Volume xx (200y), Number z, pp. 1–16

Meshless Shape and Motion Design for Multiple Deformable Objects

Bart Adams[1], Martin Wicke[2,3], Maks Ovsjanikov[2], Michael Wand[3], Hans-Peter Seidel[4], Leonidas J. Guibas[2]

[1] Katholieke Universiteit Leuven
[2] Stanford University
[3] Max Planck Center for Visual Computing and Communication
[4] Max Planck Institut für Informatik

Abstract

We present physically based algorithms for interactive deformable shape and motion modeling. We coarsely sample the objects with simulation nodes, and apply a meshless finite element method to obtain realistic deformations at interactive frame rates. This shape deformation algorithm is then used to specify keyframe poses and a smooth interpolating motion is obtained by solving for an energy-minimizing trajectory. We show how to handle collisions between different deformable objects as well as with static or moving scene objects. Secondary motion is added as a post-process by running a meshless elastic solid simulation. We enforce precomputed trajectories using control forces computed using shape matching. Key to the efficiency of our method is a sparse deformation representation and an adaptive optimization algorithm that automatically introduces new degrees of freedom in problematic regions. An accurate temporal interpolation scheme that exactly recovers rigid motions keeps the number of unknowns low and achieves realistic deformations with very few keyframes. We also show how the algorithm allows combining purely physical simulation with keyframe-based scripted animation. The presented results illustrate that our framework can handle complex shapes at interactive rates, making it a valuable tool for animators to realistically model deformable 3D shapes and their motion.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation I.3.5 [Computational Geometry and Object Modeling]: Physically based modeling

1. Introduction

Deformable shapes are used extensively in physics-based simulations for the animation of elastic and plastic solids. Although recent advances allow for interactive simulations, controlling the behavior of the deforming objects (for example by tweaking the physical forces) is difficult. Often, a tedious process of trial and error is necessary to generate a specific result. Instead of using physics-based simulations, one can explicitly model keyframe poses of the deformable object at certain points in time, and interpolate a smooth motion between the keyframes (cf. Fig. 1).

We first present an efficient modeling framework for complex deformable shapes. From a coarse set of strategically placed sample points, called nodes, a continuous deformation field is computed that adequately represents the desired shape deformation. After specifying handle constraints, meshless finite elements are used to compute the energy-optimal deformation that aims to respect the original shape, minimizing internal strain and change in volume. An efficient surface deformation algorithm enables modeling with high resolution meshes.

We extend this shape modeling framework to enable motion design of deforming shapes. Given user-specified keyframe poses at discrete time instances, we compute energy-optimal shape trajectories. As before, the meshless elasticity constraints ensure that objects deform naturally. To facilitate modeling of complex scenes, we propose to add energies to the optimization that enforce smooth trajectories, and handle collisions with other deformable objects as well as static or moving rigid scene objects. An adaptive optimization adds degrees of freedom where necessary, allowing us to efficiently model scenes with many objects and scene obstacles.

To further enhance realism, we run a dynamic elasticity simulation as a post-process. Using control forces that are generated from the computed energy-optimal trajectories, the simulated dynamics add secondary effects such as jiggling and

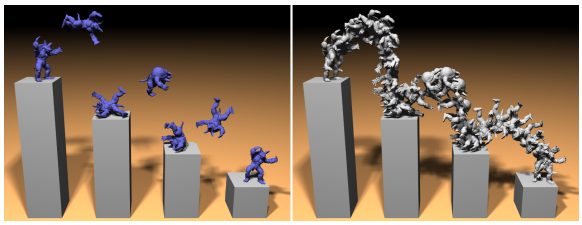

Figure 1: *Our method allows rapid modeling of deformable shapes and their motion. Left: keyframe poses obtained using our deformation algorithm. Right: smooth interpolated motion obtained using our shape interpolation algorithm.*

high-frequency vibrations to the objects. This technique also allows us to switch between designed and simulated motion by enabling or disabling the control forces.

Contributions We propose a method for interactive and realistic deformation modeling of highly complex shapes in real-time and use it as the basis for a novel algorithm for computing optimized trajectories of multiple deformable objects given keyframe poses specified by an animator. We extend existing work on meshless finite elements by introducing a novel node coupling based on material distance. We propose an efficient temporal interpolation scheme based on shape matching and motion interpolation using spherical cubic splines of unit quaternions that exactly recovers rigid motions. We propose a collision handling algorithm to enable the design of collision-free motion paths for multiple deformable objects. We discuss methods to add secondary dynamics to designed motion paths and to combine designed with simulated animations.

This paper is an improved and extended version of [AOW*08]. For sake of completeness, we discuss all components and indicate the differences and improvements where appropriate.

2. Related Work

This paper is related to a large body of work in geometric shape modeling, inverse kinematics, shape interpolation, motion planning and animation control. We limit the discussion to the most relevant work in these areas.

2.1. Shape Modeling

The core component of this paper is a deformation algorithm that allows to realistically deform a shape by specifying handle constraints. This problem has received considerable attention in recent literature. In the following we summarize the most closely related work and refer to [NMK*05, BS08] for extensive surveys on other deformation models.

A number of related methods perform shape deformations by direct mesh optimization such as [SK04, SYBF06, LSCO*04, SLCO*04]. The underlying motivation of these methods is to explicitly preserve the local shape properties while applying user-specified deformations. In a conceptually similar way, inverse-kinematics based methods restrict the space of natural deformations by either exploring the set of example poses [SZGP05] or by inferring the deformations on the skeleton structure of the shape [SZT*07]. Although mesh-based methods give a high degree of freedom in manipulating the shape, they suffer from the restrictive complexity of constraining and estimating per-vertex deformations. Multiresolution methods such as [BK03, SYBF06, BSS07] have been designed to improve efficiency.

Motivated by the intuition that in many scenarios, shape deformations can be encoded using only a few motion parameters, researchers have proposed techniques that use deformable models with a significantly reduced dimensionality as compared to the full geometric complexity (e.g., [JP02, JT05, DSP06, AFTCO07]). In this paper we use a reduced space deformation technique that allows computing long animation sequences for highly detailed deformable shapes. Of the large body of work in this area, the most immediately pertinent are [BPGK06] in which a prism based shell energy is formulated and solved efficiently, and [BPWG07] where a similar elastic energy is extended to rigid volumetric cells. Although the latter provides a simplified deformation field, it is both topology unaware and employs an interpolation scheme that results in solving a large sparse linear system making it prohibitively slow in our setting. Huang et al. [HSL*06] present a gradient domain mesh deformation technique that preserves volume and rigidity of limb segments of articulated figures. They propose a subspace technique by solving the problem on a control mesh. Somewhat differently, Funck et al. [vFTS06] design a set of vector field based deformation tools that guarantee non-intersecting and volume-preserving shape deformations. Their system seems to be more geared towards model creation as opposed to shape deformations. Our work is most similar in spirit to [SSP07] where the deformation field is discretized, solved for and interpolated using a sparse topology graph. Although we use a similar paradigm, we avoid estimating the rotation and translation components of the deformation field separately, and employ an interpolation scheme which guarantees first-order consistency. Moreover, our method introduces less unknowns for the same number of nodes. Finally, Stoll [Sto07] presents a tetrahedral deformation approach that iterates between a linear Laplacian step and a differential update step. Again, their deformation interpolation method is not guaranteed to be consistent.

Note that our deformation algorithm is also related to work that uses barycentric-like coordinates to interpolate the deformation field inside a coarse control mesh (see [FKR05, LKCOL07, JMD*07] for different flavors). Unlike these methods, however, we restrict the space of possible deformations to only include realistic, shape-preserving deformations. This facilitates the animator's task and allows intuitive shape modeling by only constraining or dragging points on the shape itself, without having to model and deform a cage.

2.2. Motion Design

Rigid (Multi-)Body Motions A significant number of researchers have tackled the problem of designing and controlling animations involving rigid bodies. Among others, Isaacs and Cohen [IC87] and Barzel and Barr [BB88] propose ways to control rigid body simulations using control forces and inverse dynamics. The seminal work of Witkin and Kass [WK88] allows the animator to specify a sparse set of spacetime constraints and the resulting motion is computed using sequential quadratic programming. Witkin and Popovic [WP95] present a simple technique for editing keyframe animations based on warping. Popovic et al. [PSE*00] present an intuitive interface that allows interactive editing of rigid body positions and velocities, while the system interactively updates the computed motion. They propose a way to handle collisions by separating the simulation function in a pre-collision, collision and post-collision part. In [PSE03], a sketching interface is presented that extends [PSE*00] and allows intuitive motion sketching for multiple rigid bodies possibly connected with joints. More recently, Hofer et al. [HPR04, Hof04] solve the problem of finding a smooth rigid body motion that interpolates given keyframes using curve design algorithms. They obtain smooth collision-free motions for multiple rigid bodies. Twigg and James [TJ07] present a method to obtain the desired result of a multibody simulation by computing many slightly perturbed example simulations in parallel on a cluster while the user is allowed to browse and modify them interactively. In [TJ08], the authors go against the usual flow of time and propose methods to allow backward time-stepping for rigid body simulations. This makes their method particularly useful for specifying final and intermediate states, but the method cannot be constrained by user-designed initial *and* final states.

A large body of literature treats the problem of motion planning in robotics. A recent overview can be found in [LaV06].

Motion Design for Deformable Objects With the emergence of efficient shape deformation algorithms (cf. Sec. 2.1), authors have worked to extend motion modeling algorithms to handle deformable objects. Kondo et al. [KKiA05] present a dynamic simulation method for deformable objects that lets the user specify keyframes or modify trajectories which are then used to guide the physics-based deformable simulation. Exact keyframe interpolation is hard to achieve and keyframes have to be spaced relatively far apart in order to obtain realistic and stable results. Wojtan et al. [WMT06] blend user defined constraints with physically based simulation by seeking an optimal set of external control forces to minimize a non-linear least squares system. A force is defined per particle per frame, which makes the system rather large for long animations and many particles. Jeon and Choi [HJ07] present a control technique for deformable objects represented as mass-spring systems. Xu et al. [XZY*07] propose a gradient domain based method for editing deforming mesh sequences. To maintain scalability, their method requires a manually designed coarse control mesh. The work of Kilian et al. [KMP07] allows for designing deformable motions by morphing between shapes along geodesic paths in shape space. Their method is computationally too expensive to handle many deformable objects.

This paper extends the work of [AOW*08] in which the authors present a non-linear optimization strategy to compute motions given user-defined constraints. Their temporal motion representation lacks consistency and does not recover rotations, which results in an excessive number of temporal sample points to represent even simple deformable motions, making animation of many objects intractable. We solve these problems by introducing an improved temporal interpolation method that exactly recovers rigid motions and hence drastically reduces the number of necessary unknowns.

Secondary Dynamics Bergou et al. [BMWG07] present a process, dubbed tracking, which enhances a rough animation or simulation of a surface with physically simulated detail. Kass and Anderson [KA08] solve the spacetime constraints problem over the domain of complex numbers where the imaginary part of the solution defines a phase angle that the authors use to control and generalize the oscillatory behavior of a deformable animation. The resulting *wiggly splines* allow intentional oscillations that are easy to control with familiar tools by the animators. Our work is also related to the volumetric formulation in Capell et al. [CGC*02], where skeletal deformations are extended to the entire shape in a way that simulates secondary motion. In this work, we also propose a technique to add secondary dynamics such as jiggling, but use a method more closely related to the one of Shi et al [SZT*08], where the authors present an approach to enrich skeleton-driven animations with physically-based secondary deformations using stable target forces. The proposed technique in this paper not only allows adding secondary dynamics to an existing motion, but it also allows seamless switching between purely physical animations and designed animations.

3. Overview

This paper discusses a modeling framework for shape deformations and motions of deformable objects. Both components rely on the same meshless spatial discretization of an object, which we describe in Sec. 4. We represent deformations as a vector field discretized at sample points, called nodes, that are carefully chosen, such that only few samples are needed even for complex objects.

Sec. 5 discusses how to use the deformation representation for interactive shape modeling. Given modeling constraints set by the user, we minimize a quasi-static continuum elasticity energy to find a natural deformation that satisfies these constraints.

In Sec. 6, we generalize the optimization approach for shape modeling to motion modeling. We discretize the deformation field in space and time, and optimize for a time-dependent field which minimizes an energy measuring both deformations and smoothness of the trajectory. We also add energy terms for collision handling, both between deformable objects and with static or moving scene objects. We describe an adaptive optimization scheme that refines the temporal sampling only where necessary, thus allowing for simulation of large scenes with many objects.

Since the motions obtained by optimization are smooth, there are no dynamic effects such as jiggling and vibrations. In Sec. 7, we show how secondary dynamics effects can be added by running a dynamic simulation on the deformable objects. The computed trajectories are then only used to obtain control forces that steer the dynamic simulation. This also allows us to seamlessly combine motions obtained by keyframe design and pure simulation: By simply switching the control forces on and off, we can choose between designed and simulated motions.

4. Shape Representation

Our deformation representation is based on classical meshless finite elements (see [FM03] for a good overview), that were recently introduced in computer graphics for physically based animations (e.g., [MKN*04, PKA*05, GQ05, GP07]). We use a similar formulation as [MKN*04], but instead of computing the dynamic behavior, we solve for the deformation field given user-specified position constraints. Moreover, in the work of [MKN*04], nodes are coupled based on Euclidean distance, which does not allow adequate deformation modeling for nearby, but topologically separated features (e.g. two adjacent fingers in a hand model). This problem was addressed by Pauly et al. [PKA*05] for the simulation of fracturing materials by using a visibility driven transparency criterion [OFTB96]. However, this method only allows separating parts cut by a crack surface and does not allow defining an appropriate nodal coupling for general shapes. To resolve these problems, we propose a novel algorithm that defines an adequate nodal coupling by using distances within the material. A similar technique was proposed by Steinemann et al. [SOG06]. However, they approximate the material distances using a graph structure, resulting in distances that are not smooth and do not converge towards the continuous solution. We however propose to approximate the material distances using the fast marching method [Set99] which indeed results in a smooth distance field and convergence towards the correct solution. Even for low sampling densities, this technique results in proper deformations, while maintaining the flexibility of traditional meshless algorithms such as easy (re-)sampling and a smooth and consistent deformation field representation.

In the remainder of this section, we first discuss the meshless deformation field representation in Sec. 4.1. We then describe the nodal sampling and coupling algorithm in Sec. 4.2. Finally, Sec. 4.3 shows how the shape's boundary mesh can be efficiently deformed using the deformed nodes.

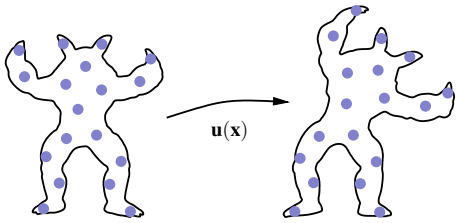

Figure 2: *We represent the shape's deformation at discrete nodes* $\mathbf{x}_i$ *and use meshless shape functions to approximate the deformation field over the whole shape as* $\mathbf{u}(\mathbf{x}) = \sum_i \Phi_i(\mathbf{x})\mathbf{u}_i$.

4.1. Deformation Field

We follow [MKN*04, AOW*08] and represent a shape's deformation field by a coarse set of N nodes, strategically sampled over the shape's volume using the algorithm detailed in Sec. 4.2. Each node at position $\mathbf{x}_i \in \mathbb{R}^3$ is assigned a displacement vector $\mathbf{u}_i \in \mathbb{R}^3$. Using meshless function approximation theory [FM03], we define a continuous displacement field from these vectors as (see also Fig. 2)

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{N} \Phi_i(\mathbf{x})\mathbf{u}_i. \tag{1}$$

Hence, a point $\mathbf{x}$ in the undeformed shape is matched to the position $\mathbf{f}(\mathbf{x}) = \mathbf{x} + \mathbf{u}(\mathbf{x})$ in the deformed shape. The shape functions $\Phi_i(\mathbf{x})$ are defined as

$$\Phi_i(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})[\mathbf{M}(\mathbf{x})]^{-1} w_i(\mathbf{x})\mathbf{p}(\mathbf{x}_i), \tag{2}$$

where $\mathbf{p}(\mathbf{x})$ is a complete polynomial basis of order n in $\mathbb{R}^3$ and $w_i(\mathbf{x}) = \max(0, (1 - d^2(\mathbf{x}, \mathbf{x}_i)/r_i^2)^3)$ is a smoothly decaying locally supported weight function (with the nodal support radius r_i and a suitable distance function $d(\cdot,\cdot)$). The moment matrix $\mathbf{M}(\mathbf{x})$ is defined as

$$\mathbf{M}(\mathbf{x}) = \sum_{i=1}^{N} w_i(\mathbf{x})\mathbf{p}(\mathbf{x}_i)\mathbf{p}^T(\mathbf{x}_i). \tag{3}$$

By using compactly supported weight functions the deformation field is locally only defined by a limited set of nodal deformations. We will use $n = 1$ and hence $\mathbf{p}(\mathbf{x}) = [1\ \mathbf{x}]^T$ as the basis to guarantee linear consistency in the meshless deformation field approximation.

The shape deformation algorithms that will be presented in this paper, use constraints and forces that require the gradient $\nabla\mathbf{u}$ of the displacement field, where $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$. These partial derivatives to the k-th component of $\mathbf{x}$ ($k = 1,2,3$), are obtained as

$$\frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}_{(k)}} = \sum_{i=1}^{N} \frac{\partial \Phi_i(\mathbf{x})}{\partial \mathbf{x}_{(k)}} \mathbf{u}_i, \tag{4}$$

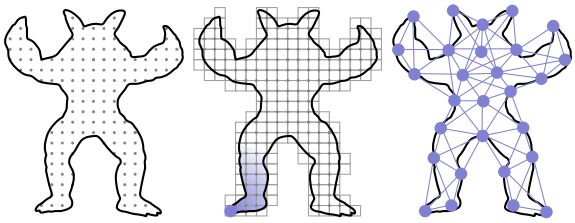

Figure 3: *Topology aware nodal sampling of the shape's interior. Left: the set of candidate points consists of the mesh vertices and a dense set of interior grid points. Middle: nodes are created iteratively by picking the point furthest away from all previously created nodes. A grid-based fast marching method is used to compute distances within the shape. The figure shows the first node and its influence region. Right: the resulting nodal coupling adequately separates parts that are close in Euclidean distance, but far in topological sense such as the legs in the picture.*

where $\partial\Phi_i(\mathbf{x})/\partial\mathbf{x}_{(k)}$ is computed from Eq. 2 using the product rule as

$$\begin{aligned}\frac{\partial\Phi_i(\mathbf{x})}{\partial\mathbf{x}_{(k)}} &= \frac{\partial w(\mathbf{x})}{\partial\mathbf{x}_{(k)}}\mathbf{p}^T(\mathbf{x})\mathbf{M}(\mathbf{x})^{-1}\mathbf{p}(\mathbf{x}_i)\\ &+ w(\mathbf{x})\mathbf{p}^T(\mathbf{x})\frac{\partial\mathbf{M}(\mathbf{x})^{-1}}{\partial\mathbf{x}_{(k)}}\mathbf{p}(\mathbf{x}_i)\\ &+ w(\mathbf{x})\frac{\partial\mathbf{p}^T(\mathbf{x})}{\partial\mathbf{x}_{(k)}}\mathbf{M}(\mathbf{x})^{-1}\mathbf{p}(\mathbf{x}_i),\end{aligned} \tag{5}$$

and by using the fact that

$$\frac{\partial(\mathbf{M}^{-1})}{\partial\mathbf{x}_{(k)}} = -\mathbf{M}^{-1}\left(\frac{\partial\mathbf{M}}{\partial\mathbf{x}_{(k)}}\right)\mathbf{M}^{-1} \tag{6}$$

with $\partial\mathbf{M}/\partial\mathbf{x}_{(k)}$ obtained from Eq. 3.

4.2. Nodal Sampling & Coupling

Given a sampling of the object with N nodes, there are $3N$ parameters in the deformation field representation. Our goal is to keep this number as low as possible, while allowing realistic shape deformations. When using a coarse set of nodes, it is important to appropriately define the nodal influence regions (i.e., the non-zero extent of the weight functions $w_i(\mathbf{x})$ in Eq. 2). Using Euclidean distances introduces undesirable undersampling artifacts when nodes influence parts of the shape that are not connected (for example, a node in one leg of a human would incorrectly influence the other leg). In this section we propose a fast nodal sampling algorithm that strategically covers the shape with a low number of nodes while guaranteeing appropriate nodal coupling.

We use farthest point sampling to create nodes from a set of candidate points defined as the union of the mesh vertices and a dense set of interior grid points (see Fig. 3, left). Our method iteratively picks the point $\mathbf{x}_i$ from this set that is farthest away from the already created nodes $\mathbf{x}_{i-1}, \mathbf{x}_{i-2}, \ldots, \mathbf{x}_0$ until the whole shape is sufficiently covered (the first node is picked randomly). The distance $d(\mathbf{x}, \mathbf{x}_i)$ to the node $\mathbf{x}_i$ is computed within the shape by solving the Eikonal equation

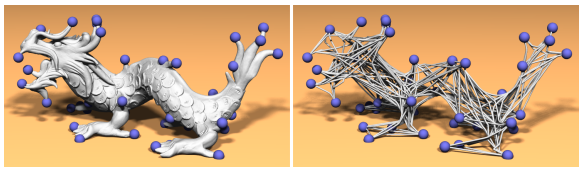

Figure 4: *Nodal sampling for the dragon model. The right image shows the coupling of the nodes as their shortest connecting paths within the dragon. The lengths of these paths are computed using a fast marching algorithm and are used in the shape function computations. Note that this results in a nodal coupling that respects the topology of the object.*

using a grid-based fast marching method [Set99] (see also the middle image in Fig. 3). This distance represents the *material distance* and corresponds to the length of the shortest path from $\mathbf{x}$ to $\mathbf{x}_i$ without leaving the shape. Using this distance in the weight function $w_i(\mathbf{x})$ to define the nodal shape functions (Eq. 2) ensures that nodes influence the appropriate regions. As noted above, this allows adequate modeling of shape deformations with nearby features such as the fingers of a hand, or the legs of a human.

In the current implementation we prescribe a uniform nodal influence radius $r_i = r$. Note that to guarantee non-singular moment matrices, necessary for deformation interpolation (cf. Eq. 3), every vertex $\mathbf{x}$ in the shape has to be within the support radius r of at least 4 non-planar nodes. We ensure this criterion during the node creation by keeping a count, for every $\mathbf{x}$, of the number of its covering nodes, i.e., those nodes $\mathbf{x}_i$ that are within a material distance r_i of $\mathbf{x}$. The sampling algorithm ends if every point is covered by at least 4 non-planar nodes. The proposed strategy results in a roughly uniform and sufficiently dense nodal sampling.

Note that the fast marching algorithm produces a first order accurate approximation to the in-material distances [Set99] and therefore it converges to the true distances if the grid resolution is increased. Moreover, the computed approximate distances are continuous, which guarantees smooth shape and surface deformations. Because the nodal sampling and coupling is computed in a preprocessing step and does not change during interaction, performance is not a real issue. Typical samplings take on the order of 1 to 5 seconds. Fig. 4 shows the resulting sampling for the dragon.

4.3. Surface Deformation

The approximation scheme described in Sec. 4.1 defines a space deformation: We can evaluate the deformation field at any position $\mathbf{x}$ that lies in the support of the nodes. Hence, we can use a high resolution surface and deform it using the deformation defined by the nodal displacements. In our algorithms we use triangle meshes, but the discussed surface deformation algorithm extends trivially to other explicit boundary representations, such as point-sampled surfaces represented as sets of surfels [PZvBG00].

In a preprocessing step, we compute for each mesh vertex $\mathbf{x}$ the set of nodes that have non-zero support at the vertex. Given these nodes, the shape functions $\Phi_i(\mathbf{x})$ and the gradient of the shape functions $\nabla\Phi_i(\mathbf{x})$ are computed using Eq. 1 and Eq. 4 respectively. This computation is only done once before beginning a modeling session. During modeling, the deformed vertex position $\mathbf{x}'$ is computed using Eq. 1 as $\mathbf{x}' = \mathbf{f}(\mathbf{x})$. Note that this amounts to computing a linear combination of the neighboring nodes' deformation vectors using the precomputed shape functions. Similarly, the updated (unnormalized) vertex normal $\mathbf{n}'(\mathbf{x})$ can be computed from the gradient of the deformation field as [Bar84]

$$\mathbf{n}'(\mathbf{x}) = \nabla\mathbf{f}^{-1^T}(\mathbf{x})\mathbf{n}(\mathbf{x}). \tag{7}$$

Updating the vertex normals using the above equation can be quite expensive since it involves one 4×4 matrix inversion for each boundary mesh vertex. During interactive manipulation we therefore use the following approximation

$$\mathbf{n}'(\mathbf{x}) = \nabla\mathbf{f}(\mathbf{x})\mathbf{n}(\mathbf{x}). \tag{8}$$

This is a good approximation as long as the deformation field is locally rigid, i. e., in the absence of shearing or (anisotropic) scaling. Since this approximation is made locally for *each* mesh vertex separately, it is sufficiently accurate in practice.

Using this approximation, we can further improve performance and reduce the matrix-vector multiplications to scalar-vector multiplications by expanding (8) to

$$\mathbf{n}'(\mathbf{x}) = \mathbf{n}(\mathbf{x}) + \sum_{i=1}^{N}(\nabla\Phi_i^T(\mathbf{x})\mathbf{n}(\mathbf{x}))\mathbf{u}_i, \tag{9}$$

where the scalars $\nabla\Phi_i^T(\mathbf{x})\mathbf{n}(\mathbf{x})$ are constant and can be precomputed. Again, this amounts to adding to the undeformed normal a weighted sum of the displacement vectors $\mathbf{u}_i$, where the weights are the precomputed $\nabla\Phi_i^T(\mathbf{x})\mathbf{n}(\mathbf{x})$.

The deformed vertex position and normal can be efficiently computed on the GPU. We store for each vertex $\mathbf{x}$ the indices to the node neighbors and the accompanying scalars $\Phi_i(\mathbf{x})$ and $\nabla\Phi_i^T(\mathbf{x})\mathbf{n}(\mathbf{x})$ in GPU texture memory. During modeling, we only have to send the computed nodal displacement vectors $\mathbf{u}_i$ to the graphics board. Due to the coarse sampling with nodes, this data is typically several orders of magnitude smaller than the number of vertices. Using multiple render passes and fragment shaders, we compute and write the position and normal deformation for each vertex to intermediate texture memory. In a final render pass, we update the vertex position and normal in a vertex shader using two final texture lookups to retrieve this information.

A similar GPU implementation in the context of skeletal deformations based on ideas of [PBMH02] and [LHK*04] is detailed in [RLN06].

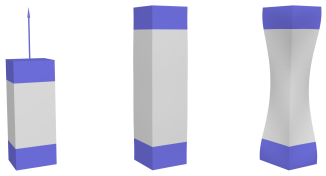

Figure 5: *Illustration of the effect of the different shape modeling constraints. Left: handle constraints are specified to fix the bottom of the box and to move the top to the desired position. Middle: with only the strain minimization constraint the total volume is increased by 53%. Right: the total volume remains within 3% of the original when the volume constraint is added.*

5. Shape Deformations

We use the deformation discretization introduced in the last section for interactive shape modeling. We define energy terms that penalize strain and changes in the shape's volume, while enforcing the user's input constraints. We then find the deformation by minimizing this energy. The complexity of the resulting optimization problem only depends on the number of nodes. Thus, the resulting deformation framework can be used to interactively model even complex shapes. It will also serve as the core component in the keyframe interpolation and motion planning framework that we will discuss in Sec. 6.

The goal for shape modeling is to find a continuous deformation field $\mathbf{f}(\mathbf{x}) = \mathbf{x}+\mathbf{u}(\mathbf{x})$ that maps the shape to its deformed pose given following constraints (see also Fig. 5).

Handle Constraints Handle constraints restrict the movement of certain points of the shape. For example, the user may want to fix the legs while pulling one of the arms of the model to deform its shape. Thus, a handle constraint simply states that the deformation field $\mathbf{f}(\mathbf{x}_k)$ should move a given point $\mathbf{x}_k$ to a prescribed target position $\mathbf{x}'_k$. Given a set of K handle constraints $(\mathbf{x}_k, \mathbf{x}'_k)$, we will minimize:

$$E_{\text{handle}} = \sum_{k=1}^{K}\|\mathbf{f}(\mathbf{x}_k) - \mathbf{x}'_k\|^2. \tag{10}$$

Minimal Strain Given a deformation field $\mathbf{f}$, the Green-Saint-Venant's non-linear strain tensor is defined as $\nabla\mathbf{f}^T(\mathbf{x})\nabla\mathbf{f}(\mathbf{x}) - \mathbf{I}$, where $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$. Hence, to obtain realistic shape deformations that have minimal strain, we will minimize:

$$E_{\text{strain}} = \int_{\mathbf{x}\in\mathcal{V}}\|\nabla\mathbf{f}^T(\mathbf{x})\nabla\mathbf{f}(\mathbf{x}) - \mathbf{I}\|_F^2 d\mathbf{x}, \tag{11}$$

where the integration is over the (undeformed) shape's volume $\mathcal{V}$ and $\|\cdot\|_F$ is the Frobenius norm. To facilitate optimization, we will only penalize strain at the nodal positions. This leads to the discretized equation

$$E_{\text{strain}} = \sum_{i=1}^{N} V_i\|\nabla\mathbf{f}^T(\mathbf{x}_i)\nabla\mathbf{f}(\mathbf{x}_i) - \mathbf{I}\|_F^2. \tag{12}$$

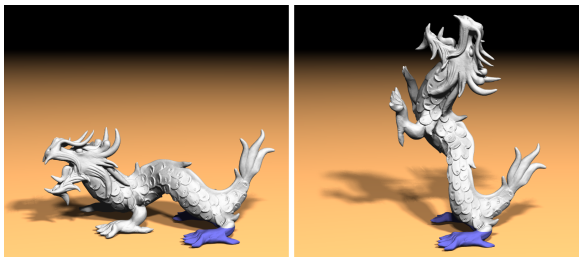

Figure 6: *Deformation of the dragon model obtained using a coarse set of only* 60 *nodes. The nodal deformations are computed on the CPU, while the high resolution surface is deformed faithfully on the GPU. The interaction was performed at a rate of* 55 *fps for the model with* 100*k vertices and* 10 *fps for the model with* 500*k vertices.*

Here, $\nabla \mathbf{f}(\mathbf{x}) = \mathbf{I} + \nabla \mathbf{u}(\mathbf{x})$, where $\nabla \mathbf{u}(\mathbf{x})$ is computed using the analytic derivative formula of Eq. 4. The scaling by the node volume $V_i = 4/3\pi r_i^3$ can be omitted when using uniform node radii ($r_i = r$). In the following we will write down the discretized equation (cf. Eq. 12) and leave out the continuous one (cf. Eq. 11) for the sake of brevity.

Volume Preservation The deformation field preserves the shape's volume if and only if $|\nabla \mathbf{f}(\mathbf{x})| = 1$ over the whole shape. Thus, the deformed shape's volume matches its original volume as closely as possible if we minimize

$$E_{\text{volume}} = \sum_{i=1}^{N} V_i (|\nabla \mathbf{f}(\mathbf{x}_i)| - 1)^2. \qquad (13)$$

The optimal deformation field $\mathbf{f}(\mathbf{x})$ can now be found by minimizing the total sum of constraint energies:

$$E_1 = \lambda_1 E_{\text{handle}} + \lambda_2 E_{\text{strain}} + \lambda_3 E_{\text{volume}}, \qquad (14)$$

where the parameters λ_1, λ_2 and λ_3 vary the contribution of each of the different constraints. It can be easily seen from Eq. 10, 12 and 13 that E_1 is a multivariate polynomial of total degree 6 in the unknowns (the fictitious nodal displacements $\mathbf{u}_i$). Minimizing Eq. 14 hence requires a non-linear solver. Note however that taking analytic derivatives with respect to the unknowns is straightforward, since $\mathbf{f}(\mathbf{x})$ is linear with respect to the unknown $\mathbf{u}_i$'s.

An example of a deformation of a high resolution dragon model is shown in Fig. 6.

6. Deformable Shape Motions

The meshless approximation discussed in Sec. 4.1 provides us with an efficient representation of the deformation of an object. In order to represent a time-varying deformation field of a deformable shape in motion, we sample the shape's deformation at discrete frames $j \in \{1, \ldots T\}$. At each frame, the deformation is described in terms of the nodal displacements using Eq. 1.

$$\mathbf{f}(\mathbf{x}, t_j) = \mathbf{x} + \mathbf{u}(\mathbf{x}, t_j) = \mathbf{x} + \sum_{i=1}^{N} \Phi_i(\mathbf{x}) \mathbf{u}_{i,t_j}, \qquad (15)$$

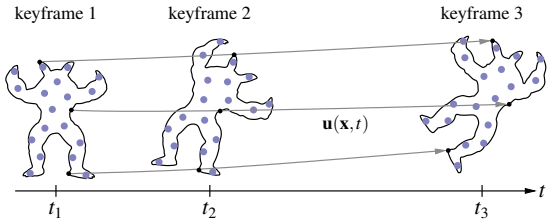

Figure 7: *The goal is to find a smooth motion of the deformable shape that interpolates the keyframes. We solve for one displacement vector $\mathbf{u}_{i,t_j}$ for each node i in each frame t_j.*

where $\mathbf{u}_{i,t_j}$ is the deformation of node i in frame j. See Fig. 7 for an illustration. The motion of the deformable object is completely defined by these nodal displacements $\mathbf{u}_{i,t_j}$, which are the unknowns we will solve for using an energy minimization approach.

6.1. Temporal Interpolation

Eq. 15 describes the deformation field at discrete frames j. To allow continuous evaluation, this deformation field has to be interpolated in time. We presented a temporal interpolation scheme based on meshless approximations similar to the one used in Sec. 4.1 in [AOW*08]. However, as illustrated in Fig. 8, the interpolation scheme from [AOW*08] has the inherent problem that it does not recover rigid body motions exactly: Given rigidly transformed keyframes, the temporal interpolation might not describe a rigid transformation. We improve on this method and propose an interpolation algorithm based on shape matching and quaternion interpolation. Contrary to [AOW*08], the new interpolation scheme recovers rigid motions exactly. Moreover, it is interpolating at the frames, greatly simplifying the motion optimization algorithm.

To interpolate the motion given by a set of frames, we factor out the rotational component of the deformation and interpolate it separately. For each frame, we compute the rotational part $\mathbf{R}_j = \mathbf{R}(t_j)$ of the deformation using shape matching against the undeformed shape as proposed in [Hor87]. The deformation can then be described as the sum of rotation and detail deformation

$$\mathbf{f}(\mathbf{x}, t_j) = \mathbf{R}(t_j)(\mathbf{x} - \mathbf{c}_0) + \mathbf{c}_\mathbf{f}(t_j) + \mathbf{d}(\mathbf{x}, t_j), \qquad (16)$$

where the average of node positions in the original shape, $\mathbf{c}_0$, is the center of the rotations $\mathbf{R}_j$, and $\mathbf{c}_\mathbf{f}(t) = 1/N \sum_i \mathbf{f}(\mathbf{x}_i, t)$ denotes the average of the deformed node positions at time t. After the per-frame rotations are obtained, we compute the detail deformation at the frames, $\mathbf{d}(\mathbf{x}, t_j)$, and interpolate both separately using spherical cubic Hermite splines and Catmull-Rom splines, respectively. The interpolated deformation $\mathbf{f}(\mathbf{x}, t)$ at an arbitrary time t can then be computed with Eq. 16 using the interpolated quantities.

The interpolation of the non-rotational parts of the deformation, $\mathbf{c}_\mathbf{f}(t_j)$ and $\mathbf{d}(\mathbf{x}, t_j)$, is performed using Catmull-Rom

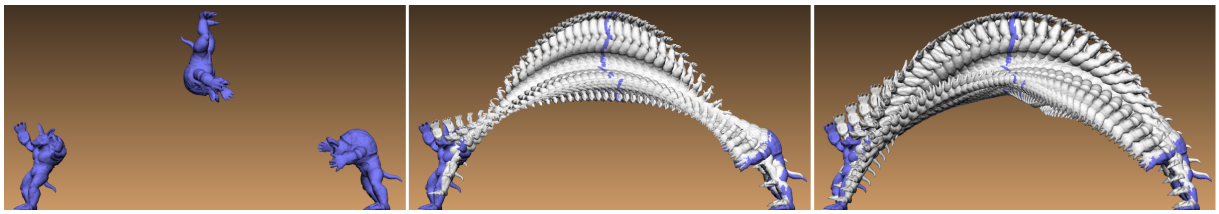

Figure 8: *Left: The user specifies 3 keyframes. Middle: Interpolated motion between the 3 keyframes using the method of [AOW*08]. The armadillo's shape is not well preserved. Right: Interpolation result using the method presented in this paper. The salto motion is nicely recovered even with only 3 frames.*

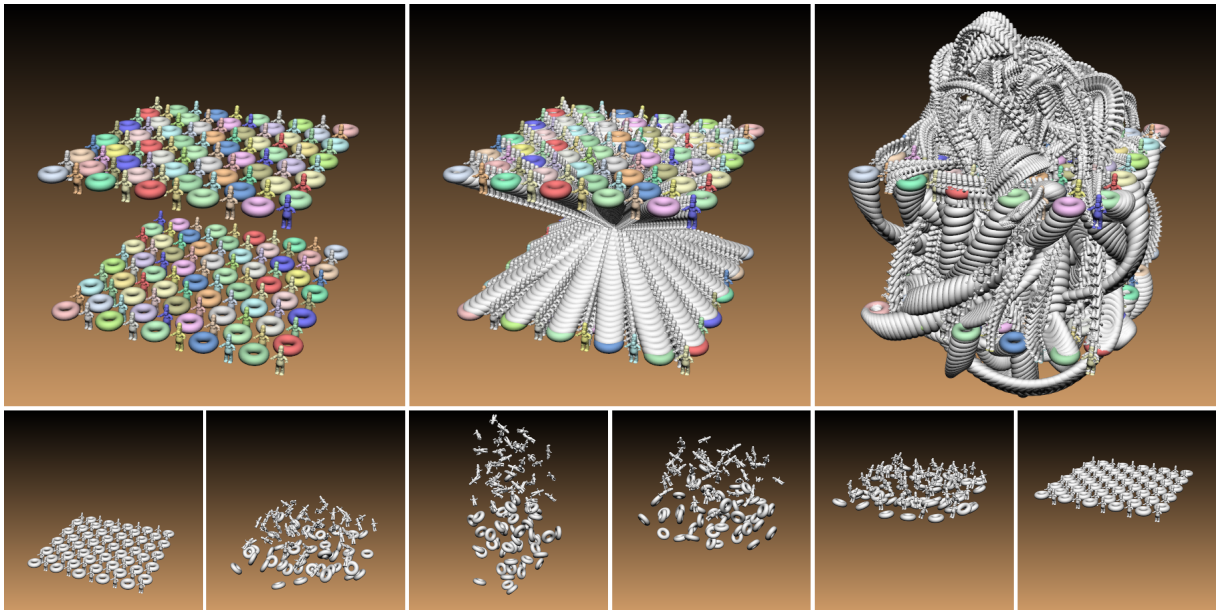

Figure 9: *Computing a collision-free motion for 100 deformable objects. Top left: User-specified begin and final keyframe poses. Top middle: Each object has to move up to the opposite side as illustrated by the linearly interpolated motion. Top right: Resulting collision-free trajectories computed by our algorithm. Bottom row: 6 frames of the computed motion.*

splines, providing C^1 continuity as well as continuous, analytic derivatives.

Interpolating the rotations is more involved: We represent the rotations as quaternions, and interpolate them using spherical cubic Hermite splines. These splines are C^1-continuous, interpolating, and lead to closed-form expressions [Sho85, Ebe09]. Of particular importance for our algorithm is the ability of taking analytical derivatives to evaluate angular velocities. All relevant equations are given in App. A.

To evaluate the energy function, we need to compute not only positions, but also velocities of each node. Using Eq. 34 in the appendix, we can compute the angular velocity $\omega(t)$ represented by the time-varying quaternion $\mathbf{R}(t)$. The interpolated velocity is then

$$\mathbf{v}(\mathbf{x},t) = \omega(t) \times (\mathbf{x}-\mathbf{c}) + \frac{\partial \mathbf{d}(\mathbf{x},t)}{\partial t}. \qquad (17)$$

It is easy to see that this interpolation strategy indeed recovers all rigid motions exactly. The temporal deformation field representation also has the nice property that space and time are decoupled. As a result, spatial shape functions are not affected when up-sampling the temporal resolution.

Fig. 8 compares our new interpolation algorithm with the one proposed in [AOW*08] where temporal interpolation is performed using the meshless approximation equations. As can be seen in the image, a much improved motion is obtained using the shape matching approach.

6.2. Deformable Motion Field Optimization

Once we define an energy function, we have all ingredients for finding time varying motion fields by optimization. Within a frame t_j, we use the strain and volume preserving penalties as defined before in Sec. 5. To solve for a temporally changing deformation field, we add temporal smooth-

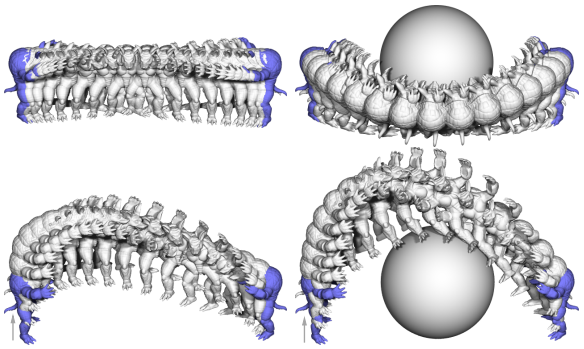

Figure 10: *Illustration of the effect of the different temporal constraints. Top left: smooth interpolation between two keyframes using the keyframe and acceleration constraints. Lower left: result after prescribing the velocity in the first frame (gray arrow). Top right: result after adding an obstacle. Bottom right: result after adding the velocity and obstacle constraints.*

ness constraints as well as collision avoidance and handling constraints. See Fig. 10 for an illustration.

Since a scene can contain multiple deformable shapes s, we will write $\mathbf{f}_s$ for the deformation field, N_s for the number of nodes and T_s for the number of frames of shape s. The total number of shapes is denoted by S.

Position Constraints At any frame, we can constrain the position of any node to an arbitrary point.

$$E_{\text{position}} = \sum_{s=1}^{S} \sum_{j=1}^{T_s} \sum_{k} \|\mathbf{f}_s(\mathbf{x}_k, t_j) - \mathbf{x}'_{k,t_j}\|^2, \qquad (18)$$

where $\mathbf{x}'_{k,t_j}$ is the desired position of node k in frame j (at time t_j). The index k runs over all nodes that are constrained at t_j.

We use position constraints to implement keyframes: A keyframe locks an object in a desired shape at a certain time. In the system, the user specifies keyframes by simply moving and deforming the shape using the deformation algorithm described in Sec. 5. We can also constrain individual nodes to allow for partial deformations.

Shape Preservation In each frame and for all shapes, we minimize the strain energy (12) to obtain realistic shape deformations

$$E_{\text{strain}} = \sum_{s=1}^{S} \sum_{j=1}^{T_s} \sum_{i=1}^{N_s} \|\nabla \mathbf{f}_s^T(\mathbf{x}_i, t_j) \nabla \mathbf{f}_s(\mathbf{x}_i, t_j) - \mathbf{I}\|_F^2. \qquad (19)$$

To avoid shape inversions and to preserve the original volume, we also minimize the volume preservation energy (13)

$$E_{\text{volume}} = \sum_{s=1}^{S} \sum_{j=1}^{T_s} \sum_{i=1}^{N_s} (|\nabla \mathbf{f}_s(\mathbf{x}_i, t_j)| - 1)^2. \qquad (20)$$

Temporal Smoothness To obtain smooth deformable motions, we minimize the following energy that penalizes accelerations

$$E_{\text{acceleration}} = \sum_{s=1}^{S} \sum_{j=1}^{T_s} \sum_{i=1}^{N_s} \left(\frac{\partial^2 \mathbf{f}_s(\mathbf{x}_i, t_j)}{\partial t^2} \right)^2. \qquad (21)$$

The second derivatives are approximated using finite differences. Since we use cubic Hermite interpolation to compute the node trajectories, the second derivatives are not well-defined at the frames, and cannot be evaluated analytically. We opted against using splines with higher-order continuity (for example natural cubic splines) since those would give each frame global influence, which greatly degrades optimization performance, while adding no visible improvement.

Whenever a discontinuity in the motion is desired, this energy is disabled.

Velocity Constraint Velocity constraints simply constrain the velocity of a node k at frame j to a fixed value $\mathbf{v}_{k,t_j}$:

$$E_{\text{velocity}} = \sum_{s=1}^{S} \sum_{j=1}^{T_s} \sum_{k} \|\mathbf{v}_s(\mathbf{x}_k, t_j) - \mathbf{v}_{k,t_j}\|^2, \qquad (22)$$

where k again runs over all constrained nodes and the velocity of a node is given by Eq. 17. Velocity constraints are used for example to obtain smooth transitions from a physically simulated animation to a user-designed motion path.

Collision Avoidance We add collision avoidance energies to adequately resolve motion paths for different kinematically scripted and deformable objects.

Assuming that scene obstacles can be represented by a time dependent signed distance field $d(\mathbf{x},t)$ and that a point $\mathbf{x}$ is penetrating at time t if $d(\mathbf{x},t) \geq 0$, we obtain the following energy function that prevents nodes to penetrate obstacles

$$E_{\text{obstacles}} = \sum_{s=1}^{S} \sum_{j=1}^{T_s} \sum_{i=1}^{N_s} \max[0, d(\mathbf{f}_s(\mathbf{x}_i, t_j), t_j)]^2. \qquad (23)$$

Fig. 11 illustrates this collision avoidance energy for a moving obstacle.

The above approach can be extended to avoid collisions between deformable shapes. Assume a signed distance field $d_s(\mathbf{x},t)$ is given for each shape s in its *undeformed* state, we can then avoid shape-shape collisions by minimizing

$$E_{\text{deformables}} = \sum_{\substack{s,t=1 \\ s \neq t}}^{S} \sum_{j=1}^{T_t} \sum_{i=1}^{N_t} \max[0, d_s(\mathbf{f}_s^{-1}(\mathbf{f}_t(\mathbf{x}_i, t_j), t_j), t_j)]^2. \qquad (24)$$

Evaluating $\mathbf{f}_s^{-1}$ at arbitrary positions requires computation of shape functions, which can be expensive during optimization. To avoid unnecessary computations we use a bounding sphere hierarchy that is constructed from the deformed node positions. Eq. 24 is only evaluated when a sphere-sphere

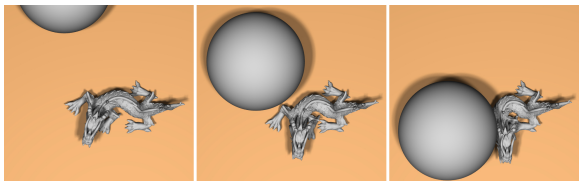

Figure 11: *The dragon deforms to avoid the moving obstacle. Note how the dragon anticipates the impact of the ball thanks to the acceleration constraint.*

collision between two shapes is reported. Since each shape is represented by only a small number of nodes, updating the sphere hierarchies can be performed efficiently. Fig. 9 illustrates the deformable collision avoidance constraint for a scene with 100 deformable objects.

Given a discrete spatial and temporal sampling for possibly multiple deformable objects, the optimal temporal shape deformations $\mathbf{f}_s(\mathbf{x},t)$ are found by minimizing a weighted sum of the different energies, similar to Eq. 14:

$$\begin{aligned} E_2 = {} & \lambda_1 E_{\text{position}} + \lambda_2 E_{\text{strain}} + \lambda_3 E_{\text{volume}} \\ & + \lambda_4 E_{\text{acceleration}} + \lambda_5 E_{\text{velocity}} \\ & + \lambda_6 E_{\text{obstacles}} + \lambda_7 E_{\text{deformables}}. \end{aligned} \tag{25}$$

Again, E_2 is a polynomial of degree 6 in the unknowns, the nodal deformations of all nodes of all shapes in all frames. Since the temporal interpolation algorithm is interpolating and since finite differences in time are used to approximate accelerations, optimizing E_2 proceeds very similar to optimizing E_1.

Note however that ensuring a collision-free state for example at the discrete frames t_j, does not guarantee this property at every time instance, since the interpolation algorithm is collision oblivious. Similar considerations hold for the shape preservation energies. In the following we discuss the adaptive temporal sampling strategy that we introduced in [AOW*08] that resolves these issues.

6.3. Adaptive Temporal Sampling

In the (single object and single frame) deformation modeling part of Sec. 5, the total number of unknowns to solve for is $3N$, where N is the number of nodes. In the motion planning setting, the total number of unknowns increases to $\sum_{s=1}^{S} 3N_sT_s$, where S is the number of shapes, N_s the number of nodes and T_s the number of frames used for shape s. To keep this number sufficiently low, we use an adaptive time sampling strategy that introduces frames iteratively in problematic regions (see Fig. 12 for an illustration with one deformable shape).

Initially, we only have frames t_j that correspond to the keyframes specified by the user (see top left image in Fig. 12). We optimize the displacement field as discussed above and evaluate the error for each shape at a dense number of frames in between the frames t_j (we typically evaluate at 10 intermediate frames). We then introduce for each shape a new frame t_j at the time instance t_{max} where the error is maximal. We solve again and iterate until a desired accuracy is obtained. Note that in each step when we introduce a new frame at time t_{max}, we initialize the nodes' deformation vectors at the new frame using the temporal interpolation algorithm of Sec. 6.1. This yields a good initial guess for the subsequent solve.

The proposed adaptive sampling strategy greatly reduces the number of unknowns and introduces frames only at problematic regions, for example when there is high acceleration, or when the deforming shape is penetrating an obstacle (see for example Fig. 12).

6.4. Local Minima

Since we formulate the problem of finding the optimal shape motions as a non-linear, non-convex optimization problem, it is quite possible that the solution we find is a local minimum of the energy function, rather than a global one. In addition, in some situations it is not possible to distinguish between a set of equally plausible trajectories using the objective function alone. For instance, if an obstacle is located exactly between two poses of a deforming object, the solutions that avoid the obstacle on either of its sides are equally likely.

The tight coupling of our deformation module with the motion optimization facilitates the resolution of these scenarios. In particular, we solve the problem of local minima by giving the user control over the initial state of the optimization. The user can specify the intermediate positions of the deformable objects without specifying keyframe correspondence constraints. These intermediate positions are given to the optimization procedure as an initial value for the minimizer of the objective function. This moves the optimization out of one local minimum, and into another, which is closest to the manually specified initial value. In simple scenarios, such as the one described above, this procedure allows to disambiguate between a set of, possibly local, minima. This is particularly useful during adaptive temporal sampling, because it allows the user to control the general direction of the motion without specifying handle constraints.

7. Adding Dynamics

The shape and motion optimization algorithms presented in this paper use the same meshless deformation field representation as used before in the graphics community for physically based simulation of elastic materials [MKN*04]. Below we show how such physical simulations can be used in a post-process to add secondary effects such as jiggling to a designed deformable shape motion and how designed and purely physically simulated motion can be combined to offer additional flexibility in designing realistic animations.

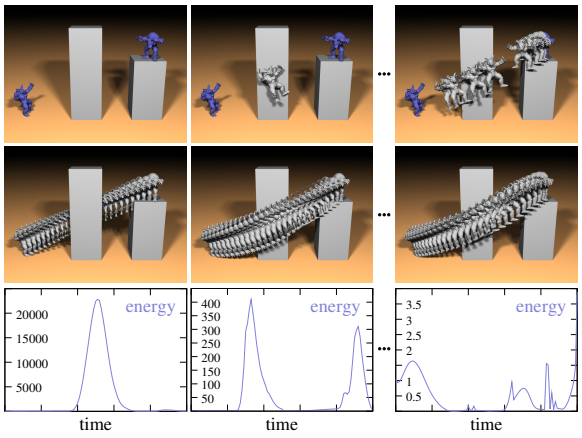

Figure 12: *The user wants to find a smooth motion between two deformed shapes of the armadillo, that are specified as keyframes in the top left image. The linearly interpolated motion between these two frames is inadequate and has high energy where the armadillo moves through one of the obstacles (left column). Our algorithm adds a new frame and solves for the armadillo's deformed shape at this time step. The resulting motion largely avoids the obstacles (middle column). This process is iterated until a sufficiently low energy is obtained for 10 frames (right column).*

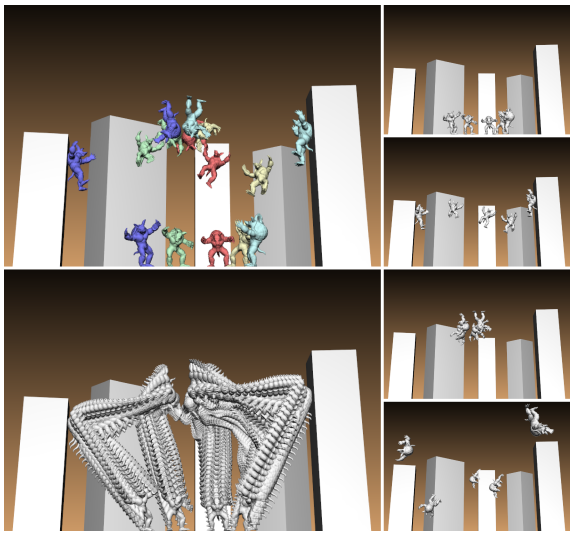

Figure 13: *Illustration of the effect of secondary dynamics and the combination of designed and physically simulated motion. Top left: The user specifies 3 keyframes for each of the 5 armadillos. Lower left: Resulting motion computed using the algorithm of Sec. 6.2. Right: 4 frames of the resulting motion including secondary dynamics and continued physical simulation.*

7.1. Secondary Dynamics

Given an object's computed temporal deformation field, we can add secondary motion effects by running a meshless elastic solid simulation. This simulation is run as a post-process, and control forces make objects follow the precomputed trajectories.

Assume the temporal shape deformation field computed in Sec. 6.2 is given by $\mathbf{g}(\mathbf{x},t)$, the goal is to compute a modified deformation field $\mathbf{f}(\mathbf{x},t)$ that closely follows $\mathbf{g}$, but includes secondary motions. Starting with the deformed shape $\mathbf{f}(\mathbf{x},0)=\mathbf{g}(\mathbf{x},0)$ we advance $\mathbf{f}$ forward in time using a combination of shape matching and explicit force-based time integration.

Shape Matching In a first step we rigidly move $\mathbf{f}$ as close as possible to the goal shape $\mathbf{g}$ by rigidly matching the nodes $\mathbf{f}(\mathbf{x}_i,t)$ to the target node positions $\mathbf{g}(\mathbf{x}_i,t)$ [Hor87]. The best match for node i is given by

$$\begin{aligned}\tilde{\mathbf{f}}(\mathbf{x}_i,t) &= \mathbf{R}(t)(\mathbf{f}(\mathbf{x}_i,t)-\mathbf{c}_\mathbf{f}(t))+\mathbf{c}_\mathbf{g}(t)\\ &= \mathbf{g}(\mathbf{x}_i,t)-\mathbf{d}(\mathbf{x}_i,t),\end{aligned} \tag{26}$$

where $\mathbf{c}_\mathbf{f}(t)$ and $\mathbf{c}_\mathbf{g}(t)$ are the average node positions of $\mathbf{f}$ and $\mathbf{g}$, respectively, and $\mathbf{R}(t)$ and $\mathbf{d}(\mathbf{x},t)$ are defined as in Sec. 6.1. Before computing any forces, we kinematically move the nodes to their best rigid match $\tilde{\mathbf{f}}$.

Force Computation We compute nodal control forces that attract the nodes to their precomputed trajectory similar to [MHTG05, SZT*08], using $\mathbf{g}$ as the target positions. Given a time-step Δt, the control forces are

$$\mathbf{F}_i^{\text{target}} = \frac{\mathbf{d}(\mathbf{x}_i,t)}{\Delta t^2}. \tag{27}$$

Besides these target forces, we add shape preservation forces that try to restore the deformed shape to its rest shape. These forces will give rise to secondary motions such as wiggling. Following [MKN*04], the elastic force is given by

$$\mathbf{F}_i^{\text{elastic}} = -\sigma\nabla_{\tilde{\mathbf{u}}_i}\varepsilon, \tag{28}$$

with strain $\varepsilon=\nabla\tilde{\mathbf{f}}^T(\mathbf{x}_i,t)\nabla\tilde{\mathbf{f}}(\mathbf{x}_i,t)-\mathbf{I}$ (cf. Eq. 12) and stress $\sigma=\mathbf{C}\varepsilon$ where $\mathbf{C}$ a constitutive matrix that defines the stress-strain relationship. We also add volume conservation forces to avoid undesirable shape inversions (cf. Eq. 13)

$$\mathbf{F}_i^{\text{volume}} = -\frac{k_v}{2}\nabla_{\tilde{\mathbf{u}}_i}(|\nabla\tilde{\mathbf{f}}(\mathbf{x}_i,t)|-1)^2, \tag{29}$$

where k_v is user-defined constant.

With external forces such as gravity and penalty-based collision response forces, we obtain the total force $\mathbf{F}_i=\mathbf{F}_i^{\text{target}}+\mathbf{F}_i^{\text{elastic}}+\mathbf{F}_i^{\text{volume}}+\mathbf{F}_i^{\text{external}}$ which is used in an explicit Euler step to compute the final node positions.

7.2. Combining Designed and Simulated Motion

If we omit the control forces, the above formulation is a regular physical simulation similar to [MKN*04]. We can therefore seamlessly extend a computed motion trajectory continuing to run the physics simulation after the last keyframe. The added secondary effects ensure that the designed and simulated motion parts transition smoothly. An example of

vertices \ nodes	20	50	100	200
100k	4.6/4.4	9.5/4.6	15.2/4.7	31.6/5.5
250k	4.9/51.5	9.6/52.2	14.5/51.8	33.0/55.2
500k	5.1/95.9	9.4/94.8	15.6/97.6	32.5/102.9

Table 1: *Timing statistics (ms) for the dragon deformation of Fig. 6 for different numbers of vertices and nodes. Each entry shows the average time spent solving on the CPU and on deforming and rendering the triangle mesh on the GPU.*

combined designed and simulated motion for multiple deformable objects is shown in Fig. 13.

Reversely, one can also design a motion path to follow the result of a purely physically based simulation. This can be done by taking the last frame of the simulation and using it to define the boundary conditions of the first keyframe for the designed part. The node positions and velocities can be set using the constraints of Eq. 18 and Eq. 22 respectively to ensure a smooth transition.

8. Results & Discussion

We implemented our algorithms in C++ and used Cg for the fragment and vertex shaders that compute the deformed mesh vertices. Our models are given as triangle meshes. In the preprocessing step we compute a regular distance field and use the same grid for the nodal sampling and the fast marching. Inversion of the moment matrices is performed using the Cholesky decomposition code of the JAMA/C++ linear algebra package. The energy minimization problems are solved using the non-linear LBFGS solver from OPT++.

Fig. 6 shows the result of a real-time deformation of the dragon model. We obtain an interaction rate of 10 fps for the model of 500k vertices and at least 55 fps for the decimated model of 100k vertices, both sampled with 60 nodes. Detailed timings for varying numbers of vertices and nodes are given in Table 1. This result was obtained on a 3.2 GHz Intel Pentium D CPU with an NVIDIA GeForce 8800 graphics board.

Fig. 1 shows the designed motion from 7 keyframe poses of the armadillo represented by a triangle mesh of 166k vertices and sampled with 66 nodes. To obtain realistic bouncing behavior, we allow C^1 discontinuities in the resulting motion path. This is achieved by omitting the acceleration constraint at the respective frames. Fig. 11 shows the resulting motion of a deforming dragon in the presence of a moving obstacle. The initial and final keyframe are set to the undeformed dragon and its back feet are fixed over the whole time interval. The intermediate motion is computed automatically. The dragon is sampled with 59 nodes. Fig. 9 illustrates the scalability of our approach. 100 objects in a grid exchange positions. Our algorithm is able to resolve all collisions, and computes smooth, energy minimal motion paths for all objects. Each object is sampled with 20 nodes.

Fig. 13 shows how we can combine a designed keyframe animation with a purely physical simulation when no keyframes

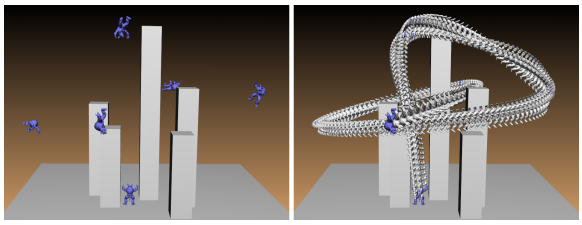

Figure 15: *Interpolating a smooth motion from 6 keyframes.*

are given. Added secondary dynamics effects make the designed motions appear more dynamic, and render the transition from designed to simulated motion smooth. The armadillos are each sampled with 46 nodes. We can also switch back to designing motions if the physical simulation does not arrive at a satisfactory solution. Fig. 14 shows an example in which a motion path is designed using keyframes, part of the motion is computed using a physical simulation, before switching back to keyframe animation to achieve a particular outcome. The girl character is sampled with 31 nodes.

Table 2 shows timings for the motion examples, all of which were measured on an Intel 2.53GHz CPU.

We used the setting of Fig. 15 to compare our improved interpolation algorithm with the one of [AOW*08]. In this example a smooth motion is computed interpolating 6 keyframes. Using the algorithm presented in this paper, the solver converges after introducing 9 more frames, resulting in 15 frames and 2070 unknowns to solve for (the armadillo is sampled with 46 nodes). Using the method of [AOW*08], the algorithm converges with the same energy only after inserting 46 frames, making the number of unknowns 7176. The solve time was 8 seconds and 20 seconds respectively.

8.1. Limitations

As discussed earlier, one of the principal limitations of our motion design system is that the energies that we formulate may have several global and local minima. Our method of manually controlling the initial guess for the deformation at any intermediate frame allows to partly overcome this problem. However, finding the global minimum of a non-convex, non-linear function is a hard problem, and there are certainly pathological cases where manual control will not lead to the desired result.

scene	# variables	solve time
Fig. 1	2772	7.2 seconds
Fig. 11	2478	6.7 seconds
Fig. 9	102000	18.6 minutes
Fig. 13	6210	6.1 seconds
Fig. 14	1581	2.3 seconds
Fig. 15	2070	8.0 seconds

Table 2: *Statistics for the examples shown in this paper.*

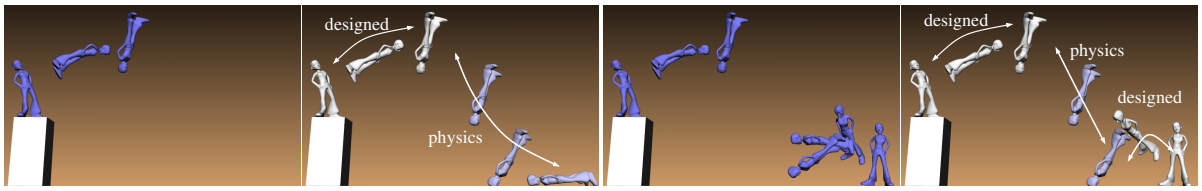

Figure 14: *Illustration of the combination of designed motion and motion generated from a physics simulation. Left two images: The user specifies 3 keyframes and lets the physics take over after the last keyframe (these frames are colored in light blue on the right). Right two images: The user wants the character to land on its feet instead of falling flat on the ground plane. Our system allows stopping the physics simulation at a certain time and inserting new keyframes that describe the desired motion. This sequence consists of 3 different parts: designed, simulated and finally designed again. We also add secondary dynamics in this example to obtain dynamic behavior and a smooth transition between the different motion parts.*

In this paper we opted for using a low-dimensional deformation model, while allowing for high resolution boundary surfaces. Since the optimization algorithm only depends on the number of nodes and not on the number of mesh vertices, this enables motion design for high quality surfaces at reasonable rates. However, the choice of the number and placement of nodes limits the possible deformations. An adaptive nodal sampling would allow the animator to introduce more nodes where more degrees of freedom are required or to dynamically adapt the nodal sampling during optimization if more flexibility is desired.

The added secondary motion mostly consists of high-frequency vibrations around the shape's rest pose. Low frequency modes are typically not present since the designed animation is optimized to be energy optimal and shape matching is performed on all nodes to align the simulated shape before computing elastic forces. An approach similar to TRACKS [BMWG07] could resolve this issue by performing the shape matching step on a coarser set of nodes, allowing more freedom in the original nodes. The added secondary effects also result in a motion that not necessarily aligns with the keyframes. Small deviations from the specified keyframes are possible and hard to control.

Combining physically simulated and scripted motion can still be tricky and requires some training by the animator, particularly when the scripted motion follows the simulated one. When scripting the end of an animation, the first keyframe of the designed sequence has to exactly correspond to the final frame of the simulated sequence. This prohibits changing the first part of the animation after the final part is designed. Also, manual adjusting of the timing and frame rate of the different motion parts is necessary when combining different motions. In particular, discontinuities in the velocities and accelerations have to be avoided. Smooth velocities can be enforced using the velocity constraints described in Sec. 6.2. However, even discontinuities in accelerations can lead to visual artifacts. Currently, only careful modeling can ensure no such discontinuities are visible when switching between designed to simulated motion.

Finally, our current energy formulation does not contain a term for preservation of angular momentum, which would increase the overall realism of the motion. Formulating such a term should be relatively straightforward, given the formula for angular velocity in the Appendix. We leave this for future work.

9. Conclusion

We have presented a shape modeling and motion design framework that allows users to quickly and easily generate new poses and motion paths for many objects. A novel nodal and an adaptive temporal sampling algorithm ensure adequate sampling while keeping the number of unknowns low. At all times during the shape and motion modeling process the objects are guaranteed to preserve their shape thanks to the physically inspired modeling constraints. Collisions between the objects and the environment, as well as among the objects, are avoided. We have also described a method for adding secondary physical deformations to the generated motions, improving the dynamic behavior, and creating the possibility of combining designed and simulated motion.

Acknowledgments The authors wish to acknowledge the support of NSF grants ITR 0205671 and FRG 0354543, NIH grant GM-072970, DARPA grant HR0011-05-1-0007, and the Max-Planck Center for Visual Computing and Communication. Bart Adams is funded as a post-doctoral researcher by the Fund for Scientific Research, Flanders (F.W.O.-Vlaanderen). We would like to thank the Stanford University Computer Graphics Laboratory for providing the armadillo model and XYZ RGB Inc. for providing the dragon model.

References

[AFTCO07] AU O. K.-C., FU H., TAI C.-L., COHEN-OR D.: Handle-aware isolines for scalable shape editing. *ACM Trans. Graph. 26*, 3 (2007).

[AOW*08] ADAMS B., OVSJANIKOV M., WAND M., SEIDEL H.-P., GUIBAS L. J.: Meshless modeling of deformable shapes and their motion. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008).

[Bar84] BARR A. H.: Global and local deformations of solid primitives. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 21–30.

[BB88] BARZEL R., BARR A. H.: A modeling system based on dynamic constraints. *Computer Graphics (Proceedings of ACM SIGGRAPH 88) 22*, 4 (1988).

[BK03] BOTSCH M., KOBBELT L.: Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum 22*, 3 (2003).

[BMWG07] BERGOU M., MATHUR S., WARDETZKY M., GRINSPUN E.: Tracks: toward directable thin shells. *ACM Trans. Graph. 26*, 3 (2007), 50.

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: Primo: coupled prisms for intuitive surface modeling. In *Proceedings of the fourth Eurographics symposium on geometry processing* (2006), Eurographics Association.

[BPWG07] BOTSCH M., PAULY M., WICKE M., GROSS M.: Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3 (2007).

[BS08] BOTSCH M., SORKINE O.: On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (2008).

[BSS07] BOUBEKEUR T., SORKINE O., SCHLICK C.: Simod: Making freeform deformation size-insensitive. In *IEEE/Eurographics Symposium on Point-Based Graphics 2007* (2007).

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21*, 3 (2002), 586–593.

[DSP06] DER K. G., SUMNER R. W., POPOVIĆ J.: Inverse kinematics for reduced deformable models. *ACM Trans. Graph. 25*, 3 (2006).

[Ebe09] EBERLY D.: Quaternion algebra and calculus. http://www.cs.brown.edu/courses/cs224/papers/eberly99.pdf, 1999, retrieved September 2009.

[FKR05] FLOATER M. S., KÓS G., REIMERS M.: Mean value coordinates in 3d. *Comput. Aided Geom. Des. 22*, 7 (2005).

[FM03] FRIES T.-P., MATTHIES H. G.: *Classification and Overview of Meshfree Methods*. Tech. rep., TU Brunswick, Germany Nr. 2003-03, 2003.

[GP07] GROSS M., PFISTER H.: *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., 2007.

[GQ05] GUO X., QIN H.: Real-time meshless deformation: Collision detection and deformable objects. *Comput. Animat. Virtual Worlds 16*, 3-4 (2005).

[HJ07] HONGJUN JEON A. M.-H. C.: Interactive motion control of deformable objects using localized optimal control. In *International Conference on Robotics and Automation* (2007).

[Hof04] HOFER M.: *Variational motion design in the presence of obstacles*. PhD thesis, Vienna University of Technology, 2004.

[Hor87] HORN B. K. P.: Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A 4*, 4 (1987), 629–642.

[HPR04] HOFER M., POTTMANN H., RAVANI B.: From curve design algorithms to the design of rigid body motions. *Vis. Comput. 20*, 5 (2004).

[HSL*06] HUANG J., SHI X., LIU X., ZHOU K., WEI L.-Y., TENG S.-H., BAO H., GUO B., SHUM H.-Y.: Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3 (2006).

[IC87] ISAACS P. M., COHEN M. F.: Controlling dynamic simulation with kinematic constraints. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM, pp. 215–224.

[JMD*07] JOSHI P., MEYER M., DEROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3 (2007).

[JP02] JAMES D. L., PAI D. K.: Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. Graph. 21*, 3 (2002).

[JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Trans. Graph. 24*, 3 (2005).

[KA08] KASS M., ANDERSON J.: Animating oscillatory motion with overlap: wiggly splines. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (New York, NY, USA, 2008), ACM, pp. 1–8.

[KKiA05] KONDO R., KANAI T., ICHI ANJYO K.: Directable animation of elastic objects. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on computer animation* (2005), ACM.

[KMP07] KILIAN M., MITRA N. J., POTTMANN H.: Geometric modeling in shape space. *ACM Trans. Graph. 26*, 3 (2007).

[LaV06] LAVALLE S. M.: *Planning Algorithms*. Cambridge University Press, 2006.

[LHK*04] LUEBKE D., HARRIS M., KRÜGER J., PURCELL T., GOVINDARAJU N., BUCK I., WOOLLEY C., LEFOHN A.: Gpgpu: general purpose computation on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes* (New York, NY, USA, 2004), ACM, p. 33.

[LKCOL07] LIPMAN Y., KOPF J., COHEN-OR D., LEVIN D.: Gpu-assisted positive mean value coordinates for mesh deformations. In *Proceedings of the fifth Eurographics symposium on geometry processing* (2007), Eurographics Association.

[LSCO*04] LIPMAN Y., SORKINE O., COHEN-OR D., LEVIN D., RÖSSL C., SEIDEL H.-P.: Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International* (2004), IEEE Computer Society Press.

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deforma-

tions based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478.

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. *Proceedings of 2004 ACM SIGGRAPH Symposium on Computer Animation* (2004).

[NMK*05] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically Based Deformable Models in Computer Graphics. In *Eurographics: State of the Art Report* (2005).

[OFTB96] ORGAN D., FLEMING M., TERRY T., BELYTSCHKO T.: Continuous meshless approximations for nonconvex bodies by diffraction and transparency. *Comput. Mechanics 18* (1996).

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3 (July 2002), 703–712. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. *ACM Trans. Graph. 24*, 3 (2005).

[PSE*00] POPOVIĆ J., SEITZ S. M., ERDMANN M., POPOVIĆ Z., WITKIN A.: Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH 2000* (2000), ACM Press/Addison-Wesley Publishing Co.

[PSE03] POPOVIĆ J., SEITZ S. M., ERDMANN M.: Motion sketching for control of rigid-body simulations. *ACM Trans. Graph. 22*, 4 (2003).

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 335–342.

[RLN06] RHEE T., LEWIS J. P., NEUMANN U.: Real-time weighted pose-space deformation on the gpu. *Comput. Graph. Forum 25*, 3 (2006), 439–448.

[Set99] SETHIAN J. A.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[Sho85] SHOEMAKE K.: Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 245–254.

[SK04] SHEFFER A., KRAEVOY V.: Pyramid coordinates for morphing and deformation. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium* (2004), IEEE Computer Society.

[SLCO*04] SORKINE O., LIPMAN Y., COHEN-OR D., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2004), Eurographics Association.

[SOG06] STEINEMANN D., OTADUY M. A., GROSS M.: Fast arbitrary splitting of deforming objects. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Sept. 2006), pp. 63–72.

[SSP07] SUMNER R. W., SCHMID J., PAULY M.: Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (2007).

[Sto07] STOLL C.: *A Volumetric Approach to Interactive Shape Editing*. Research Report MPI-I-2007-4-004, Max-Planck-Institut für Informatik, June 2007.

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3 (2006).

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3 (2005).

[SZT*07] SHI X., ZHOU K., TONG Y., DESBRUN M., BAO H., GUO B.: Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM Trans. Graph. 26*, 3 (2007).

[SZT*08] SHI X., ZHOU K., TONG Y., DESBRUN M., BAO H., GUO B.: Example-based dynamic skinning in real time. *ACM Trans. Graph. 27*, 3 (2008), 1–8.

[TJ07] TWIGG C. D., JAMES D. L.: Many-worlds browsing for control of multibody dynamics. *ACM Trans. Graph. 26*, 3 (2007).

[TJ08] TWIGG C. D., JAMES D. L.: Backward steps in rigid body simulation. *ACM Trans. Graph. 27*, 3 (2008), 1–10.

[vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. *ACM Trans. Graph. 25*, 3 (2006).

[WK88] WITKIN A., KASS M.: Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), ACM, pp. 159–168.

[WMT06] WOJTAN C., MUCHA P. J., TURK G.: Keyframe control of complex particle systems using the adjoint method. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 15–23.

[WP95] WITKIN A., POPOVIC Z.: Motion warping. In *Proceedings of ACM SIGGRAPH 95* (1995), ACM.

[XZY*07] XU W., ZHOU K., YU Y., TAN Q., PENG Q., GUO B.: Gradient domain editing of deforming mesh sequences. *ACM Trans. Graph. 26*, 3 (2007).

Appendix A: Quaternion Splines

A good derivation of this material can be found in [Ebe09]. For reference, we only include the equations necessary for implementation.

Given a set of n frame rotations $\mathbf{R}_1 \dots \mathbf{R}_j \dots \mathbf{R}_n$, we want to compute an interpolated rotation $\mathbf{R}(t)$. Let $t_j < t < t_{j+1}$, such that $\mathbf{R}(t)$ lies in the spline segment connecting $\mathbf{R}_j = \mathbf{R}(t_j)$ and $\mathbf{R}_{j+1} = \mathbf{R}(t_{j+1})$.

If two rotations are given as unit quaternions $\mathbf{p}$ and $\mathbf{q}$, a spherical linear interpolation between them can be expressed as

$$S^0(\mathbf{p},\mathbf{q},\alpha) = \mathbf{p}(\mathbf{p}^{-1}\mathbf{q})^{\alpha}, \tag{30}$$

where $\alpha \in [0,1]$ is the blending parameter. Similar to the de Casteljau algorithm, we can construct a higher-order curve by concatenating several spherical linear interpolations, yielding

$$S(\mathbf{p},\mathbf{a},\mathbf{b},\mathbf{q},\alpha) = S^0(S^0(\mathbf{a},\mathbf{b},\alpha),S^0(\mathbf{p},\mathbf{q},\alpha),2\alpha(1-\alpha)). \tag{31}$$

It can be shown that the derivative of Eq. 31 is given by

$$\begin{aligned} S'(\mathbf{p},\mathbf{a},\mathbf{b},\mathbf{q},\alpha) &= \mathbf{U}\left[(2-4\alpha)\mathbf{W}^{\beta}\log(\mathbf{W}) + \beta\mathbf{W}^{\beta-1}\mathbf{W}'\right] \\ &+ \mathbf{U}'\left[\mathbf{W}^{\beta}\right], \end{aligned} \tag{32}$$

where $\beta = 2\alpha(1-\alpha)$, $\mathbf{U} = S^0(\mathbf{p},\mathbf{q},\alpha)$, $\mathbf{V} = S^0(\mathbf{a},\mathbf{b},\alpha)$, and $\mathbf{W} = \mathbf{U}^{-1}\mathbf{V}$. Then $\mathbf{U}' = \mathbf{U}\log(\mathbf{p}^{-1}\mathbf{q})$, $\mathbf{V}' = \mathbf{V}\log(\mathbf{a}^{-1}\mathbf{b})$, and $\mathbf{W}' = \mathbf{U}^{-1}\mathbf{V}' + \mathbf{U}^{-2}\mathbf{U}'\mathbf{V}$.

Turning back to the interpolation problem outlined above, we will now connect the rotations $\mathbf{R}_j$ and $\mathbf{R}_{j+1}$ using a spline segment $\mathbf{S}_j(t) = S(\mathbf{R}_j,\mathbf{a}_j,\mathbf{b}_j,\mathbf{R}_{j+1},\frac{t-t_j}{t_{j+1}-t_j})$. In order to ensure C^1-continuity, we have to choose $\mathbf{a}_j$ and $\mathbf{b}_j$ such that the tangents at the frames match: $S'_j(t_{j+1}) = S'_{j+1}(t_{j+1})$. While there are several ways of achieving this, using central differences works well and yields $\mathbf{a}_j = \mathbf{b}_{j-1}$, where

$$\mathbf{a}_j = \mathbf{R}_j \exp\left(-\frac{\log(\mathbf{R}_j^{-1}\mathbf{R}_{j+1}) + \log(\mathbf{R}_j^{-1}\mathbf{R}_{j-1})}{4}\right). \tag{33}$$

Thus, $\mathbf{R}(t) = S(\mathbf{R}_j,\mathbf{a}_j,\mathbf{a}_{j+1},\mathbf{R}_{j+1},\frac{t-t_j}{t_{j+1}-t_j})$ for $t_j < t < t_{j+1}$, and the spline segments connect in a $\mathcal{C}^1$-continuous manner.

Angular Velocity

Given a time-dependent rotation represented by a quaternion $\mathbf{Q}(t)$, we can compute the angular velocity represented by this rotation as

$$\omega = 2\frac{\partial \mathbf{Q}(t)}{\partial t}\mathbf{Q}^{-1}(t). \tag{34}$$

Note that ω is a purely virtual quaternion, which can be trivially converted to a 3-vector.