Animated Mesh Approximation With Sphere-Meshes

JEAN-MARC THIERY
Delft University of Technology
ÉMILIE GUY and TAMY BOUBEKEUR
LTCI, CNRS, Télécom-ParisTech, Université Paris-Saclay
and
ELMAR EISEMANN
Delft University of Technology

Performance capture systems are used to acquire high-quality animated 3D surfaces, usually in form of a dense 3D triangle mesh. Extracting a more compact yet faithful representation is often desirable, but existing solutions for animated sequences are surface based, which leads to a limited approximation power in the case of extreme simplification. We introduce animated sphere-meshes, which are meshes indexing a set of animated spheres. Our solution is the first to output an animated volumetric structure to approximate animated 3D surfaces and optimizes for the sphere approximation, connectivity, and temporal coherence. As a result, our algorithm produces a multiresolution structure from which a level of simplification can be selected in real time, preserving a faithful approximation of the input, even at the coarsest levels. We demonstrate the use of animated sphere-meshes for low-cost approximate collision detection. Additionally, we propose a skinning decomposition, which automatically rigs the input mesh to the chosen level of detail. The resulting set of weights are smooth, compress the animation, and enable easy edits.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Object hierarchies*

General Terms: Algorithms

Additional Key Words and Phrases: Animated shape approximation, simplification, abstraction

This work has been partially funded by the European Commission under contract FP7-323567 Harvest4D and the Intel VCI at Saarland University.
Authors' addresses: J.-M. Thiery; email: jeanmarc.thiery@gmail.com; E. Guy; email: emieguy@gmail.com; T. Boubekeur; email: boubek@gmail.com; E. Eisemann; email: e.eisemann@tudelft.nl.

DOI: http://dx.doi.org/10.1145/2898350

ACM Reference Format:
Jean-Marc Thiery, Émilie Guy, Tamy Boubekeur, and Elmar Eisemann. 2016. Animated mesh approximation with sphere-meshes. ACM Trans. Graph. 35, 3, Article 30 (May 2016), 13 pages.
DOI: http://dx.doi.org/10.1145/2898350

1. INTRODUCTION

Modern performance capture systems automatically generate high-resolution 3D animated meshes from real-world objects and characters [Vlasic et al. 2008; de Aguiar et al. 2008a]. However, the resulting output sequences are mostly targeted for "replay"; each frame is represented independently.

In this context, high-level control mechanisms, based on geometric approximation, are often desirable to perform shape and motion processing, analysis, and editing. The underlying structures should capture the global spatial embedding of the animated shape and remain coarse enough to act as intuitive intermediate representations to edit the sequence. A number of such approaches have been proposed to reconstruct control structures having a shape-driven 3D spatial embedding such as an *animation skeleton* or a *deformations cage*. The former offers a natural layout to model articulated objects, while the latter is more suited for volume evolution modeling and maps better to nontubular geometries. Ideally, one would like to benefit from both worlds, using a medial structure, such as the *medial axis transform* [Blum 1967] (MAT), which provides explicitly both an inner skeleton and local thickness models. At a coarse scale, such a solution would be a valid alternative to both cages and skeletons. At finer scales, the volumetric nature of the medial axis is problematic, requiring an untractable amount of values, which describe geometry far away from the location of interest (i.e., the surface). In the case of static meshes, *sphere-meshes* [Thiery et al. 2013] tackle this problem. A sphere-mesh is a multiresolution mesh structure (edges, faces) indexing a set of spheres, which are optimized to properly approximate the input geometry locally, when linearly interpolated on the sphere-mesh simplices. Such a volume approximation compactly represents the shape as a convolution surface or a (simpler) primitive sum, proving useful for a variety of applications such as surface analysis [Siddiqi and Pizer 2008], shadowing [Wang et al. 2006], and proximity queries [Stolpner et al. 2012].

Unfortunately, volumetric structures and abstractions, such as the MAT or 3D "blobs" [Muraki 1991], exist mostly for static shapes. For example, the MAT cannot be used to represent consistently animated 3D data, since the MAT of a shape varies strongly and in unexpected ways along the animation, even for smooth and isometric transformations of the shape.

In this article, we approximate animated mesh sequences with *animated* sphere-meshes. Our algorithm outputs a nested hierarchy

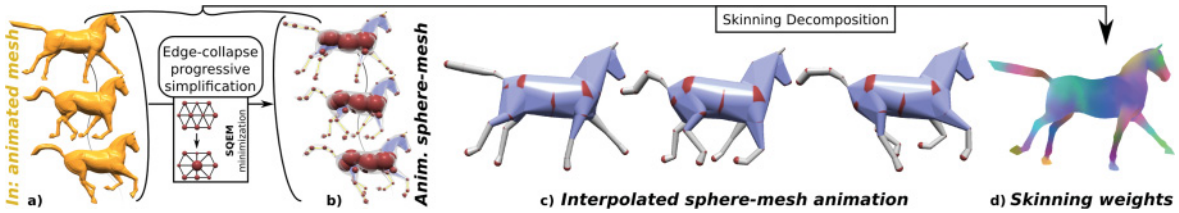


Fig. 1. Our algorithm takes as input an animated surface mesh (a) and outputs an animated sphere-mesh: a nonmanifold mesh indexing a set of animated spheres (b): spheres in red, edges in yellow, triangles in blue. The input animation can be approximated efficiently by interpolating the spheres on the simplices of the sphere-mesh (c): interpolated edges in gray, interpolated triangles in blue. We perform a skinning decomposition of the animation (d) using a sphere-mesh to define a level of detail. The resulting weight maps allow us to efficiently compress the data via linear blend skinning and to define new poses.

of simplified animated sphere-meshes that evolve from surface structures at fine scales to more volumetric structures at coarse scales, keeping a consistent connectivity during the entire animated sequence. Our work is the first to provide a time-consistent volumetric approximation of an animated surface mesh at different levels of detail. Existing applications using this representation can thus trivially be used directly on our data, when they could previously use static or manually designed animated convolution surfaces only. We illustrate this point with an application to approximate collision detection. As a second application, we show how to perform automatic rigging of the input animation using our animated sphere-mesh as a rigging structure for the original data. The resulting skinning weights are smooth enough for high-end shape modeling applications and reproduce faithfully the complex deformations learned from the input animation. Additionally, we introduce mixed weights to benefit from geometrically salient structures in regions that lack detailed motions along the animation.

Specifically, we make the following contributions (see Figure 1):

—An approximation algorithm to build an animated sphere-mesh from the input (Section 3)
—A spherical quadric error metric for animated meshes (Section 3)
—An error minimization algorithm (Section 4.1.1), which is compatible with temporal-coherence optimization (Section 4.2)
—An optional connectivity improvement (Section 4.3)
—A skinning decomposition method (Section 6.2), based on the chosen simplification level of the animated sphere-mesh, resulting in smooth weights considering either animation or geometric weights, depending on their appropriateness

We present the results for our approximation (Section 5) and skinnning decomposition (Section 6.2), before discussing our work (Section 7).

2. RELATED WORK

Animated-Mesh Simplification. Contrary to static-mesh simplification [Talton 2004], few approaches address animated-mesh simplification. Furthermore, the output of existing methods is generally an animated triangle mesh [Mohr and Gleicher 2003; DeCoro and Rusinkiewicz 2005; Kircher and Garland 2005; Landreneau and Schaefer 2009; Zhang et al. 2010] or a set of triangle meshes with minimal frame-to-frame connectivity transformations [Houle and Poulin 2001; Huang et al. 2005; Payan et al. 2007]. Most of these techniques rely on a quadric error metric (QEM) [Garland and Heckbert 1997] to define the cost of collapsing an edge to a 3D vertex by summing the QEMs over all frames. Such a solution exhibits two major advantages. First, edge decimation is fast and efficient, which is crucial for processing large animations. Second, starting from consistent meshes and decimating corresponding edges in all frames simultaneously maintains the output connectivity along the animation, which simplifies storage and editing. We build upon such a solution, but in contrast to previous approaches, we opt for an animated volumetric approximation, which has been shown to be favorable for coarse simplifications [Thiery et al. 2013].

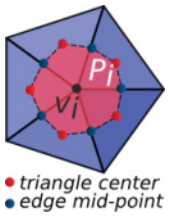
Skinning from Animations. For purely geometric animated-mesh compression, most work focuses on algebraic approximations (generally assimilated by nonlinear dimensionality reduction), such as skinning decomposition techniques. The latter approximate the input animation via a single mesh (often, a frame of the animation) and additional data in the form of skinning weights and a skeleton animation [de Aguiar et al. 2008b; Le and Deng 2014], or simply a set of *proxy-bone* transformations [Kavan et al. 2010; James and Twigg 2005; Le and Deng 2012], which can be either rigid or arbitrary. The input is then approximated by linear blend skinning (LBS) [Magnenat-Thalmann et al. 1988; Lewis et al. 2000]. These methods tend to produce a single bone per rigid region, considering the motion of the input mesh rather than its complete animated geometry (e.g., if the head of an animal is globally rigid along the animation, a single bone will be found). Additionally, the skeleton connectivity is usually deduced from an adjacency graph of the rigid parts in the mesh. Our approach makes it possible to select a skeletal domain taking the full animated geometry into account and to choose an appropriate level of detail before deriving a particular skinning decomposition.

Sphere-Meshes. In Thiery et al. [2013], the QEM decimation algorithm [Garland and Heckbert 1997] is modified to approximate the sum of squared distances of the tangent planes in a region to a sphere instead of a point. Iteratively, the edge inducing the lowest error according to this metric is collapsed. The resulting nested hierarchy of sphere-based approximations can then be traversed in real time, making it possible to progressively navigate from surface structures to volumetric structures by decreasing the number of spheres. Nonetheless, this approach did not consider animation, nor does there exist any technique for building an animated simplified volumetric representation. One important reason could be that most existing techniques for the static case rely on a MAT, which is inconsistent during animation when determined per frame.

Technical Background

We briefly present the edge-collapse decimation framework described in Thiery et al. [2013], which we extend in our system. It takes as input a surface mesh (preferably but not necessarily manifold or closed, and possibly containing some wire edges) and outputs a nested hierarchy of coarser meshes with a sphere associated with each of the vertices. The input mesh is then approximated at multiple resolutions by linearly interpolating the spheres on each of the triangles and edges of a sphere-mesh in the hierarchy.

Computation from Triangle Meshes. At first, each vertex v_i is associated with a region of the mesh called its barycentric cell P_i (see inset figure). In each simplification step, an edge is collapsed and the region associated with the newly created vertex is set as the union of the regions of the two collapsed vertices. The cost associated with each edge collapse is defined as the integral of the squared distance from the tangent planes in the region to a 3D sphere, whose radius and position are optimized to minimize this energy. All edges are collapsed iteratively, and these collapse operations are ordered by increasing cost in a priority queue.



When optimizing the geometry of a sphere approximating a region, its diameter is constrained to be smaller than the *width* [Gärtner and Herrmann 2001] of the region, essentially to avoid impossibly large spheres approximating planar surfaces.

The cost of a partition into K regions $\{I_k, \mathbf{s}_k\}_{k\leq K}$ is

$$\mathfrak{C}(\{I_k, \mathbf{s}_k\}_{k\leq K}) = \sum_{k\leq K} \mathbf{Q}_{I_k}(\mathbf{s}_k),$$

where $\{I_k\}_{k\leq K}$ is a partition of the vertex's barycentric cells $\{P_i\}$, $\mathbf{Q}_{I_k}$ is the sum of the spherical quadric error metrics (SQEMs) of the vertices in the set I_k, and $\mathbf{s}_k$ is a sphere associated with the region k. Formally, $\mathbf{Q}_{I_k}(q,r) = \sum_{i\in I_k} \int_{\xi\in P_i} SQEM_{(p_\xi,n_\xi)}(q,r)d\sigma_\xi$, where

$$SQEM_{(p_\xi,n_\xi)}(q,r) := ({n_\xi}^{\mathrm{T}} \cdot (p_\xi - q) - r)^2 \qquad (1)$$

is the squared distance between the plane intersecting $p_\xi \in \mathbb{R}^3$ with normal orientation $n_\xi \in \mathbb{R}^3$ and a sphere $\mathbf{s} := (q,r) \in \mathbb{R}^3 \times \mathbb{R}^+$, consisting of its center q and radius r, and $d\sigma_\xi$ denotes the infinitesimal surface element. The sphere s_k is optimized to best approximate the region I_k (in the sense that it minimizes its associated quadric $\mathbf{Q}_{I_k}$). As explained in detail in Thiery et al. [2013], the SQEM is sensitive to the normal orientation, and minimizing it while constraining the radius of the solution to be positive prevents approximating concave patches with a single sphere.

Hierarchy Traversal. At each decimation step, the locations and radii of the collapsed and created spheres are recorded, as well as the edges/triangles that are deleted or created. All these events are stored in an array, representing a sphere-mesh hierarchy. After the decimation process, the user can navigate in real time in the sphere-mesh hierarchy to find the desired level of detail. Following Hoppe [1996], we note $\mathcal{M}_0$ as the input mesh and $\mathcal{M}_\tau$ as the mesh resulting from the τ^{th} recorded edge-collapse operation (i.e., $\mathcal{M}_\tau$ is obtained by collapsing an edge of $\mathcal{M}_{\tau-1}$).

Surface Extraction. A surface approximating the input mesh can be extracted from a sphere-mesh as a convolution surface [Bloomenthal and Shoemake 1991] between the base mesh and the spheres. In our work, spheres are interpolated along the edges and faces of the base mesh. An interpolated edge corresponds therefore to a cone cut by orthogonal planes at each edge extremity, and an interpolated triangle corresponds to a triangular prism with three faces from the edges' extrusion and two triangular faces representing the lower and upper crust. For the sake of simplicity, although convolution-surface extraction algorithms exist [McCormack and Sherstyuk 1998; Zanni et al. 2013], we mesh a sphere-mesh (if needed) by contouring the signed distance to the sphere-mesh using a marching cube algorithm.

Rendering. The sphere-mesh primitives are rendered efficiently on the GPU using the geometry shader [de Toledo and Lévy 2008], without any surface extraction.

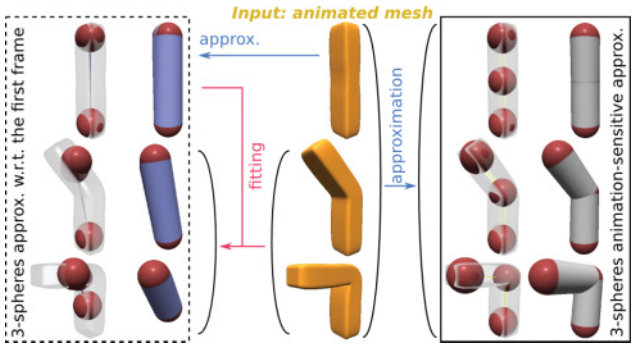


Fig. 2. The input animation is approximated with a sphere-mesh with three spheres with time-consistent connectivity. On the left, two spheres on top almost coincide because the sphere-mesh geometry is optimized w.r.t. the first frame, while this choice is not adequate for the complete animation. On the right, the sphere-mesh geometry is optimized w.r.t. all frames simultaneously, resulting in a geometry that is better suited to approximate the input animation.

3. ANIMATED SPHERE-MESHES

Problem Statement. We aim at approximating a mesh animation with F frames (meaning F different triangle meshes having the same connectivity) with an animated sphere-mesh (a mesh where each vertex is associated with a time-varying sphere). The linear interpolation of these spheres across the sphere-mesh triangles and edges results in the animated approximating shape.

Approximation Algorithm Overview. We cast our shape approximation problem into a partitioning of the barycentric cells P_i^f around each mesh vertex v_i in $\mathcal{M}_0$. The resulting regions of the partitioned animated input mesh are approximated via animated spheres. These spheres, together with the connectivity (deduced from the dual of the partition), results in an animated sphere-mesh.

We initialize the animated sphere-mesh with $\mathcal{M}_0$ and simplify it by iteratively collapsing its edges (equivalently, this means that regions are merged in the animated mesh), while minimizing a cost function. The output is, thus, an animated sphere-mesh hierarchy.

For a given frame $f \in [\![1, F]\!]$, the SQEM Q_i^f for a barycentric cell P_i^f describes the squared distances of the tangent planes in P_i^f:

$$Q_i^f(q,r) := \int_{\xi\in P_i^f} SQEM_{(p_\xi,n_\xi)}(q,r)d\sigma_\xi. \qquad (2)$$

The cost of a partition into K regions $\{I_k, \{\mathbf{s}_k^f\}_{f\leq F}\}_{k\leq K}$ is then

$$\mathfrak{C}\left(\left\{I_k, \left\{\mathbf{s}_k^f\right\}_{f\leq F}\right\}_{k\leq K}\right) := \sum_{k\leq K}\sum_{f\leq F} Q_{I_k}^f\left(\mathbf{s}_k^f\right), \qquad (3)$$

where $\{I_k\}_{k\leq K}$ is a partition of barycentric cells and $\mathbf{s}_k^f = (q_f^k, r_f^k)$ is a sphere approximating region I_k in frame f (in the sense that it minimizes $Q_{I_k}^f := \sum_{i\in I_k} Q_i^f$). By combining the cost of all frames, the entire animation is taken into account. This step is crucial (see Figure 2); a simple three-sphere fit to a capsule in the first frame (dashed box) is less suitable than a three-sphere approximation considering the entire sequence.

Radius Constraints. Similarly to Thiery et al. [2013], we bound each sphere's diameter during the optimization process, to avoid overly large spheres approximating planar regions. This bound, $R(P_{I_k})$ for a region I_k of the barycentric cell partition, is computed

based on an analysis of I_k over the animation. Since this step is linked to the spheres' optimization, the details are given in Section 4. To enforce smoothness in the reconstruction, we also constrain each sphere to have a constant radius over the animation. The motivation is that many animations (e.g., character and animal animations, etc.) are meant to be volume preserving, in which case the radius should be constant. We discuss this point further in Sections 4.2 and 5. Formally, it implies

$$\begin{cases} r\left(\mathbf{s}_k^f\right) = r_k \;\forall f \in [\![1, F]\!] \\ 0 \le r_k \le R(P_{I_k}) = \min_{f\in[\![1,F]\!]} R\left(P_{I_k}^f\right). \end{cases} \tag{4}$$

Quadrics in $\mathbb{R}^{3F+1}$

To apply the simplification process on animated meshes, we need

$$Q_{v_i}(\bar{\mathbf{s}}) := \sum_{f=1}^{F} \int_{\xi\in P_i^f} SQEM_{(p_\xi, n_\xi)}(q_f, r) d\sigma_\xi, \tag{5}$$

where $P_i := \{P_i^f\}_{f\le F}$ is the F-tuple of barycentric cells of vertex v_i and $\bar{\mathbf{s}}$ is an animated sphere. In this section, we derive a closed-form expression for this quadric.

Since we enforce that all F instances of the spheres have the same radius, we can denote an *animated* sphere $\bar{\mathbf{s}} \equiv ({q_1}^\mathrm{T}, \ldots, {q_F}^\mathrm{T}, r)^\mathrm{T} \in \mathbb{R}^{3F+1}$, that is, the concatenation of its F sphere centers $\{q_f\}_{f\le F}$ and radius r.

Noting that each barycentric cell P_i is contained in the set of triangles $t_j \in T_1(v_i)$, where $T_1(v_i)$ is the set of triangles adjacent to vertex v_i, and that the distance from an oriented point (p_ξ, n_ξ) to a sphere (q_f, r) is constant on any triangle, we can rewrite Equation (5):

$$Q_{v_i}(\bar{\mathbf{s}}) = \sum_{t_j\in T_1(v_i)} \sum_f w_{ij}^f SQEM_{(p_j^f, n_j^f)}(q_f, r), \tag{6}$$

where p_j^f (n_j^f, respectively) denotes the center (normal, respectively) of the triangle t_j^f (t_j in frame f), and $w_{ij}^f := \mathrm{area}(t_j^f)/3$.

$Q_{v_i}(\bar{\mathbf{s}})$ is itself an error quadric, as it is the sum of quadrics $Q_{v_i,t_j} := \sum_f w_{ij}^f SQEM_{(p_j^f, n_j^f)}$ for $t_j \cap P_i$. To find an expression for Q_{v_i}, it is thus sufficient to sum up the quadrics Q_{v_i,t_j}, which are given by

$$Q_{v_i,t_j} =: \frac{1}{2}\bar{\mathbf{s}}^\mathrm{T}\cdot \bar{A}_{ij}\cdot\bar{\mathbf{s}} - \bar{b}_{ij}^\mathrm{T}\cdot\bar{\mathbf{s}} + \bar{c}_{ij}, \tag{7}$$

with

$$\begin{cases} \bar{A}_{ij} := \left[\begin{array}{c|c|c|c} M_1 & & & N_1 \\ \hline & \ddots & & \vdots \\ \hline & & M_F & N_F \\ \hline {N_1}^\mathrm{T} & \cdots & {N_F}^\mathrm{T} & W \end{array}\right] & \in \mathcal{S}^{3F+1} \\ \bar{b}_{ij} := \left({b_1}^\mathrm{T}, \ldots, {b_F}^\mathrm{T}, B\right)^\mathrm{T} & \in \mathbb{R}^{3F+1} \\ \bar{c}_{ij} := \sum_f \left({N_f}^\mathrm{T}\cdot p_j^f\right)^2 & \in \mathbb{R}, \end{cases} \tag{8}$$

where $\mathcal{S}^n$ denotes the set of symmetric matrices of size $n\times n$ and

$$\begin{cases} M_f := 2w_{ij}^f n_j^f\cdot {n_j^f}^\mathrm{T} & \in \mathcal{S}^3 \\ N_f := 2w_{ij}^f n_j^f & \in \mathbb{R}^3 \\ W := 2\sum_f w_{ij}^f & \in \mathbb{R} \end{cases} \qquad \begin{cases} b_f := \left({N_f}^\mathrm{T}\cdot p_j^f\right) n_j^f & \in \mathbb{R}^3 \\ B := \sum_f \left({N_f}^\mathrm{T}\cdot p_j^f\right) & \in \mathbb{R} \end{cases}$$

Importance-Driven Distribution. Note that it is also possible to introduce a spatially varying importance function $K^f(\xi)$ in the process (e.g., taking into account the saliency of the mesh at each frame) by setting $w_{ij}^f = \int_{\xi\in P_i^f\cap t_j^f} K^f(\xi) d\sigma_\xi$ (which justifies indexing all terms over both the vertex and the triangle indices in previous equations). Following Thiery et al. [2013], for all results, we used $K^f(\xi) := 1 + BBD^2({\kappa_2}^f(\xi)^2 + {\kappa_1}^f(\xi)^2)$, where ${\kappa_2}^f(\xi)^2 + {\kappa_1}^f(\xi)^2$ denotes the total curvature at point ξ in frame f and BBD the average over the F frames of the bounding-box diagonal length. This measure prevents very small structures from disappearing too early during the optimization.

4. APPROXIMATION ALGORITHM

The simplification algorithm partitions the original mesh $\mathcal{M}_0$ into regions by successively simplifying a sphere-mesh. The initial sphere-mesh is defined as the input mesh, along with the quadrics Q_i associated with the barycentric cells of each vertex v_i (following Equation (2)). The spheres $\bar{s}_i$ of the sphere-mesh are optimized as $\bar{s}_i = \mathrm{argmin}_{\bar{s}}\, Q_i(\bar{s})$. The resulting approximation error is then $Q_i(\bar{s}_i)$. The closed-form solution for the minimization process is presented in Section 4.1.

The algorithm performs one edge collapse of the sphere-mesh at a time, the one leading to the lowest approximation error. To this extent, each possible edge collapse together with the resulting approximation error is placed in a priority queue $\mathcal{Q}$; that is, given an edge linking two vertices u and v with corresponding error quadrics Q_u and Q_v, this edge (u, v) is placed in $\mathcal{Q}$ with the approximation error for the quadric $Q_u + Q_v$ as a priority.

When taking the top edge (u, v) from $\mathcal{Q}$, we collapse it in the sphere-mesh to a new vertex w with $Q_w := Q_u + Q_v$. Remaining edge-collapse suggestions involving u and v are removed from $\mathcal{Q}$, while for all edges from w to a neighbor x, the (w, x) edge-collapse suggestion is added to $\mathcal{Q}$. The algorithm iterates until $\mathcal{Q}$ is empty. Notice that Q_w implicitly encodes the sphere associated with w: the mesh structure mainly encodes connectivity, while position and radius at vertices can be derived via the minimization of the quadric error.

Radius Bound Computation. Following Thiery et al. [2013], when determining the minimizing sphere, we bound the maximal radius to avoid flat regions being approximated by overly large spheres, as mentioned in Section 3. Specifically, for two vertices u and v with associated regions P_u and P_v, Thiery et al. suggest setting the radius bound to $\mathbf{R} := \mathbf{3/4}\mathcal{W}(\mathbf{P_u}\cup\mathbf{P_v})$, where $\mathcal{W}(\mathcal{X})$ denotes the width of the set $\mathcal{X}$ in $\mathbb{R}^3$.

The width (i.e., the minimum extent over all directions [Gärtner and Herrmann 2001]) is computed as follows. At the initialization stage, we compute the extent of the barycentric cell P_i^f of the vertices v_i along a given set of directions $k_j \in \mathbb{R}^3$ ($|k_j| = 1$). The extents are given by $M_j(P_i^f) := \max_{\xi\in P_i^f}(\xi^\mathrm{T}\cdot k_j)$ and $m_j(P_i^f) := \min_{\xi\in P_i^f}(\xi^\mathrm{T}\cdot k_j)$ and are used to define a width $\mathcal{W}(P_i^f) := \min_j |M_j(P_i^f) - m_j(P_i^f)|$ for frame f. Finally, the width $\mathcal{W}(P_i) := \min_f(\mathcal{W}(P_i^f))$ is defined as the minimum over the F frames. The width of $P_u^f \cup P_v^f$ is computed from the extents of P_u^f and P_v^f, which are obtained via $M_j(P_u^f \cup P_v^f) = \max(M_j(P_u^f), M_j(P_v^f))$ and $m_j(P_u^f \cup P_v^f) = \min(m_j(P_u^f), m_j(P_v^f))$. During a collapse, we update the extents for direction k_j, following these formulae.

Triangle Inversion. Additionally, we suggest forbidding collapsing two vertices as soon as it induces triangle inversions. For each

vertex v, we check all triangles $t \in T_1(v)$ and detect an inversion when $\sum_f \sqrt{\text{area}(t_f)\text{area}(t'_f)} n(t_f)^{\mathrm{T}} \cdot n(t'_f) < 0$, with t' being the geometry of t after the collapse. In other words, we test if large triangles are inverted for a significant amount of frames.

4.1 Sphere Optimization

Here, we describe the derivation of the minimizer $\text{argmin}_{\bar{s}}\, Q(\bar{s})$, under the radius constraints $0 \le r \le R$.

4.1.1 *General Case.* For convenience, in the following, we keep the notations introduced in Equation (8) to describe the block elements of the matrix $\bar{A}$ and the vector $\bar{b}$, which correspond to the quadric we want to minimize (the block structure never changes).

Quadric Minimization (Invertible). Minimize $E(\bar{\mathbf{s}}) = \frac{1}{2}\bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A} \cdot \bar{\mathbf{s}} - \bar{b}^{\mathrm{T}} \cdot \bar{\mathbf{s}}$, with $\bar{A}$, $\bar{b}$ of the form given by Equation (8), subject to $0 \le r \le R$.

Assuming all block matrices M_f are invertible, the global minimizer $\bar{\mathbf{s}}^*$ (without inequality constraints) is given by $\bar{A} \cdot \bar{\mathbf{s}}^* = \bar{b}$ ($\Leftrightarrow \overrightarrow{\nabla}_{\bar{s}} E(\bar{s}^*) = \vec{0}$), which leads to

$$\begin{cases} r^* = \frac{B - \sum_f {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot b_f}{W - \sum_f {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot N_f} \\ q_f^* = M_f^{-1} \cdot (b_f - r^* N_f) \quad \forall f \in [\![1, F]\!]. \end{cases}$$

The solution to the problem with inequality constraints is given by

$$\begin{cases} r = \min(\max(r^*, 0), R) \\ q_f = M_f^{-1} \cdot (b_f - r N_f) \quad \forall f \in [\![1, F]\!]. \end{cases} \tag{9}$$

If the denominator $(W - \sum_f {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot N_f)$ in the expression of r^* is null, we set $r = 0$ and $q_f = M_f^{-1} \cdot b_f \forall f$. However, this measure is only a sanity check to avoid numerical instabilities and this case rarely occurred in our experiments.

4.1.2 *Degenerate Cases.* Care must be taken if some of the matrices M_f are noninvertible (i.e., the quadrics are degenerate). For the sake of simplicity, we assume that $[\![1, D]\!]$ are the indices of the degenerate quadrics.

In the context of edge-collapse simplification, such cases are usually handled by constraining the optimal sphere position q_f to be located on the edge (u_f, v_f) that is about to be collapsed [Garland and Heckbert 1997]: $q_f = u_f + \lambda_f \vec{d}_f$, with $\vec{d}_f := v_f - u_f$. This restriction leads to a quadratic problem of size $D + 3(F - D) + 1$ with $2D + 2$ linear inequality constraints. Solving this problem via active-set methods (see Lawson and Hanson [1974]) is very costly for high-dimensional problems with a high number of inequalities, and, in our experiments, it happened that all F quadrics were degenerate. Therefore, we constrain all variables λ_f to equal a single value λ, that is, $q_f = u_f + \lambda \vec{d}_f \forall f \in [\![1, D]\!]$, leading to the following quadratic problem of size $3(F - D) + 2$:

Quadric Minimization (Not Invertible). Minimize $E(\bar{\mathbf{s}}) = \frac{1}{2}\bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A} \cdot \bar{\mathbf{s}} - \bar{b}^{\mathrm{T}} \cdot \bar{\mathbf{s}}$, with $\bar{A}$, $\bar{b}$ of the form given by Equation (10), subject to $0 \le \lambda \le 1$ and $0 \le r \le R$:

$$\begin{cases} \bar{A} = \left[\begin{array}{c|ccc|c} \mu & & & & \nu \\ \hline & M_{D+1} & & & N_{D+1} \\ \hline & & \ddots & & \vdots \\ \hline & & & M_F & N_F \\ \hline \nu & {N_{D+1}}^{\mathrm{T}} & \cdots & {N_F}^{\mathrm{T}} & W \end{array}\right] & \in \mathcal{S}^{3(F-D)+2} \\ \bar{b} = (\beta_1, {b_{D+1}}^{\mathrm{T}}, \ldots, {b_F}^{\mathrm{T}}, \beta_2)^{\mathrm{T}} & \in \mathbb{R}^{3(F-D)+2} \\ \bar{s} = (\lambda, {q_{D+1}}^{\mathrm{T}}, \ldots, {q_F}^{\mathrm{T}}, r)^{\mathrm{T}} & \in \mathbb{R}^{3(F-D)+2}, \end{cases} \tag{10}$$

with the various scalars μ, ν, β_1, β_2 defined as

$$\begin{cases} \mu := \sum_{f \le D} \vec{d}_f^{\,\mathrm{T}} \cdot M_f \cdot \vec{d}_f & \nu := \sum_{f \le D} {N_f}^{\mathrm{T}} \cdot \vec{d}_f \\ \beta_1 := \sum_{f \le D} (b_f - M_f \cdot u_f)^{\mathrm{T}} \cdot \vec{d}_f & \beta_2 := B - \sum_{f \le D} {N_f}^{\mathrm{T}} \cdot u_f. \end{cases}$$

For this problem, active set methods are efficient because, despite its size, the number of inequalities is low. Alternatively, we provide a closed-form solution for the minimization, which we used for all examples and results in Appendix A.

4.2 Temporal Coherence

The use of a constant sphere radius implicitly captures deformations and leads to a high consistency. Nonetheless, the position of the spheres being optimized per frame can lead to loss of temporal *correlation* with the input at very coarse scales, depending on the input. This section describes an algorithm to improve temporal coherence while only slightly reducing approximation accuracy.

One could enforce a smooth trajectory for each sphere. However, this approach is restrictive because it is sensitive to the time sampling of the input and not all sequences are smooth animations. Further, enforcing directly null time derivatives on trajectories results in unnatural and overly smooth animations.

Instead, we add a soft-attach term to the center of the sphere in each frame during the optimization, biasing its position toward a point moving consistently along the animation of the mesh. To this extent, we choose the barycenter of the corresponding region in $\mathcal{M}_0$, which the vertex in mesh $\mathcal{M}_\tau$ is associated with. Hereby, we avoid assumptions about the sampling of the input animation and mimic the temporal coherence of the input.

Formally, for an edge (u, v) with corresponding error quadrics Q_u and Q_v, the associated quadric becomes $Q_{uv} = Q_u + Q_v + \delta_{\mathbf{TC}} q_{uv}$ instead of $Q_u + Q_v$, with $\delta_{\mathbf{TC}}$ the temporal coherence weight. Let σ_u^f and c_u^f be the area and the mean point of the region P_u corresponding to the vertex u in frame f; we then define q_{uv} as

$$q_{uv}(\bar{\mathbf{s}}) = \sum_{f \le F} \sigma_{uv}^f ||q_f - c_{uv}^f||^2, \tag{11}$$

where $\sigma_{uv}^f = \sigma_u^f + \sigma_v^f$, $c_{uv}^f = \frac{\sigma_u^f c_u^f + \sigma_v^f c_v^f}{\sigma_u^f + \sigma_v^f}$. In other words, $q_{uv}(\bar{\mathbf{s}})$ defines the sum of the squared distances between the sphere centers and $\{c_{uv}^f\}_{f \le F}$ over all frames. In practice, following Equation (8), the following elements need to be added to the quadric $Q_u + Q_v$:

$$\begin{cases} M_f \leftarrow M_f + 2\delta_{\mathbf{TC}} \cdot \sigma_{uv}^f \cdot I_3 & \forall f \le F \\ b_f \leftarrow b_f + 2\delta_{\mathbf{TC}} \cdot \sigma_{uv}^f \cdot c_{uv}^f & \forall f \le F \\ \bar{c} \leftarrow \bar{c} + \delta_{\mathbf{TC}} \cdot \sum_{f \le F} \sigma_{uv}^f ||c_{uv}^f||^2. \end{cases} \tag{12}$$

4.3 Connectivity Optimization

Most edge collapses, especially for the first levels, maintain a good connectivity. A verification after each edge collapse is thus not justified due to the involved costs. However, at coarse scales, undesired artifacts in the form of large and thin triangles crossing the original surface can occur, which is a typical pitfall of edge-decimation-based simplification algorithms. These artifacts are more likely to occur whenever the associated regions in $\mathcal{M}_0$ differ significantly from the Voronoï regions of the vertices (in our context, spheres) of $\mathcal{M}_\tau$. This last observation motivated the use of (centroïdal) Voronoï tesselation algorithms, which are considered the most effective for 2D surface and 3D volume remeshing. It also inspires ours, which exhibits a similar behavior.

We propose to optimize the connectivity only after selecting the desired level of simplification τ. We start by associating each vertex v_i of the original mesh $\mathcal{M}_0$ to a set of candidate sphere indices $S(i)$ in $\mathcal{M}_\tau$, which are the sphere to which v_i was collapsed and all its adjacent spheres in $\mathcal{M}_\tau$. This set is relatively small in practice, as it is bounded by the highest vertex degree in $\mathcal{M}_\tau$ plus one. For each vertex v_i, an index s_i among $S(i)$ is chosen as the one minimizing the sum of distances along the animation:

$$s_i = \underset{(\{q_f\}_{f\leq F}, r)\in S(i)}{\operatorname{argmin}} \sum_{f\leq F} \left(\left|\left|v_i^f - q_f\right|\right| - r\right).$$

The connectivity of $\mathcal{M}_\tau$ is redefined entirely based on the dual of the sphere-index labeling in $\mathcal{M}_0$: **(1)** for each triangle (v_a, v_b, v_c) of $\mathcal{M}_0$ whose vertices are labeled with three different spheres s_a, s_b, and s_c, the corresponding triangle (s_a, s_b, s_c) and edges are added, unless they already exist; **(2)** for each triangle or edge of $\mathcal{M}_0$ whose vertices have two different sphere labels, a corresponding edge is added to $\mathcal{M}_\tau$, unless it already exists.

5. RESULTS

We implemented our shape approximation method in C++ and report its performance on an Intel Core2 Duo running at 2.5GHz with 4GB of main memory. The algorithm is automatic and computes the full hierarchy. Afterward, the user controls one parameter (the target number of spheres). In Figure 3, we show results on performance capture, as well as on synthetic animations.

We report timings and distances to the input (computed with *Metro* [Cignoni et al. 1998]) in Table I, which serve as numerical comparisons to QEM simplification. For a fair comparison, we allow triangles to degenerate to wire edges for QEM. However, it results in invalid shapes, since QEM does not model volumes.

Sphere-meshes tend to better approximate shapes than traditional surface meshes for a low number of primitives. We roughly halve reconstruction errors for the Capoeira, Samba, Jumping, and Flamingo sequences, using various numbers of spheres ranging from 10 to 42. In contrast, if many primitives are required (with respect to the geometric complexity of the shape), sphere-meshes tend to have smaller reconstruction-error ratios (e.g., roughly 1.2 for the Hand and Cat sequences, when approximating with 32 and 85 spheres, respectively). Sphere-meshes and traditional QEM-driven triangle meshes start with the same animated input mesh. Hence, they are basically equivalent at the first levels of simplification, but with a decreasing number of primitives, sphere-meshes evolve progressively from surface to volumetric structures.

Interestingly, when comparing the *Horse-gallop* and *Horse-collapse* sequences, which both contain 46 spheres, we observe that animated sphere-meshes tend to become volumetric if the input mesh deformations are volume preserving. Consequently, the sphere-mesh nicely adapts and behaves either more like a skeleton or like rigs depending on the type of deformation; for example, a skeleton could animate efficiently the *Horse-gallop* sequence, whereas a set of surface rigs would be more efficient at animating the *Horse-collapse* sequence.

Figure 4 illustrates a similarly challenging example because the *Samba* model contains large tubular parts (legs, arms, etc.), which can be approximated by single edges, while the dress model is better represented using surface structures; that is, it is highly deformed in a nonrigid way during the animation. Even with a low number, such as 22 spheres, our sphere-meshes recover the fine details of the dress as well as the simpler arm and leg structures. When using only seven spheres, an abstract structure taking a plausible skeleton form emerges. Note that the extracted surface remains manifold, even in the presence of wire edges in the sphere-mesh.

Analysis. Figure 5 shows results where each sphere is allowed to have a time-varying radius, which is bounded per frame only. The left side shows how strongly the resulting sphere-mesh varies, even for "volume-constant" animations. The right side shows that the resulting sphere-mesh no longer exhibits a proper structure in the collapsed parts, which leads to an unnatural connectivity as well (see Figure 3 for a comparison to our approach). Our solution results in an improved sphere placement and behavior during animation, which makes it applicable for skinning decompositions (see next section). The latter would be impossible when using varying radii.

Figure 6 illustrates the effect of our temporal coherence method. Connectivity and preserved features might change for the same number of spheres and produce more natural simplifications for input animations exhibiting strong temporal coherence. In particular, as emphasized within the figure, spheres remain in the vicinity of the same input vertices and avoid sliding over the geometry. In this example, it is also visible that our solution is less dependent on the time sampling of the input sequence. Our temporal constraints are based on the coherence of the original input, which makes sure that no overly smooth motion is enforced where it is not present. Consequently, the use of the temporal coherence constraint also leads to more constant-length edges in the simplified animated sphere-mesh over the course of the animation, at the cost of a slightly less optimal geometric approximation.

Figure 7 illustrates the impact of the connectivity optimization step. While the Hausdorff errors might already be acceptable before this optimization and are not necessarily reduced significantly by a modification of the sphere-mesh connectivity, some elements might be undesirable. An example is the presence of primitives, such as elongated triangles, which cross the input surface or lead to locally inverse connectivity. Figure 7 illustrates several such cases. In the first example (left), the tail of the horse, defined as a long triangle, crosses the surface due to a nonconvex partitioning of the input mesh $\mathcal{M}_0$. The second example (middle) depicts a triangle, which introduces an unwanted link, with the ankle of the left leg being connected to the hip of the right leg. Finally, the third example (right) shows that, due to inadequate local connections, complex geometry inversions are introduced. All these connectivity problems are solved by our nonparametric optimization step, as illustrated next to the circled issue.

6. APPLICATIONS

Our algorithm is the first to approximate automatically animated surfaces using an animated convolution surface. Existing applications using this representation can thus trivially be used directly with our data, when they could previously use static or manually

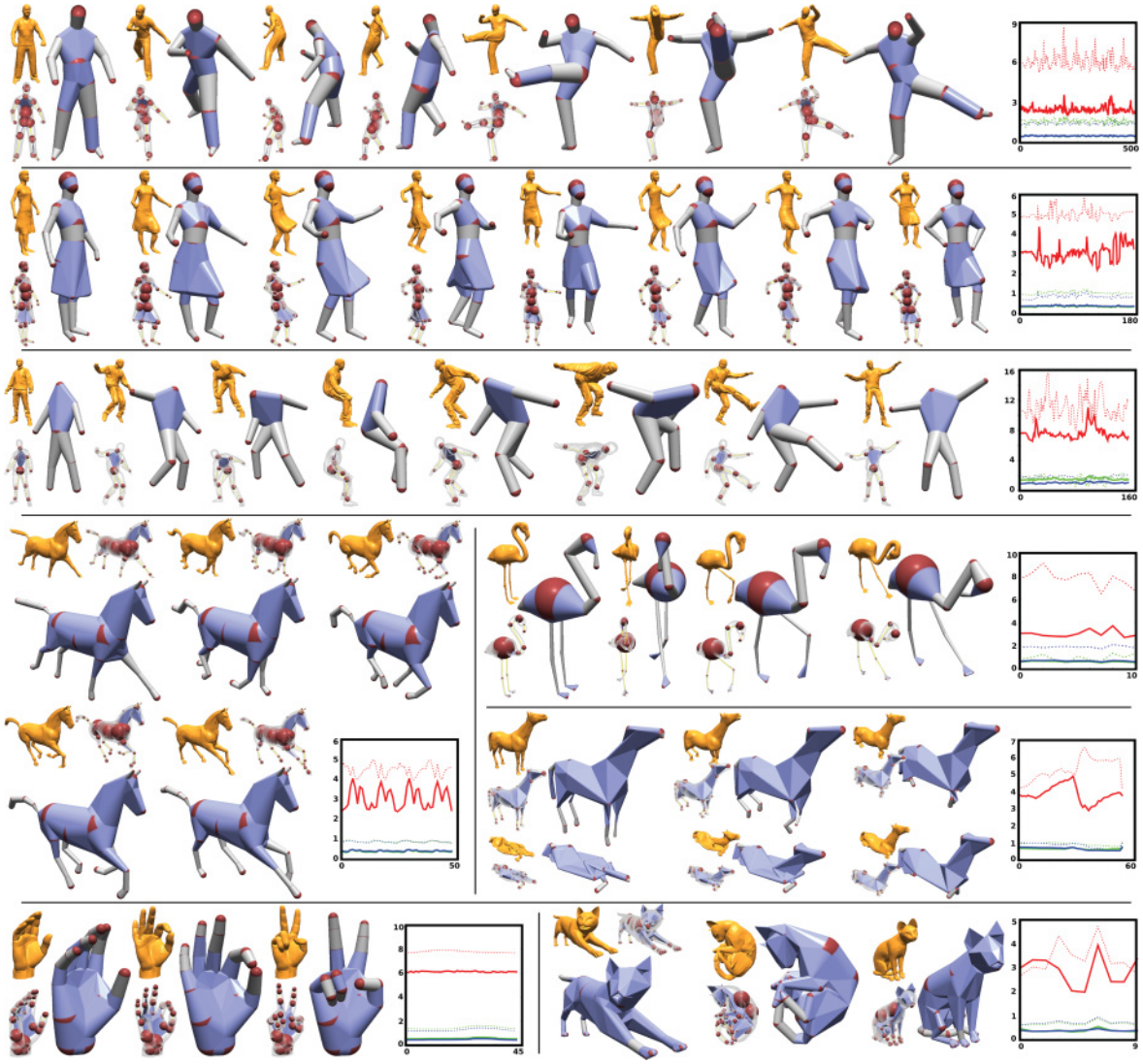


Fig. 3. With $\delta_{\mathbf{TC}} = \mathbf{0}$, **no connectivity optimization**. For each animation, we show the input mesh (gold), the constructed sphere-mesh with transparent input, and the interpolated sphere-mesh. Solid (dashed, respectively) curves plot distances between our (QEM, respectively) approximation and the input. Red curves: Hausdorff distance (H), blue curves: mean distance from the approximation to the original model (M21), green curves: mean distance from the original model to the approximation (M12) (distances in percentages of the bounding-box diagonal).

Table I. Performance and Timings for Our Animated Sphere-Mesh Approximation Algorithm. The Initialization Time Excludes the Animation Parsing. #S / #E / #T: Number of Spheres, Wire Edges, and Triangles in the Output Sphere-Mesh, Respectively. We Compare Our Approximation with QEM Simplification for the Same Number of Primitives (Smallest Error in **Bold**). H, M21, M12 (See Figure 3) Are Averaged Over the Animation

		Init.	Decim.	Animated Sphere-Mesh				QEM Simplification			
Input Anim.	(#V / #T / #F)	(ms)	(ms)	(#S / #E / #T)	H	M12	M21	(#V / #T)	H	M12	M21
Capoeira	(19,988 / 39,972 / 499)	65,338	186,482	20 / 9 / 12	**2.45**	**0.46**	**0.47**	20 / 31	6.05	1.58	1.38
Samba	(9,971 / 19,938 / 175)	8,037	16,302	40 / 12 / 46	**3.05**	**0.37**	**0.39**	40 / 60	4.97	1.05	0.86
Jump	(10,002 / 20,000 / 150)	7,221	15,488	10 / 6 / 4	**7.62**	1.53	**1.02**	10 / 10	11.11	**1.47**	1.76
Flamingo poses	(26,394 / 52,895 / 11)	2,252	5,628	20 / 8 / 10	**3.03**	**0.62**	**0.67**	20 / 21	7.79	0.96	1.89
Horse-gallop	(8,431 / 16,843 / 48)	2,338	5,394	46 / 17 / 50	**3.06**	**0.41**	**0.43**	46 / 76	4.47	0.89	0.89
Horse-collapse	(8,431 / 16,843 / 54)	2,554	4,915	46 / 9 / 61	**3.84**	**0.65**	**0.66**	46 / 76	5.29	0.93	0.85
Hand	(7,929 / 15,855 / 44)	2,120	4,412	34 / 8 / 44	**6.11**	**0.55**	**0.47**	34 / 62	7.77	1.44	1.26
Cat poses	(7,207 / 14,410 / 10)	489	1,283	85 / 5 / 142	**2.86**	**0.38**	**0.37**	85 / 166	3.34	0.66	0.70

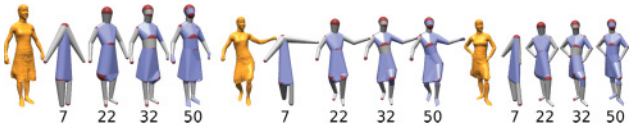


Fig. 4. Samba model animation approximated with animated sphere-meshes with seven, 22, 32, and 50 spheres.

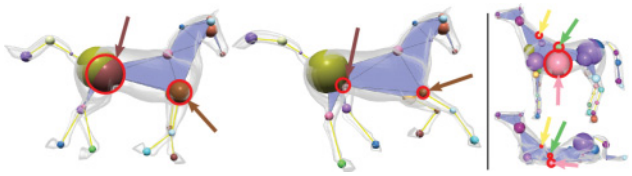


Fig. 5. Results obtained with a time-varying sphere radius.

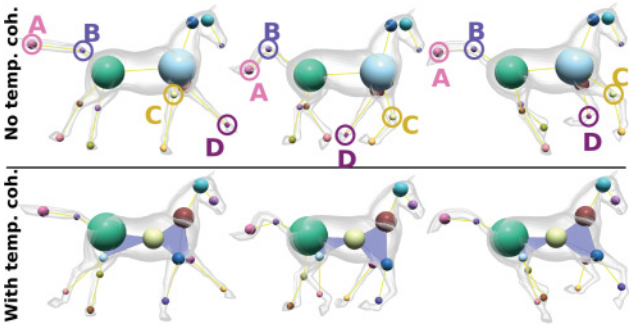


Fig. 6. Comparison of results obtained without (top row) and with temporal coherence (bottom row, $\delta_{\mathbf{TC}} = \mathbf{10^5}$) using 15 spheres. Notice how, without our temporal coherence method, the spheres (A, B, C, D) slide along the mesh to best fit the input mesh.

designed animated convolution surfaces only. The first application we showcase, approximate collision detection with animated meshes (Section 6.1), is a simple illustration of this. The second application we present, skinning decomposition using our sphere-mesh as a decomposition domain (Section 6.2), is more involved and relies on the fact that our approximation method outputs geometries that are compatible with the specific nature of the input animations (as already discussed when comparing the *Horse-gallop* and *Horse-collapse* sequences in Section 5).

6.1 Approximate Collision Detection

Our animated sphere-mesh can be used as a direct low-budget substitute for collision detection with the animated object. By testing the distance from a point to the primitives of the animated sphere-mesh directly, approximate collision detection can be achieved at high frame rates without the building of any complex structure.

Figure 8 illustrates the advantages of such a solution for approximate collision detection. An animated sphere-mesh with few spheres presents enough geometric details to be a cheap yet plausible substitute to the complete animated mesh. Compared to low-detail surface meshes (see Figure 8), the normal field of low-detail sphere-meshes is smooth enough to avoid directional artifacts after the bounces of the particles. In Appendix B, we give formulas for the construction of the sphere-mesh primitives, as well as for intersecting them with particles (represented as spheres on segments).

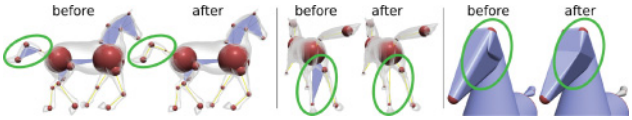


Fig. 7. Comparison of results obtained without and with connectivity improvement for various settings.

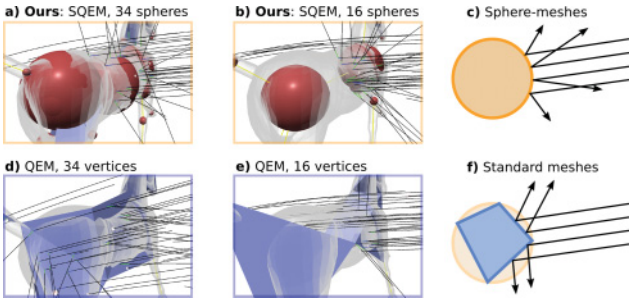


Fig. 8. We compare the use of an animated sphere-mesh (a, b, c) against a standard mesh (d, e, f) for approximate collision detection with particles. The normal field on large tubular structures is smooth, and so are the resulting bounces on the sphere-mesh (compare SQEM with QEM for the same number of primitives). When using 16 vertices (e), a large number of particles go through the input shape and fail to bounce, whereas our 16-sphere approximation (b) provides acceptable results.

6.2 Skinning Decomposition with Rigid Bones

We use a strategy similar to Le and Deng [2012] and decompose the animation on our sphere-mesh via linear-blend skinning (LBS). The bones for LBS correspond to the spheres of the sphere-mesh, which is provided by the user as input for our decomposition scheme. The user can therefore chose any level of complexity he or she desires to obtain for the decomposition. The output of the decomposition is a set of weights $\{w_{ij}\}$, which rig the rest pose to the sphere-mesh, and bone transformations $\{R_j^f, T_j^f\}$, which, together with the weights, allow for reconstruction of the input animation. Further, the artist can use the derived weights with the sphere-mesh to create new animations exhibiting the input animation's partial motion.

Notations: v_i^f is the position of vertex i in frame f, μ_i^f is the area of its barycentric cell in frame f, $B(i)$ is the set of bones that have an influence over vertex i, w_{ij} is the weight of bone j at vertex i, p_i is the rest pose vector of vertex i in $\mathbb{R}^3$ (we initialize the rest pose with the first frame), and R_j^f and T_j^f are the rotation and the translation of the bone j in frame f, respectively. Please note the slight abuse of notation, allowing us to consider a bone j as an index.

Constraints: $\{w_{ij}\}$ are optimized under a *nonnegativity constraint* ($w_{ij} \geq 0 \forall i, j$), an *affinity constraint* ($\sum_j w_{ij} = 1 \forall i$), and a *sparsity constraint*, that is, a small set $B(i) := \{j | w_{ij} \neq 0 \forall j\}$, which is the set of bones having an influence on v_i. While the sparsity mostly serves acceleration purposes, the other two constraints are important for stability during editing processes.

In contrast to Le and Deng [2012], we fix $B(i)$ once and for all based on the animated sphere-mesh, which serves as the skeletal domain for the decomposition. For a given vertex v_i in the first frame, we define $B(i)$ as the set of those sphere-mesh spheres whose Voronoï cells on the input mesh are adjacent to the cell in which v_i is located. In other words, all adjacent spheres have a potential influence, but none of the others.

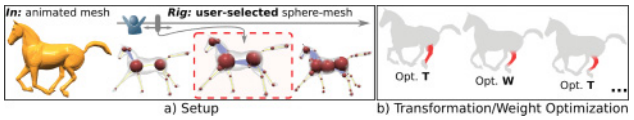


Fig. 9. **(a)** The input for the skinning decomposition is a **user-selected** LoD of the animated sphere-mesh hierarchy, which is traversable in real time. **(b)** A few iterations of successive optimizations of the bones' transformations (**T**) and the vertex's weights (**W**).

Figure 9 illustrates the optimization process. At first, the user picks a level of detail from the animated sphere-mesh hierarchy, which is an intuitive way to give control over the degrees of freedom desired in the output skinning decomposition. At each step, we optimize the vertex weights, while fixing the bones' transformations R_j^f and T_j^f. Next, we optimize these bone transformations bone by bone and frame by frame, while fixing the vertex weights. Finally, we also optimize for the rest pose after a few (typically five) iterations.

Transformations Optimization. Similarly to Le and Deng [2012], we optimize the transformations of the bones one after the other. When optimizing the one of bone j, we minimize

$$E_{R_j^f,T_j^f} = \sum_{i\in I(j)} \mu_i^f \left\| w_{ij}\left(R_j^f \cdot p_i + T_j^f\right) - q_{ij}^f \right\|^2,$$

for all frames f, where q_{ij}^f is the resulting point without the contribution of bone j: $q_{ij}^f := v_i^f - \sum_{k\in B(i),\neq j} w_{ik}(R_k^f \cdot p_i + T_k^f)$, with R_j^f and T_j^f as variables and $I(j)$ denoting the vertices that are influenced by the bone j (i.e., $I(j) = \{i | j \in B(i)\}$), under the only constraint that R_j^f is a rotation. Details for this optimization can be found in Le and Deng [2012], up to the fact that the reader should introduce the various weights $\{\mu_i^f\}$ in the formulas.

Minor bones. As explained in the work of Le and Deng [2012], such a transformation-optimization process can introduce instabilities with bones that have minor influence over vertices ($\{j | \sum_i w_{ij}^2 < \epsilon\}$). We detect them using the same criterion (bone j is unstable if $\sum_i w_{ij}^2 < 3$) and solve the issue by setting the transformation of the bone to an average of the transformations of its adjacent bones (following the sphere-mesh connectivity).

Vertex Rest-Pose Optimization. We optimize the rest pose p_i of each vertex v_i, by minimizing for each vertex v_i independently

$$E_{p_i} = \sum_f \mu_i^f \left\| \left(\sum_{j\in B(i)} w_{ij} R_j^f\right) \cdot p_i - \left(v_i^f - \sum_{j\in B(i)} w_{ij} T_j^f\right) \right\|^2,$$

which boils down to solving a simple quadratic system in $\mathbb{R}^3$.

Mixed Weights Optimization. Previous work (e.g., Le and Deng [2012]) optimizes weights with the sole purpose of reconstructing the input animation. The control structure (i.e., the bones) and the weights, which result from this process, exhibit a level of complexity that is directly driven by the complexity of the input animation (i.e., a completely rigid animation will result in a single bone with all mesh vertices having a weight of 1 w.r.t. this bone, rendering this decomposition useless for artistic editing, for example).

In contrast, we provide the control structure as input to our skinning decomposition scheme, and we *enforce* the desired level of complexity of the weight maps. We do so by initializing our weights with *geometric weights* (we use Baran and Popović [2007], but our method makes no assumptions on the input geometric method, and other geometric skinning weights could be used instead (e.g., Jacobson et al. [2011])) and progressively refine our weights as to better fit the input animation.

The core idea is that geometric weights should be favored in regions of the mesh where they are able to reproduce the input animation, as they allow for more general transformations. In other regions of the mesh, where these are not able to reproduce the input animation well, the weights should be learned from the animation.

We optimize the weights $\{w_{i\cdot}\}^{t+1}$ of vertex i at step $t+1$ by minimizing the following energy:

$$\begin{aligned} E_{w_i}^{t+1} = \alpha_i^{t+1} \Bigg(& \sum_f \mu_i^f \left\| \sum_{j\in B(i)} w_{ij}^{t+1}\left(R_j^f \cdot p_i + T_j^f\right) - v_i^f \right\|^2 \\ & + \gamma \sum_f \mu_i^f \sum_{j\in B(i)} \left(w_{ij}^{t+1} - \tilde{w}_{ij}^t\right)^2 \Bigg) \\ & + \left(1-\alpha_i^{t+1}\right)\gamma \sum_f \mu_i^f \sum_{j\in B(i)} \left(w_{ij}^{t+1} - w_{ij}^t\right)^2, \end{aligned}$$

where $\gamma := 10^{-4} BBD^2$ is a normalization constant that compensates for the differences in the range between reconstruction and weight errors based on the bounding box diagonal, $\tilde{w}_{ij}^t := \sum_{v_k\in B(v_i;3\sigma)} h_\sigma(v_i, v_k) w_{kj}^t$ is a filtered version of $\{w_{ij}^t\}$ (we use a Gaussian kernel over a geodesic disk centered in v_i, with σ equaling 3% of the bounding box diagonal), and α_i^{t+1} is the blending factor between animation-driven weights and the geometric alternative. α_i^{t+1} should be close to one if information needs be extracted from the input animation. Otherwise, α_i^{t+1} should be low, leading to mostly geometric weights being used.

Our strategy to determine α_i^{t+1} is as follows. We first compute the weights $\hat{w}_{ij}^{t+1}$ for all vertices v_i with $\alpha_i = 1$. Next, we compute the reconstruction errors of the vertices v_i when considering the newly found weights $\hat{w}_{ij}^{t+1}$ and the weights w_{ij}^t of the previous iteration:

$$\begin{cases} e_i^{\text{before}} = \sum_f \mu_i^f \,||\sum_j w_{ij}^t (R_j^f \cdot p_i + T_j^f) - v_i^f||^2 \\ e_i^{\text{after}} = \sum_f \mu_i^f \,||\sum_j \hat{w}_{ij}^{t+1} (R_j^f \cdot p_i + T_j^f) - v_i^f||^2 \end{cases}$$

Finally, we set a large value to α_i for the vertices v_i where the reconstruction error decreases significantly when compared to the differences in the weights $\delta_i = ||\{\hat{w}_{ij}^{t+1}\}_j - \{w_{ij}^t\}_j||^2$. Specifically, noting $\gamma_i = (e_i^{\text{before}} - e_i^{\text{after}})_+$, the decrease of the reconstruction error of vertex v_i; $\gamma = \max_i \gamma_i$, the maximum decrease; and $\delta = \max_i \delta_i$, the maximum weights change, we set $\alpha_i^{t+1} = \frac{0.8\gamma_i/\gamma}{\gamma_i/\gamma + \delta_i/\delta}$.

Results. Figure 10 shows the skinning reconstruction of the *Horse-gallop*, *Horse-collapse,* and *Samba* inputs decomposed onto sphere-meshes with 46 spheres. Errors (blue curves) are computed as

$$MSE(f) = \frac{\sum_i \mu_i^f \,||\sum_j w_{ij}(R_j^f \cdot p_i + T_j^f) - v_i^f||^2}{BBD^2 \sum_i \mu_i^f}. \qquad (13)$$

We also plot the errors for skinning decompositions with 34 (green curves) and 22 (red curves) spheres. As pointed out in Section 5, the two horse sequences strongly differ in shape and local density and exhibit differences locally in their type of structures (volumetric/surface), even though both input animations share the same mesh topology and connectivity. These examples show the strength of our approach: when converting the mesh animations into linear blend skinning, our animated sphere-mesh adapts automatically and appears as a flexible rig structure that naturally takes the

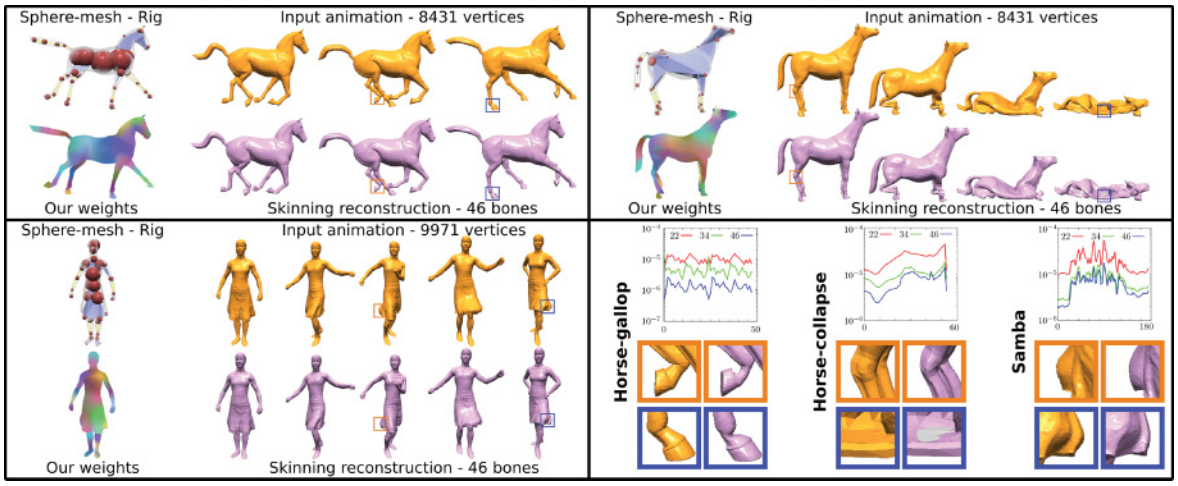


Fig. 10. Skinning reconstructions of three sequences with 46 bones. Close-ups show artifacts that are essentially loss of volume or presence of self-intersections. **Bottom right:** mean squared reconstruction error for 46 (blue), 34 (green), and 22 (red) bones. y-axis is logarithmic.

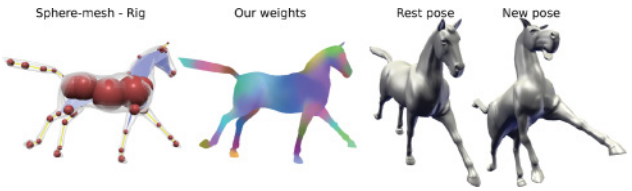


Fig. 11. **Pose editing using mixed weights:** skinning weights learned solely from the *Horse* motion provide a faithful decomposition for the body and the legs but lack a proper layout for the head, while our mixed weights also account for geometry and provide the necessary degrees of freedom to alter the grouped geometric features, even if they collectively undergo the same motion.

deformations of the input mesh into account, resulting in either a more volumetric or a more surface-oriented representation.

Using the animated sphere-mesh as a prior to derive the bones impacting the reconstruction of each original vertex leads to a fast algorithm: all our skinning decompositions were performed in a couple of minutes. Moreover, by initializing our iterative optimization scheme with smooth geometric weights, we output weight maps that are efficiently learned from the animation wherever it is pertinent, while falling back to the geometric weights elsewhere.

Figure 11 shows an example of a new pose generated using our mixed weights. While traditional skinning decomposition methods can learn weights on the body of the model based on its motion in the raw sequence, they fail at providing the user with deformation handles in the parts of the shape that are deformed rigidly, even if they exhibit interesting structures (e.g., the head). On the contrary, our mixed weights account for both details in the motion and in the geometry, leading to an optimized mix of both properties.

7. DISCUSSION

Skinning Decomposition. Table II summarizes the differences from previous works.

(i) Our method is the first to take a geometry (i.e., the animated sphere-mesh) as input, whereas other methods require a desired number of bones. Hereby, a user can easily determine the parameters to obtain the expected precision and wanted level of detail. This is critical, especially if the skinning decomposition method requires several minutes to compute; for example, Le and Deng [2014] report an execution time of 384 minutes for the decomposition of the *Samba* sequence on a commodity laptop similar to ours.

(ii) The comparably high performance of our method results from the use of a coarse geometry (our animated sphere-mesh), which is optimized to respect the geometric details that are present in the input animation. It is even several orders of magnitude faster than the method of Le and Deng [2014] (all our decompositions were performed in less than 3 minutes, even for the aforementioned *Samba* sequence—a speedup of two orders of magnitude).

(iii) Our method is the only existing one to output mixed weights, which leads to important additional handles for editing applications, which is one of the main goals of previous work.

(iv) On the downside, similarly to the method of Le and Deng [2012], our method outputs weights w.r.t. the spheres in our examples, which are free handles and are not constrained to respect any rigidity during the animation; this differs from the explicit edges of Le and Deng [2014], which have fixed length over the animation and their extremities connected by joints. However, not any animation can be approximated by a rigid skeleton (e.g., sequences with large stretching).

Table II. Our Method Takes a Geometry (GEO) as Input Parameters for the Decomposition, Whereas Others Take a Number of Bones (NOB). It Is the Only One to Output Mixed Weights. Our Method Uses Free Handles (FH: The Spheres of the Sphere-Mesh), Contrary to Le and Deng [2014], Which Outputs Rigid Bones

	Ours	Le and Deng [2012]	Le and Deng [2014]
Input Param.	GEO	NOB	NOB
Rapidity	++	+	−
Mixed weights	Yes	No	No
Handles	FH	FH	BONES

Finally, just like other skinning decomposition techniques, relying on the input connectivity for the geometric construction of the skeletal domain can lead to poor animation reproduction. Currently,

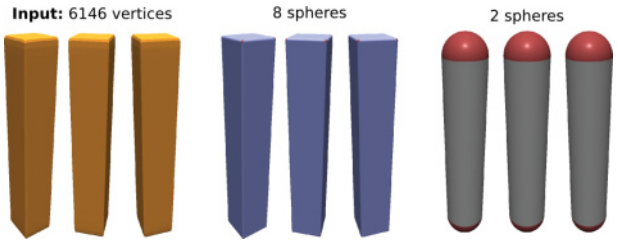


Fig. 12. The input capsule is undergoing rotation around its dominant axis. The motion is still visible when using an eight-sphere approximation, whereas two spheres are not sufficient to capture it.

an input sequence containing a lot of disconnected parts leads to a disconnected animated sphere-mesh.

Memory Complexity. Presently, the input animation has to reside in memory, preventing us from approximating very large animations. We thus cannot immediately generalize our approach to a streaming scenario where a sphere-mesh would be refined on demand. A possible solution could be to rely only on an analysis of the most significant (e.g., least redundant) poses to compute the animated sphere-mesh. This one could then be fit to the other poses.

Surface Extraction. As in Thiery et al. [2013], a surface reconstruction based on the animated sphere-mesh can be seen as a Minkowski sum of an oriented mesh and a sphere with spatially varying radius (some triangles may be double-sided though, and wire edges describe complex orientations, as the degeneracy of a cylinder over themselves). Yet, we extract the outer surface of the complete interpolation of the spheres over the sphere-mesh, which is equivalent to the Minkowski sum only when all triangles are double-sided and there is no boundary in the sphere-mesh.

Noise. Filtering topological and geometric noise from the input sequence prior to our approximation scheme is clearly a promising direction for future research. While the former may involve highly nontrivial nonhomotopic mappings, the latter could be adapted from existing filtering techniques for static meshes.

Abstraction Limitations. Finally, extreme simplifications of the input mesh can fail to depict the motion of the input sequence. Figure 12 illustrates this point: when an eight-sphere approximation of the capsule is used, the motion is still visible, whereas it becomes invisible when only two spheres are used. As future work, one could think of attaching a local frame to each sphere and encode an *interpolating function* on each edge and triangle so as to encode a local coordinate system to the animated sphere-mesh. This could be a step toward parameterizing the animated sphere-mesh (e.g., for texturing).

Conclusion

We proposed a shape approximation algorithm efficiently converting an animated mesh sequence into an *animated sphere-mesh*, which is a mesh indexing a set of animated spheres. To do so, we introduced a new optimization scheme tailoring the animated spheres robustly to capture the animated shape at a given level of detail. Additionally, we showed how connectivity and temporal coherence can be optimized. The resulting animated sphere-mesh models the animated mesh sequence from a fine-resolution surface representation to a coarse volumetric one, based on a single user-defined scale value, which still captures the dominant motions and geometric entities in the raw data, even at the coarsest levels.

In contrast to skeleton-based or cage-based performance-capture reverse-engineering systems, our alternative can locally model tubular structures and provide a convincing volumetric approximation for all other components. We demonstrated its effectiveness on a collection of nontrivial examples and compared it to purely surface-based approximation methods.

Based on the resulting animated sphere-mesh, we showed how to rig a single mesh of the original sequence with it, reproducing faithfully the full animated sequence. The underlying linear-blend skinning map is smooth and accounts for both the animation and the geometry of the original sequence. Hereby, regions with poor motion but salient structures, as well as simple geometry with singular motions, are rendered editable.

Our work is the first to output an animated volumetric structure to approximate animated 3D surfaces. Volumetric structures of static geometry already have many applications. Hand-designed animated generalized cylinders are used for tasks such as collision detection in modern games [Sambavaram 2007]. We showed that our approach leads to high-quality skinning decompositions and offers flexibility to the artists in this context. Our work has the potential to serve as an enabler for future work, as advanced computer graphics frameworks may build upon our representation to address new challenges, and we believe it is a step toward a unified framework for volumetric shape and motion modeling and analysis.

APPENDIX

A. QUADRIC MINIMIZATION (NOT INVERTIBLE)

Minimize $E(\bar{\mathbf{s}}) = \frac{1}{2}\bar{\mathbf{s}}^{\mathrm{T}} \cdot \bar{A} \cdot \bar{\mathbf{s}} - \bar{b}^{\mathrm{T}} \cdot \bar{\mathbf{s}}$, with $\bar{A}$, $\bar{b}$ of the form given by Equation (10), subject to $0 \leq \lambda \leq 1$ and $0 \leq r \leq R$.

The global minimizer $\bar{\mathbf{s}}^*$ (without inequality constraints) is given by $\bar{A} \cdot \bar{\mathbf{s}}^* = \bar{b}$, leading to

$$\begin{cases} \mu\lambda + \nu r = \beta_1 & (E1) \\ M_f \cdot q_f + r N_f = b_f \; \forall f > D & (E2_f) \\ \nu\lambda + \sum_{f>D} N_f{}^{\mathrm{T}} \cdot q_f + W r = \beta_2 & (E3)\,, \end{cases}$$

$$\Leftrightarrow \begin{cases} \mu\lambda + \nu r = \beta_1 & (E1 := E1) \\ \nu\lambda + W_2 r = \beta_3 & (E2 := E3 - \sum_{f>D} N_f{}^{\mathrm{T}} M_f^{-1} E2_f) \\ q_f = M_f^{-1} \cdot (b_f - r N_f) \, \forall f > D & (E3_f := M_f^{-1} \cdot (E2_f - r N_f))\,, \end{cases}$$

with $W_2 := W - \sum_{f>D} N_f{}^{\mathrm{T}} M_f^{-1} N_f$ and $\beta_3 := \beta_2 - \sum_{f>D} N_f{}^{\mathrm{T}} M_f^{-1} b_f$.

The solution without inequality constraints is therefore given by

$$\begin{cases} \begin{pmatrix} \lambda^* \\ r^* \end{pmatrix} = \operatorname{argmin} \left\| \begin{bmatrix} \mu & \nu \\ \nu & W_2 \end{bmatrix} \cdot \begin{pmatrix} \lambda \\ r \end{pmatrix} - \begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} \right|^2 \\ q_f^* = u_f + \lambda^* \vec{d}_f & \forall f \leq D \\ q_f^* = M_f^{-1} \cdot (b_f - r^* N_f) & \forall f > D. \end{cases}$$

We note that $C := \begin{bmatrix} \mu & \nu \\ \nu & W_2 \end{bmatrix}$ and $\mathcal{D}_R := [0,1] \times [0,R]$. Note that $\begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} \in Im(C)$; otherwise, the gradient of the quadric could never be null. Noting $C^\dagger$ as the pseudo-inverse of C, several cases need to be considered based on the dimension of C's kernel $\dim(Ker(C))$:

(1) If $\dim(Ker(C)) = 0$ ($\Leftrightarrow Ker(C) = \emptyset$):
—If $\begin{pmatrix} \lambda^* \\ r^* \end{pmatrix} = C^{-1} \cdot \begin{pmatrix} \beta_1 \\ \beta_3 \end{pmatrix} \in \mathcal{D}_R$:

$$\begin{cases} \lambda = \lambda^* \, r = r^* \\ q_f = u_f + \lambda \vec{d}_f & \forall f \leq D \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$

—otherwise: optimize on $\partial\mathcal{D}_R$ (see next paragraph)

(2) if $\dim(Ker(C)) = 1$:
—If $\{C^\dagger \cdot \begin{pmatrix}\beta_1\\ \beta_3\end{pmatrix} + Ker(C)\} \cap \mathcal{D}_R \neq \emptyset$, then choose $\begin{pmatrix}\lambda\\ r\end{pmatrix}$ in $C^\dagger \cdot \begin{pmatrix}\beta_1\\ \beta_3\end{pmatrix} + Ker(C)$ with smallest radius, and
$$\begin{cases} q_f = u_f + \lambda \vec{d}_f & \forall f \leq D \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$
—otherwise: optimize on $\partial\mathcal{D}_R$ (see next paragraph)

(3) otherwise: the space of solutions of $\overrightarrow{\nabla}_{\mathbb{S}} E = \vec{0}$ is a two-dimensional space, and we choose the solution
$$\begin{cases} \lambda = 1/2\, r = 0 \\ q_f = u_f + 1/2 \vec{d}_f & \forall f \leq D \\ q_f = M_f^{-1} \cdot b_f & \forall f > D \end{cases}$$

Optimization on $\partial\mathcal{D}_R$. $\partial\mathcal{D}_R$ is composed of four segments: $\partial\mathcal{D}_R = \{0,1\} \times [0,R] \cup [0,1] \times \{0,R\}$. The minimizer on $\partial\mathcal{D}_R$ is then the solution found with minimal cost over these segments.

Fixing λ to $\hat{\lambda} \in \{0,1\}$: The minimizer on $\{\hat{\lambda}\} \times \mathbb{R}$ is given by
$$\begin{cases} r^* = \frac{\beta_4 - \sum_{f>D} {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot b_f}{W - \sum_{f>D} {N_f}^{\mathrm{T}} \cdot M_f^{-1} \cdot N_f} \\ q_f^* = M_f^{-1} \cdot (b_f - r^* N_f) & \forall f > D \end{cases}$$
with $\beta_4 := \beta_2 - \hat{\lambda}\nu$.

If the denominator of r^* is null, we ignore this step and fix r as well to 0 and R and keep the solution with minimal cost. Otherwise:

—If $0 \leq r^* \leq R$: the minimizer on $\{\hat{\lambda}\} \times [0,R]$ is given by
$$\begin{cases} r = r^* \\ q_f = u_f + \hat{\lambda}\vec{d}_f & \forall f \leq D\,. \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$
—Otherwise, we fix r as well to 0 or R and keep the solution with minimal cost.

Fixing r to $\hat{r} \in \{0,R\}$: The minimizer on $\mathbb{R} \times \{\hat{r}\}$ is given by
$$\begin{cases} \lambda^* = (\beta_1 - \nu\hat{r})/\mu \\ q_f^* = M_f^{-1} \cdot (b_f - \hat{r} N_f) & \forall f > D. \end{cases}$$
If the denominator μ is null, we ignore this step and fix λ as well to 0 and 1 and keep the solution with minimal cost. Otherwise:

—If $0 \leq \lambda^* \leq 1$: the minimizer on $[0,1] \times \{\hat{r}\}$ is given by
$$\begin{cases} r = \hat{r} \\ q_f = u_f + \lambda^* \vec{d}_f & \forall f \leq D\,. \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D \end{cases}$$
—Otherwise, we fix λ as well to 0 or 1 and keep the solution with minimal cost.

Fixing (λ, r) to $(\hat{\lambda}, \hat{r}) \in \{0,1\} \times \{0,R\}$: The minimizer of the energy when fixing r and λ is simply given by
$$\begin{cases} r = \hat{r} \\ q_f = u_f + \hat{\lambda}\vec{d}_f & \forall f \leq D \\ q_f = M_f^{-1} \cdot (b_f - r N_f) & \forall f > D. \end{cases}$$

B. INTERSECTION WITH SPHERE-SEGMENTS

We aim at detecting the intersection between a sphere-segment $[(e_0;r),(e_1;r)]$ (a sphere with radius r traveling continuously from point e_0 to point e_1) and a sphere-mesh. This can be achieved by testing the intersections between the segment and the various geometric primitives of the sphere-mesh and selecting the closest valid one. In the following, we describe the cases for intersecting the segment with spheres, capsules, and thick triangles. We note $e_\lambda := e_0 + \lambda\overrightarrow{e_0 e_1}$ as a parameterization of (e_0, e_1).

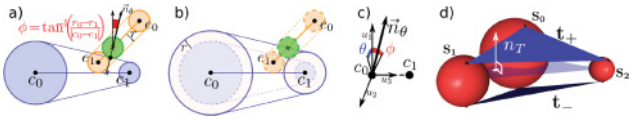


Fig. 13. Intersecting a capsule (blue) and a sphere-segment (orange) (a) is equivalent to intersecting a thicker capsule and a segment (b). (c): Cone's normal parameterization. (d): Interpolation of three spheres on a (transparent blue) triangle.

Against spheres: There exists an intersection with a sphere $(C;R)$ if the following can be satisfied:
$$\exists \lambda \in [0,1] \mid ||e_\lambda - C||^2 = (R+r)^2, \tag{14}$$
which boils down to solving a simple quadratic polynomial in λ.

Against capsules: Detecting the intersection between the segment and a capsule defined as the interpolation of two spheres $(c_0;r_0)$ and $(c_1;r_1)$ (with $r_0 \geq r_1$) is equivalent to detecting the intersection between the segment $[e_0,e_1]$ and the capsule thickened by r (see Figures 13(a) and 13(b)). We will therefore focus on the second problem. We note $c_\mu := c_0 + \mu\overrightarrow{c_0 c_1}$ as a parameterization of the line (c_0,c_1), and $\vec{n}_\theta$ as one of the cone's normal ($\vec{n}_\theta := \cos(\theta)\cos(\phi)u_1 + \sin(\theta)\cos(\phi)u_2 + \sin(\theta)u_3$, see Figure 13(b)).
There exists an intersection if the following can be satisfied:
$$\exists (\lambda,\mu,\theta) \mid e_\lambda = c_\mu + (r_0 + r + \mu(r_1 - r_0))\vec{n}_\theta. \tag{15}$$

Computing $(\text{Equation (15)})^{\mathrm{T}} \cdot u_3$ and $((\text{Equation (15)})^{\mathrm{T}} \cdot u_1)^2 + ((\text{Equation (15)})^{\mathrm{T}} u_2)^2$ gives
$$\lambda\overrightarrow{e_0 e_1}^{\mathrm{T}} \cdot u_3 - \mu(\overrightarrow{c_0 c_1}^{\mathrm{T}} \cdot u_3 + (r_1 - r_0)\sin(\phi)) = \overrightarrow{e_0 c_0}^{\mathrm{T}} \cdot u_3 + r_0 \sin(\phi)$$
$$(\overrightarrow{c_0 e_0}^{\mathrm{T}} \cdot u_1 + \lambda\overrightarrow{e_0 e_1}^{\mathrm{T}} \cdot u_1)^2 +$$
$$(\overrightarrow{c_0 e_0}^{\mathrm{T}} \cdot u_2 + \lambda\overrightarrow{e_0 e_1}^{\mathrm{T}} \cdot u_2)^2 = (r_0 + r + \mu(r_1 - r_0))^2 \cos(\phi)^2.$$

Combining these last equations leads to a simple second-order polynomial in (λ,μ). Solving this quadric provides the solution, assuming it respects: $0 \leq \lambda, \mu \leq 1$.

Against thick triangles: The intersection between the segment and a thick triangle defined as the interpolation of the three spheres $(c_0;r_0)$, $(c_1;r_1)$, and $(c_2;r_2)$ with normal n_T can be found by checking the intersections against the capsules (edges of the thick triangle) and two additional triangles $t_+ = (p_0^+, p_1^+, p_2^+)$ and $t_- = (p_0^-, p_1^-, p_2^-)$ (see Figure 13(c)), which are constructed as follows:

—Compute $\phi_{ij} = \tan^{-1}((r_i - r_j)/||c_j - c_i||), \forall 0 \leq i < j \leq 2$.
—Compute p_i as the intersection of the three planes:
—$\{\overrightarrow{c_i p_i}^{\mathrm{T}} \cdot n_T = 0\}$,
—$\{\overrightarrow{c_i p_i}^{\mathrm{T}} \cdot \overrightarrow{c_i c_j} = r_i ||\overrightarrow{c_i c_j}|| \sin(\phi_{ij})\} \forall j \neq i, 0 \leq j \leq 2$.
—Compute $p_i^{+/-}$ as $p_i^{+/-} = p_i +/- \sqrt{r_i^2 - ||\overrightarrow{c_i p_i}||^2} n_T$.

Note that t_+ and t_- do not always exist (e.g., if the three sphere centers are coplanar).

ACKNOWLEDGMENTS

The captured performance data were provided courtesy of the research group 3D Video and Vision-based Graphics of the

Max-Planck-Center for Visual Computing and Communication (MPI Informatik/Stanford).

REFERENCES

I. Baran and J. Popović. 2007. Automatic rigging and animation of 3d characters. *ACM Transactions on Graphics (TOG)* 26 (2007), 72.

J. Bloomenthal and K. Shoemake. 1991. Convolution surfaces. In *ACM SIGGRAPH Computer Graphics*, Vol. 25. ACM, 251–256.

H. Blum. 1967. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, Weiant Wathen-Dunn (Ed.). MIT Press, Cambridge, 362–380.

P. Cignoni, C. Rocchini, and R. Scopigno. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174.

E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. 2008a. Performance capture from sparse multi-view video. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 98.

E. de Aguiar, C. Theobalt, S. Thrun, and H.-P. Seidel. 2008b. Automatic conversion of mesh animations into skeleton-based animations. *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 389–397.

R. de Toledo and B. Lévy. 2008. Visualization of industrial structures with implicit GPU primitives. In *Proceedings of the International Symposium on Visual Computing (ISVC'08)*.

C. DeCoro and S. Rusinkiewicz. 2005. Pose-independent simplification of articulated meshes. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. ACM, 17–24.

M. Garland and P. S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., 209–216.

B. Gärtner and T. Herrmann. 2001. Computing the width of a point set in 3-space. In *Abstracts for the 13th Canadian Conference on Computational Geometry* (2001), 101–103.

H. Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 99–108.

J. Houle and P. Poulin. 2001. Simplification and real-time smooth transitions of articulated meshes. In *Graphics Interface 2001*. 55–60.

F.-C. Huang, B.-Y. Chen, Y.-Y. Chuang, and M. Ouhyoung. 2005. Animation model simplifications. In *Computer Graphics Workshop*.

A. Jacobson, I. Baran, J. Popovic, and O. Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics* 30, 4 (2011), 78.

D. L. James and C. D. Twigg. 2005. Skinning mesh animations. *ACM Transactions on Graphics (TOG)* 24 (2005), 399–407.

L. Kavan, P.-P. Sloan, and C. O'Sullivan. 2010. Fast and efficient skinning of animated meshes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 327–336.

S. Kircher and M. Garland. 2005. Progressive multiresolution meshes for deforming surfaces. In *Proceedings of the 2005 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation*. ACM, 191–200.

E. Landreneau and S. Schaefer. 2009. Simplification of articulated meshes. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 347–353.

C. L. Lawson and R. J. Hanson. 1974. *Solving Least Squares Problems*. Vol. 161. SIAM.

B. H. Le and Z. Deng. 2012. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 199.

B. H. Le and Z. Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 84.

J. P. Lewis, M. Cordner, and N. Fong. 2000. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., 165–172.

N. Magnenat-Thalmann, R. Laperrire, and D. Thalmann. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface*. Citeseer.

J. McCormack and A. Sherstyuk. 1998. Creating and rendering convolution surfaces. In *Computer Graphics Forum*, Vol. 17. Wiley Online Library, 113–120.

A. Mohr and M. Gleicher. 2003. Deformation sensitive decimation. *Technical Report* (2003).

S. Muraki. 1991. Volumetric shape description of range data using "blobby model". *SIGGRAPH Computer Graphics* 25, 4 (July 1991), 227–235. DOI:http://dx.doi.org/10.1145/127719.122743

F. Payan, S. Hahmann, and G.-P. Bonneau. 2007. Deforming surface simplification based on dynamic geometry sampling. In *IEEE International Conference on Shape Modeling and Applications, 2007 (SMI'07)*. IEEE, 71–80.

R. Sambavaram. 2007. Cylinder Collision. Insomniac Games Tech Team Presentation. Retrieved from http://www.insomniacgames.com/supporting-cylinder-collision/.

K. Siddiqi and S. M. Pizer. 2008. *Medial Representations: Mathematics, Algorithms and Applications*. Vol. 37.

S. Stolpner, P. Kry, and K. Siddiqi. 2012. Medial spheres for shape approximation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 6 (June 2012), 1234–1240. DOI:http://dx.doi.org/10.1109/TPAMI.2011.254

J. O. Talton. 2004. A Short Survey of Mesh Simplification Algorithms. University of Illinois at Urbana-Champaign (2004).

J.-M. Thiery, E. Guy, and T. Boubekeur. 2013. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 178.

D. Vlasic, I. Baran, W. Matusik, and J. Popović. 2008. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 97.

R. Wang, K. Zhou, J. Snyder, X. Liu, H. Bao, Q. Peng, and B. Guo. 2006. Variational sphere set approximation for solid objects. *Visual Computer* 22, 9 (2006), 612–621.

C. Zanni, A. Bernhardt, M. Quiblier, and M.-P. Cani. 2013. SCALe-invariant integral surfaces. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 219–232.

S. Zhang, J. Zhao, and B. Wang. 2010. A local feature based simplification method for animated mesh sequence. In *Proceedings of the 2010 2nd International Conference on Computer Engineering and Technology (ICCET'10)*, Vol. 1. IEEE, V1–681.

Received October 2015; revised February 2016; accepted February 2016