Enriching Triangle Mesh Animations With Physically Based Simulation

Yijing Li, *Member, IEEE,* Hongyi Xu, *Member, IEEE,* and Jernej Barbič, *Member, IEEE,*

Abstract—We present a system to combine arbitrary triangle mesh animations with physically based Finite Element Method (FEM) simulation, enabling control over the combination both in space and time. The input is a triangle mesh animation obtained using any method, such as keyframed animation, character rigging, 3D scanning, or geometric shape modeling. The input may be non-physical, crude or even incomplete. The user provides weights, specified using a minimal user interface, for how much physically based simulation should be allowed to modify the animation in any region of the model, and in time. Our system then computes a physically-based animation that is constrained to the input animation to the amount prescribed by these weights. This permits smoothly turning physics on and off over space and time, making it possible for the output to strictly follow the input, to evolve purely based on physically based simulation, and anything in between. Achieving such results requires a careful combination of several system components. We propose and analyze these components, including proper automatic creation of simulation meshes (even for non-manifold and self-colliding undeformed triangle meshes), converting triangle mesh animations into animations of the simulation mesh, and resolving collisions and self-collisions while following the input.

Index Terms—Computer Graphics, Animation, physically based modeling, animation system, directable simulation, FEM, collisions.

1 Introduction

The problem of combining keyframe animation with physically based simulation is one of the most interesting and important ones in computer animation, and has been examined in several publications in recent years. In this system paper, we look into an important specific problem that has to date not received substantial analysis: how to improve *existing* animations of three-dimensional solids represented as triangle meshes (characters, tissue, plants, creatures, etc.) by making them better obey physics, yet simultaneously, for purpose of artistic direction and control, preserve the input in a controllable way in space and time. The input to our method is a triangle mesh animation, obtained using any method, such as keyframe animation, geometric shape modeling, or scanned from real subjects using computer vision. The input animation may be crude, non-physical or incomplete. Our method makes it possible to replace the input triangle mesh animation with physically based simulation, or properly blend the two, smoothly and selectively, in desired mesh regions and time intervals. This makes it possible to preserve the input wherever needed, but improve it with physically based simulation in the target regions. We investigate the system pipeline and simulation aspects that are needed in order to obtain a robust and versatile system. Furthermore, our method is designed so that it only modifies the triangle mesh dynamic deformable detail that is *representable* by the chosen tetrahedral mesh. This makes it possible to rapidly "physify" the input using a coarse tetrahedral mesh, which only modifies the coarse part of the motion, whereas the input dynamic spatial high-frequency motion is preserved.

In animation production, animations are often created in layers, such as, first the skeletal animation, followed by muscles, fat, skin and then clothing and hair. The output of each of these layers is "baked" as a triangle mesh animation, and then serves as input to the next layer, or is sent to the renderer. Instead, our "physification" method serves as an additional layer in the pipeline that simply transforms an input triangle mesh animation into another triangle mesh animation, and can as such be selectively employed with a minimal change to the established computer animation pipelines. We note that several of the animation production layers often use rigging deformers of various form (skeletal, wire, blendshape, cluster, lattice, etc.). Although one could in principle employ a rigging-aware physically based method, this is seldom an easy investment for a film or games studio, as it requires re-writing complex code and re-training the artists. Furthermore, for inputs that are obtained using geometric shape modeling techniques (Figure 14), or acquired from real subjects using computer vision (Figure 1), a rig is not available, necessitating a method that can work with triangle meshes directly.

Our method works as follows. We first generate a quality tetrahedral mesh to the undeformed triangle mesh. We demonstrate how to do so even in the presence of self-collisions in the undeformed triangle mesh (Figure 2). Next, we convert the input into a physically based form, by fitting a tetrahedral mesh animation to the embedded triangle mesh input animation using a fast Newton solver. The tetrahedral mesh animation is then generated using physically based simulation, subject to the constraint that it has to follow the input motion, to a degree controllable by the artist in space and time. The artist exerts this controls by painting a scalar field on the input mesh. We automate the painting so that only a minimal UI input is required. Our method 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 the tradeoff between physics and the input, we provide two methods: (1) using an implicit blending force during the simulation, and (2) blending the shapes after the simulation. For both methods, we demonstrate how to incorporate collision and self-collision response that simultaneously preserves the input motion. Figure 4 gives an overview of our system. The end result is an *artistically directed simulation*: it conforms to

- *All authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA, 90089.*
  E-mail: yijingl@usc.edu, hongyixu@usc.edu, jnb@usc.edu.

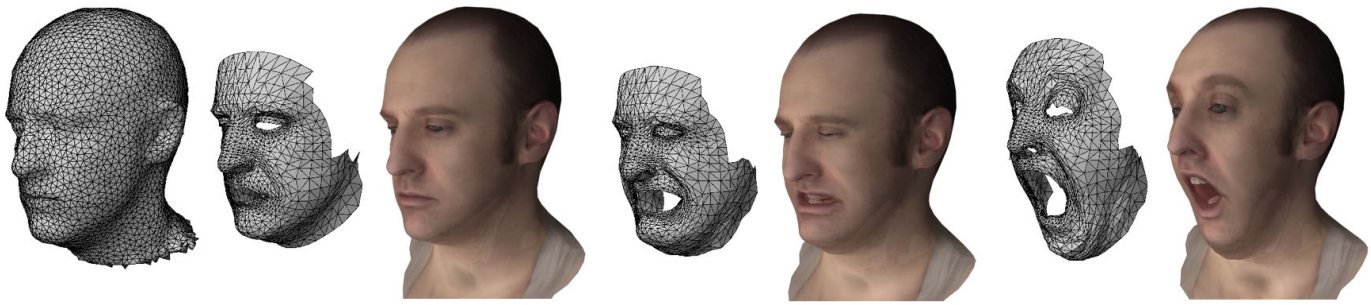


Fig. 1: ***Completing a tracked face using physics:*** *Due to occlusions, vision tracking systems cannot capture all parts of a human head (mouth cavity, inner lips, tongue, etc.). We utilize physically based simulation to produce the deformations of the untracked parts. Left (the input): tetrahedral mesh (18,391 tets), the tracked triangle mesh (49,924 triangles) and the neutral complete triangle mesh of the head. Middle, right (the output): the deformed tracked mesh and the complete mesh computed using our method. 960 frames, 1.35 sec per frame.*

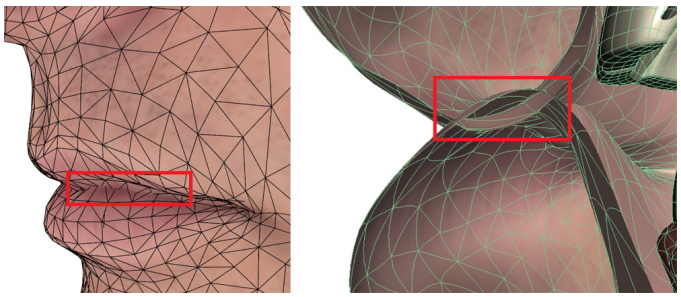


Fig. 2: Self-collisions in the input mesh (human face). *Left: the lips collide in the highlighted region. Right: an interior view of the self-collision. Our method works with such ill-formed inputs.*

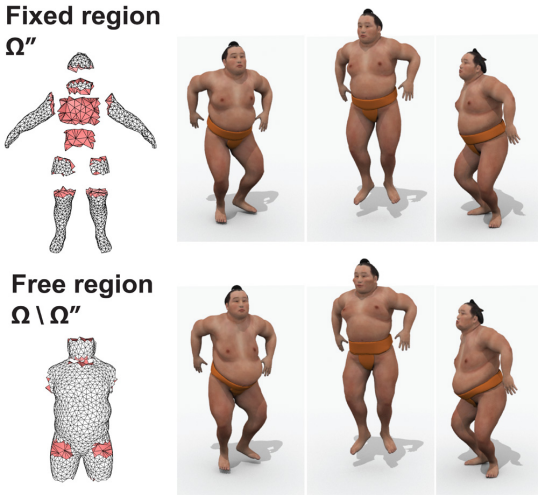


Fig. 3: Adding physics to a keyframed sumo wrestler animation. *Left-most column: the fixed* Ω'' *and free* $\Omega \setminus \Omega''$ *region of the sumo tetrahedral mesh. Free regions include the body, thighs, cheeks and derrière. The top row on the right shows several frames of the input motion. The bottom row on the right shows our result with secondary motion added.*

the input by a prescribed amount, but is enriched with physics, secondary motion and collision response. Our method operates in the full FEM space without model reduction, simulating both global and local deformations. In addition to providing good dynamics, our method can also infer missing data in a triangle mesh, making it suitable for completing deformed animation frames based on a rest configuration template.

2 Related Work

2.1 Adding Physics

The problem of simulating deformable objects has been well-studied since the pioneering work of [1]. Several publications have improved simulation versatility and efficiency, for example [2], [3], [4], [5], and robustness [6], [7]. A common theme in computer animation is to simulate soft tissue based on the motion of an embedded skeleton or character rig [5], [8], [9], [10], [11], [12], [13]. Wiggly splines were used for direct manipulation of physical oscillatory motion [14], and Shi [15] enhanced skeleton-driven animations with secondary motion extracted from sample sequences. Hahn [16], [17] formulated the equations of motion in the rig space so that some rig parameters can evolve via physics. Different from these previous methods that *generate* soft tissue motion from the character rig, our method does not require a rig, and works with and corrects an *existing* triangle mesh animation. Several methods [18], [19], [20], [21] enriched coarse cloth animations with spatial high-frequency effects such as wrinkles and folds. Among them, Bergou [18] achieved this by running a detailed mesh simulation constrained to the input coarse animation. We also use constraint-based dynamics, but with a different design goal. We *modify* the coarse (spatially low-frequency) component of the animation, as representable by the chosen simulation mesh, and *preserve* high-frequency motion, whereas Bergou's method *preserves* coarse motion and *generates* high-frequency detail. We compare our method to Bergou's method in Section 9.

2.2 Animation Design and Control

Several papers addressed the design and editing of elastic physically based simulations using keyframes [22], [23], [24] or spacetime constraints [25], [26], [27]. Additionally, Kondo [28] allowed animation editing by both keyframes and trajectories. Barbič [29] directed physical simulations to given input trajectories. Coros [30] controlled deformable objects by changing their rest shapes. Barbič [31] introduced spacetime Greens functions for interactive animation editing. These papers mainly focus on direct user control. Although they can enrich animations with physics, they require special models (e.g., model reduction or

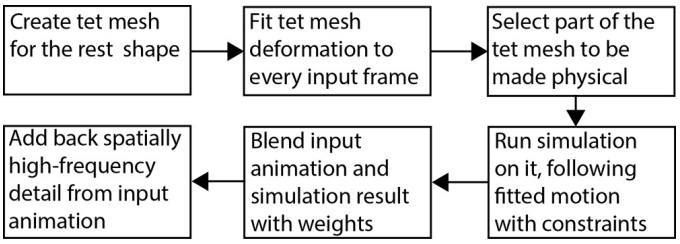


Fig. 4: Overview. *This diagram corresponds to the kinematic blending method. Implicit blending is performed simultaneously while running the simulation.*

specific definitions of elastic forces), or operate under specific assumptions. We aim to achieve the goal using commonly used methods such as Newton's method and full-space FEM with boundary constraints, so that the method is better understood, more general and more likely to be adopted in practice.

2.3 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 term is usually used to determine the positions of the interior vertices. Choi and Szymczak [32] proposed fitting volumetric meshes to animated surfaces by minimizing a warped linear elasticity energy and surface distance. Wuhrer [33] used point clouds that are a subset of the surface vertices of the volumetric mesh to infer material properties, forces and interior displacements. Paillé [34] compressed a volume grid to fit its boundary to the input surface. Volumetric mesh tracking for 2D and 3D images is also used in biomechanics [35]. Our volumetric mesh fitting process differs from these methods in that these methods tracked the surface of a volumetric mesh. In contrast, we apply *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.

2.4 Inverse Caging

Inverse caging is the process of finding a deformed cage (and sometimes its weights) that best generates the given shape. The vertices of a volumetric mesh can be treated as the handles of a cage that controls an embedded mesh. Lu [36] and Savoye and Franco [37] adopted Laplacian coordinates to deform or fit a cage. Chen [38] 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, Thiery [39] computed the maximum volume submatrix of the interpolation matrix, to fit a cage animation to a triangle mesh animation. Savoye [40] converted performance mesh animation into cage-based animation using an linear estimation framework. 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 avoid interior contact on triangle mesh deformation like collisions in concave regions (Figure 17). Barbič [22] 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. We utilize temporal coherence to accelerate Newton solve on the fitting energy. In contrast to their model reduction approach which is limited to global low-frequency deformations, our outputs are full-dimensional, model local deformations and preserve input animation detail up to the volumetric mesh resolution.

3 Overview

The input to our method is a neutral pose of a triangle mesh Γ, and an animation of a submesh $\Gamma' \subseteq \Gamma$. The animation of Γ' consists of frames $i = 0, \ldots, T$, each given by a displacement vector away from the neutral pose of Γ'. This input animation can be created using any means available: it can be keyframe-animated, 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 $\Gamma' \subsetneqq \Gamma$ case occurs, for example, with the human face geometry, where Γ 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 Γ' 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 Γ that is identical to the input animation in a user-selected subset $\Gamma'' \subseteq \Gamma'$. Elsewhere on Γ', 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 Γ' 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 Γ', unlike typical physically based simulation that either requires a computationally slow detailed tetrahedral mesh to produce rich detail, or else detail is destroyed using coarse meshes.

An overview of our method is shown in Figure 4, and pseudocode is given in Figure 5. 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 boundary constraints to produce physical motion on selected parts of the mesh (Section 5), and optionally preserve the dynamic spatial high-frequency detail (Section 6). 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 kinematically blending the input with the simulation result (Section 7).

4 Fitting a Simulation Mesh

To physically enrich the input (non-manifold) triangle mesh animation, it is first necessary to equip it with a simulation

Fig. 5: *Add Physics to Animation*

Input: Γ, submesh Γ', $\{\bar{q}_i\}$
Output: $\{q_i\}$
procedure ADD_PHYSICS($\Gamma, \Gamma', \{\bar{q}_i\}$)
  $\{\Omega, C, C'\} \leftarrow$ CREATE_TET_MESH(Γ, Γ') ▷ C, C' are interpolation matrices to Γ, Γ'
  $\{\bar{u}_i\} \leftarrow$ FIT_ANIMATION($\Omega, \Gamma', C', \{\bar{q}_i\}$)
  assign $\Omega'' \subseteq \Omega$ that follows input
  if use implicit blending force **then**
    initialize blending force f_w according to weights W
  else
    $f_w \leftarrow 0$
  end if
  $\{u_i\} \leftarrow$ SIMULATE($\Omega, \Omega'', \{\bar{u}_i\}, f_w$)
  if use kinematic blending **then**
    $\{u_i\} \leftarrow$ BLEND_MOTIONS($\{u_i\}, \{\bar{u}_i\}, W$)
  end if
  $\{q_i\} \leftarrow C\{u_i\}$ ▷ interpolate u_i to q_i
  if preserve high-frequency detail **then**
    $\{q_i\} \leftarrow$ PRESERVE_DETAIL($\Omega, C', \bar{q}_i, \{\bar{u}_i\}, \{u_i\}$)
  end if
  return $\{q_i\}$
end procedure

(tetrahedral) mesh capable of providing physics. In this section, we explain how we generate a simulation mesh, and fit an animation of it to the input triangle mesh animation.

4.1 Generating a Simulation Mesh

Given the input triangle mesh Γ, we create a tetrahedral simulation mesh Ω that encloses the space occupied by the undeformed triangle mesh. Many existing tools such as *Tetgen* [41] and *NetGen* [42] can generate good-shaped tetrahedral meshes given manifold surface triangle meshes. In order to generate a quality tetrahedral mesh from an arbitrary, non-manifold input, we first compute a signed distance field using the method described in [43]. Next, we use an isosurface meshing algorithm [44] to generate a manifold triangular surface. Recent work by Sacht [45] can be used to coarsen the surface mesh with collision avoidance. Finally, we build a tetrahedral mesh using constrained Delaunay tetrahedralization [46], [47]. The resulting tetrahedral mesh encloses the space of the input mesh and has good quality. However, extra effort is needed to accommodate a common practical situation: the undeformed triangle mesh contains self-intersections (such as in the case of human lips in Figure 2 and dragon horns in Figure 14), causing the mesh to "weld". We will present our solution to this problem in Section 8.

By adjusting the isosurface and tetrahedral mesher parameters, the user can obtain a tetrahedral mesh at whatever resolution desired. Our method then automatically adds physical effects that are resolved by the chosen tetrahedral mesh, but keeps the input high-frequency spatial component of the animation unmodified. Note that this is different from merely preserving the spatially high-frequency detail in the neutral pose: our method does not preserve just the static geometric detail of the neutral pose (as has been done in many prior papers), but also the input spatially high-frequency *animation* of the detailed geometry (Figure 6).

Fig. 6: Preserving dynamic spatial high-frequency detail: *Top: input animation of the plant withering (aging), containing spatial high-frequency motion on the leaves. The model and animation were downloaded from the Internet. Middle: physically based simulation loses high-frequency detail. Bottom: our result preserves high-frequency detail. Tetrahedral mesh can be coarse and is shown overlaid. Our fitting process can be seen as extracting the spatial low-frequency part of the motion and modifying it using physically based simulation. The high-frequency motion can be added back depending on needs.*

4.2 Fitting the Simulation Mesh Animation

For every input frame of the triangle mesh Γ', we compute a tetrahedral 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 tetrahedral mesh. Given the tracked triangle mesh with t vertices and its deformation $\bar{q}_i \in \mathbb{R}^{3t}$ at a frame i, the deformation of the tetrahedral mesh $\bar{u}_i \in \mathbb{R}^{3n}$ is obtained by minimizing

$$\bar{u}_i = \arg\min_u \frac{1}{2}\|C'u - \bar{q}_i\|^2_\Lambda + \gamma\Phi(u), \tag{1}$$

where $C' \in \mathbb{R}^{3t\times 3n}$ is the interpolation matrix that interpolates tetrahedral mesh deformation to Γ', $\Phi(u)$ is an internal elastic energy for deformation u to regularize the shape, and $\gamma > 0$ determines the tradeoff between the two terms. Diagonal matrix Λ is used to weight the interpolation energy according to the surface area of each vertex, $\|x\|^2_\Lambda = x^T\Lambda x$. We choose the invertible St.Venant-Kirchhoff (StVK) elastic energy [6] because it is robust, because it preserves volume under large stretches better than the linear co-rotational material (Figure 7), and because its evaluation speed is similar to the corotational linear FEM [48]. One limitation of such invertible models is that the energy is not consistent in the extrapolated regions corresponding to extreme compression and inversion. We did not encounter any related stability problems; but if needed, the issue can be remedied using [49]. 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 [50] or [51]. We report γ in Table 1 and discuss the parameter tuning in Section 9.

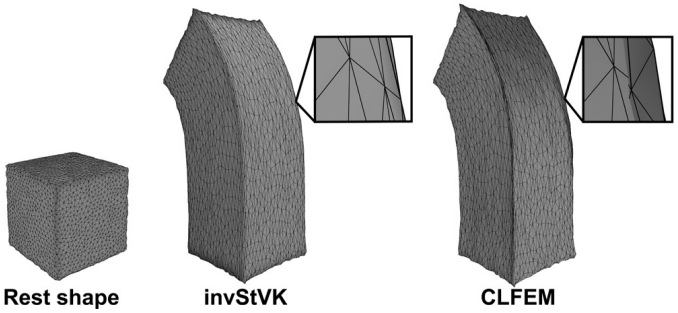


Fig. 7: Comparison of invertible StVK to corotational linear FEM. *The input is a severely stretched cube triangle mesh. With the same amount of total fitting error, the shape produced by the corotational linear FEM (right) has self-intersections along the cube edge, whereas the invertible StVK (middle) has no such artifacts.*

For each frame i, given an initial guess, we compute the tetrahedral mesh deformation at that frame using a process similar to numerical timestepping. 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 $\Phi(u)$, which can be computed easily. We compute the derivative of the fitting energy and set it to zero:

$$C'^T \Lambda(C'u - \bar{q}) + \gamma F_{\text{int}}(u) = 0, \tag{2}$$

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

$$C'^T \Lambda\Big(C'\Delta u + C'u^0 - \bar{q}\Big) + \gamma\Big(K(u^0)\Delta u + F_{\text{int}}(u^0)\Big) = 0. \tag{3}$$

Here, $K(u^0)$ is the tangent stiffness matrix of the elastic energy Φ evaluated at u^0. By solving the above sparse system for Δu, we obtain the next iterative approximation for $\bar{u}_i$. The iteration continues until the relative difference between the two consecutive iterations becomes small enough (1% in our examples). For the first frame in the animation, we use the rest shape as the initial guess and apply a line search after each iteration to stabilize the optimization. For the other frames, we use the deformation of the previous frame as the initial guess.

Such a Newton's method is faster than a gradient-only optimization method such as the conjugate gradient optimizer [52] 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 $\Phi(u)$ and the linear system solve, which can be easily parallelized. One can instead parallelize the solver across multiple frames. However, such independent parallel solves present a difficulty for maintaining temporal coherence across the fitted frames, whereas our sequential implementation is less likely to create temporal artifacts. In practice it gives good results without noticeable temporal artifacts.

5 Constrained Simulation

After the tetrahedral 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 tetrahedral mesh Ω'' of Ω that will be constrained during the simulation to follow the input.

We compute the animation of $\Omega \setminus \Omega''$ using a physically based simulation where the vertices in Ω'' serve as boundary constraints. The equation of motion of a FEM nonlinear deformable object is

$$M\ddot{u} + D(u)\dot{u} + f_{int}(u) = f_{ext}(t), \tag{4}$$

$$\text{subject to } u^c = \bar{u}^c, \tag{5}$$

where M is mass matrix, u is the vertex displacements of Ω, $f_{int}(u)$ is the internal elastic force, $f_{ext}(t)$ is the external force including gravity and contact forces, and $D(u)$ is the damping matrix. The vertex displacements of Ω'' obtained during the fitting process and during the simulation are denoted by $\bar{u}^c$ and u^c, respectively. We use the same invertible StVK energy as in the fitting process because of its robustness and fast evaluation. We adopt Rayleigh damping in all our examples: $D(u) = \alpha M + \beta K(u)$, where α and β are Rayleigh damping parameters and $K(u)$ is the tangent stiffness matrix at u.

We use $v = \dot{u}$ to represent vertex velocities. Given the state (v_k, u_k) at frame k, we perform one iteration of the implicit backward Euler integrator to get Δv so that the velocity at next frame is $v_{k+1} = v_k + \Delta v$, and the displacement is $u_{k+1} = u_k + \Delta t v_{k+1}$.

In order to compute Δv, we partition the linear system into

$$\begin{bmatrix} A^{ff} & A^{fc} \\ A^{cf} & A^{cc} \end{bmatrix} \begin{bmatrix} \Delta v^f \\ \Delta v^c \end{bmatrix} = \begin{bmatrix} b^f \\ b^c \end{bmatrix}, \tag{6}$$

$$\begin{bmatrix} A^{ff} & A^{fc} \\ A^{cf} & A^{cc} \end{bmatrix} = A = M + \Delta t D(u_k^0) + \Delta t^2 K(u_k^0), \tag{7}$$

$$\begin{bmatrix} b^f \\ b^c \end{bmatrix} = b = \Delta t(f_{ext} - f_{int}(u_k^0) - (\Delta t K(u_k^0) + D(u_k^0))v_k), \tag{8}$$

where Δv^c and Δv^f correspond to the DOFs in Ω'' and $\Omega \setminus \Omega''$, respectively, Δt is the timestep and u_k^0 is the linearization state for internal elastic forces, discussed further below. Since we have the boundary constraint $\Delta v^c = \Delta\bar{v}^c$ that the motion in Ω'' should follow the fitted motion, we only need to solve

$$A^{ff}\Delta v^f = b^f - A^{fc}\Delta\bar{v}^c. \tag{9}$$

We use PARDISO, a direct sparse solver. While a fully implicit Euler involves multiple iterations of the Newton-Raphson method, we in practice solve Equation 9 only once. In our experiments, for a fixed amount of computation cost, it was better to simply subdivide the timestep, than to invest the same amount of computation time into multiple Newton-Raphson iterations of a single timestep. We made a comparison where we ran (A) one timestep of size Δt with five Newton iterations, versus (B) five timesteps of size $\Delta t/5$ with one Newton iteration. We found that the largest stable timestep under (B) is 5-10$\times$ times larger than (A), plus (B) is higher quality (less damped) due to the smaller timestep.

We experimented with two choices for the evaluation state u_k^0 for internal forces. We observe that it can have a substantial visual influence on the output animation (Figure 8). One choice, $u_k^0 = u_k + \Delta t v_k$, is used by Bergou [18], while the other is $u_k^0 = u_k$. We observed that the latter produces stabler simulations, allowing very large timesteps with minimal tuning, at the cost of more damped motion. We give a control on the initial guess as $u_k^0 = u_k +$

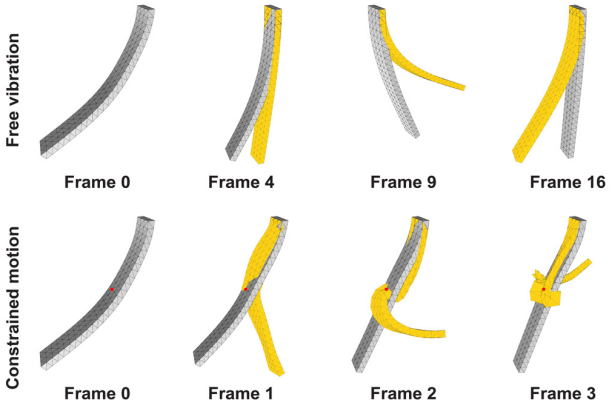


Fig. 8: Comparison of different internal force evaluation strategies: *A simple bar vibrates with one end fixed. The yellow shape was obtained using $u_k^0 = u_k + \Delta t v_k$ (Bergou's initial guess). The gray shape was obtained using $u_k^0 = u_k$ (the other). The initial displacement and velocity are the same in both methods. In the first row, $\Delta t = 0.003$. Bergou's initial guess exhibits a less-damped motion. In the second row, $\Delta t = 0.1$. A position constraint is added to one vertex (shown as a red dot). The other initial guess is stable under the large timestep, whereas the Bergou's is not.*

$\delta \Delta t v_k$, $0 \le \delta \le 1$, so that the user can choose to prefer accuracy or stability given same amount of computation cost.

To keep the integrator stable, we subdivide the animation timestep if necessary. One limitation of our constrained simulation is that the boundary constraints on Ω'' only provide a $\mathcal{C}^0$ deformation continuity across the interface between Ω'' and $\Omega \setminus \Omega''$. This is because we use linear tetrahedral elements. Deformations are linear per element and $\mathcal{C}^0$ across elements. However, in order for non-$\mathcal{C}^1$ 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. The user can also use the blending methods described in Section 7 to blend the simulated motion around the interface with the scripted motion to reduce possible discontinuity.

6 Preserving High-Frequency Detail

After the simulation, we obtain the output deformation q_i of Γ at frame i as $q_i = Cu_i$, where C interpolates tetrahedral mesh deformation to Γ. If the input triangle mesh animation contains spatial high-frequency motion, 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 spatial detail present in the undeformed mesh Γ is always preserved. In our method, we preserve the *dynamic* detail (as opposed to only static) as follows (see also Figure 9). At each frame, for each tracked triangle mesh vertex, we store the difference between the input vertex displacement and the displacement interpolated using the fitted tetrahedral mesh deformations $\bar{u}_i$. This difference is then rotated and added to the interpolated displacement obtained using simulation mesh deformations u_i, effectively reproducing the high-frequency detail

$$q_i' = C'u_i + R_i(\bar{q}_i - C'\bar{u}_i), \tag{10}$$

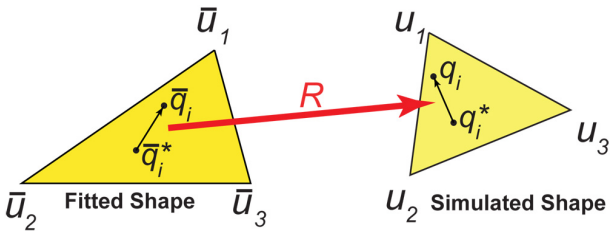


Fig. 9: Preserving dynamic high-frequency detail: *The difference between the input vertex displacement $\bar{q}_i$ and the interpolated displacement $\bar{q}_i^*$ of the fitted triangle $\bar{u}_1\bar{u}_2\bar{u}_3$ (left) is rotated by the rotation component R of the deformation gradient between the two triangles (2D example). It is then added to the interpolated displacement q_i^* of the simulated triangle $u_1u_2u_3$ (right) to give the final displacement q_i.*

where $q_i' \in \mathbb{R}^{3t}$ is the final displacement on the tracked mesh Γ' at frame i, $R_i \in \mathbb{R}^{3t \times 3t}$ is a block diagonal matrix formed by 3×3 rotation matrices. Each rotation matrix is extracted by polar decomposition from the relative deformation gradient between the fitted and simulated tetrahedron containing each vertex.

This procedure has the property that the dynamic detail on Γ'' is always preserved. For any triangle mesh vertex in Γ'' chosen to follow the input, its embedding tetrahedron should be in Ω'', which is constrained throughout the simulation. Consequently, the deformations of this tetrahedron in the input shape and in the simulated shape are identical, at any frame. The rotation matrix is thus always identity, causing $q_i' = \bar{q}_i$. After the transfer of the detail, the vertex position is guaranteed to be identical to the input; i.e., the input is exactly preserved in Γ''. The result of this procedure is shown in Figure 6. We also considered an alternative method where the tetrahedral deformation gradient is directly used to transform $\bar{q}_i - C'\bar{u}_i$. While this alternative method worked on most examples, it failed in cases where the input animation causes large compressions in the fitted tetrahedral mesh shape (Figure 10). This produced nearly singular deformation gradients on compressed tetrahedra, resulting in mesh "spikes" when the fitting procedure causes the triangle mesh vertex to escape out of its original tetrahedron. Altering barycentric coordinates for each individual frame does not work well either. It would have difficulties if the tetrahedral mesh self-collides during the fitting, which leaves multiple tetrahedra containing the same embedded vertex, or for vertices outside of the mesh with poorly conditioned neighboring tetrahedra. Our method does not suffer from these issues. One limitation of using rotations is that they have slight discontinuities at tetrahedra boundaries. In practice, the rotations of neighboring tetrahedra are similar, and we did not observe any artifacts.

7 Balancing Input Motion with Physics

In the previous section, the output animation comes either from physical simulation, or from the artist input. We now describe and compare two procedures that permit the artists to control the balance between physics and the input animation. We discuss their advantages and disadvantages. The balance is prescribed by spatially-varying tetrahedral mesh vertex weights $w_i \ge 0$. We explain how the artists can easily generate such weights in Section 7.3.

Fig. 10: High-frequency detail transfer failure *produces spikes on the mesh (left) when using tetrahedral deformation gradients to transfer detail from highly compressed tetrahedra, as contrasted with the smooth shape (right) produced by transfer with the rotation component (our method). An alternative method to preserve high-frequency detail is to alter barycentric coordinates for the triangle mesh at each frame. This method also suffers from such a "spike" failure.*

7.1 Implicit Blending Force

In the first method, we add an implicitly integrated external force to drive the mesh toward the input, scaled by the weights

$$f_w = W(\bar{u} - u), \tag{11}$$

where $W \in \mathbb{R}^{3n\times 3n}$ is the diagonal matrix of weights w_i for each vertex. Higher weights generate motion closer to the input. In industry, such forces are often referred to as the kin(ematic) springs. They can be integrated implicitly in Equation 9. Because Equation 11 is perfectly linearized, the Jacobian of this force is exact. The implicit integration thus has an exact prediction of the force. Besides, the presence of W increases only diagonal entries in the system matrix, making it more well-conditioned. As a result, the integration of this force is highly stable. We have experimentally observed stable convergence to $\bar{u}$ as W grows to extremely large values such as 10^{30}, well beyond values that produce visually identical motion as $\bar{u}$. It is convenient for the user to express weights on the interval $[0,1]$; we re-map them to $[0,\infty]$ using the function $f(x) = \eta(x^{1/\zeta})$, where the user can adjust parameters $\eta > 0$ and $\zeta > 1$. 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. The disadvantage of this method is that the relationship between the weights and the input vs. physics tradeoff, while monotonic, is only indirect.

7.2 Kinematic Blending

In the second approach, we use a construction similar to as-rigid-as-possible interpolation [23], [53] to blend the output physical shapes and the input shapes. We decompose the deformation gradient for each tetrahedron into a rotation and symmetric matrix, blend them separately according to the weights, and solve a linear system to combine the deformation gradients of all tetrahedra as in rotation-strain coordinate warping [54]. Weights used in this procedure are intuitive as they can be assigned directly on $[0,1]$. The method permits easy post-simulation tuning of weights, upon which one can resolve for the blended animation without re-simulating physics.

Such an approach, however, is not compatible with contact resolution. We use the following strategy to resolve contact. After each simulation step, we use kinematic blending to produce a blended shape. If any collision is detected in the blended shape, penetration depths are collected and penalty forces calculated with respect to the blended shape (Section 8). These penalty forces cannot be applied to the simulation mesh directly, since the blended shape usually has a different shape and orientation as the simulation mesh. Therefore, a per-tetrahedron rotation is calculated, based on the deformation gradient of each tetrahedron of the blended shape relative to the same tetrahedron in the simulation mesh. We then rotate the penalty forces with these rotations to get the correct force direction before adding the contact forces to the integrator. This strategy works reasonably well in our elephant example (see the supplementary video and Figure 13). A more precise method could compute the gradient of the blending function (such as in [27]); we regard this as too complicated and did not pursue this direction. We implemented and compared the two methods (Figure 13 and the accompanied video). We prefer the implicit blending force to the kinematic blending because it enables collision handling without any special additional provisions, albeit at some loss of weight intuitiveness.

7.3 Generating Weights

In this section, we explain how we can easily determine suitable weights $w_i \geqslant 0$ for any vertex i of the simulation mesh $\Omega \setminus \Omega''$, for blending purposes. Larger weights bring the mesh closer to the input motion. We provide an interface to compute all the weights using a bi-Laplace solver. The Laplacian matrix L is defined as

$$w^T L w = \sum_{1\leq i<j\leq m} \sigma_{i,j}(w_i - w_j)^2, \tag{12}$$

$$\sigma_{i,j} = \begin{cases} 1 & \text{if vertex } i \text{ and } j \text{ share an edge,} \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Here, $w \in \mathbb{R}^{m\times 1}$ is the vector of the weights, and m is the number of vertices in $\Omega \setminus \Omega''$. The action of the tetrahedral mesh bi-Laplacian $L^T L$ is defined similarly as in [55], [56], except that we define Laplacian on vertices instead of tetrahedra. We then minimize

$$\frac{1}{2} w^T L^T L w, \tag{14}$$

$$\text{subject to} \quad Sw = \bar{w}, \tag{15}$$

where S is the selection matrix, and $\bar{w}$ are the user-defined weights on a few vertices. Because the constraints are linear and the objective is quadratic, the minimization can be performed using a single sparse linear system solve.

8 Collisions

Collisions may occur and need to be handled in various parts of our pipeline. Robust handling of collisions is generally a difficult, but an essential task for any physically based simulator. This section explains how we address collisions.

8.1 Self-colliding Undeformed Triangle Mesh

When the undeformed triangle mesh has regions that are very close to each other, it is difficult to construct a tetrahedral mesh without welding these parts together. This problem becomes even worse if the input triangle mesh has self-intersections. To avoid confusion, we argue that there are two types of self-intersections. The first type is where different mesh components are put together to form

a complete object. For example, the elephant tusks are separate meshes that collide with the body. We do not wish to separate self-intersections of this kind. The second self-intersection type appears because of errors in the modeling, or because they are too tedious to address manually in existing workflows. We want to remove such self-intersections. Our meshing pipeline addresses the first type; it will produce a connected tetrahedral mesh without any special treatment.

There has been work on removing self-collision in the rest mesh. Sacht [57] converted the mesh into a collision-free shape to tetrahedralize it correctly. However, they can only process two-manifold meshes and focused mostly on sphere-topology shapes. Jacobson [58] provided robust inside-outside segmentation even if the mesh has self-intersection, but did not give a method to remove self-intersections. We argue that it is impossible to properly process self-collision given only the geometry. This is because the collision type (first or second) depends also on the semantics (the meaning of the geometry). Therefore, we give a general method to remove self-collisions of the second type, at the cost of additional user interaction. The pseudocode is given in Figure 12. In order to identify the second type, we ask the user to select a deformed pose Γ^* of the mesh where the regions are more separated. This can either be an input animation frame, or an individual pose from another source. We then build a tetrahedral mesh Ω^* for Γ^*, and apply the tetrahedral mesh fitting method (Section 4.2), to deform Ω^* towards the undeformed triangle mesh Γ. This produces a smooth tetrahedral mesh Ω around Γ. We then use Ω as the undeformed tetrahedral mesh for the rest of our method. Because the fitting process does not handle collisions, the simulation mesh Ω may still be self-colliding. It is, however, topologically correct, not welded, and does not pose any simulation issues. We note that self-colliding undeformed tetrahedral meshes were also used in [59], [60].

One technical challenge that we need to address is that self-colliding tetrahedral meshes introduce ambiguity in building the interpolation matrix C that interpolates tetrahedral mesh deformations to the triangle mesh (line 11 in Figure 12). In the self-colliding region, a triangle mesh vertex P can be contained in at least two tetrahedra. To resolve this, we find out the tetrahedron e that contains P in the collision-free shape Ω^*, and then compute barycentric coordinates for e and P on Ω. Due to the fitting error, P might be moved outside of e on Ω. However, because Ω is topologically correct, we can form a line segment between the centroid of e and P on Ω, and then walk down the line segment in the tetrahedron e's topological neighborhood to find the containing tetrahedron. If the point lies outside of all tetrahedra, we use the last traversed tetrahedron to embed it. A 2D illustration is shown in Figure 11.

8.2 Collision Mesh Creation

The input triangle mesh may contain ill-formed triangles and non-manifold components not suitable for collision detection. Therefore, we re-use the method of [43] to create a manifold collision surface mesh. The mesh resolution can be adjusted to trade accuracy for collision speed.

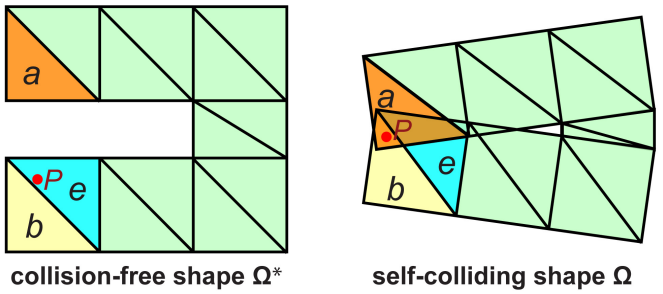


Fig. 11: Ambiguity in embedding: *A vertex P is embedded in triangle e on Ω^*. After fitting, P is embedded by both triangle a and b on the self-colliding mesh Ω. To find the correct triangle to embed, we start at e on Ω, go towards P in e's neighborhood, and arrive at b. Since b embeds P, we stop and declare b as the containing triangle of P on Ω.*

Fig. 12: *Create a Tetrahedral Mesh and Its Interpolation Matrices*

Input: Γ, Γ'
Output: Ω, C, C'
1: **procedure** CREATE_TET_MESH(Γ, Γ')
2: **if** Γ is collision-free **then**
3: $\Omega \leftarrow$ MESHING_PIPELINE(Γ)
4: $C \leftarrow$ INTERP_MATRIX(Ω, Γ)
5: **else**
6: find a collision-free shape Γ^*
7: $\Omega^* \leftarrow$ MESHING_PIPELINE(Γ^*)
8: $C^* \leftarrow$ INTERP_MATRIX(Ω^*, Γ^*)
9: $\bar{u}^* \leftarrow$ FIT_ANIMATION($\Omega^*, \Gamma^*, C^*, \Gamma - \Gamma^*$)
10: $\Omega \leftarrow \Omega^* + \bar{u}^*$
11: $C \leftarrow$ INTERP_MATRIX_WITH_COLLISION(Ω, Γ, C^*)
12: **end if**
13: $C' \leftarrow$ TRACKED_INTERP_MATRIX(Γ, Γ', C)
14: **return** Ω, C, C'
15: **end procedure**

8.3 Collisions in Triangle Mesh Animation

The input triangle mesh animation may contain self-colliding frames. We do not resolve collision during the fitting because such collisions can be resolved during the simulation, as long as at least one of the colliding regions are in $\Omega \setminus \Omega''$. If collisions appears on the first frame, we add additional "pre-roll" copies of the first frame before the animation to allow penalty forces to resolve collisions during the pre-roll.

8.4 Collision Detection and Response During the Simulation

After a collision mesh is created, we embed it into the simulation mesh Ω using barycentric coordinates. At runtime, we perform collision detection using a bounding volume hierarchy [61]. For each colliding collision mesh vertex c, its penalty force f_c is calculated based on the penetration depth. We compute the penetration depth in a manner similar to [62]. The force f_c is redistributed to the vertices of the tetrahedron that embeds c, weighted by its barycentric coordinates [63],

$$f_i = w_i^c f_c, \tag{16}$$

statistics	elephant	dragon	bird	head	plant	sumo
vtx	8,400	25,002	408	25,600	961	15,482
tri	16,636	50,000	7,878	49,924	1,600	30,642
tet-vtx	3,850	1,590	3,949	5,729	200	2,717
tet	14,942	5,387	11,846	18,391	413	7,767
free tet-vtx	1,621	1,401	2,414	–	199	1,642
free tet	5,354	4,801	6,915	–	410	4,599
frames	178	420	299	960	120	650
sim-frames	1,602	2,500	600	–	120	3,900
fitting	0.54s	0.11s	0.78s	1.35s	0.024s	0.24s
γ	10^{-10}	10^{-9}	10^{-5}	10^{-8}	10^{-2}	10^{-9}
sim	8.37s	0.68s	0.084s	–	0.0058s	0.30s

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 $\Omega \setminus \Omega''$; frames=#graphical frames; sim-frames=#simulation frames, including intermediate frames that were inserted for stable simulation; fitting, sim = time for fitting the tetrahedral deformations and simulation for one graphical frame, respectively; γ: parameter used in fitting. Intel Xeon 2.3GHz 2×6 cores, 32 GB memory. In the elephant example, 95% of the simulation time is spent processing collisions. The head example only uses frame fitting; hence, no simulation time is reported.*

where f_i is the collision force on tetrahedral mesh vertex i, and w_i^c is the barycentric weight of c with respect to i.

9 Results

We demonstrate our method using several examples. Statistics are available in Table 1.

Elephant: In our first example (Figure 13), we enrich a walking elephant motion with physics. The motion was created by an artist in Maya, using rigging and keyframing. We use our method to add secondary motion to the ears, trunk, belly and tail, by putting them into $\Gamma \setminus \Gamma''$. In each of these regions, we also adjusted a single Young's modulus value to make the region more or less elastic. The motion has 178 frames. It took three minutes to generate the tetrahedral mesh. Then, we iterated seven times to fit the first several frames of the animation, searching for a good γ (Equation 1). This process took eight minutes including user interaction and running the fitting. It took about 90 seconds to produce the entire fitted animation with the desired γ. Collision detection and contact resolution were enabled in the simulation. Figure 13 shows our blended result, where we blend between a physically based simulation and the input with different weights, using the blending method of Equation 11. We also show the result of using kinematic blending: it produces a similar motion as implicit blending. In Figure 18, we demonstrate that we can also enrich input artist animations with *large* 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 should collide with the deformable character, without substantially changing the original character motion.

Dragon: The second example (Figure 14) is an Asian dragon animation created by a surface deformation method [64]. The dragon's horns (nearly) self-collide with the head in the input mesh, and standard methods produce welded tetrahedral meshes with incorrect topology. We use the method described in Section 8 to create an undeformed tetrahedral mesh with correct un-welded topology, despite self-collisions. We spent three minutes generating a (non-neutral) self-intersection-free mesh in Maya. Then it took three minutes to create a tetrahedral mesh, and one minute to produce the topologically correct neutral tetrahedral mesh. Next, it took six iterations in four minutes of user time to fit the initial frames, and then 46 seconds to fit the animation. The feet of the dragon are constrained to perform a rising action similar to the one used in [65]. We include the entire dragon except the back feet into $\Gamma \setminus \Gamma''$. Implicit blending forces are added on the head and front feet. Our method produces vivid motion on the horns, whiskers, body and tail. By adjusting the weights temporally on the head and front feet, we make them follow the input strictly during the initial part of the animation, and exhibit secondary motion afterwards.

In Figure 15, we compare our method to TRACKS [18]. The original TRACKS is designed for cloth simulation, and we extend it by building Petrov-Galerkin constraints between the input triangle mesh animation and the embedded mesh animation driven by solid FEM simulation. The triangle mesh displacements v in those constraints $\mathcal{G}(v) = 0$ are then substituted with $Cu = v$, to create simulation constraints for the tetrahedral mesh Ω. We find that TRACKS constraints suppress global, spatially low-frequency motion. The generated high-frequency motion is less pronounced on solids than on cloth. In comparison, our method imposes no restrictions on $\Gamma \setminus \Gamma''$ to allow secondary motion.

Bird: We add physics to a keyframed bird animation as our third example (Figure 16). The model and the animation were purchased online in a 3D model store. We place the legs into Γ'', so that the animator's previous work on ground contact is preserved. Similarly, we place the ends of the hands into Γ'', so that the hand motion follows the input, which can be seen as a variant of inverse kinematics. The rest of the hand including the arm feathers, however, is placed into $\Gamma \setminus \Gamma''$, which gives secondary motion. We also add physics to the head and eyes.

Sumo Wrestler: In our fourth example (Figure 3), we enrich the motion of a dancing sumo wrestler with secondary motion on the belly, cheeks, derrière and thighs. We place the hands and legs (except thighs) into Γ'' to follow the artist's input. We also put the neck, and parts of the back and the hips into Γ'', confining the motion to reasonable poses. We put the lower half of the head into $\Gamma \setminus \Gamma''$, to produce secondary motion on the cheeks. The rest of the head is in Γ''. Our method produces large deformations when the wrestler is dancing and jumping. The thighs and cheeks exhibit jiggling motion as well.

Head: In our fifth example, we complete the deformed shapes of a human head triangle mesh Γ, based on the input motion of a submesh Γ' (Figure 1). The input motion was computed using a proprietary optical flow-like tracking algorithm at a major computer animation company. Due to occlusions, it is missing all the deformation detail in the inner lips and the mouth cavity. We compare our method to a standard surface deformation method [64] (Figure 17). We find that the surface deformation method produces collisions, especially in the mouth cavity and inside the eyes, as these regions contain extremely concave geometry. Surface deformation methods also require a single 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 tetrahedral mesh Ω for the entire head using the procedure described in Section 8. Then we fitted tetrahedral mesh deformations $\bar{u}_i$ based on the input tracked triangle mesh motion.

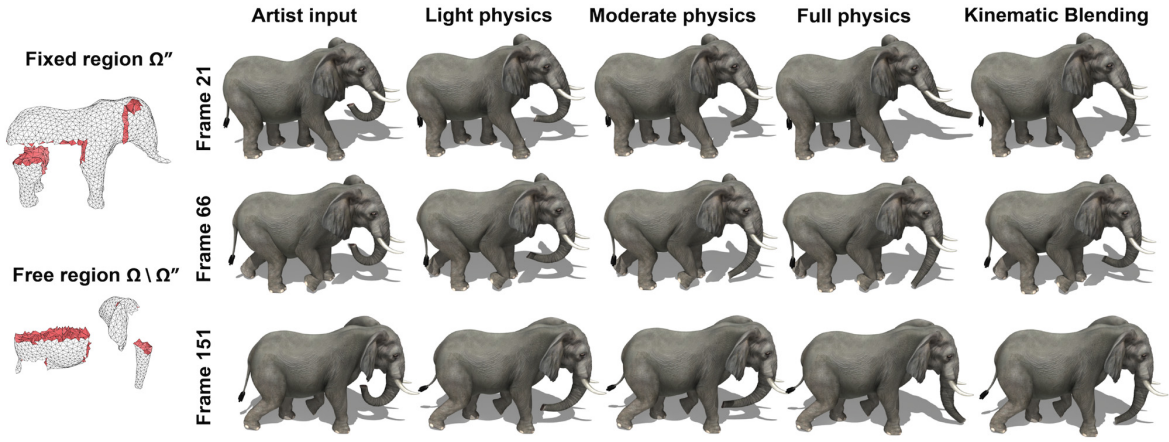


Fig. 13: Adding physics to an elephant walk cycle. *Left-most column: the fixed Ω'' and free $\Omega \setminus \Omega''$ region of the elephant tetrahedral mesh. Free regions include the trunk, ears, belly and tail. Left four columns: Each column is a separate animation with a different blending weight (Equation 11) between the artist input and physics. Implicit blending method was used. From left to right, the weights are decreased from following the input 100%, to a 100% physically enriched result. In $\Gamma \setminus \Gamma''$, our method produces secondary motion according to the weights, whereas in Γ'', the output is identical to the input. Right-most column: kinematic blending with 50% physics. It produces similar results as the implicit blending method, but suffers from less intuitive tuning of collision handling, such as between the legs and belly.*

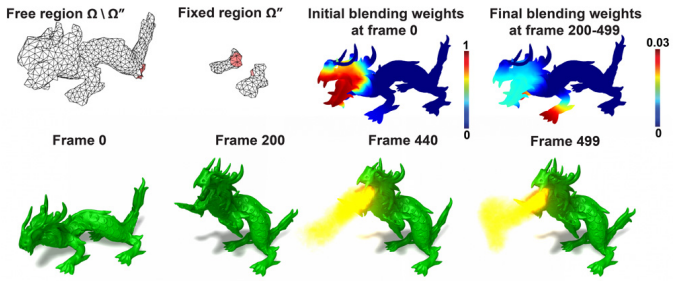


Fig. 14: Adding physics to a dragon animation. *The fixed region consists of the two back feet. The blending weights (third and fourth subfigures in the first row) change through time to follow input at first, and add secondary motion afterwards. Note that the weights on the back feet are ignored because they are in Γ''. The second row shows the resulting animation where the dragon rises up. For cinematic effect, we add a particle-effect fire animation. Secondary motion is present in the horns, whiskers, body and tail.*

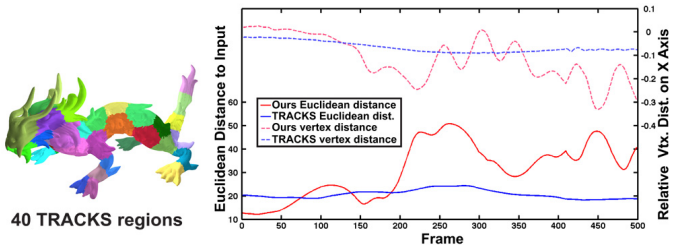


Fig. 15: Our method produces more global motion than TRACKS *with 40 automatically-generated regions. Two comparisons are made at each frame: (1) the Euclidean distance between the displacement $u_k \in \mathbb{R}^{3n}$ and the input frame $\bar{u}_k \in \mathbb{R}^{3n}$, and (2) the difference in the x-axis position of a selected dragon vertex between the output and input.*

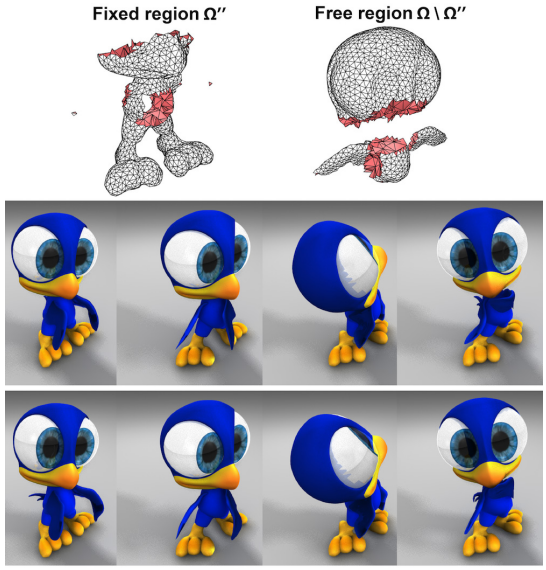


Fig. 16: Adding physics to a keyframed bird animation. *The top row shows the fixed Ω'' and free $\Omega \setminus \Omega''$ bird regions. Free regions are the head, belly and wings, except the wing tips which are fixed to drive the wings, similar to inverse kinematics. The middle row shows several frames of the input motion. The bottom row shows our result with subtle secondary motion added. This example does not use blending.*

Since the tracked data already contains facial dynamics, we skip the simulation part of our pipeline, and interpolate the complete mesh Γ using $\bar{u}_i$. The teeth are handled separately by extracting the closest rigid transformation to the interpolated shapes because they are rigid. In this example, we did not add back the input high-frequency detail as the input detail contains tracking noise, rather than something that was intelligently designed by the artist (as in Figure 6). Actually, the input data contains tracking imperfections, mostly along the perimeter of the lips, which our method resolved.

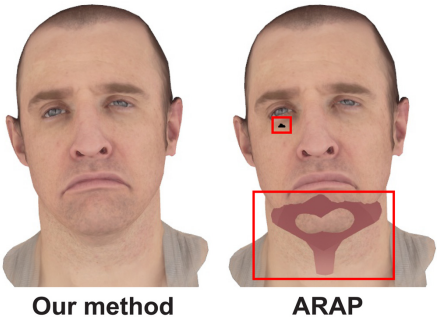


Fig. 17: Comparison to as-rigid-as-possible energy: *Left: the deformation produced by our method. Right: the shape produced by as-rigid-as-possible energy [Sorkine and Alexa 2007]. In order to improve the as-rigid-as-possible result, we clamp the negative weights in the as-rigid-as-possible energy; otherwise, the as-rigid-as-possible method causes the obtuse mesh triangles to collapse. We performed the comparison by setting the positions of the tracked mesh Γ' as constraints for Γ and minimized the as-rigid-as-possible energy. Lacking interior information, the as-rigid-as-possible energy resulted in a colliding shape at extremely concave regions such as the mouth cavity and eyelids, whereas our method produces a good result.*

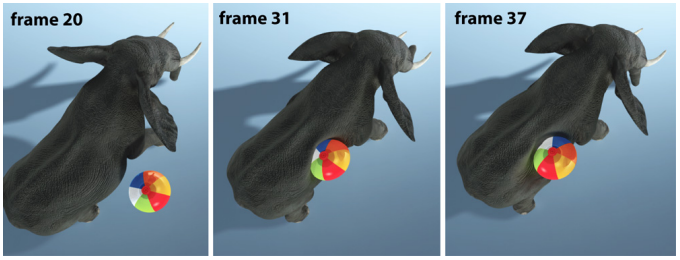


Fig. 18: Adding physically based contact to artist animations: *The soft ball is launched at the elephant. The ears, trunk, belly and the tail are enriched with physics just as in Figure 13. Our system performs collision detection between the elephant and the ball. Upon impact, contact forces are computed and added to both objects, resulting in local contact deformations on both objects, all the while the elephant still generally follows the original animation.*

We processed 12 facial animations, each containing about 80 frames. Fitting each of these animations took 1.8 minutes on average.

Fitting Parameters: The meaning of γ changes if one adjusts the material stiffness (Youngs modulus). Our default Young's modulus for fitting is $10^7 N/m^2$. Some examples require heterogeneous materials, in which case we keep the average Young's modulus in the same range. We use Poisson's ratio of 0.45 in all the examples.

A wide range of γ is used across our examples. A good γ depends not only on the size and overall stiffness of the object, but also on the input animation quality. In the elephant example, the input animation has quite large non-smooth deformations on the legs which generate huge internal elastic potential. We have to use a very small γ (10^{-10}) to achieve a tight fit. In contrast, the motion of the plant is very gentle and smooth, and even a large γ can produce a good result. Besides, a large γ (10^{-2}) helps filter out the local high-frequency dynamic details, which we remove during fitting, and add them back after the simulation.

10 Conclusion and Future Work

We have augmented general, arbitrary triangle mesh animations with physics. Our method enables artistically driven simulation of three-dimensional solid objects. We allow balance between input and physics. We augment coarse tetrahedral mesh simulation with the preservation of spatial dynamic detail.

Our system has the limitation that the input animation must be 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 wants physically based animations that do obey the input at least somewhat; otherwise, one can simply run standard physically based simulation. We augment coarse tetrahedral mesh simulation with the preservation of spatial dynamic detail. While our method is not tied to a particular interpolation method, barycentric coordinates are only $\mathcal{C}^0$ at the tetrahedral boundaries. Smoother interpolation methods such as mean value coordinates [50] could be used instead. Our method cannot generate deformations smaller than the volumetric mesh resolution. Therefore, it can be hard to create wrinkles and other small details. In practice, one desires a mesh that resolves the appropriate deformation detail as well as performs efficiently. This fundamental tradeoff in computer simulation requires experimentation and user iteration.

If the input deformations are unreasonable or chaotic, fitting may produce severely strained volumetric mesh animations, which in turn will make the simulation unstable. Adding anatomical detail to our solid models would improve animation realism. We would also like to extend our method to shells (cloth), and apply model reduction to accelerate the simulation.

Acknowledgments

This research was sponsored in part by the National Science Foundation (CAREER-1055035, IIS-1422869), the Sloan Foundation, the Okawa Foundation, and USC Annenberg Graduate Fellowship to Yijing Li and Hongyi Xu.

References

[1] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically Deformable Models," *Computer Graphics (Proc. of ACM SIGGRAPH 87)*, vol. 21(4), pp. 205–214, 1987.

[2] D. Baraff and A. P. Witkin, "Large Steps in Cloth Simulation," in *Proc. of ACM SIGGRAPH 98*, July 1998, pp. 43–54.

[3] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr, "Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling," in *Proc. of ACM SIGGRAPH 2001*, August 2001, pp. 31–36.

[4] M. Müller and M. Gross, "Interactive Virtual Materials," in *Proc. of Graphics Interface*, 2004, pp. 239–246.

[5] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis, "Efficient elasticity for character skinning with contact and collisions," in *ACM Trans. on Graphics (TOG)*, vol. 30, no. 4. ACM, 2011, p. 37.

[6] G. Irving, J. Teran, and R. Fedkiw, "Invertible Finite Elements for Robust Simulation of Large Deformation," in *Symp. on Computer Animation (SCA)*, 2004, pp. 131–140.

[7] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw, "Robust Quasistatic Finite Elements and Flesh Simulation," in *Symp. on Computer Animation (SCA)*, 2005, pp. 181–190.

[8] S. Capell, M. Burkhart, B. Curless, T. Duchamp, and Z. Popović, "Physically based rigging for deformable characters," in *Symp. on Computer Animation (SCA)*, Jul. 2005, pp. 301–310.

[9] N. Galoppo, M. A. Otaduy, S. Tekin, M. Gross, and M. C. Lin, "Soft articulated characters with fast contact handling," in *Computer Graphics Forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 243–253.

[10] J. Kim and N. S. Pollard, "Fast simulation of skeleton-driven deformable body characters," *ACM Trans. on Graphics (TOG)*, vol. 30, no. 5, p. 121, 2011.

[11] F. Faure, B. Gilles, G. Bousquet, and D. K. Pai, "Sparse meshless models of complex deformable solids," in *ACM Trans. on Graphics (SIGGRAPH 2011)*, vol. 30, no. 4, 2011, p. 73.

[12] T. Kim and D. L. James, "Physics-based character skinning using multidomain subspace deformations," *IEEE Trans. on Visualization and Computer Graphics*, vol. 18, no. 8, pp. 1228–1240, 2012.

[13] L. Liu, K. Yin, B. Wang, and B. Guo, "Simulation and control of skeleton-driven soft body characters," *ACM Trans. on Graphics (SIGGRAPH Asia 2013)*, vol. 32, no. 6, p. 215, 2013.

[14] M. Kass and J. Anderson, "Animating oscillatory motion with overlap: Wiggly splines," *ACM Trans. on Graphics (SIGGRAPH 2008)*, vol. 27, no. 3, pp. 28:1–28:8, 2008.

[15] X. Shi, K. Zhou, Y. Tong, M. Desbrun, H. Bao, and B. Guo, "Example-based dynamic skinning in real time," in *ACM Transactions on Graphics (SIGGRAPH 2008)*, vol. 27, no. 3, 2008, pp. 29:1–29:8.

[16] F. Hahn, S. Martin, B. Thomaszewski, R. Sumner, S. Coros, and M. Gross, "Rig-space physics," *ACM Trans. on Graphics (SIGGRAPH 2012)*, vol. 31, no. 4, pp. 72:1–72:8, 2012.

[17] F. Hahn, B. Thomaszewski, S. Coros, R. W. Sumner, and M. Gross, "Efficient simulation of secondary motion in rig-space," in *Symp. on Computer Animation (SCA)*, 2013, pp. 165–171.

[18] M. Bergou, S. Mathur, M. Wardetzky, and E. Grinspun, "TRACKS: Toward directable thin shells," *ACM Trans. on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, pp. 50:1–50:10, 2007.

[19] H. Wang, F. Hecht, R. Ramamoorthi, and J. F. O'Brien, "Example-based wrinkle synthesis for clothing animation," in *ACM Trans. on Graphics (TOG)*, vol. 29, no. 4. ACM, 2010, p. 107.

[20] D. Rohmer, T. Popa, M.-P. Cani, S. Hahmann, and A. Sheffer, "Animation wrinkling: augmenting coarse cloth simulations with realistic-looking wrinkles," in *ACM Trans. on Graphics (SIGGRAPH Asia 2010)*, vol. 29, no. 6, 2010, p. 157.

[21] M. Müller and N. Chentanez, "Wrinkle meshes," in *Symp. on Computer Animation (SCA)*, 2010, pp. 85–92.

[22] J. Barbič, M. da Silva, and J. Popović, "Deformable object animation using reduced optimal control," *ACM Trans. on Graphics (SIGGRAPH 2009)*, vol. 28, no. 3, pp. 53:1–53:9, 2009.

[23] J. Huang, Y. Tong, K. Zhou, H. Bao, and M. Desbrun, "Interactive shape interpolation through controllable dynamic deformation," *IEEE Trans. on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 983–992, 2011.

[24] K. Hildebrandt, C. Schulz, C. von Tycowicz, and K. Polthier, "Interactive spacetime control of deformable objects," *ACM Trans. on Graphics (SIGGRAPH 2012)*, vol. 31, no. 4, pp. 71:1–71:8, 2012.

[25] S. Li, J. Huang, F. de Goes, X. Jin, H. Bao, and M. Desbrun, "Space-time editing of elastic motion through material optimization and reduction," *ACM Trans. on Graphics (SIGGRAPH 2014)*, vol. 33, no. 4, pp. 108:1–108:10, 2014.

[26] S. Li, J. Huang, M. Desbrun, and X. Jin, "Interactive elastic motion editing through space–time position constraints," *Computer Animation and Virtual Worlds*, vol. 24, no. 3-4, pp. 409–417, 2013.

[27] C. Schulz, C. von Tycowicz, H.-P. Seidel, and K. Hildebrandt, "Animating deformable objects using sparse spacetime constraints," *ACM Trans. on Graphics (SIGGRAPH 2014)*, vol. 33, no. 4, pp. 109:1–109:10, 2014.

[28] R. Kondo, T. Kanai, and K. ichi Anjyo, "Directable animation of elastic objects," in *Symp. on Computer Animation (SCA)*, Jul. 2005, pp. 127–134.

[29] J. Barbič and J. Popović, "Real-time control of physically based simulations using gentle forces," *ACM Trans. on Graphics (SIGGRAPH Asia 2008)*, vol. 27, no. 5, pp. 163:1–163:10, 2008.

[30] S. Coros, S. Martin, B. Thomaszewski, C. Schumacher, R. Sumner, and M. Gross, "Deformable objects alive!" *ACM Trans. on Graphics (SIGGRAPH 2012)*, vol. 31, no. 4, p. 69, 2012.

[31] J. Barbič, F. Sin, and E. Grinspun, "Interactive editing of deformable simulations," *ACM Trans. on Graphics (SIGGRAPH 2012)*, vol. 31, no. 4, 2012.

[32] J. Choi and A. Szymczak, "Fitting solid meshes to animated surfaces using linear elasticity," *ACM Trans. on Graphics (TOG)*, vol. 28, no. 1, pp. 6:1–6:10, 2009.

[33] S. Wuhrer, J. Lang, M. Tekieh, and C. Shu, "Finite element based tracking of deforming surfaces," *Graphical Models*, vol. 77, pp. 1–17, 2015.

[34] G.-P. Paillé, P. Poulin, and B. Lévy, "Fitting polynomial volumes to surface meshes with voronoï squared distance minimization," in *Computer Graphics Forum*, vol. 32, no. 5, 2013, pp. 103–112.

[35] K. Ø. Noe and T. S. Sørensen, "Solid mesh registration for radiotherapy treatment planning," in *Biomedical Simulation*. Springer, 2010, vol. 5958, pp. 59–70.

[36] H. Lu, C. Xian, G. Li, and Z. Zhang, "EC-CageR: error controllable cage reverse for animated meshes," *Computers & Graphics*, vol. 46, pp. 138–148, 2014.

[37] Y. Savoye and J.-S. Franco, "CageIK: dual-Laplacian cage-based inverse kinematics," in *Articulated Motion and Deformable Objects*. Springer, 2010, vol. 6169, pp. 280–289.

[38] L. Chen, J. Huang, H. Sun, and H. Bao, "Cage-based deformation transfer," *Computers & Graphics*, vol. 34, no. 2, pp. 107–118, 2010.

[39] J.-M. Thiery, J. Tierny, and T. Boubekeur, "CageR: Cage-Based Reverse Engineering of Animated 3D Shapes," in *Computer Graphics Forum*, vol. 31, no. 8, 2012, pp. 2303–2316.

[40] Y. Savoye and J.-S. Franco, "Conversion of performance mesh animation into cage-based animation," in *ACM SIGGRAPH ASIA 2010 Posters*. ACM, 2010, p. 2.

[41] H. Si and A. TetGen, "A quality tetrahedral mesh generator and three-dimensional delaunay triangulator," *Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany*, 2006.

[42] J. Schoberl, "NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules," *Comput.Visual.Sci*, vol. 1, pp. 41–52, 1997.

[43] H. Xu and J. Barbič, "Signed distance fields for polygon soup meshes," in *Proc. of Graphics Interface*, 2014, pp. 35–41.

[44] J.-D. Boissonnat and S. Oudot, "Provably good sampling and meshing of surfaces," *Graphical Models*, vol. 67, no. 5, pp. 405–451, 2005.

[45] L. Sacht, E. Vouga, and A. Jacobson, "Nested cages," *ACM Trans. on Graphics (SIGGRAPH Asia 2015)*, vol. 34, no. 6, p. 170, 2015.

[46] H. Si, "Adaptive tetrahedral mesh generation by constrained Delaunay refinement," *International Journal for Numerical Methods in Engineering*, vol. 75, pp. 856–880, 2008.

[47] H. Si and K. Gärtner, "3d boundary recovery by constrained delaunay tetrahedralization," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 11, pp. 1341–1364, 2011.

[48] F. S. Sin, D. Schroeder, and J. Barbič, "Vega: Non-linear fem deformable object simulator," in *Computer Graphics Forum*, vol. 32, no. 1. Wiley Online Library, 2013, pp. 36–48.

[49] A. Stomakhin, R. Howes, C. Schroeder, and J. M. Teran, "Energetically consistent invertible elasticity," in *Symp. on Computer Animation (SCA)*, 2012, pp. 25–32.

[50] T. Ju, S. Schaefer, and J. Warren, "Mean value coordinates for closed triangular meshes," in *ACM Trans. on Graphics (SIGGRAPH 2005)*, vol. 24, no. 3, 2005, pp. 561–566.

[51] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, "Harmonic coordinates for character articulation," in *ACM Trans. on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, 2007, pp. 71:1–71:9.

[52] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical recipes: The art of scientific computing*, 3rd ed. Cambridge University Press, Cambridge, UK, 2007.

[53] M. Alexa, D. Cohen-Or, and D. Levin, "As-rigid-as-possible shape interpolation," in *Proc. of ACM SIGGRAPH 2000*, 2000, pp. 157–164.

[54] E. Sifakis and J. Barbic, "FEM simulation of 3D deformable solids: a practitioner's guide to theory, discretization and model reduction," in *ACM SIGGRAPH 2012 Courses*, 2012, p. 20.

[55] H. Zhang, "Discrete combinatorial laplacian operators for digital geometry processing," in *Proceedings of SIAM Conference on Geometric Design and Computing*. Nashboro Press, 2004, pp. 575–592.

[56] B. Bickel, M. Baecher, M. Otaduy, W. Matusik, H. Pfister, and M. Gross, "Capture and modeling of non-linear heterogeneous soft tissue," *ACM Trans. on Graphics (SIGGRAPH 2009)*, vol. 28, no. 3, pp. 89:1–89:9, 2009.

[57] L. Sacht, A. Jacobson, D. Panozzo, C. Schüller, and O. Sorkine-Hornung, "Consistent volumetric discretizations inside self-intersecting surfaces," in *Computer Graphics Forum*, vol. 32, no. 5. Wiley Online Library, 2013, pp. 147–156.

[58] A. Jacobson, L. Kavan, and O. Sorkine-Hornung, "Robust inside-outside segmentation using generalized winding numbers," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 33, 2013.

[59] J. Teran, E. Sifakis, S. S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw, "Creating and simulating skeletal muscle from the visible human data set," *IEEE Trans. on Visualization and Computer Graphics*, vol. 11, no. 3, pp. 317–328, 2005.

[60] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure, "Preserving topology and elasticity for embedded deformable models," *ACM Trans. on Graphics (SIGGRAPH 2009)*, vol. 28, no. 3, pp. 52:1–52:9, 2009.

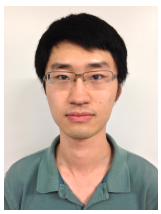
[61] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection," in *Proc. of ACM SIGGRAPH 96*, 1996, pp. 171–180.

[62] R. Bridson, R. Fedkiw, and J. Anderson, "Robust Treatment of Collisions, Contact, and Friction for Cloth Animation," *ACM Trans. on Graphics (SIGGRAPH 2002)*, vol. 21, no. 3, pp. 594–603, 2002.

[63] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw, "Hybrid simulation of deformable solids," in *Symp. on Computer Animation (SCA)*, 2007, pp. 81–90.

[64] O. Sorkine and M. Alexa, "As-rigid-as-possible surface modeling," in *Symp. on Geometry processing*, vol. 4, 2007, pp. 109–116.

[65] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt, "PriMo: Coupled Prisms for Intuitive Surface Modeling," in *Eurographics Symp. on Geometry Processing*, 2006, pp. 11–20.



Yijing Li is a PhD student in the Department of Computer Science, Viterbi School of Engineering, University of Southern California. His research interests are computer graphics and physically based animation. He received his undergraduate degree from Tsinghua University in 2013.

Hongyi Xu Hongyi Xu is a PhD student in computer science at the University of Southern California. He obtained his BS degree from Zhejiang University. His research interests are in computer graphics, physically based animation, contact and interactive physics.

Jernej Barbič is an associate professor of computer science at USC. In 2011, MIT Technology Review named him one of the Top 35 Innovators under the age of 35 in the world (TR35). Jernej published several papers on nonlinear solid deformation modeling, collision detection and contact, and interactive design of deformations and animations. He is the author of Vega FEM, an efficient free C/C++ software physics library for deformable object simulation. Jernej is a Sloan Fellow (2014).