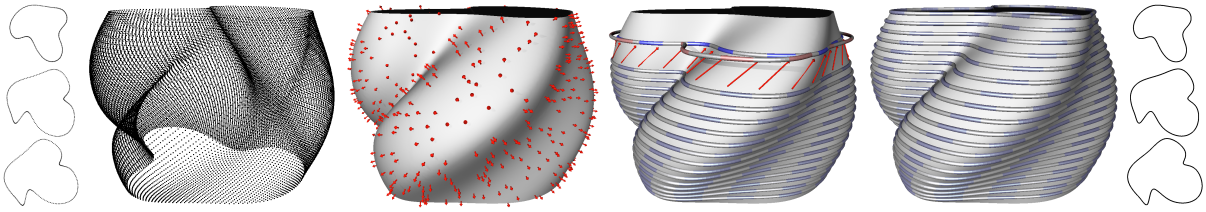

Figure 2: *Reconstructing time-deforming 2-D curves. From left to right: single 2-D point clouds; time-space point cloud; reconstructed time-space surface with some normals; intermediate step; reconstructed curves in 3-D; reconstructed curves.*

z-axis). By averaging the normals of the triangles adjacent to a vertex $\mathbf{v}_i \in \mathbb{R}^3$ we obtain consistently oriented normals $\mathbf{n}_i \in \mathbb{R}^3$ for all vertices of the inputted mesh sequence.

We then construct a time-space point cloud with normals $\mathcal{P}$ by following the approach of Mitra et al. in [MFO*07]: For each three-dimensional vertex $\mathbf{v}_i = (x_i, y_i, z_i)^T$ we create a corresponding four-dimensional vertex $\tilde{\mathbf{v}}_i = (x_i, y_i, z_i, w)^T \in \mathbb{R}^4$ by adding an additional coordinate $w = \delta \cdot t$. Here, t is the index of the time-frame at which the vertex $\mathbf{v}_i$ was captured and δ is a scaling factor. Since we want $\mathcal{P}$ to be sampled uniformly in space and time, we compute a triangulation of the first time-frame and choose δ to be the median of all edge lengths in this triangulation. As time-space normals $\{\tilde{\mathbf{n}}\} \in \mathbb{R}^4$ we use the original normals $\{\mathbf{n}\} \in \mathbb{R}^3$ and add 0 as fourth coordinate, i.e. $\tilde{\mathbf{n}}_i = (\mathbf{n}_i, 0)^T$. Although these normals are only a rough approximation to the real normals of the kinematic surface described by $\mathcal{P}$, they are adequate for our purpose. The subsequent computation of the implicit function f will use the normals only to define inside and outside.

The time-space point cloud $\mathcal{P}$ is now defined as $\mathcal{P} = \{(\tilde{\mathbf{v}}_i, \tilde{\mathbf{n}}_i); \tilde{\mathbf{v}}_i, \tilde{\mathbf{n}}_i \in \mathbb{R}^4\}$. For the remainder of this paper, we will use a tilde "˜" to distinguish four-dimensional points from three-dimensional points, i.e. $\mathbf{b} \in \mathbb{R}^3$ and $\tilde{\mathbf{b}} \in \mathbb{R}^4$.

3. Four-Dimensional Surface Approximation

For reconstructing the implicit function f that approximates the time-space surface defined by the time-varying point cloud $\mathcal{P}$, we mainly follow the *Multi-level Partition of Unity (MPU) Implicits* approach from Ohtake et al. [OBA*03]. Their approach operates in $\mathbb{R}^3$ and uses an octree containing local quadrics at its leaf nodes that approximate the parts of the surface in the node's proximity. An approximation to the whole surface is obtained by using weighting functions that blend together the local implicit functions. The octree is built on-the-fly during evaluation, thus making it highly efficient when used together with optimized polygonization or ray tracing methods.

In order to respect the additional dimension of time, we developed an MPU implementation that operates in $\mathbb{R}^4$ and works on an entire four-dimensional point cloud. Therefore, all points $(\tilde{\mathbf{v}}_i, \tilde{\mathbf{n}}_i) \in \mathcal{P}$ are stored in a hierarchical 4-D structure called a hextree [YS83], and are used to compute local shape functions for the approximation to the time-space surface.

Analogous to the original method, our approach uses different shape functions for local surface reconstruction:

- generic 4-D quadrics
- low-variate (3-D) quadrics in local coordinates
- clusters of low-variate quadrics that approximate edges and corners

The first two types of quadrics are implemented straightforward using the same computational steps as described in the original approach. However, for fitting local edges and corners, we use a more generalized approach in order to detect these local features. Therefore, we implemented an algorithm similar to *k*-means clustering or the *LBG* vector quantization algorithm [GG91]. This algorithm tries to divide the local set of points into k clusters of points with similar normals. The computation domain of the algorithm is the "normal ball" of the local points, i.e. all clusters and points actually are normals. The algorithm starts with one cluster center $\tilde{\mathbf{c}}_0$ and works as follows:

* set new center $\tilde{\mathbf{c}}_i$ into cluster with largest normal variance
* do clustering with all current cluster centers:
  * for each normal $\tilde{\mathbf{n}}_j$, find cluster center $\tilde{\mathbf{c}}_i$ where $\tilde{\mathbf{n}}_j \cdot \tilde{\mathbf{c}}_i = \max$ and assign $\tilde{\mathbf{n}}_j$ to cluster C_i
  * recompute cluster centers $\tilde{\mathbf{c}}_i$:
    $$\tilde{\mathbf{c}}_i' = \sum_{j \in C_i} \tilde{\mathbf{n}}_j, \qquad \tilde{\mathbf{c}}_i = \frac{\tilde{\mathbf{c}}_i'}{\|\tilde{\mathbf{c}}_i'\|}$$
  * repeat until movement of cluster centers falls below a certain threshold ε_m
* stop if we have enough clusters, otherwise add new cluster center and repeat loop

Finding the optimal number of clusters k for partitioning the normals is a problem which cannot be solved generally. In our implementation, we stop clustering if the sum of variances of all clusters found so far falls below a certain threshold:

$$\sum_{i=1}^{k} \frac{1}{\kappa_i} \sum_{j=1}^{\kappa_i} \|\tilde{\mathbf{c}}_i - \tilde{\mathbf{n}}_j\|^2 < \varepsilon_c \quad ,$$

where κ_i is the number of normals in cluster C_i. In our implementation, we set $\varepsilon_c = 0.1$. Once we found a suitable clustering of the local points, reconstruction of the local feature

is achieved by creating a low-variate quadric for each cluster and combining them using Boolean operations [OBA*03].

Using the MPU implementation described above, we are able to reconstruct the implicit function $f : \mathbb{R}^4 \rightarrow \mathbb{R}$ whose zero-set represents the desired time-space surface. Results of our 4-D MPU reconstruction are shown in the accompanying video.

4. Extracting the Template Mesh

In order to compute an initial template mesh which can then be propagated over time, we sample the implicit approximation f to the time-space surface at the initial time-step on a regular grid into a 3-D volume. We then use a modified marching cubes [LC87] algorithm to extract a template mesh $\mathcal{M}^0$ which, in accordance to previous literature [WJH*07], we will call the *urshape*. Since the main application of our method is the reconstruction of geometry from real-time range scans - which are surfaces with boundaries - we need to care about these boundaries when extracting the urshape. To account for this fact we modified the marching cubes algorithm such that the iso-surface extraction is restricted to cells which are close to the point cloud. Simply speaking, we do only process volume cells which have at least one point of the input point cloud within their circumsphere.

Triangular meshes produced by the marching cubes algorithm suffer from badly shaped triangles which makes them not well suited for direct visualization or subsequent mesh processing. By employing simple Laplacian mesh smoothing we obtain more homogeneous triangles. To correct the effect of the smoothing in surface normal direction (surface shrinking) we project the vertices of the smoothed urshape back onto the zero-set of the implicit function f as explained in Section 5.1.

5. Propagating the Template Mesh Along the Time Axis

To obtain a coherent animated mesh for the entire captured sequence, we slide the template mesh $\mathcal{M}^0$ along the four-dimensional time-space surface defined by $f = 0$ while trying to preserve the mesh's intrinsic characteristics. Our algorithm for propagating the urshape model is a generalization of *as-rigid-as-possible surface modelling* by Sorkine et al. [SA07]. We therefore start with a brief summary of their work. Details of our implementation will then be discussed in the following sections.

The problem statement in [SA07] is as follows: Given a triangular mesh $\mathcal{S}$ with n vertices $\{\mathbf{p}\}$ and m triangles, find new vertex positions $\{\mathbf{p}'\}$ for the mesh $\mathcal{S}'$ so that they meet a certain boundary condition (which in [SA07] is that the new positions of some vertices $\mathbf{p}'_j$ are fixed) such that the mesh is locally deformed as rigid as possible. Local rigidity is thereby defined on a per cell $\mathcal{C}_i$ (namely the triangle fan of a vertex $\mathbf{p}_i$) basis. Thus, they optimize for the global deformation which minimizes the deviation of all (unknown) cell deformations $\mathbf{R}_i$ from being rigid. Sorkine et al. propose the following energy functional:

$$E(\{\mathbf{p}'\},\{\mathbf{R}\}) = \sum_{i=1}^{n} \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}'_i - \mathbf{p}'_j) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j) \right\|^2, \tag{1}$$

where w_{ij} are the well known cotangent weights [PP93] of the edges connecting the vertices $\mathbf{p}_i$ and $\mathbf{p}_j$. The energy functional $E(\{\mathbf{p}'\},\{\mathbf{R}\})$ is non-linear and depends on two sets of variables. Nevertheless, the minimum of $E(\{\mathbf{p}'\},\{\mathbf{R}\})$ can be found using a simple iterative strategy: First, an initial guess for the deformed mesh vertices $\{\mathbf{p}'\}$ is estimated. This guess is then used to minimize E for the optimal rotations $\{\mathbf{R}\}$ for fixed $\{\mathbf{p}'\}$. Then, for the now given set of fixed rotations $\{\mathbf{R}\}$ they find the vertex positions $\{\mathbf{p}'\}$ which minimize E. These steps are repeated until convergence.

Before going on, let us fix some notation. Lower indices refer to a particular vertex and upper indices refer to a particular time ($\mathbf{p}_i^t \in \mathbb{R}^3$ is then the three-dimensional position of vertex $\mathbf{p}_i$ at time t). By $\mathcal{M}^0$ we denote the template mesh (urshape) which has been extracted at the first time-step. Let $\{\mathbf{p}^0\} \in \mathbb{R}^3$ be the vertices of $\mathcal{M}^0$. $\mathcal{N}(i)$ is the set of vertices that are connected to the vertex $\mathbf{p}_i^0$ by an edge in $\mathcal{M}^0$. Since the connectivity of the evolving mesh stays fixed over time, this neighborhood relation does also hold for any subsequent mesh $\mathcal{M}^t$. $\mathcal{M}^t$ and $\{\mathbf{p}^t\}$ are the mesh and its vertices at time t, respectively. $\tilde{\mathbf{p}}_i^t \in \mathbb{R}^4$ is the spacio-temporal counterpart of the vertex $\mathbf{p}_i^t \in \mathbb{R}^3$ with the time as fourth coordinate, i.e. $\tilde{\mathbf{p}}_i^t = (\mathbf{p}_i^t, t)^T$.

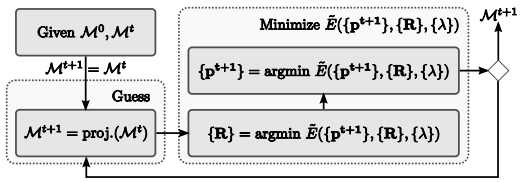

Figure 3: *Optimization loop.*

Assume we are given the urshape $\mathcal{M}^0$ and a deformed mesh $\mathcal{M}^t$ with vertices $\{\mathbf{p}^t\}$ at time t and we wish to compute the mesh $\mathcal{M}^{t+1}$ for the next time-step $(t+1)$ which deforms as little as possible while following the movement of the underlying kinematic surface $f = 0$. Thus, we need to find the vertex positions $\{\mathbf{p}^{t+1}\}$ which minimize equation (1) subject to the constraints that $f(\tilde{\mathbf{p}}_i^{t+1}) = 0$. These boundary conditions are non-linear and therefore the optimization with respect to these constraints would be computationally expensive. A fundamental property of any kinematic surface generated by a rigid motion is that the trajectories of all points $\{\tilde{\mathbf{p}}^t\}$ are perpendicular to the surface's normal field [MFO*07, PW01]. Under the assumption that the deformation between two consecutive time-frames is locally close to rigid, we can use this property to reformulate the aforementioned constraint as

$$(\tilde{\mathbf{p}}_i^{t+1} - \tilde{\mathbf{p}}_i^t) \cdot \nabla f(\tilde{\mathbf{p}}_i^t) = 0, \tag{2}$$

where $\nabla f(\tilde{\mathbf{p}}_i^t)$ is the 4-D normal of the time-space surface at time t at vertex $\mathbf{p}_i$ and $(\tilde{\mathbf{p}}_i^{t+1} - \tilde{\mathbf{p}}_i^t)$ is the four-dimensional movement at $\tilde{\mathbf{p}}_i^t$. Using Lagrange multipliers $\{\lambda\}$ to model the boundary condition (2), we arrive at following energy functional:

$$\tilde{E}(\{\mathbf{p}^{\mathbf{t+1}}\},\{\mathbf{R}\},\{\lambda\}) = \\ = \sum_{i=1}^{n} \sum_{j\in\mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}_i^{t+1} - \mathbf{p}_j^{t+1}) - \mathbf{R}_i(\mathbf{p}_i^0 - \mathbf{p}_j^0) \right\|^2 + \\ \sum_{i=1}^{n} \lambda_i (\tilde{\mathbf{p}}_i^{t+1} - \tilde{\mathbf{p}}_i^t) \cdot \nabla f(\tilde{\mathbf{p}}_i^t) \qquad (3)$$

The vertices $\{\mathbf{p}^{\mathbf{t+1}}\}$, which minimize the above energy $\tilde{E}$ are exactly the coordinates of the mesh $\mathcal{M}^{t+1}$ which describes an as-rigid-as-possible deformation of the urshape $\mathcal{M}^0$. Since the vertices $\{\mathbf{p}^{\mathbf{t+1}}\}$ fulfill the constraint given by equation (2), they do further lie on or at least close to (due to the assumption that the movement between consecutive frames is rigid) the kinematic surface $f = 0$ at time $(t+1)$.

The minimum of $\tilde{E}$ is found using an iterative approach as depicted in Figure 3: Using an initial guess (Sec. 5.1) for the yet unknown vertices $\{\mathbf{p}^{t+1}\}$, the per cell rotations $\{\mathbf{R}_i\}$ which map the cells of $\mathcal{M}^0$ to those of $\mathcal{M}^{t+1}$ are estimated (Sec. 5.2). Then, for fixed $\{\mathbf{R}\}$, the target function $\tilde{E}$ is minimized in order to find a revised estimate for the vertices $\{\mathbf{p}^{t+1}\}$ (Sec. 5.3). If the maximum number of iterations is not yet reached, the vertices $\{\mathbf{p}^{t+1}\}$ are projected onto the time-space surface at time $(t+1)$ again and the iteration is continued.

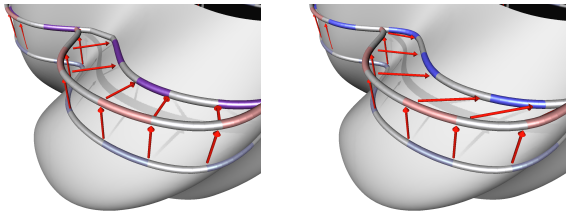

Figure 4: *Propagating the template mesh. (left) after projection; (right) final fit after energy minimization.*

5.1. Estimating the Initial Guess for $\mathcal{M}^{t+1}$

The time-space surface defined by $f = 0$ which we computed earlier offers a convenient way to compute an initial guess for the mesh $\mathcal{M}^{t+1}$ at time $(t+1)$. To estimate the vertex positions $\{\mathbf{p}^{t+1}\}$ of $\mathcal{M}^{t+1}$, we take the vertices $\{\mathbf{p}^t\}$ from the previous step and slide them onto the time-space surface $f = 0$ at time $t+1$. Therefore, the four-dimensional points $\tilde{\mathbf{p}}_i^{t+1} = (\mathbf{p}_i^t, (t+1))^T$ are projected onto the kinematic surface using a gradient descent method. By setting the fourth component of ∇f to zero during this iteration, we assure that the points are indeed projected onto the kinematic surface at time $(t+1)$. The coordinates of the vertices $\{\mathbf{p}^{t+1}\}$ are then set to the spatial coordinates of the projected points $\{\tilde{\mathbf{p}}^{t+1}\}$ (cf. Figure 4, left).

Note that all vertices of $\mathcal{M}^{t+1}$ now lie on the time-space surface at time $(t+1)$, and thus $\mathcal{M}^{t+1}$ is a valid reconstruction of the animation at time $(t+1)$. However, using solely this projection based approach to propagate the template mesh would result in serious surface sliding and potentially overlapping triangles.

5.2. Determining Optimal Rotations

Now that we have estimated a guess for the deformed mesh $\mathcal{M}^{t+1}$, we can use it to compute the rotations $\{\mathbf{R}\}$ which best map the cells $\{\mathcal{C}^0\}$ of the urshape onto the cells of $\mathcal{M}^{t+1}$. This is done by minimizing the energy functional E for the given set of vertices $\{\mathbf{p}^{\mathbf{t+1}}\}$ (we use the energy functional E from equation (1) here instead of $\tilde{E}$ because the $\{\mathbf{R}\}$ are not constrained by the boundary condition (2)). Following [SA07], we compute a covariance matrix

$$\mathbf{S}_i = \sum_{j\in\mathcal{N}_i} w_{ij} (\mathbf{p}_i^0 - \mathbf{p}_j^0)(\mathbf{p}_i^{t+1} - \mathbf{p}_j^{t+1})^T$$

for each cell $\mathcal{C}_i$ independently. Given the singular value decomposition of $\mathbf{S}_i = \mathbf{U}_i \Sigma_i \mathbf{V}_i^T$, the rotation $\mathbf{R}_i$ for the cell $\mathcal{C}_i$ which minimizes E can be computed as

$$\mathbf{R}_i = \mathbf{V}_i \mathbf{U}_i^T. \qquad (4)$$

5.3. Computing Optimal Vertex Positions $\{\mathbf{p}^{t+1}\}$

In order to find the vertex positions $\{\mathbf{p}^{t+1}\}$ which minimize the energy functional (3) for given rotations $\{\mathbf{R}\}$, we compute the gradient

$$\nabla\tilde{E} = \left(\frac{\partial\tilde{E}}{\partial\mathbf{p}_1^{t+1}}, \ldots, \frac{\partial\tilde{E}}{\partial\mathbf{p}_n^{t+1}}, \frac{\partial\tilde{E}}{\partial\lambda_1}, \ldots, \frac{\partial\tilde{E}}{\partial\lambda_n} \right)$$

of $\tilde{E}$. Let $\tilde{\mathbf{n}}_i^t = (nx_i^t, ny_i^t, nz_i^t, nw_i^t)^T$ be the four-dimensional normal of the kinematic surface $f = 0$ at point $\mathbf{p}_i^t$ and time t. Thus, $\tilde{\mathbf{n}}_i^t$ corresponds to the gradient of the implicit function f at the 4-D point $\tilde{\mathbf{p}}_i^t$: $\tilde{\mathbf{n}}_i^t = \nabla f(\tilde{\mathbf{p}}_i^t)$. The partial derivatives of $\tilde{E}$ with respect to $\mathbf{p}_i^{t+1} = (x_i^{t+1}, y_i^{t+1}, z_i^{t+1})^T$ and λ_i are given as:

$$\frac{\partial\tilde{E}}{\partial\mathbf{p}_i^{t+1}} = \sum_{j\in\mathcal{N}(i)} 4w_{ij} \left(\left(\mathbf{p}_i^{t+1} - \mathbf{p}_j^{t+1}\right) - \frac{1}{2}(\mathbf{R}_i + \mathbf{R}_j)\left(\mathbf{p}_i^0 - \mathbf{p}_j^0\right) \right) \\ + \lambda_i \cdot \mathbf{n}_i^t$$

$$\frac{\partial\tilde{E}}{\partial\lambda_i} = \tilde{\mathbf{p}}_i^{t+1} \cdot \tilde{\mathbf{n}}_i^t - \tilde{\mathbf{p}}_i^t \cdot \tilde{\mathbf{n}}_i^t,$$

where $\mathbf{n}_i^t \in \mathbb{R}^3$ are the spatial coordinates $(nx_i^t, ny_i^t, nz_i^t)^T$ of $\tilde{\mathbf{n}}_i^t$. Using the fact that $\tilde{\mathbf{p}}_i^t = (\mathbf{p}_i^t, w^t)^T$ is the four-dimensional counterpart of the vertex $\mathbf{p}_i^t$ and that the temporal distance $\upsilon = w^{t+1} - w^t$ between the points $\tilde{\mathbf{p}}_i^{t+1}$ and $\tilde{\mathbf{p}}_i^t$ is well known to be constant for each time-frame, we can rewrite the partial derivative w.r.t λ_i as:

$$\frac{\partial\tilde{E}}{\partial\lambda_i} = \mathbf{p}_i^{t+1} \cdot \mathbf{n}_i^t - \mathbf{p}_i^t \cdot \mathbf{n}_i^t + \upsilon \cdot nw_i^t$$

By setting the gradient $\nabla\tilde{E}$ to zero, we arrive at the following sparse linear system of equations:

$$\begin{aligned}\frac{\partial\tilde{E}}{\partial\mathbf{p}_i^{t+1}} &: \sum_{j\in\mathcal{N}(i)} w_{ij}\left(\mathbf{p}_i^{t+1}-\mathbf{p}_j^{t+1}\right)+\lambda_i\cdot\mathbf{n}_i^t \\ &= \sum_{j\in\mathcal{N}(i)}\frac{w_{ij}}{2}\left(\mathbf{R}_i+\mathbf{R}_j\right)\left(\mathbf{p}_i^0-\mathbf{p}_j^0\right) \qquad (5)\\ \frac{\partial\tilde{E}}{\partial\lambda_i} &: \mathbf{p}_i^{t+1}\cdot\mathbf{n}_i^t=\mathbf{p}_i^t\cdot\mathbf{n}_i^t-\upsilon\cdot nw_i^t\end{aligned}$$

Since the partial derivatives of $\tilde{E}$ w.r.t $\mathbf{p}_i^{t+1}$ are three-dimensional vectors, the above system of linear equations is of dimension $4n\times 4n$. The positions of the vertices $\{\mathbf{p}_i^{t+1}\}$ which minimize $\tilde{E}$ for the given set of rotations can now be computed by solving the equations in (5). To efficiently solve these equations we use the sparse LU-factorization provided by the UMFPACK [DD97] library. Note that all factors on the left side of the equations in (5) are fixed during the iteration for computing the mesh $\mathcal{M}^{t+1}$ from the mesh $\mathcal{M}^t$. It is therefore sufficient to factor the system only once during the first iteration step. In the following steps we only need to recompute the right-hand sides of the equations in (5) and do back-substitution to obtain the set of vertices $\{\mathbf{p}_i^{t+1}\}$ which minimize $\tilde{E}$. Figure 4 (right) shows the results of this iteration for a 2-D example.

5.4. Implementation Details

It can be seen in Figure 5 that the MPU method used to approximate the time-space surface f behaves badly at surface boundaries. Since we assume that the object for which we wish to reconstruct a coherent deforming mesh model was recorded by a single real-time 3-D scanner, it is presumably that the template model is not entirely covered by point samples in each time-frame (e.g. when recording a face which turns to one side, a part of the face will be occluded and therefore no samples will be captured for this part). If a part of the template mesh is no longer supported by sample points in a time-frame, it is therefore no longer feasible to enforce that this part is following the movement of the approximated kinematic surface $f=0$. To prevent such parts from being deformed unnaturally, we remove the constraint (2) from all mesh vertices that lost track with the underlying point cloud. To test whether a vertex $\mathbf{p}_i^t$ of the mesh $\mathcal{M}^t$ is still in touch with the original point cloud, we use the test which we previously used to restrict the iso-surface extraction, i.e. we check if an ε-ball around $\mathbf{p}_i^t$ contains any points of $\mathcal{P}$. The constraints for a vertex $\mathbf{p}_i^t$ can be removed by simply setting the respective Lagrange multiplier λ_i to zero. Therefore, the line containing the partial derivative w.r.t λ_i in the linear system of equations (5) is replaced by $\lambda_i=0$. Thus, the positions of the unconstrained vertices will be optimized only such that the mesh deforms as-rigid-as-possible while the corresponding parts of the mesh will follow the motion nicely.

data set	hand	face$_1$	face$_2$	igea
# points	3,766K	5,669K	4,153K	599K
# frames in	100	100	51	30
# frames out	100	100	100	100
MPU-error	0.003	0.01	0.01	0.003
mc-res.	120	120	120	80
iterations	5	4	3	10
urshape	65	323	171	73
sliding	1,650	3,911	3,109	2,011

Table 1: *Parameters and computation time (in seconds) for the data sets shown in Figure 6.*

6. Results

We used our algorithm to compute coherent deforming mesh models for several data sets ranging from simple synthetic data to real-time captured 3-D data sets with noise and scanning artifacts. Single frames of the generated animated meshes are shown in Figure 6, the complete animations can be found in the accompanying video. To expose potential surface drifting, all meshes were textured using texture coordinates computed from the first mesh.

Table 1 gives an overview of the computation times and the parameters used for the data sets shown in this paper. The first three rows contain the total number of points of the input point clouds in thousands, the number of inputted point clouds and the number of frames computed for the animated mesh sequence. The next rows give parameters for the maximum error of the MPU approximation, the marching cubes resolution which was used to generate the urshape and the (fixed) number of iterations used during the sliding of the template mesh. The final rows show timings in seconds for the computation of the urshape and the propagation of the urshape, respectively. All tests were performed on a single core Pentium-4 3.4GHz with 2GB RAM. Although the reconstruction of the entire animations took up to an hour for the real-world data sets, the amount of time required to reconstruct one single time-frame of the animation was only 17-40 seconds.

As can be seen in the video, the computed animated meshes follow the motion of the point clouds well while unnatural surface drifting cannot be observed.

Since the 4-D MPU reconstruction incorporates points from several time-steps, our algorithm implicitly exploits the coherence between neighboring time-steps which results in a higher signal-to-noise ratio. Figure 5 shows a comparison between the original 3-D MPU implementation [OBA*03] and our algorithm. Note that the artifacts at the surface boundaries in the center image would also be reproduced by our implementation if we would not stop iso-surface extraction at object boundaries. Due to the improved signal-to-noise ratio we are able to eliminate noise while preserving small features like the fine crinkles on the palm.

Limitations: The time-space formulation used in this work

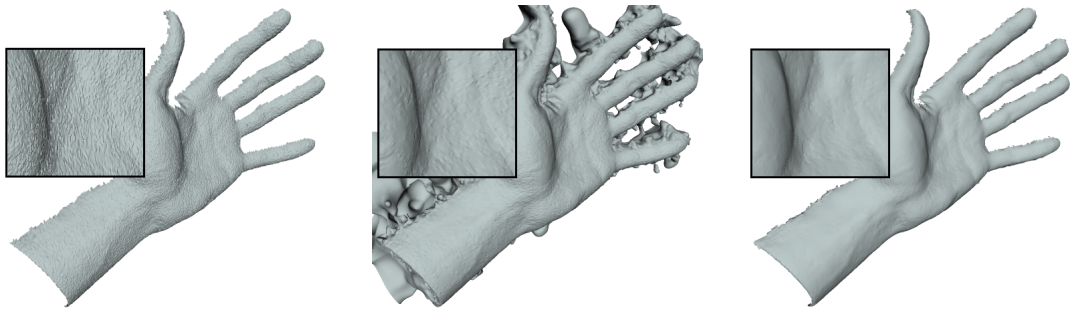

Figure 5: *Comparing noise sensitivity. (left) Original range scan; (middle) MPU reconstruction of a single time-frame; (right) Our reconstruction which incorporates data from neighboring time-frames.*

requires the input data to be sampled densely in both time and space as it is done by modern real-time shape acquisition devices. Theoretically, if the temporal sampling is too sparse or the movement between two consecutive time-frames is too fast, we fail in constructing a valid approximation to the time-space surface and therefore cannot reconstruct an animated mesh for the recorded sequence.

7. Conclusions and Future Work

We presented a novel approach for reconstructing animated meshes from point clouds sampled from time-deforming geometry directly in the time-space domain. By reconstructing the point clouds entirely in $\mathbb{R}^4$ we are able to compute meshes at arbitrary time instances which allows us to continuously generate models in-between the discrete time-steps at which the point clouds have been recorded. An advantage of the proposed time-space reconstruction is that the input points are not required to be sampled at distinct time-steps. It can also handle data sets whose points are arbitrary distributed in time. This property overcomes the restriction used in prior publication, that all movement is supposed to happen between two consecutive time-frames, and enables the acquisition of time-deforming 3-D sequences using "non-one-shot" scanners like fast laser line scanners.

We showed that our algorithm is applicable directly to raw scanner output data. As opposed to previous methods, no preprocessing like point decimation is required.

The computationally most expensive part of our implementation is the permanent evaluation of the implicit function f during the propagation of the template mesh along the kinematic surface. We think that by using a multi-resolution urshape model, we could dramatically improve the performance of our algorithm. Due to the high memory consumption of the MPU approximation, our algorithm is currently limited to reconstruct point clouds containing up to approximately 8 million points. In order to reconstruct sequences of several hundred point clouds we would have to realize a streaming variant of MPU approximation.

7.1. Acknowledgements

We wish to thank Thibaut Weise and Sören König for providing the real-time 3-D data sets and Yutaka Ohtake for making his implementation of MPU implicits publicly available.

References

[ACP02] ALLEN B., CURLESS B., POPOVIĆ Z.: Articulated body deformation from range scan data. *ACM Trans. Graph. 21*, 3 (2002), 612–619.

[ASS06] AMJOUN R., SONDERSHAUS R., STRASSER W.: Compression of complex animated meshes. In *Computer Graphics International* (2006), pp. 606–613.

[ATR*08] AHMED N., THEOBALT C., RÖSSL C., THRUN S., SEIDEL H.-P.: Dense correspondence finding for parametrization-free animation reconstruction from video. In *CVPR 2008* (2008), p. to appear.

[DD97] DAVIS T. A., DUFF I. S.: An unsymmetric-pattern multifrontal method for sparse *LU* factorization. *SIAM Journal on Matrix Analysis and Applications 18*, 1 (1997), 140–158.

[GG91] GERSHO A., GRAY R. M.: *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[KG07] KÖNIG S., GUMHOLD S.: Image-based motion compensation for structured light scanning of dynamic scenes. In *Dyn3D '07: Proceedings of the Dynamic 3D Imaging workshop* (2007), pp. 173–181.

[KHYS02] KÄHLER K., HABER J., YAMAUCHI H., SEIDEL H.-P.: Head shop: generating animated head models with anatomical structure. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), ACM, pp. 55–63.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: ACM SIGGRAPH 1987 Papers* (New York, NY, USA, 1987), ACM Press, pp. 163–169.

[LS01] LANGE R., SEITZ P.: Solid-state time-of-flight range camera. *Quantum Electronics, IEEE Journal of 37*, 3 (2001), 390–397.

[MFO*07] MITRA N. J., FLORY S., OVSJANIKOV M., GELFAND N., GUIBAS L., POTTMANN H.: Dynamic geometry

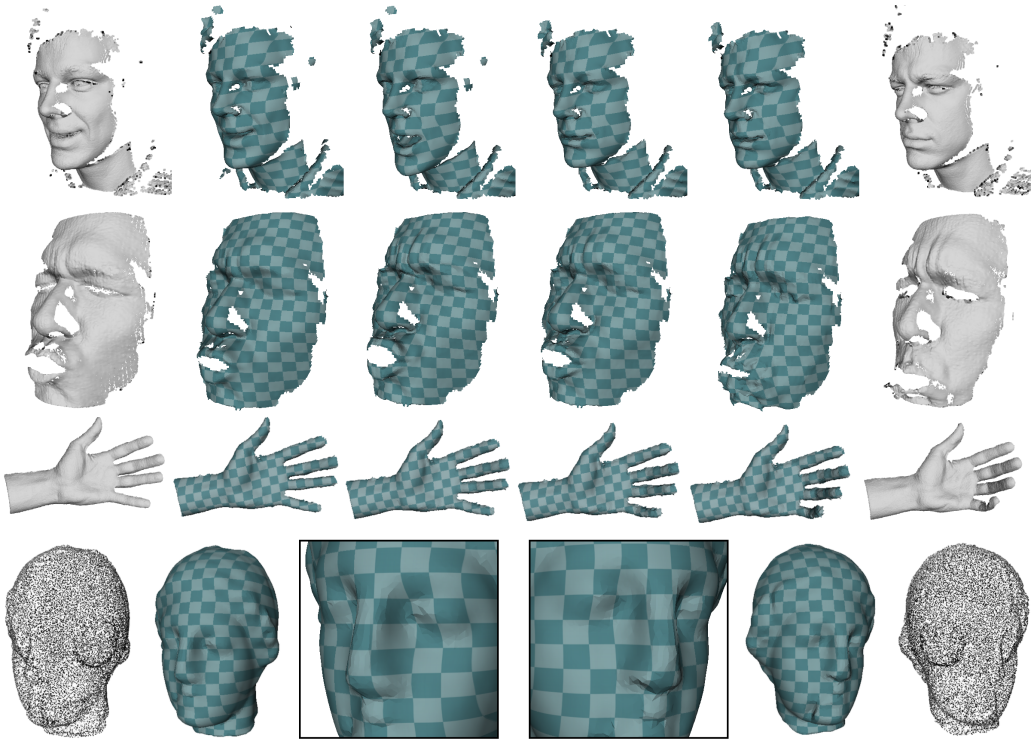

Figure 6: *Animated meshes reconstructed by our algorithm. The leftmost/rightmost image shows the first/last time-frame of the animated point clouds, single frames of the computed animated meshs are shown in between. First row: face$_1$ data set [WLG07]; second row: face$_2$ data set [KG07]; third row: hand data set [WLG07]; last row: synthetic igea data set.*

registration. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), Eurographics Association, pp. 173–182.

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM Press, pp. 463–470.

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2*, 1 (1993), 15–36.

[PW01] POTTMANN H., WALLNER J.: *Computational Line Geometry*. Springer, Heidelberg, Germany, 2001.

[RHHL02] RUSINKIEWICZ S., HALL-HOLT O., LEVOY M.: Real-time 3d model acquisition. In *SIGGRAPH '02: ACM SIGGRAPH 2002 Papers* (New York, NY, USA, 2002), ACM Press, pp. 438–446.

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), Eurographics Association, pp. 109–116.

[Shi04] SHINYA M.: Unifying measured point sequences of deforming objects. In *Proceedings of 3DPVT '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 904–911.

[SKR*06] STOLL C., KARNI Z., RÖSSL C., YAMAUCHI H., SEIDEL H.-P.: Template deformation for point cloud fitting. In *Symposium on Point-Based Graphics* (2006), pp. 27–35.

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM Press, pp. 399–405.

[SY07] SCHAEFER S., YUKSEL C.: Example-based skeleton extraction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), Eurographics Association, pp. 153–162.

[WJH*07] WAND M., JENKE P., HUANG Q., BOKELOH M., GUIBAS L., SCHILLING A.: Reconstruction of deforming geometry from time-varying point clouds. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), Eurographics Association, pp. 49–58.

[WLG07] WEISE T., LEIBE B., GOOL L. V.: Fast 3d scanning with automatic motion compensation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)* (June 2007), pp. 1–8.

[YS83] YAU M.-M., SRIHARI S. N.: A hierarchical data structure for multidimensional digital images. *Communication of the ACM 26*, 7 (1983), 504–515.

[ZH06] ZHANG S., HUANG P.: High-resolution, real-time three-dimensional shape measurement. *Optical Engineering 45*, 12 (2006), 123601.