Abstract

We give a method to enrich arbitrary triangle mesh animations with physics. The input is a triangle mesh animation obtained using any method, such as keyframed animation or 3D scanned using computer vision techniques. The user provides weights for how much physics should affect the output animation in any region of the model. Our method then solves for a physically-based animation that is constrained to the input animation to the degree prescribed by these weights. This permits parts of the model to strictly follow the input positions, to be completely constraint-free and purely follow physics, and anything in between. The output is a triangle mesh animation that has been enriched with physics, providing secondary motion and collision resolution. Our method can also infer missing data in a triangle mesh, making it possible to use it to complete deformed animation frames based on a rest configuration template.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling, I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: triangle mesh animation, physically based modeling, directable simulation, FEM

1 Introduction

In computer animation practice, both traditional animation and physically based methods are widely used. Traditional animation gives the artist full control, but also burdens her with the task of maintaining realism. Physically based methods easily provide realism, but are difficult to direct. Although progress has been made, it is still difficult to use the two techniques together.

In this paper, we give an approach to combine triangle mesh animations with physics. The input to our method is a triangle mesh animation, obtained using any method, e.g., artist keyframe animation, or scanned from real subjects using computer vision. While many methods exist to provide physics on a skinned or rigged input, we aim to provide the solution for general triangle mesh animations. The input animation may be crude, non-physical or incomplete. Our method works as follows:

1. We first convert the input into a physically based form, by fitting a tetrahedral mesh animation to the embedded triangle mesh input animation (“inverse caging”).

2. The tet mesh animation is then simulated using physics, subject to the constraint that it has to follow the input motion.

3. The user controls the degree to which input is followed by painting a scalar field over the input mesh. This makes it possible to strictly follow the input in parts of the mesh, let the simulation evolve completely based on physics in other parts, and anything in between. To achieve this, we provide two methods: (1) using an implicit driving force during the simulation, and (2) blending the shapes after the simulation.

The end result is an artistically directed simulation: it conforms to the input by prescribed amounts, but is enriched with physics, secondary motion and collision resolution. Our method can also infer missing data in a triangle mesh, making it possible to use it to complete deformed animation frames based on a rest configuration template.

Figure 2 gives an overview of our system; we further elaborate on it in Section 3. We explain the simulation mesh creation and fitting to input in Section 4. We describe the constrained simulation which produces physical motions on the selected parts of the mesh in Section 5. The two methods of balancing between input motion and physical results are addressed in Section 6. Finally, we show experiment results in Section 7 and conclude in Section 8.
2 Related work

Adding Physics: Several methods ([Bergou et al. 2007; Wang et al. 2010; Rohmer et al. 2010]) enriched coarse cloth animations with spatially high-frequency effects such as wrinkles and folds. Among them, Bergou achieved this by running a detailed mesh simulation that is constrained to the input coarse animation. We also use constraint-based dynamics. However, our method is opposite to them: it adds physics at the coarse scale resolved by the tet mesh and we give a way to preserve input animation high-frequency detail that cannot be resolved by the tet mesh. The simulations in [Bergou et al. 2007] usually run on high-resolution meshes whereas our method is designed to run on a coarse simulation mesh. Wiggly splines were introduced to provide direct manipulation and physical oscillatory motion [Kass and Anderson 2008]. Shi [2008] enhanced skeleton-driven animations with secondary motion extracted from sample sequences. Unlike prior methods, we give artists the freedom to spatially balance between physical output and artist input, and we address contact. Hahn [2012; 2013] formulated the equations of motions in the rig space so that some rig parameters can evolve via physics. Several researchers employed physically based skinning to generate physical behaviors on skinned meshes [Capell et al. 2002; McAdams et al. 2011; Kim and Pollard 2011; Faure et al. 2011; Liu et al. 2013]. Our method targets general triangle mesh animations where no rig or skinning is provided, such as our head example (Figure 1). It operates in the full FEM space without model reduction, simulating both global and local deformations. It supports any triangle mesh input animation and preserves input animation detail.

Animation Design and Control: Several papers addressed the design and editing of elastic physically based simulations using keyframes [Barbić et al. 2009; Huang et al. 2011; Hildebrandt et al. 2012] or spacetime constraints [Li et al. 2014; Li et al. 2013; Schulz et al. 2014]. Additionally, Kondo [2005] allowed animation editing by both keyframes and trajectories. Barbić [2008] directed physical simulations to given input trajectories. Coros [2012] controlled deformable objects by changing their rest shapes. Barbić [2012] introduced spacetime Greens functions to allow interactive deformable object animation editing. These papers focus on direct user control whereas our goal is to enrich an existing animation with physics. Most previous papers assume that a volumetric mesh animation is given as input, whereas we provide a way of fitting it to the input triangle mesh animation.

Volumetric Mesh Tracking and Fitting: When fitting a volumetric mesh animation to a triangle mesh animation, the surface vertices of the volumetric mesh are usually over-determined by the given triangle mesh geometry, whereas the interior vertices are in the opposite situation. Therefore, a smoothing or elastic energy term is usually used to determine the positions of the interior vertices. Choi and Szymczak [2009] proposed fitting volumetric meshes to animated surfaces by minimizing a warped linear elasticity energy and surface distance. Wuhrer et al. [2015] used point clouds that are a subset of the surface vertices of the volumetric mesh to infer material properties, forces and interior displacements. Paillé et al. [2013] proposed a method to compress a volume grid to fit its boundary to the input surface. Volumetric mesh tracking for 2D and 3D images is also used in biomechanics [Noe and Sørensen 2010]. Our volumetric mesh fitting process differs from these methods in that these methods tracked the surface of a volumetric mesh. In contrast, we work with embedded simulation, a widely useful technology in computer animation where the tracked triangle mesh is not necessarily on the volumetric mesh surface, but may be embedded (deep) into the volumetric mesh, with the goal of subsequent physically based animation.

Inverse Caging: Inverse caging is the process of finding a deformed cage (and sometimes its weights) that best interpolates the input vertices. The vertices of a volumetric mesh can be treated as the handles of a cage that controls an embedded mesh. Lu et al. [2014] and Savoye and Franco [2010] adopted Laplacian coordinates to deform or fit a cage. Chen et al. [2010] used Green coordinates as cage weights and transferred the deformation gradients of the input mesh to a cage. In order to address over-determined systems occurring in the fitting problem, Thierry et al. [2012] computed the maximum volume submatrix of the interpolation matrix, to fit a cage animation to a triangle mesh animation. These methods focus on geometric caging where the cage is typically a surface mesh, with applications in animation compression and geometric editing. We focus on volumetric mesh fitting using a 3D solid physical energy, including interior vertices, for physically based simulation. Our method can handle dynamic simulation contact, e.g., interior collisions in concave regions (Figure 8). Barbić et al. [2009] used a related method to generate volumetric mesh deformations from given triangle meshes. However, they only used fitted deformations as animation keyframes, whereas we fit meshes to an entire animation. Therefore, we do not minimize a fitting energy individually and independently for every frame. Instead, for each frame, we approximate the fitting energy using a second-order expansion around the previous frame, and then minimize the quadratic function using a linear system solve; such temporal coherence greatly accelerates the fitting process. In contrast to their model reduction work which is limited to global low-frequency deformations, our outputs are full-dimensional, model local deformations, support contact and preserve input animation detail.

3 Overview

The input to our method is a neutral pose of a detailed triangle mesh \( \Gamma \), and an animation of a submesh \( \Gamma' \subseteq \Gamma \). The animation of \( \Gamma' \) consists of frames \( i = 0, \ldots, T \), each given by a displacement vector away from the neutral pose of \( \Gamma \). This input animation can be created using any means available: it can be keyframe- and itself animated using some physical process, or it can be scanned from a real subject. In several examples, we will simply have \( \Gamma' = \Gamma \). The \( \Gamma' \subseteq \Gamma \) case occurs with the human face geometry, where \( \Gamma \) refers to the entire human head, including the complete lips (both inside and outside the mouth cavity) and the internal mouth structure (the "artist mesh"). The computer vision tracking system, however, only manages to track a proper subset \( \Gamma' \) that misses the interior part of the lips, the internal mouth cavity and large peripheral regions of the face (Figure 1).

The output of our method is an animation of \( \Gamma \) that is identical to the input animation in a user-selected subset \( \Gamma'' \subseteq \Gamma' \). Elsewhere on \( \Gamma' \), it strikes a balance between physically-balanced simulation and following the input animation. This balance is user-controlled: it
can be spatially-varying and painted on the mesh by the artist. The animation in \( \Gamma \setminus \Gamma' \) is reasonably and automatically extended from \( \Gamma' \) using our physical process. If desired, the output animation also obeys collisions.

Our algorithm is designed so that it can optionally preserve the input detail that is beyond the resolution of the chosen tetrahedral mesh, everywhere on \( \Gamma' \). Our outputs are rich with detail, unlike typical physically based simulation that either requires a computationally slow detailed tet mesh, or else detail is destroyed using coarse meshes.

An overview of our method is shown in Figure 2. First, a tetrahedral simulation mesh is built to enclose the input triangle mesh (Section 4.1), and then fitted to the input animation (Section 4.2). We run a physically based simulation with constraints to produce physical motion on selected parts of the mesh, and optionally preserve the spatially high-frequency detail (Section 5). The user can control how much physics is added to the animation by painting weights on the tetrahedral mesh. The tradeoff between input animation and physics is achieved either via implicitly integrated forces driving the simulation to the input animation, or by geometrically blending the input with the simulation result (Section 6).

4 Creating a physical structure

In order to physically enrich the input (non-manifold) triangle mesh animation, it is first necessary to equip it with a structure capable of providing physics. In this section, we explain how we generate a simulation mesh and fit an animation of it to the input animation.

4.1 Generating a simulation mesh

Given the input triangle mesh \( \Gamma \), we create a tetrahedral simulation mesh \( \Omega \) that encloses the space occupied by the rest shape of the triangle mesh. This can be done using a variety of methods; we use [Xu and Barbi 2014] because it can handle non-manifold geometry and produces a tetrahedral mesh that encloses the space of the triangle mesh. We note that our method does not require the tet mesh to be detailed; actually, we encourage the use of coarse meshes (compared to the resolution of the input triangle mesh) as they are computationally fast and often sufficient to provide the required physical effects. By adjusting the standard tet meshing parameters, the user can obtain a tet mesh at whatever resolution desired. Our method then automatically adds physical effects that are resolved by the chosen tet mesh, but keeps the input spatially high-frequency triangle mesh deformations unmodified (Figure 3).

4.2 Fitting simulation mesh animation to input triangle mesh animation

For every input frame of the triangle mesh \( \Gamma' \), we compute a tet mesh deformation that best conforms to it. We do so using an energy function that balances the match to the input frame and the elastic strain energy of the tet mesh. Given the input triangle mesh with \( t \) vertices and its deformation \( \bar{q}_t \in \mathbb{R}^3 \) at a frame \( t \), the deformation of the tet mesh \( \bar{u}_t \in \mathbb{R}^3 \) is obtained by minimizing

\[
\bar{u}_t = \arg \min_u \frac{1}{2} \| Cu - \bar{q}_t \|^2_A + k \varepsilon(u),
\]

where \( C \in \mathbb{R}^{3t \times 3n} \) is the interpolation matrix that interpolates tet mesh deformations to the triangle mesh, \( \varepsilon(u) \) is the invertible StVK elastic energy [Irving et al. 2004] for the deformation \( u \), and \( k \geq 0 \) determines the tradeoff between the two terms. We choose this elastic energy model because it provides a fast evaluation (same magnitude of speed as the corotational linear FEM energy) and produces robust nonlinear motion [Sin et al. 2013]. The interpolation matrix simply consists of the barycentric coordinates of every triangle mesh vertex in its containing tetrahedron. One could instead adopt other coordinates to perform the interpolation, such as [Ju et al. 2005] or [Joshi et al. 2007]. Matrix \( A \) is used to weight the interpolation energy according to the surface area of each vertex. In most of our examples, we choose \( k = 10^{-7} \) when Young’s modulus for \( \varepsilon \) is \( 10^8 \text{N/m}^2 \); otherwise, \( k \) should be scaled with Young’s modulus.

To acquire a good initial guess for the first frame, we can first use the rest shape as the initial guess and apply a large \( k \) to produce a stable shape at the first frame. Then we decrease \( k \) use this shape as the initial guess and iterate until we get a satisfactory result. In practice, we test the parameters on first few input frames before applying them to all the frames. The user can try larger values if she finds the fitted shapes are not natural enough, or smaller values if the shapes do not follow the input sufficiently.

For every frame \( i \), we need the deformation of the previous frame as the initial guess. We compute the tet mesh deformation at frame \( i \) using a process similar to numerical time-stepping. We approximate the energy in Equation 1 using a second-order Taylor series in \( u \), minimize this quadratic function, and iterate. The quadratic approximation involves the tangent stiffness matrix (second derivative) of the StVK elastic energy \( \varepsilon(u) \), which can be computed easily. We compute the derivative of the fitting energy and set it to zero:

\[
C^T A (Cu - \bar{q}) + kF_{\text{int}}(u) = 0,
\]

where \( F_{\text{int}} \) is the internal elastic force. Given an initial guess \( u^0 \) for \( u \), we can approximate Equation 2 with:

\[
C^T A \left( C\Delta u + Cu^0 - \bar{q} \right) + k \left( K(u^0)\Delta u + F_{\text{int}}(u^0) \right) = 0.
\]

Here, \( K(u^0) \) is the tangent stiffness matrix of the elastic energy \( \varepsilon \) evaluated at \( u^0 \). By solving the above sparse system for \( \Delta u \), we ob-
tain the next iterative approximation for $\bar{u}_i$. After a few iterations (at most 10), we get a reliable $\bar{u}_i$, and proceed to compute the next frame using current frame as the initial guess. Such a Newton’s method is faster than a gradient-only optimization method such as the conjugate gradient optimizer [Press et al. 2007] applied directly to the energy in Equation 1, with no visible loss of quality. We observed a 30x speedup in our examples. Although our algorithm proceeds frame by frame, it can still exploit parallelism since the bottleneck of the computation is the evaluation of the second derivative of $E(u)$, which can be easily parallelized. Our solver does not guarantee temporal coherence, but in practice it gives good results without noticeable temporal artifacts. One can instead parallelize the solver among multiple frames. However, a parallel structure adds difficulty for finding a good initial guess for every thread, and the Newton’s method is not stable when starting far away from the solution, leading to blow up. Although the conjugate gradient method is more robust against arbitrary initial guesses, we find it converges so slow that it is better to use our solver sequentially than the parallelized conjugate gradient solver.

5 Physically based simulation with constraints

After the tet mesh deformations at all frames are constructed, we use them to add physics to the input triangle mesh animation. The user first selects a subset tet mesh $\Omega^T$ of $\Omega$ that will be constrained during the simulation to follow the input. This is achieved by selecting vertices on $\Gamma$ to belong to $\Gamma^0$. The tetrahedra containing vertices of $\Gamma^0$ are then found automatically. Alternatively, $\Omega^T$ can be selected on the tet mesh directly. Note that $\Omega^0$ and $\Omega\setminus\Omega^0$ may consist of multiple connected components.

We compute the animation of $\Omega \setminus \Omega^T$ using a physically based simulation where the vertices in $\Omega^T$ serve as constraints. In the continuous setting, the problem is formulated as:

$$\ddot{u} + D(u, \dot{u})\dot{u} + f_{ext}(u) = f_{ext},$$  \hspace{1cm} (4)

$$u_{\Omega^T} = \bar{u}_{\Omega^T}.$$  \hspace{1cm} (5)

Here, $M$ is the mass matrix, $u$ is the vertex displacements, $f_{ext}(u)$ is the internal force of the deformable object, $f_{ext}$ is the external force including gravity and collision force, and $D(u, \dot{u})$ is the damping matrix. We adopt Rayleigh damping in all our examples. The vertex displacements of $\Omega^T$ obtained during the fitting process and during the simulation are denoted by $\bar{u}_{\Omega^T}$ and $u_{\Omega^T}$, respectively.

We use the implicit backward Euler integrator with the Lagrange multipliers to solve this problem, similar to [Baraff and Witkin 2001; Bergou et al. 2007]:

$$\begin{bmatrix} M + \Delta \nabla D(u_k, v_k) + \Delta t^2 K(u_k) & \Delta t \nabla g_k(u_k) \\ \Delta t \nabla g_k(u_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta \nu \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} \Delta (f_{ext} - f_{int}(u_k)) - (\Delta \nabla K(u_k) + D(u_k, v_k))v_k) \\ -g_{k+1}(u_k + \Delta \nu_k) \end{bmatrix},$$  \hspace{1cm} (6)

$$g_k(u_k) = S\bar{u}_k - \bar{\bar{u}}_k,$$  \hspace{1cm} (7)

$$v_{k+1} = v_k + \Delta \nu, \text{ and } u_{k+1} = u_k + \Delta \nu v_{k+1},$$  \hspace{1cm} (8)

where $v = \dot{u}$ is the velocity, $\lambda$ are the Lagrange multipliers and $S \in \mathbb{R}^{3n\times3n}$ is the selection matrix that selects the DOFs of $\Omega^T$ from $u$. Matrix $K(u_k)$ is the tangent stiffness matrix for the elastic energy evaluated at shape $u_k$ at frame $k$. We use the same invertible StVK energy as in the fitting process because of its fast evaluation and robust nonlinear motion. Our constraint solver differs from [Bergou et al. 2007] in that our upper-left block in the system matrix uses $K(u_k)$, whereas Bergou et al. used a prediction $K(\bar{u}_k + \Delta \nu_k)$. In our experiments, we found that using $K(u_k)$ makes the stable timestep 4x larger. We attribute this difference to the prediction of [Bergou et al. 2007] being too aggressive under complex motion.

One simple optimization for the formulation described above is that we do not need all DOFs in $\Omega^T$. Because simulation is actually performed on $\Omega \setminus \Omega^T$, we only need the vertices in $\Omega^T$ that are adjacent to any vertices in $\Omega \setminus \Omega^T$ as the constraints. In other words, we remove from the simulation any vertex in $\Omega^T$ whose neighbors are all in $\Omega^T$. This way, the system dimension is often greatly reduced, accelerating simulation. To keep the integrator stable, we subdivide the animation timestep if necessary. One limitation of our constrained simulation is that the point constraints on $\Omega^T$ only provide a $C^0$ deformation continuity across the interface between $\Omega^T$ and $\Omega \setminus \Omega^T$. However, in order for non-$C^1$ continuity to be visible, shear is generally required, and solid simulations tend to avoid shear. Continuity can also be improved with sufficient stiffness around the interface, or with additional averaging or soft constraints around the interface.

Preserving spatially high-frequency detail: After the simulation, we obtain the output deformation $q_i$ of $\Gamma$ at frame $i$ as $q_i = Cu_i$. If the input triangle mesh animation contains spatially high-frequency detail, it will be lost during this interpolation step because the simulation mesh is typically much coarser than the triangle mesh. Note that this only applies to dynamic detail in the animation; any detail present in the undeformed mesh $\Gamma$ is always preserved. In our method, we preserve the dynamic detail as follows (see also Figure 4). At each frame, for each triangle mesh vertex, we store the difference between the vertex input position and the position interpolated using the fitted tet mesh deformations $\bar{u}_i$. This difference is then transformed and added to the interpolated position obtained using simulation mesh deformations $u_i$, effectively reproducing the high frequency detail. In order to properly rotate, scale and shear the detail, we transform it using the relative tet deformation gradient:

$$q_i = Cu_i + F_i(q_{i} - C\bar{u}_i),$$  \hspace{1cm} (9)

where $F_i \in \mathbb{R}^{3n\times3n}$ is a block diagonal matrix formed by $3 \times 3$ relative deformation gradients between the fitted and simulated tet containing each vertex. The result of this procedure is shown in Figure 3.

6 Balancing input motion with physics

In the previous section, the output animation comes either from physical simulation or from the input. We now describe a procedure...
to strike a balance between physics and the input animation, which enables artist control. First, the user paints weights on the simulation mesh tetrahedra representing how much the physics should affect each tet. The painting can be greatly simplified by only requiring a few sparse values, and then propagating them smoothly over the entire mesh using a Laplace solve.

Implicit driving force: We use two methods to balance input motion and physics. In the first method, we add an additional external force to drive the mesh toward the input, scaled by the weights:

$$f_w = W(\ddot{u} - u),$$  \hfill (10)$$

where \(W \in \mathbb{R}^{3n \times 3n}\) is the diagonal matrix of weights, computed by averaging weights of the neighboring tets. Higher weights generate motions closer to the input. So Equation 4 is augmented to be:

$$M\ddot{u} + D(u, \dot{u})\dot{u} + f_{int}(u) = f_{ext} + f_w.$$  \hfill (11)

We note that this force can be integrated implicitly in Equation 6:

$$\begin{bmatrix} M + \Delta t D(u_k, v_k) + \Delta t^2 (K(u_k) + W) \\ \Delta t V g_k(u_k) \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} 0 \\ \Delta t (f_{ext} - f_{int}(u_k) + f_w(u_{k+1}) - (\Delta t (K(u_k) + W) + D(u_k, v_k)) v_k) - g_{k+1}(u_k + \Delta u_k) \end{bmatrix}.$$  \hfill (12)

The term \(f_w\) will only increase the diagonal entries in the system matrix, making the matrix more well-conditioned. So, even very large weights will not compromise stability. We have experimentally observed stable convergence to \(\ddot{u}\) as \(W\) grows to extremely large values, well beyond values that produce visually identical motions as \(\ddot{u}\). It is convenient for the user to express weights on the interval \([0,1]\); we re-map them to \([0,\infty)\) using the tan\((2K\zeta/\pi)\) function, where the user can adjust a global parameter \(\zeta > 0\). Collision and contact can be resolved during the simulation; no special handling is required. For simplicity, we use the penalty-based method to handle contact, but any contact resolution method could be used.

Post-simulation blending: The second approach uses as-rigid-as-possible interpolation [Alexa et al. 2000; Huang et al. 2011], where we simply blend the output physical shapes and the input shapes. We decompose the deformation gradient for each tet into a rotation and symmetric matrix, blend them separately according to the weights, and finally solve a linear system to combine the deformation gradients of all the tets as in rotation-strain coordinate warping [Sifakis and Barbic 2012]. This procedure does not require the tan function (weights can be used directly on \([0,1]\)) and it also permits easy post-simulation tuning of weights, but cannot directly incorporate contact.

If the simulation is fast, we recommend using the implicit driving force for its ability to handle contact. In the opposite situation where the simulation is too expensive, the user can adjust the motion faster by post-simulation blending, as the tweaking of the post-simulation blending weights does not require re-simulation.

Self-colliding rest configurations: When the undeformed triangle mesh has regions very close to each other, it is difficult to construct a tet mesh without welding these parts together. To address this problem, the user selects a frame \(k\) where the regions are more separated. We then build a tet mesh and interpolation matrix using the shape of frame \(k\). The tet mesh fitting method (Section 4.2) is then applied to the deformation \(-g_k\), producing a smooth tet mesh around the undeformed triangle mesh. We then use this tet mesh everywhere in our paper. Note that this tet mesh may be self-colliding, but all the proximity regions are topologically separated. The previous interpolation matrix is no longer valid and we need to update the barycentric coordinates. Although some tets may be colliding, we identify the tet “owning” each triangle mesh vertex as the tet that owned it at frame \(k\), and then compute the barycentric weights.

We applied this procedure in our elephant and face examples. If the user cannot find a suitable frame to resolve close regions, she can create an additional safe shape and continue the above procedure.

7 Results

In our first example (Figure 6), we enrich a walking elephant motion with physics. The motion was created in Maya by an artist using rigging and keyframing. We use our method to add secondary motion to the ears, trunk, belly and the tail, by selecting these regions and putting them into \(\Gamma^w\). For each of these regions, we also adjusted a single Young’s modulus value to make the region more or less bouncy. Statistics are available in Table 1. Collision detection and contact resolution was enabled for the simulation. Figure 6 also shows our blended result, where we blend between a physically based simulation and the input, using the blending method of Equation 10. In Figure 9, we demonstrate that we can also enrich input artist animations both with physically based secondary motion and physically based contact. Such contact can be useful when the artist already designed the primary motion of a character, and then wants to add additional (deformable or rigid) objects into the scene that collide with the deformable character, without substantially changing the original character motion.

In our second example, we add physics to a keyframed bird animation (Figure 5). The model and the animation were purchased online in a 3D model store. We place the legs into \(\Gamma^w\), so that the animator’s previous work on ground contact is preserved. Similarly, we place the ends of the hands into \(\Gamma^w\), so that the hand motion follows the input, which can be seen as a variant of IK. The rest of the hand including the arm feathers, however, are placed into \(\Gamma^w\),
which gives good secondary motion. We also add physics to the head and the eyes.

In our third example, we enrich the motion of a dancing sumo wrestler with secondary motion on the belly, cheeks, derrière and thighs (Figure 7). We place the hands and legs (except thighs) into $\Gamma''$ to follow the artist’s input. We also put the neck, and parts of the back and the hips into $\Gamma''$, confining the motion to reasonable poses. We put the lower half of the head into $\Gamma \backslash \Gamma''$, to produce secondary motion on the cheeks. The rest of the head is in $\Gamma''$. Our method produces large deformations when the wrestler is dancing and jumping. The thighs and cheeks exhibit subtle jiggling motions.

In our fourth example, we complete the deformed shapes of a human head triangle mesh $\Gamma$, based on the input motion of a submesh $\Gamma'$ (Figure 1). The input motion was computed using an optical flow-like tracking algorithm at a major computer graphics company, and is missing inner lips and all the mouth cavity detail due to occlusions. We compared our method to surface deformation methods such as [Sorkine and Alexa 2007] (Figure 8) and found that they may produce collisions, especially in the mouth cavity and the triangles inside the eyes because they contain extremely concave geometry. Surface deformation methods also require one connected surface mesh, whereas in the head model the tongue and teeth are separate triangle shells; we removed them for this comparison. We created a tet mesh $\Omega$ for the entire head using the procedure described in Section 6. Then we fitted tet mesh deformations $\bar{u}_i$ based on the input tracked triangle mesh motion. Since the tracked data already contains facial dynamics, we skip the simulation part of our pipeline, and interpolate the complete mesh $\Gamma$ using $\bar{u}_i$. Teeth are handled separately by extracting the closest rigid transformation to the interpolated shapes because they are rigid. In this example, we do not add back the input detail as in this case, the input detail contains tracking noise, rather than something that was intelligently designed by the artist (as in Figure 3). Actually, the input data contains tracking imperfections, mostly along the perimeter of the lips, which our method resolved. We processed 12 facial animations, each containing about 80 frames. Fitting each of them took 1.8 min on average.

8 Conclusion

We have augmented general, arbitrary triangle or point cloud animations with physics. Our method enables artistically driven simulation of three-dimensional solid objects. We provide two methods to balance the input with physics. We augment coarse tet mesh simulation with the preservation of spatial detail.

9 Limitations and future work

We assume that the input animation is provided at every frame; completely missing frames would require solving space-time optimization problems. Our method also requires that at least a (small) part of the model has to follow the input exactly. In the context of artist-directed animation, this requirement is natural as one typically does not want physically based animations that do not obey the input at least somewhat; otherwise, one can simply run standard physically based simulation. We augment coarse tet mesh simulation with the preservation of spatial detail. While our method is not

Table 1: Simulation statistics. vtx, tri=#triangle mesh vertices and triangles; tet-vtx, tet=#tet mesh vertices and tets; free tet-vtx, free tet=#tet mesh vertices and tets in the free region $\Omega \backslash \Omega''$; frames=#graphical frames; sim-frames=#simulation frames, including intermediate frames that were inserted for stable simulation; fitting, sim = time for fitting the tet deformations and simulation for one graphical frame. Intel Xeon 2.3GHz 2×6 cores, 32 GB memory.

<table>
<thead>
<tr>
<th>statistics</th>
<th>elephant</th>
<th>bird</th>
<th>head</th>
<th>plant</th>
<th>sumo</th>
</tr>
</thead>
<tbody>
<tr>
<td>vtx</td>
<td>8,401</td>
<td>408</td>
<td>25,600</td>
<td>961</td>
<td>15,482</td>
</tr>
<tr>
<td>tri</td>
<td>16,636</td>
<td>7,878</td>
<td>49,923</td>
<td>1,600</td>
<td>30,642</td>
</tr>
<tr>
<td>tet-vtx</td>
<td>6,685</td>
<td>3,949</td>
<td>5,729</td>
<td>200</td>
<td>2,717</td>
</tr>
<tr>
<td>tet</td>
<td>20,136</td>
<td>11,846</td>
<td>18,391</td>
<td>413</td>
<td>7,767</td>
</tr>
<tr>
<td>free tet-vtx</td>
<td>3,725</td>
<td>2,414</td>
<td>–</td>
<td>199</td>
<td>1,642</td>
</tr>
<tr>
<td>free tet</td>
<td>10,416</td>
<td>6,915</td>
<td>–</td>
<td>410</td>
<td>4,599</td>
</tr>
<tr>
<td>frames</td>
<td>178</td>
<td>299</td>
<td>960</td>
<td>120</td>
<td>650</td>
</tr>
<tr>
<td>sim-frames</td>
<td>534</td>
<td>600</td>
<td>–</td>
<td>120</td>
<td>3,900</td>
</tr>
<tr>
<td>fitting</td>
<td>1.25s</td>
<td>0.78s</td>
<td>1.35s</td>
<td>0.024s</td>
<td>0.24s</td>
</tr>
<tr>
<td>sim</td>
<td>0.35s</td>
<td>0.29s</td>
<td>–</td>
<td>0.0058s</td>
<td>0.31s</td>
</tr>
</tbody>
</table>
References


