To appear in Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2011)

Robust Real-Time Deformation of Incompressible Surface Meshes

R. Diziol[1] J. Bender[2] D. Bayer[1]

[1]Karlsruhe Insitute of Technology, Germany
[2]Graduate School CE, TU Darmstadt, Germany

Abstract

We introduce an efficient technique for robustly simulating incompressible objects with thousands of elements in real-time. Instead of considering a tetrahedral model, commonly used to simulate volumetric bodies, we simply use their surfaces. Not requiring hundreds or even thousands of elements in the interior of the object enables us to simulate more elements on the surface, resulting in high quality deformations at low computation costs. The elasticity of the objects is robustly simulated with a geometrically motivated shape matching approach which is extended by a fast summation technique for arbitrary triangle meshes suitable for an efficient parallel computation on the GPU. Moreover, we present an oscillation-free and collision-aware volume constraint, purely based on the surface of the incompressible body. The novel heuristic we propose in our approach enables us to conserve the volume, both globally and locally. Our volume constraint is not limited to the shape matching method and can be used with any method simulating the elasticity of an object. We present several examples which demonstrate high quality volume conserving deformations and compare the run-times of our CPU implementation, as well as our GPU implementation with similar methods.

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

1. Introduction

Interactive simulation of deformable objects is an important topic in computer graphics. Additionally, the preservation of volume is essential when simulating deformable models. Not only does it play an important role when simulating biological tissues, such as muscles [HJCW06], but it also provides more realistic deformations and is therefore used in shape modeling [vFTS06]. Unfortunately, achieving real-time simulations in combination with volume conservation is still a difficult task. While different methods have been presented, many of them are simply not robust or fast enough to be used in interactive applications. Furthermore, most real-time methods introduce special restrictions, such as lowering the complexity of the simulation mesh, or do not consider the volume at all. Other systems, which are robust and accurate enough, tend to be non-interactive when using thousands of geometric elements, and may be difficult to implement for a parallel computation on the GPU.

We propose an efficient and robust simulation method for incompressible deformable objects. In contrast to most systems, we only consider a closed triangle mesh for our simulation model and thus we do not need any manual preprocessing or creation of special simulation models, like tetrahedral meshes. Not requiring elements in the interior of the object not only saves computation time but also memory to store the object. Our system handles thousands of elements in real-time, does not have any stability issues, is able to conserve the volume, both globally and locally, produces high quality and visually plausible deformations, is easy to implement and benefits from the speed of today's GPUs.

The elasticity of the objects is simulated with an unconditionally stable method called shape matching [MHTG05]. This method is able to simulate visually plausible elastic and plastic deformations. Fortunately, shape matching is robust by construction which allows bigger time steps for interactive applications. The method matches points which are

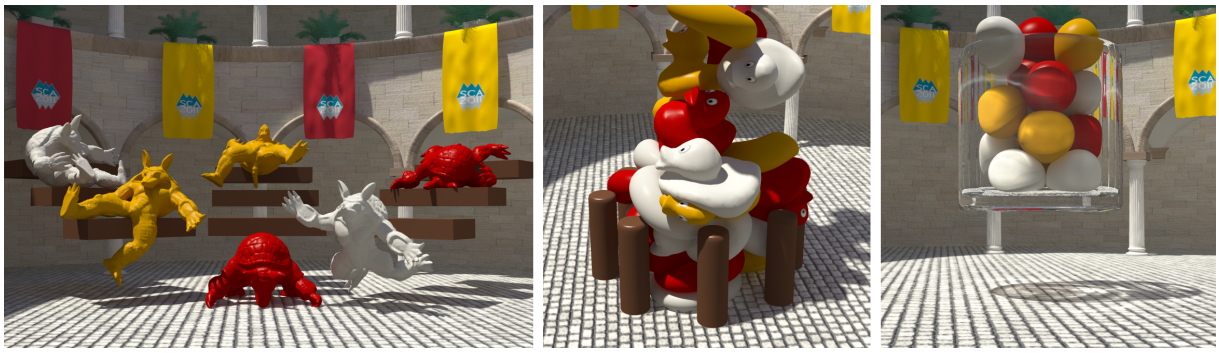

Figure 1: *Real-time deformations: Armadillos (32442 particles total), 20 ducks and 20 tori (21280 particles total), and 20 balls (7640 particles total) robustly deform with computation times of* 4.69 ms, 3.54 ms, *and* 2.34 ms *for one time step and have a maximum volume loss of* 0.1%, 0.5%, *and* 0.2% *for an individual object respectively in each scene.*

clustered in several overlapping regions to their best rigid body transformations. While the original approach is limited to only a few overlapping regions in real-time, Rivers and James [RJ07] extended the method by fast summations on regular lattices with many regions. However, their summation technique is not extendable to irregular models and is not suitable for an efficient parallel computation on the GPU. Our novel summation technique overcomes these limitations and is not limited to the shape matching method. Additionally, we preserve the volume of the object which is usually changed after resolving collisions. In order to make sure that already solved collisions are not corrupted again, the volume correction has to consider collisions. Furthermore, simply correcting positions introduces oscillations [ISF07]. We therefore propose an oscillation-free and collision-aware volume constraint which is able to conserve the volume both globally and locally. We use the volume formulation presented in [HJCW06] where volume preservation for a mass-spring system was used. This formulation enables us to define a local volume correction. While Hong et al. [HJCW06] only considered special collision forces for the local volume correction, we present a new heuristic which measures the volume change and we also consider the velocities in a similar way to eliminate oscillations. In addition, our volume constraint does not only work with the shape matching approach but can be used with any other method that simulates the elasticity of an object.

Our contributions:

- A fast and robust deformation technique for incompressible triangle meshes with thousands of elements executed on the GPU.
- A method purely based on the surface of the object, reducing the computational costs and memory consumption.
- An oscillation-free and collision-aware volume constraint for both global and local volume conservation.
- An efficient parallel summation algorithm for shape matching with arbitrary triangle meshes.

2. Related work

One of the first studies about deformable objects was [TPBF87] which presented a simulation method based on finite differences, followed by many different approaches such as the finite element method [OH99], the finite volume method [TBHF03], the boundary element method [JP99], and mass-spring systems [DSB99]. A survey of physically based deformable models in computer graphics is given in [GM97] and [NMK*05].

Different approaches have been presented to improve the speed of those simulation methods. For instance, Wu et al. [WDGT01] introduced an adaptive method for non-linear finite elements based on progressive meshes [Hop96]. Debunne et al. [DDCB01] presented an automatic space and time adaptive level of detail technique for continuous models which uses an adaptive time stepping and a non-nested multiresolution hierarchy of tetrahedral meshes. In [CGC*02], a hierarchical basis on the control lattice was defined in order to enable a level of detail adaption and therefore an interactive simulation. Grinspun et al. [GKS02] used a refinement of the basis functions instead of refining the elements to reduce the computational costs. In order to be able to cut deformable models, Steinemann et al. [SOG08] applied an adaptive shape matching method based on an octree representation using a linear fast summation technique. Adaptive methods improve the speed of the simulation but at a loss of accuracy. Instead of applying adaptive methods, real-time simulations can also be obtained by dimensional model reduction techniques. Barbič and James [BJ05] used a precomputed subspace integration method in order to get fast simulations which, on the other hand, could restrict the deformations when using a non-optimal low-dimensional subspace.

Additionally, the numerical stability has been addressed by many authors. The method of Terzopoulos et al. [TPBF87] becomes numerically ill-conditioned for stiff models. They solved this problem by using a hybrid

model which consists of a rigid and a deformable component [TW88]. In [BW98] an implicit integration method was used in order to perform a stable simulation with large time steps. To solve the numerical issues and reduce the costly computation, when using a non-linear strain tensor, Müller et al. [MDM*02] used a precomputed stiffness matrix in combination with a tensor field for local rotations. In contrast to that, Irving et al. [ITF04] introduced invertible finite elements for the robust simulation of large deformations.

Volume loss can be a distracting artifact when large deformations occur. With meshless methods, as presented in [MKN*04], it is possible to directly adjust common material properties derived from continuum mechanics, such as Poisson's ratio. However, the used moving least squares method cannot guarantee a precise conserved volume. In order to simulate realistic plastic deformations, Bargteil et al. [BWHT07] presented an enhanced plasticity model that allows volume preservation but is intended for offline simulations. Nedel and Thalmann [NT98] used additional angular springs in their mass-spring system to preserve the shape of a muscle. However, the volume of the model is not conserved accurately in their approach. Teschner et al. [THMG04] defined an energy function for each tetrahedron of their model in order to obtain forces which preserve the local volume approximately. Purely local volume conservation was presented in [ISF07]. They solved the locking problem which commonly arises when using a high Poisson's ratio with the linear finite element method. Unfortunately, their approach is not suitable for real-time applications with many elements. Global volume correction was also used in mesh skinning techniques [vFTS08] and can also be applied to the lattice shape matching approach [TK09].

3. Overview

The time integration scheme used for the simulation has to consider the elasticity of the objects, constraints between objects, such as collisions, and the objects' volumes. Volume change can occur directly by modifying positions or indirectly via velocity alterations. Trying to make an object incompressible by merely constraining the positions can cause oscillations. That is because artificially correcting positions during a time step effectively changes the corresponding velocities for that step, and thus yield overshooting in the next time step. To overcome this problem, we treat errors in positions and velocities separately, following [ISF07].

We use a closed triangle mesh with a particle located at every vertex which contains dynamic attributes like velocity and mass. Our system uses the following simple time integration scheme with step size Δt and external acceleration $\mathbf{a}$, like gravity, to advance particle positions and velocities from $(\mathbf{x}^n, \mathbf{v}^n)$ to $(\mathbf{x}^{n+1}, \mathbf{v}^{n+1})$:

1. $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t \mathbf{a}$
   $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1}$
2. Modify $\mathbf{x}^{n+1}$ and $\mathbf{v}^{n+1}$ to account for the elastic behavior.
3. Collide objects and modify $\mathbf{x}^{n+1}$ and $\mathbf{v}^{n+1}$ to achieve a collision free state and simulate friction.
4. Correct positions $\mathbf{x}^{n+1}$ to restore the volume without violating collisions.
5. Correct velocities $\mathbf{v}^{n+1}$ to avoid oscillations.

While Steps 1-3 can be realized with any method of your choice, we propose a fast and robust geometrically motivated approach leading to visually plausible results. This method, as described in Section 4, is stable even when using an explicit Euler integration scheme. As Step 3 already modifies positions to get a collision free state, the volume correction in Step 4 needs to take those collisions into account. The last step makes the velocity field divergence free to avoid a volume change in the next time step, thus eliminating oscillations. Steps 4 and 5 are described in Section 5.

4. Elastic Behavior

The elastic behavior of deformable objects can be computed in different ways. Mass-spring systems [THMG04] or more sophisticated methods like the well-known finite element method are typically used. Both methods have the drawback that they are not unconditionally stable when using a fast explicit time integration scheme. On the other hand, using a stable implicit integration scheme [BW98] is more time consuming, limiting the number of particles that can be simulated interactively. Additionally, many particles in the interior of the object have to be simulated when using volumetric elements such as tetrahedrons. As we are interested in interactive applications, we use the fast and unconditionally stable shape matching approach [MHTG05] to compute the elastic behavior. Instead of using volumetric models, only the surface of the object is considered disregarding all interior particles. With our novel fast summation technique it is possible to simulate smooth deformations with thousands of particles in real-time as shown in Figure 1.

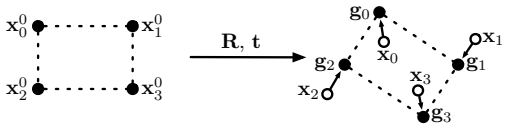

Figure 2: *Shape Matching: The undeformed points $\mathbf{x}_i^0$ are matched to the deformed points $\mathbf{x}_i$ by a rigid transformation defining goal positions $\mathbf{g}_i$. The goal positions are then used to pull the deformed points towards them.*

Shape Matching: Updating particle positions by considering the forces originating from stress and strain can cause overshooting and instabilities. This difficulty is avoided by the shape matching approach which rather defines goal positions $\mathbf{g}_i$ for all particles after evolving the particles forward in time. Using the goal positions to pull the deformed particles towards them, as shown in Figure 2, avoids the typical

overshooting problems. The goal positions are computed by finding the best rigid transformation which matches a set of given particles: given positions $\mathbf{x}_i^0$ in a non-deformed state and the integrated positions $\mathbf{x}_i = \mathbf{x}_i^{n+1}$ we find the rotation matrix $\mathbf{R}$ and translation vectors $\mathbf{t}$ and $\mathbf{t}^0$ which minimize:

$$\sum_i m_i \left(\mathbf{R} \left(\mathbf{x}_i^0 - \mathbf{t}^0 \right) + \mathbf{t} - \mathbf{x}_i \right)^2 ,$$

where m_i are weights of individual particles, typically their masses. According to [MHTG05], the optimal translation vector $\mathbf{t}$ is the center of mass and $\mathbf{t}^0$ is the initial center of mass respectively:

$$\mathbf{t}^0 = \frac{1}{M} \sum_i m_i \mathbf{x}_i^0 , \quad \mathbf{t} = \frac{1}{M} \sum_i m_i \mathbf{x}_i , \quad M = \sum_i m_i . \tag{1}$$

The matrix $\mathbf{R}$ is the rotational part of the affine transformation,

$$\mathbf{A} = \sum_i m_i \left(\mathbf{x}_i - \mathbf{t} \right) \left(\mathbf{x}_i^0 - \mathbf{t}^0 \right)^{\mathrm{T}} , \tag{2}$$

which is extracted via a polar decomposition. We use cyclic Jacobi iterations with a "warm start" (see [GVL96]) to obtain the square root $\mathbf{U}$ from the diagonal form of $\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{U}^2$ and get $\mathbf{R} = \mathbf{A}\mathbf{U}^{-1}$. Given the optimal rigid transformation for a set of particles, the goal positions,

$$\mathbf{g}_i = \mathbf{T} \begin{bmatrix} \mathbf{x}_i^0 \\ 1 \end{bmatrix} ,$$

are computed for each particle with $\mathbf{T} = [\mathbf{R} \ (\mathbf{t} - \mathbf{R}\mathbf{t}_0)]$. The goal positions define the transformed original shape minimizing the distance to the current deformed particles. Using these goal positions, we update positions and velocities according to:

$$\begin{aligned} \mathbf{v}_i^{n+1} &:= \mathbf{v}_i^{n+1} + s \frac{\mathbf{g}_i - \mathbf{x}_i}{\Delta t} \\ \mathbf{x}_i^{n+1} &= \mathbf{x}_i^n + \Delta t \, \mathbf{v}_i^{n+1} , \end{aligned} \tag{3}$$

with the user defined stiffness $s \in [0,1]$ of the object which controls how fast the particles will move to their resting state. By updating the velocities $\mathbf{v}_i^{n+1}$ using the given goal positions $\mathbf{g}_i$ we overcome the common stability issues. Additionally, the velocities can be damped before updating the final positions in order to gain more realistic results.

Simulation Model and Shape Regions: Instead of simulating many particles in the interior of the object, we simply use a closed triangle mesh for our simulation model. This allows us to simulate more particles on the surface of the object and thus ensures a smooth deformation without wasting computation time for the interior of the object. Computing only one transformation for all particles results in a rigid motion. Instead, we partition our triangle mesh into multiple overlapping regions. For each particle of the triangle mesh we define a region $\mathfrak{R}_i$. It contains all the particles in the ω-ring of the i-th particle. The region size ω is a user defined constant that further defines the stiffness of the object. Larger values of ω makes the model stiffer because the shape matching becomes more global. First, we compute the optimal rigid transformation for each region as well as the goal positions for all particles. The final goal positions are then obtained by blending the goal positions from all regions containing the particle:

$$\mathbf{g}_i = \frac{1}{|\mathfrak{R}_i|} \sum_{j \in \mathfrak{R}_i} \mathbf{T}_j \begin{bmatrix} \mathbf{x}_i^0 \\ 1 \end{bmatrix} .$$

A general problem when using shape matching is the fact that the here used method to compute the polar decomposition cannot extract a proper rotation if $\det\mathbf{A} = 0$. This is the case, when all particles in a region are coplanar which occurs quite often when purely considering the surface of an object. Therefore, we add an extra point, which is based on the surface normal, to the region when building the affine matrix $\mathbf{A}$ from Equation 2. With $\mathbf{n}_i$ being the averaged normal of all triangles containing the i-th particle, we add the point,

$$\mathbf{p} = \mathbf{x}_i + l\mathbf{n}_i ,$$

to the region $\mathfrak{R}_i$, where $l = \frac{1}{|\mathfrak{R}_i|} \sum_{j \in \mathfrak{R}_i} m_j \|\mathbf{x}_j^0 - \mathbf{t}^0\|$ is the weighted average length of all particles to the initial center of mass of the i-th region. Using the average length l, we assure that the normal has the same influence on the affine matrix as the initial points, so that the extracted rotation is not noticeably affected when $\det\mathbf{A} > 0$. As the extra point cannot completely avoid $\det\mathbf{A} < 0$, which would result in an orientation reversing orthogonal transformation instead of the closest rotation, we fix this as described in [AHB87] by simply negating the last column of $\mathbf{A}$ if necessary. The problem could also be solved by computing a singular value decompositions (SVD) $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$ using two-sided Jacobi rotations as proposed in [TKA10]. However, this method is slightly slower than our approach and requires to store the additional matrix $\mathbf{U}$ for a warm start. In contrast, using one-sided Jacobi rotations [GVL96] to compute the SVD is equally fast

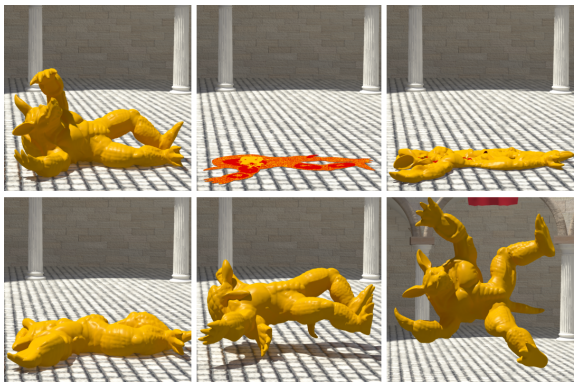

Figure 3: *An Armadillo degenerates due to disabled physics. Our method is still able to quickly recover from inverted or degenerated states after enabling the deformable physics simulation again.*

as our approach (because the non-diagonal elements of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ are implicitly set to zero) but seems to be numerically more difficult for nearly singular eigenvalues since $\mathbf{U}$ is computed by column scaling which can break the orthogonality of $\mathbf{U}$. More importantly, our experiments show that adding the extra point gives enough stability in practice and thus allows us to restore objects which are completely inverted or degenerated, as shown in Figure 3.

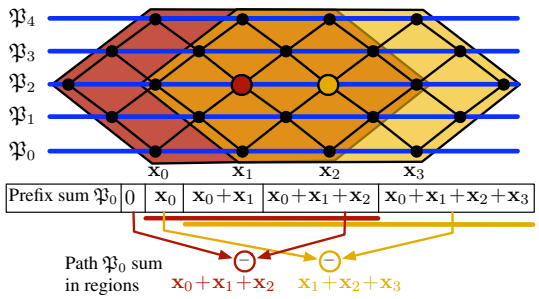

Figure 4: *Fast Summation: Two overlapping shape regions with size $\omega = 2$ of two particles (red and yellow) and the associated paths $\mathfrak{P}_i$ (blue). Computing the sum over a region is done by first computing the prefix sums over the associated paths and then adding the difference of the borders to the region's sum as shown for $\mathfrak{P}_0$.*

Fast Summation: When increasing the region size ω the naive computation of the sums of each region in Equations 1 and 2 becomes a system bottleneck. For a mesh with n regions computing the sums to construct $\mathbf{t}_i$ and $\mathbf{A}_i$ for all regions takes $O\left(\omega^d n\right)$ operations, where d is the dimension of the simulation mesh. Using a regular lattice ($d = 3$), Rivers and James [RJ07] have shown how to reduce the costs for sum computation to $O(n)$ by reusing already computed sums of neighboring regions. Unfortunately, their approach cannot be directly extended to irregular lattices or arbitrary triangle meshes. Furthermore, their approach is not suitable for an efficient parallel computation because of direct dependencies between neighboring regions. Instead, we present a novel solution for triangle meshes ($d = 2$) which takes $O(\omega n)$ instead of $O\left(\omega^2 n\right)$ operations and is suitable for an efficient parallel computation.

For each region we want to compute,

$$\begin{aligned}\mathbf{t}_i &= \frac{1}{M_i}\sum_{j\in\mathfrak{R}_i} m_j\mathbf{x}_j \\ \mathbf{A}_i &= \sum_{j\in\mathfrak{R}_i} m_j\left(\mathbf{x}_j-\mathbf{t}_i\right)\left(\mathbf{x}_j^0-\mathbf{t}_i^0\right)^{\mathrm{T}} \\ &= \sum_{j\in\mathfrak{R}_i} m_j\mathbf{x}_j\mathbf{x}_j^{0\mathrm{T}} - M_i\mathbf{t}_i\mathbf{t}_i^{0\mathrm{T}}\,, \end{aligned} \tag{4}$$

where M_i is the precomputed mass of the specific region. To quickly compute the sums for all regions, we first generate multiple paths (see below) through the edges of the triangle mesh. A path $\mathfrak{P}_i$ is a set of ordered particles $\mathbf{x}_{i_1},\ldots,\mathbf{x}_{i_n}$ which are connected through the edges of the triangle mesh (see Figure 5). The paths cover all particles of the mesh, so that each particle is part of exactly one path. The sum computation, as depicted in Figure 4, is split into two phases. First, we compute the prefix sum [SHZO07] for each path $\mathfrak{P}_i$ with elements $\mathbf{x}_{i_j}$, $j \in [1, n_i]$:

$$\bar{\mathbf{t}}_{i_j} = \sum_{k=1}^{j} m_{i_k}\mathbf{x}_{i_k}\,, \quad \bar{\mathbf{A}}_{i_j} = \sum_{k=1}^{j} m_{i_k}\mathbf{x}_{i_k}\mathbf{x}_{i_k}^{0\mathrm{T}}\,.$$

Computing all prefix sums takes $O(n)$ operations and can efficiently be done in parallel. In the second step, we compute the final sums for each region. Each region initializes its sums $\mathbf{t}_i := \mathbf{0}$ and $\mathbf{A}_i := \mathbf{0}$. If a region contains the particles of the i-th path with indices $[i_k,\ldots,i_l]$ we add,

$$\mathbf{t}_i := \mathbf{t}_i + \bar{\mathbf{t}}_{i_l} - \bar{\mathbf{t}}_{i_{k-1}}\,, \quad \mathbf{A}_i := \mathbf{A}_i + \bar{\mathbf{A}}_{i_l} - \bar{\mathbf{A}}_{i_{k-1}}\,,$$

to the region's sums. The final affine matrix of a region is then $\mathbf{A}_i := \mathbf{A}_i - M_i\mathbf{t}_i\mathbf{t}_i^{0\mathrm{T}}$. Note that the fast summation for the affine matrix only works because the sum in Equation 4 is region independent and only depends on the particles j but not the region i. Building the final sums takes $O(\omega n)$ operations because each region has roughly $2\omega+1$ paths (compare Figure 4 for a regular mesh) passing through it in an optimal path layout. Our fast summation technique is illustrated in Figure 4 for two regions. Computing the regions' sums as well as the prefix sum is done in parallel. Note that the transformation matrices $\mathbf{T}_i$ are computed analogously in parallel by first computing the prefix sums over the paths and then summing them up over the regions.

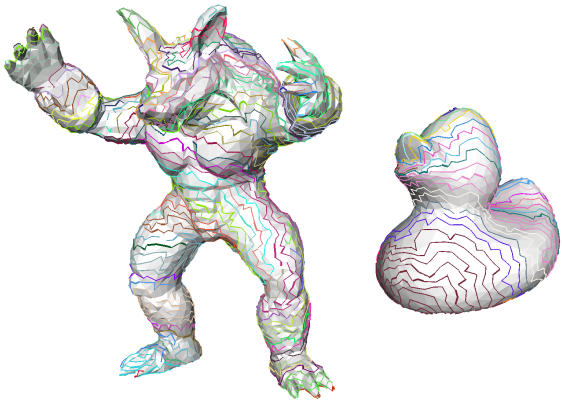

Figure 5: *The resulting paths (each in a different color) for the Armadillo and the duck model from our path generation algorithm used to compute our fast summation technique.*

Path Construction: In order to compute the sums of a region, we need to subtract values from the computed prefix sums. The layout of the paths can have a large impact on the run-time. It would be optimal if each region had only one

path that crosses it, so that the region's sum could be evaluated by a single subtraction, but unfortunately, this is not possible. We therefore aim to generate as few paths as possible so that each region has only a small number of crossing paths, reducing the required memory transactions. As it is difficult to find the most optimal path layout, our greedy path generation algorithm successively grows paths based on a heuristic to reach a good solution for this optimization problem. We start with a single vertex and add neighbors until the path length exceeds a given size or cannot be extended any further. Our heuristic tries to grow a path by choosing vertices which also have neighbors that are already part of a path. This avoids gaps between paths and places the paths in a parallel fashion. We then select the vertex which generates lines that are as parallel as possible. Therefore, we use the vertex with the smallest distance to a plane passing through the starting vertex of the current path, e. g. the X-Y-plane. Figure 5 shows the results of our path generation algorithm.

Damping: In order to damp the velocities in Equation 3, we use the stable method introduced by Müller et al. [MHHR07]. Damping is performed per region (see [RJ07]). The required inertia tensor and the angular momentum, that are used to compute the angular velocity, as well as the linear velocity, are efficiently computed in parallel using our fast summation technique. Notice that damping is not necessary to achieve a stable simulation but it can enhance the visual result.

5. Volume Preservation

Simulating deformable bodies without considering their volumes can cause a huge volume loss which might look implausible (see Figure 6). We use the volume formulation given in [HJCW06] and transform the volume integral of the vector field $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^3, \mathbf{x} \in \mathbb{R}^3$ over the volume V into a surface integral using the divergence theorem:

$$\iiint_V \nabla \cdot \mathbf{f}(\mathbf{x})\, d\mathbf{x} = \iint_{\partial V} \mathbf{f}(\mathbf{x})^{\mathrm{T}} \mathbf{n}(\mathbf{x})\, d\mathbf{x} \,,$$

where ∂V is the boundary of the volume and $\mathbf{n}$ is the outer surface normal vector. Thus, the volume of the body can be simply computed using the identity function $\mathbf{f}(\mathbf{x}) = \mathbf{x}$:

$$\iiint_V \nabla \cdot \mathbf{x}\, d\mathbf{x} = \iint_{\partial V} \mathbf{x}^{\mathrm{T}} \mathbf{n}\, d\mathbf{x} = 3V \,. \tag{5}$$

Equation 5 defines the constraint $C := \iint_{\partial V} \mathbf{x}^{\mathrm{T}} \mathbf{n}\, d\mathbf{x} - 3V_0 = 0$ for Step 4 of our integration scheme, where V_0 is the object's volume in the non-deformed state. The surface integral of Equation 5 can be written as a sum over all triangles T_i, $i = 1, \ldots, m$, formed by particle positions $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$:

$$\iint_{\partial V} \mathbf{x}^{\mathrm{T}} \mathbf{n}\, d\mathbf{x} = \frac{1}{3} \sum_{i=1}^{m} A_i (\mathbf{a}_i + \mathbf{b}_i + \mathbf{c}_i)^{\mathrm{T}} \mathbf{n}_i \,, \tag{6}$$

where A_i is the area of the i-th triangle.

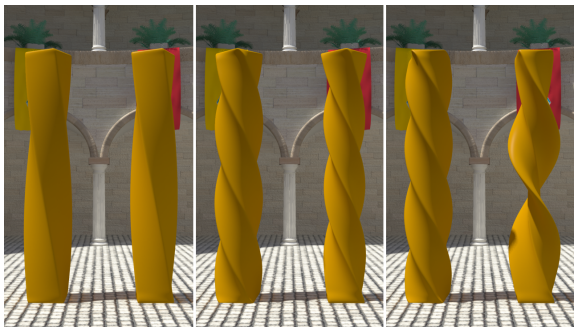

Figure 6: *Three frames of a simulated twirled bar: The volume correction (left bar) avoids implausibly large deformations (right bar) while taking only the surface into account.*

Given the positions of our particles $\mathbf{X} = [\mathbf{x}_1^{\mathrm{T}}, \ldots, \mathbf{x}_n^{\mathrm{T}}]^{\mathrm{T}}$ we want to find the offset $\Delta\mathbf{X}$ so that $C(\mathbf{X} + \Delta\mathbf{X}) = 0$. We use the position-based approach [MHHR07] to fulfill the constraint and obtain the correction for each individual particle:

$$\Delta\mathbf{x}_i = -\frac{w_i C(\mathbf{X})}{\sum_j w_j \|\nabla_{\mathbf{x}_j} C(\mathbf{X})\|^2} \nabla_{\mathbf{x}_i} C(\mathbf{X}) \,, \tag{7}$$

where w_i are weights used to localize the volume correction (see below) and $\nabla_{\mathbf{x}_i} C(\mathbf{X}) = \partial C(\mathbf{X}) / \partial \mathbf{x}_i$. In contrast to [HJCW06] we define a heuristic for the weights which enables us to define local volume correction based on the current motion of the object. Computing the gradient of $\nabla C(\mathbf{X})$ is done as follows. Rewriting the discrete surface integral from Equation 6 into a sum over all particle positions $\mathbf{x}_i$, $i = 1, \ldots, n$, yields:

$$\frac{1}{3} \sum_{i=1}^{m} A_i (\mathbf{a}_i + \mathbf{b}_i + \mathbf{c}_i)^{\mathrm{T}} \mathbf{n}_i = \frac{1}{3} \sum_{i=1}^{n} \mathbf{x}_i^{\mathrm{T}} \bar{\mathbf{n}}_i \,, \tag{8}$$

where $\bar{\mathbf{n}}_i = \sum A_j \mathbf{n}_j$ denotes the sum of the area weighted normals of all triangles containing the i-th particle. To simplify computations on the GPU, we assume fixed normals during the correction and approximate the gradient by:

$$\nabla C(\mathbf{X}) \approx \frac{1}{3} [\bar{\mathbf{n}}_1^{\mathrm{T}}, \ldots, \bar{\mathbf{n}}_n^{\mathrm{T}}]^{\mathrm{T}} \,.$$

In our experiments the approximate gradient did not change the results significantly compared to the full expression which includes the negligible terms: $\mathbf{x}_i \partial \bar{\mathbf{n}}_i / \partial \mathbf{x}_j$.

Unlike [HJCW06] we also correct velocities to avoid overshooting problems in the following time step. Differentiating Equation 5 with respect to t, under the assumption of constant V, leads to:

$$\iiint_V \nabla \cdot \mathbf{v}\, d\mathbf{x} = \iint_{\partial V} \mathbf{v}^{\mathrm{T}} \mathbf{n}\, d\mathbf{x} = 0 \,, \tag{9}$$

which turns out to be the divergence free velocity field for an incompressible object. The constraint $C := \iint_{\partial V} \mathbf{v}^{\mathrm{T}} \mathbf{n}\, d\mathbf{x} = 0$

for Step 5 of our integration scheme is corrected analogously to the position constraint.

Local Volume Preservation: If all weights w_i are equal our constraint defines a global volume correction. In order to localize the correction we choose the weights w_i in such a way that only particles for which a volume change occurred are affected. We use a heuristic to define these weights. Volume change occurs around particles which deform due to external and internal forces. Therefore, we set the weights w_i in relation to the amount of change computed during the shape matching:

$$w_i = (1-\alpha)\underbrace{\left(\frac{\|\mathbf{c}_i\|}{\sum_j \|\mathbf{c}_j\|}\right)}_{=g_i} + \alpha\frac{1}{n}\,. \tag{10}$$

Here $\mathbf{c}_i$ is the change of the particle positions from the shape matching and $\alpha \in [0,1]$ allows the user to change the behavior between local and global volume correction. When using local weights g_i, particles that stay under stress tend to participate more in the volume correction compared to particles not moving at all. Assuring that $\sum g_i = 1$, we get a smooth transition between global and local volume conservation.

Inner Structures: To further improve the behavior of the volume constraint, we mimic the interior dynamics of the object. Shape matching only propagates volume changes at the surface of the object but cannot directly propagate volume changes through the object. Therefore, we insert distance constraints into the interior of the object. The distance constraints are used to measure how fast the volume change should be propagated through the interior of the object. Given a non-deformed particle $\mathbf{x}_i^0$, we intersect the ray $\mathbf{x}_i^0 - \lambda\bar{\mathbf{n}}_i^0$, $\lambda > 0$ with the triangle mesh. We then construct a distance constraint $C(\mathbf{x}_i,\mathbf{x}_k) := \|\mathbf{x}_i - \mathbf{x}_k\| - \|\mathbf{x}_i^0 - \mathbf{x}_k^0\|$ from $\mathbf{x}_i^0$ to the nearest particle $\mathbf{x}_k^0$ of the first triangle the ray hits.

The distance constraints do not affect the positions or velocities of particles, instead they are only used to measure how fast the volume change should be propagated through the object. We modify the weights g_i of Equation 10 by

$$g_i = \frac{\beta s_i d_i + (1-\beta)\|\mathbf{c}_i\|}{\sum_j \left(\beta s_j d_j + (1-\beta)\|\mathbf{c}_j\|\right)}\,,$$

where $d_i = |C(\mathbf{x}_i,\mathbf{x}_k)|$ is the change of the length of the distance constraint weighted with the stiffness s_i. The parameter $\beta \in [0,1]$ allows the user to define how much the distance constraint affects the weights. The stiffness parameter s_i in our distance constraint accounts for different thicknesses of the object. In contrast to thick structures, thin structures should not be able to squeeze too much. To account for this behavior the user defines two stiffness values $s_{\max} \in [0,1]$ for the shortest distance constraint and $s_{\min} \in [0,1]$ for the longest distance constraint in the non-deformed model. The stiffness s_i of each individual distance constraint is the linear interpolation between $s_{\min}$ and $s_{\max}$ depending on its initial length.

Considering Collisions: If a particle collides with an obstacle, the volume correction must not violate the collision constraint. We therefore set the weights of colliding particles to zero but only for the position update and not for the velocity update. Such a strategy ensures that already resolved collisions are not violated, while the volume change still affects the colliding object through the velocities. Additionally, we smooth the weights with a Laplacian filter [DMSB99] to avoid drastic changes in the weights near collided particles. Note that the discrete Laplacian operator, namely the umbrella operator, can also be efficiently computed on the GPU using our fast summation technique.

6. Implementation Details

The elastic behavior as well as the volume constraint are computed on the GPU using CUDA. In order to avoid kernel starts for each object separately, we pack all the particles from all objects into one array and simulate all objects at once. The particles are stored in the array according to the order of the paths so we can apply the fast summation technique as described in Section 4. First, all vertices of the first path are stored in the array, then those from the second path and so on. This ordering enables us to compute all the prefix sums via one segmented prefix sum [SHZO07]. Note that we limit our path length to 512 due to numerical reasons (see below) and have a fixed segmentation for the prefix sum. Thus, we can evaluate the prefix sum with only two CUDA kernels and achieve nearly the same speed as for the normal prefix sum. While the computation of the segmented prefix sum has no random memory accesses, we store the result of the segmented prefix sum in a texture to benefit from cached texture reads in the second pass of the fast summation.

In order to compute the volume constraint in parallel, we only need to evaluate the volume integral from Equations 5 and 9 as well as the weights for the localization. The rearranged discrete integral over the vertices from Equation 8 and the weights for the localized volume constraint are computed via a segmented sum reduction. The umbrella operator, used for smoothing the weights, is again computed using our fast summation technique.

Numerical Stability: Due to the fact that we evaluate all computations on the GPU with 32 bit floating-point arithmetics, the result of the segmented prefix sum becomes numerically unstable for long paths. The evaluation of the affine transformation $\mathbf{A}$ becomes particularly unstable because of the "quadratic term" $\mathbf{x}_i\mathbf{x}_i^{0\mathrm{T}}$. We therefore limit the path length to 512. Additionally, as the rotation $\mathbf{R}$, which is computed via the polar decomposition, is independent of any translation or scaling, we can translate and scale the affine transformation $\mathbf{A}$. We simply compute all sums in an object based local coordinate system with the origin $\mathbf{x}_0^n$ and a precomputed scaling factor, so that each particle is bound to $[-1,1]^3$. Limiting the positions in the local coordinate sys-

Scene	# Particles	ω	α / β / s	Max. Vol. Loss	CPU Naive	CPU Fast Sum.	GPU Shape Matching	GPU Vol. Corr.	GPU Total
Armadillos	32442	3	0.1 / 0.1 / 0.9	0.1%	114.75	69.92	2.89	1.80	4.69
Ducks and Tori	21280	2	0.2 / 0.1 / 1	0.5%	44.88	30.2	2.03	1.51	3.54
Balls in Glass	7640	2	0.5 / 0.1 / 1	0.2%	15.86	10.58	1.20	1.14	2.34

Table 1: *Scene statistics and simulation timings: Number of simulated particles, region size ω, and simulation parameters used for each scene together with the maximum volume loss for an individual object. Timings (in ms) are listed for the naive implementation and the fast summation technique on the CPU. GPU timings are split into the shape matching part including damping and the volume correction part. GPU timings also include the time needed to copy data between CPU and GPU.*

tem together with the limited path length enables us to compute the deformation without numerical problems.

Visualization: Even though we use highly detailed triangle meshes, it might not be most beneficial to simulate all small details. Instead, a coarse simulation model can be used with the fine details only added for the sake of visualization. To achieve smooth deformations of fine details, we use the idea of Phong-shading. This has also been used in the field of modeling with multiresolution hierarchies to preserve fine details [KVS99]. The idea is to assign each fine detail vertex $\mathbf{p}$ to a triangle of the non-deformed coarse simulation mesh and reconstruct the vertex after the deformation based on the triangle's vertices and its interpolated normals at the corners. As the bilinear interpolated normals over the triangles define a continuous normal field, we can find a base point $\mathbf{q}$ on a triangle $T(\mathbf{a},\mathbf{b},\mathbf{c})$ by setting the function

$$\mathbf{F} : (\alpha,\beta) \rightarrow (\mathbf{p} - \mathbf{q}(\alpha,\beta)) \times \mathbf{n_q}(\alpha,\beta)$$

to zero, where $\mathbf{q} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$ is the point on the triangle with barycentric coordinates α, β, and $\gamma = 1 - \alpha - \beta$ and $\mathbf{n_q} = \alpha\mathbf{n_a} + \beta\mathbf{n_b} + \gamma\mathbf{n_c}$ the according bilinear interpolated normal. The barycentric coordinates are precomputed by solving the bivariate quadratic equation $\mathbf{F}(\alpha,\beta) = \mathbf{0}$ with several Newton-iteration steps. With the signed distance $d = \text{sign}\left((\mathbf{p}-\mathbf{q})^T\mathbf{n_q}\right)\|\mathbf{p}-\mathbf{q}\|$ and the barycentric coordinates we can reconstruct the detail vertex at any time by evaluating $\mathbf{p}^{n+1} = \mathbf{q}^{n+1} + d\mathbf{n}_\mathbf{q}^{n+1}$.

7. Results

All of our examples use an Intel Core i7 950 with a NVIDIA GeForce GTX 470. Although our robust simulation method can handle large time steps, we run all simulations with a fixed time step size of 5 ms to ensure that our discrete collision detection does not miss any collisions. We use a bounding volume hierarchy to prune unnecessary tests and only perform vertex-triangle collisions. For the inner structures we use the parameters $s_{\min} = 0.01$ and $s_{\max} = 0.1$ in all scenarios. Additionally, the visualization of our models uses the technique as described in Section 6.

As a stress test, four deformable spheres, each consisting of 1562 particles, are squeezed by a plate, as depicted in Figure 7. We compare the global to the local volume correction technique, with and without inner structures. Using inner

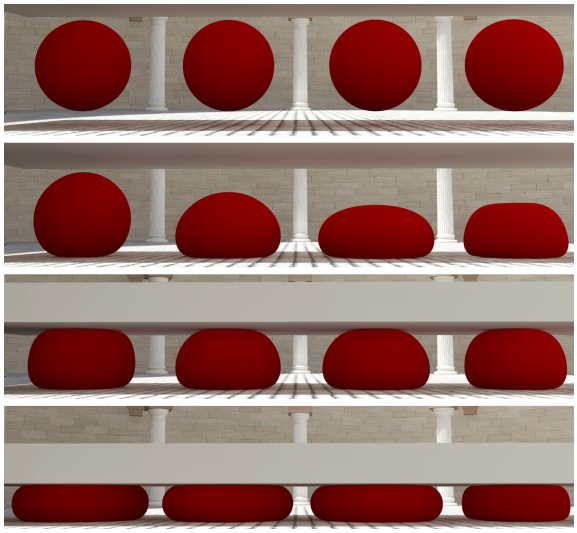

Figure 7: *Volume comparison of four spheres squeezed by a plate (left to right): Global volume conservation, local volume conservation with distance constraints ($\beta = 0.1$), local volume conservation without distance constraints ($\beta = 0$), and without volume conservation at all. The maximum volume loss is 0.6%, 0.7%, 0.7%, and 40% for each sphere.*

structures avoids overly excessive squeezing of the sphere. While the sphere with global volume correction does not change its shape considerably when it hits the ground, the local volume correction allows larger deformations. Although we only use a triangle surface for our simulation model, large deformations such as the twirl in Figure 6 can be visually plausibly simulated if the volume is taken into account. Additionally, our method is able to recover from completely inverted or degenerated states, as shown in Figure 3.

Run-times for the complex scenes from Figure 1 are summarized in Table 1. These run-times are measured without taking collision detection into account, which is executed in parallel and takes up to 9 ms in our scenes. Due to the fact that our collision detection is currently performed on the CPU, the simulation data has to be copied between CPU and GPU in each time step. Timings from the GPU implementation include damping as well as the time needed to copy the data between CPU and GPU. It is also important to

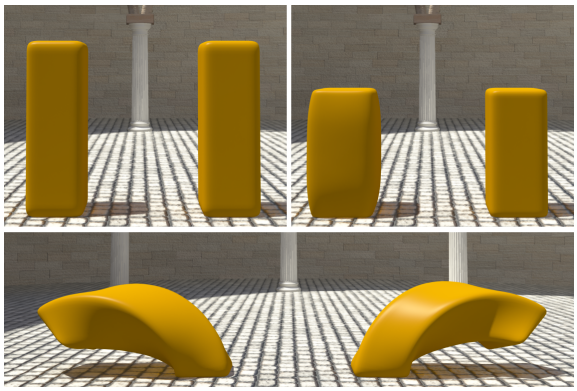

Figure 8: *Comparison between fast lattice shape matching (right) and our method (left): Both methods produce similar deformations but our method also considers the volume.*

mention that starting CUDA kernels and preparing memory transactions take some time, which results in run-times of about 1.5 ms even when simulating only a few particles.

While it is difficult to compare our method to physically accurate methods such as [ISF07], we compare our method to other visually plausible methods, such as the one in [RJ07]. Even when using only the surface, our method produces similar deformations compared to the lattice shape matching method as shown in Figure 8, but considers the volume correctly. As the run-times do not differ when using multiple objects with few particles or one object with many particles, we compare the run-times of both approaches using a cube with roughly 30k surface particles for our approach and $31^3 \approx 30$k lattice particles, which results in 5402 surface particles. We compare the run-times (see Table 2) of the unoptimized lattice shape matching (LSM), fast lattice shape matching (FastLSM), naive shape matching for surfaces on the CPU, our fast shape matching for surfaces on the CPU and our GPU implementation for different region sizes ω. The timings from all surface shape matching methods also contain the times for computing the volume correction. Even though the FastLSM is faster in theory, our fast summation technique benefits from caching effects and thus is equally fast on the CPU. The non-constant run-time of the FastLSM approach is due to special computations on the boundary. We also noticed this behavior in the sample source code available from [RJ07], which unfortunately is even slower as it is not optimized. The GPU implementation is, depending on the region size, 11-40 times faster than the naive surface shape matching and achieves a speedup of factor 15 compared to both fast summation techniques. Considering that we need only a fraction of particles on the surface compared to a volumetric model the actual speedup is even higher.

We want to mention that our approach needs slightly bigger region sizes to reproduce the same stiffness as the lattice based shape matching because of the missing inner particles. Even though the volume constraint is not momentum conserving we did not encounter any ghost forces and we believe that the velocity correction helps to avoid them. All together our method is not intended for physically correct simulations but is practical for visually plausible real-time deformations.

Method	$\omega = 1$	$\omega = 2$	$\omega = 3$	$\omega = 4$	$\omega = 5$
LSM	65.1	240.3	617.9	1436.2	2615.8
FastLSM	46.7	47.9	51.6	57.2	65.2
Naive surface SM	37.3	53.8	82.9	121.5	169.8
Fast surface SM	42.5	46.7	51.6	56.2	62.1
GPU surface SM	3.2	3.6	3.9	4.1	4.2

Table 2: *Run-time comparison (in ms) between the different shape matching methods for different region sizes ω with a cube consisting of roughly* 30k *particles. While both fast summation techniques have similar run-times, the GPU implementation achieves a speedup of factor* 15.

8. Conclusion

We proposed a robust deformation method for incompressible objects. Due to our oscillation-free and collision-aware volume constraint, it is possible to simulate our models by purely taking the surface of the object into account. Considering only the surface eliminates typical problems such as tetrahedron inversion. Our method can even recover easily from completely inverted or degenerated states. With our novel fast summation technique that is designed for an efficient parallel computation on the GPU, we can robustly simulate highly detailed surfaces with thousands of particles, obtaining visually plausible deformations. Comparing our method to similar geometrically motivated methods, we obtain a speedup of factor 15. Considering that we only need a fraction of particles because we do not require any particles in the interior of the object, the actual speedup is even higher. In future work we want to integrate a continuous collision detection on the GPU, thus removing the need to copy data between CPU and GPU.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable feedback, Dieter Finkenzeller for making the textures of our demonstration scenes and the Stanford Scanning Repository for the Armadillo model. The duck model is provided courtesy of Marco Attene by the AIM@SHAPE Shape Repository.

References

[AHB87] ARUN K. S., HUANG T. S., BLOSTEIN S. D.: Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9* (1987), 698–700. 4

[BJ05] BARBIČ J., JAMES D. L.: Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Trans. on Graphics 24*, 3 (July 2005), 982–990. 2

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proc. of SIGGRAPH 98* (1998), ACM, pp. 43–54. 3

[BWHT07] BARGTEIL A. W., WOJTAN C., HODGINS J. K., TURK G.: A finite element method for animating large viscoplastic flow. *ACM Trans. on Graph. 26*, 3 (July 2007), 16:1–16:8. 3

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Interactive skeleton-driven dynamic deformations. In *Proc. of SIGGRAPH 2002* (2002), ACM, pp. 586–593. 2

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic real-time deformations using space & time adaptive sampling. In *Proc. of SIGGRAPH 2001* (2001), ACM, pp. 31–36. 2

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. of SIGGRAPH 99* (1999), ACM, pp. 317–324. 7

[DSB99] DESBRUN M., SCHRÖDER P., BARR A.: Interactive animation of structured deformable objects. In *Proc. of SIGGRAPH 99* (1999), ACM, pp. 1–8. 2

[GKS02] GRINSPUN E., KRYSL P., SCHRÖDER P.: CHARMS: a simple framework for adaptive simulation. In *Proc. of SIGGRAPH 2002* (2002), ACM, pp. 281–290. 2

[GM97] GIBSON S. F., MIRTICH B.: *A survey of deformable modeling in computer graphics*. Tech. Rep. TR-97-19, Mitsubishi Electric Research Lab., Cambridge, MA, 1997. 2

[GVL96] GOLUB G., VAN LOAN C.: *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996. 4

[HJCW06] HONG M., JUNG S., CHOI M., WELCH S.: Fast volume preservation for a mass-spring system. *IEEE Comput. Graph. Appl. 26* (2006), 83–91. 1, 2, 6

[Hop96] HOPPE H.: Progressive meshes. In *Proc. of SIGGRAPH 96* (1996), ACM, pp. 99–108. 2

[ISF07] IRVING G., SCHROEDER C., FEDKIW R.: Volume conserving finite element simulations of deformable models. *ACM Trans. on Graphics 26*, 3 (July 2007), 13:1–13:6. 2, 3, 9

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2004), Eurographics Association, pp. 131–140. 3

[JP99] JAMES D. L., PAI D. K.: Artdefo: accurate real time deformable objects. In *Proc. of SIGGRAPH 99* (1999), ACM, pp. 65–72. 2

[KVS99] KOBBELT L., VORSATZ J., SEIDEL H.-P.: Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. Theory and Appl. 14* (1999), 5–24. 8

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2002), ACM, pp. 49–54. 3

[MHHR07] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *Journal of Vis. Commun. and Image Represent. 18* (2007), 109–118. 6

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. on Graphics 24*, 3 (July 2005), 471–478. 1, 3, 4

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2004), Eurographics Association, pp. 141–151. 3

[NMK*05] NEALEN A., MUELLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. In *Eurographics: State of the Art Report* (December 2005). 2

[NT98] NEDEL L. P., THALMANN D.: Real time muscle deformations using mass-spring systems. In *Proc. of the Comput. Grap. Int.* (1998), pp. 156–165. 3

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *Proc. of SIGGRAPH 99* (1999), ACM, pp. 137–146. 2

[RJ07] RIVERS A. R., JAMES D. L.: FastLSM: fast lattice shape matching for robust real-time deformation. *ACM Trans. on Graphics 26*, 3 (July 2007), 82:1–82:6. 2, 5, 6, 9

[SHZO07] SENGUPTA S., HARRIS M., ZHANG Y., OWENS J. D.: Scan primitives for gpu computing. In *Proc. of the 22nd ACM SIGGRAPH/Eurographics Symp. on Grap. Hardware* (2007), Eurographics Association, pp. 97–106. 5, 7

[SOG08] STEINEMANN D., OTADUY M. A., GROSS M.: Fast adaptive shape matching deformations. In *Proc. of the 2008 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2008), Eurographics Association, pp. 87–94. 2

[TBHF03] TERAN J., BLEMKER S., HING V. N. T., FEDKIW R.: Finite volume methods for the simulation of skeletal muscle. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2003), Eurographics Association, pp. 68–74. 2

[THMG04] TESCHNER M., HEIDELBERGER B., MULLER M., GROSS M.: A versatile and robust model for geometrically complex deformable solids. In *Proc. of the Comput. Grap. Int.* (2004), pp. 312–319. 3

[TK09] TAKAMATSU K., KANAI T.: Volume-preserving lsm deformations. In *Proc. of SIGGRAPH Asia Sketches* (2009). 3

[TKA10] TWIGG C. D., KAČIĆ-ALESIĆ Z.: Point cloud glue: constraining simulations using the procrustes transform. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2010). 4

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Computer Graphics (Proc. of SIGGRAPH 87)* (1987), vol. 21, ACM, pp. 205–214. 2

[TW88] TERZOPOULOS D., WITKIN A.: Physically based models with rigid and deformable components. *IEEE Comput. Graph. Appl. 8* (1988), 41–51. 3

[vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. *ACM Trans. on Graphics 25*, 3 (July 2006), 1118–1125. 1

[vFTS08] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Volume-preserving mesh skinning. In *Vision Modeling and Visualization* (2008), pp. 409–414. 3

[WDGT01] WU X., DOWNES M. S., GOKTEKIN T., TENDICK F.: Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. In *Computer Graphics Forum* (2001), pp. 349–358. 2