Eurographics Symposium on Geometry Processing (2007)
Alexander Belyaev, Michael Garland (Editors)

Elastic Secondary Deformations by Vector Field Integration

Wolfram von Funck[1] Holger Theisel[2] Hans-Peter Seidel[1]

[1]MPI Informatik, Germany
[2]Bielefeld University, Germany

Abstract

We present an approach for elastic secondary deformations of shapes described as triangular meshes. The deformations are steered by the simulation of a low number of simple mass-spring sets. The result of this simulation is used to define time-dependent divergence-free vector fields whose numerical path line integration gives the new location of each vertex. This way the deformation is guaranteed to be volume-preserving and without self-intersections, giving plausible elastic deformations. Due to a GPU implementation, the deformation can be obtained in real-time for fairly complex shapes. The approach also avoids unwanted intersections in the case of collisions in the primary animation. We demonstrate its accuracy, stableness and usefulness for different kinds of primary animations/deformations.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

Animations and deformations are standard problems in Computer Graphics for which a variety of solutions exist. For the treatment of many applications, a deformation or animation can be divided into a primary and a secondary structure [OZH00]. A primary animation/deformation performs rather large and global changes of the shape. Typical examples are keyframe animations, interactive movement of solids, rigid body simulations including collision detection, or the interactive deformation of shapes. In addition to this, secondary deformations perform rather small changes of the shapes but contribute significantly to the realism of the scene. Secondary deformations can be explicitly modeled (e.g., facial animations or lip synchronization in animated full-body human characters), or they can be derived from the primary animation/deformation by assuming certain elastic material properties of the shape. An example of secondary deformations are jiggling and bouncing effects on the moving skin on a human body [PH06]. Part 1 of the accompanying movie (in the following called movie 1) illustrates a primary motion without and with secondary deformation.

This paper presents an approach to model elastic secondary deformations by constructing and integrating time-dependent divergence-free vector fields. Elastic deformations are connected to certain intrinsic forces and tend to move back to the original shape if these forces cease. This way, jiggling and oscillating effects can be represented. The deformation described here is steered by incorporating basic mechanical laws coming from the primary animation/deformation. In fact, a low number of mass-spring sets is simulated to steer the deformation. The results of this simulation are used for an on-the-fly construction of time-dependent vector fields, delivering the new position of each vertex by a numerical path line integration. It guarantees that the deformations are volume-preserving, and without (local or global) self-intersections.

To model secondary deformations which are derived from a primary motion, two general approaches are possible. Firstly, physically based approaches consider the inherent structure of the shape to simulate the deformations. Secondly, heuristic approaches describe the deformation by a low number of parameters in order to obtain plausible and fast deformations without the explicit consideration of physical laws. The technique presented here can be considered as a compromise between the two general approaches. While the deformation is described by a simple heuristic model (a lower number of mass-spring sets), their kinematic simulation is exact as well as the deformation itself is guaranteed to preserve important physical properties (volume, avoidance

of self-intersections). In particular, it turns out that these properties are strong enough to yield plausible deformations if the mass-spring sets are placed in an appropriate way.

The rest of the paper is organized as follows: section 2 reviews related work. Section 3 describes our vector field based approach. Section 4 describes how our model can handle collisions in the primary animations. Section 5 describes details of our GPU based implementation. Section 6 applies our technique to different approaches for the primary animations: interactive moving of solid bodies, keyframe animations, rigid body simulations, and interactive deformations. We evaluate and compare our method in section 7, while conclusions are drawn in section 8.

2. Related work

There is a huge body of approaches to define animations and deformations. Here we only mention approaches which explicitly deal with secondary deformation/motion, or which can incorporate secondary motion effects. In general, all approaches aim in finding appropriate combinations of physically exact simulations and simplifying assumptions to get plausible deformations at interactive rates.

Mass-spring systems are an intuitive technique to deform objects realistically. Initially used for facial modeling [PB81], other fields like skin, fat and muscle simulation [CHP89, TW90, TW91, NT98] and interactive animation of structured deformable objects [DSB99] have been addressed using mass-spring systems. In order to simulate larger mass-spring systems in real-time, [GEW05] developed a GPU implementation which simulates spring elongation and compression on the graphics card and renders the deformed surface.

The Finite Difference Method has been introduced by [TPBF87] as a tool to simulate elastically deformable models. The Finite Element Method has been used to simulate elastic [DDCB01, GKS02, MDM*02] and elastoplastic fracturing [OBH02, MG04] material. The method has also been used to obtain interactive skeleton-driven deformations [CGC*02, ZG05] and physically based rigging of deformable characters [CBC*05]. By employing modal analysis, which reduces the computational complexity of such simulations remarkably, [JP02, CK05] were able to obtain complex deformations at ineractive rates on the GPU. [TBHF03] used the Finite Volume Method to simulate skeletal muscle. [JP99] proposed the Boundary Element Method for simulating deformable objects accurately and interactively.

The above-mentioned approaches have in common that they rely on connected structures like mass-spring networks, grids or volumetric meshes. In contrast to this, mesh free methods [DC96, Ton98] abandon connectivity. Point based animation allows for animation of elastic, plastic and melting objects [MKN*04]. Here, both the particles on which the simulation is carried out and the object surface are represented by points without connectivity.

Trying to strictly adhere the laws of physics in order to get realistic deformations often comes at the cost of performance. This makes it difficult to simulate scenarios of moderate complexity in real-time. Non-physically motivated approaches sacrifice realism for performance. [MHTG05] proposed a geometrically motivated model for simulating deformable objects. This mesh free approach replaces energies by geometric constraints and forces by distances between current and goal positions. This way, the dynamic simulation is efficient and unconditionally stable.

The idea of constructing divergence-free vector fields to define interactive (primary) deformations was recently introduced in [vFTS06]. There, a 3D vector field $\mathbf{v}$ is constructed as

$$\mathbf{v}(\mathbf{x}) = \nabla p(\mathbf{x}) \times \nabla q(\mathbf{x}) \tag{1}$$

where p, q are C^2 continuous piecewise scalar fields which are defined over three different regions. The regions are set implicitly by a scalar field $r(\mathbf{x})$ and two thresholds r_i, r_o. In the inner region (i.e., at locations $\mathbf{x}$ with $r(\mathbf{x}) < r_i$), p and q are certain simple analytical (e.g., linear or quadratic) scalar fields $e(\mathbf{x})$ and $f(\mathbf{x})$ respectively. In the outer region ($r_o \leq r$), p and q hold $p = q \equiv 0$. In the intermediate region, p and q are smoothly blended as $p = (1-b)\cdot e + b \cdot 0$ and $q = (1-b)\cdot f + b \cdot 0$ where $b = b(r(\mathbf{x}))$ is a blending function given in Bezier representation as

$$b(r) = \sum_{i=0}^{4} w_i \, B_i^4\left(\frac{r - r_i}{r_o - r_i}\right). \tag{2}$$

In (2), B_i^4 describe the well-known Bernstein polynomials, and $(w_0, \ldots, w_4) = (0,0,0,1,1)$. [vFTS06] considers different choices of r, e, f for getting different modeling metaphors. Due to this direct construction of divergence-free vector fields, the approach is efficient, and at the same time produces physically plausible results thanks to the volume-preservation and the prevention of self-intersections.

3. Our approach

The main idea of our approach is to control the secondary deformation by a low number of parameters which we describe as a number of mass-spring sets (section 3.1). For each of the mass-spring sets we define a local deformation which is based in the integration of a divergence-free vector field (section 3.2). Based on this, different ways of composing them are possible (section 3.3). Also, the impact of a particular deformation can be limited to certain parts of the shape (section 3.4). Section 3.5 describes how to place and parametrize the mass-spring sets. Section 3.6 describes how a level-of-detail approach can be incorporated into our concept.

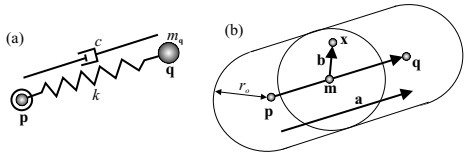

Figure 1: *(a) a mass-spring set:* $\mathbf{p}$ *is a fixed point inside the shape,* $\mathbf{q}$ *is a freely moving point, equipped with a mass* $m_\mathbf{q}$*; (b) configuration to define deformation* $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ *and vector field* $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$.

3.1. Mass-spring sets

Mass-spring systems are popular and well-understood tools to simulate various phenomena, among them deformations. Here we use a very simple one-degree-of-freedom mass-spring system and call it a *mass-spring set* (this naming should reflect that this is indeed one of the simplest mass-spring systems one can imagine). It consists of two points $\mathbf{p},\mathbf{q}$ which are connected by a spring with the stiffness k and the damping parameter c. The spring has a length of zero, i.e. $\mathbf{p}$ and $\mathbf{q}$ coincide in the rest state. While $\mathbf{p}$ is considered as a fixed anchor point inside a shape, $\mathbf{q}$ is equipped with a certain mass $m_\mathbf{q}$. Then the position of $\mathbf{q}$ can be simulated by basic rules of mechanics if the shape (and therefore $\mathbf{p}$ inside it) undergoes a primary motion. In fact, our simulation includes spring damping, inertia, and the repeated conversion of potential and kinetic energy. See the corresponding section on mass-spring systems in [NMK*06] for an explanation of these methods. Figure 1a illustrates a mass-spring set. Movie 3 (part 2) shows a simulation of 7 mass-spring sets with different characteristics using an interactive moving of the shape.

3.2. Constructing the deformation

For a mass-spring set with the points $\mathbf{p},\mathbf{q}$ at a certain location, we define a space deformation $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} : \mathbb{R}^3 \to \mathbb{R}^3$ with the following properties:

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{p}) = \mathbf{q} \tag{3}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x}) = \mathbf{x} \quad \text{for} \quad dist(\mathbf{x},\overline{\mathbf{p}\,\mathbf{q}}) > r_o \tag{4}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} \text{ is } C^2 \text{ continuous} \tag{5}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} \text{ is volume preserving} \tag{6}$$

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o} \text{ does not produce self-intersections} \tag{7}$$

where $dist(\mathbf{x},\overline{\mathbf{p}\,\mathbf{q}})$ describes the minimal Euclidean distance between a point $\mathbf{x}$ and the line segment $\overline{\mathbf{p}\,\mathbf{q}}$. The positive value r_o controls the area of impact of the deformation. The main idea to get $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ is to construct a divergence-free time-dependent vector field $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t)$ and obtain $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ as the result of a path line integration of $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$. To do so, we define the auxiliary vector field

$$\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t) = \gamma(\alpha\,\mathbf{a} + \beta\,\mathbf{b}) \tag{8}$$

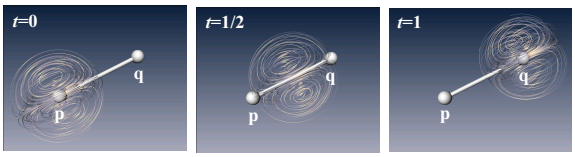

Figure 2: $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t)$ *with* $t = 0, \frac{1}{2}, 1$ *for* $r_o = \frac{1}{2}\|\mathbf{q}-\mathbf{p}\|$.

with

$$\mathbf{a} = \mathbf{q}-\mathbf{p} \quad , \quad \mathbf{m} = (1-t)\,\mathbf{p} + t\,\mathbf{q} \quad , \quad \mathbf{b} = \mathbf{x}-\mathbf{m} \tag{9}$$

$$\alpha = r_o^2 - 5\,\mathbf{b}^2 \quad , \quad \beta = 4\,\mathbf{a}\,\mathbf{b} \quad , \quad \gamma = \frac{\left(r_o^2-\mathbf{b}^2\right)^3}{\left(r_o^2\right)^4}.$$

Figure 1b illustrates this. Then it is a straightforward exercise in algebra to show that $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$ has the following properties:

$$\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}((1-t)\,\mathbf{p} + t\,\mathbf{q}\,,\,t) = \mathbf{q}-\mathbf{p} \tag{10}$$

$$\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \mathbf{0}_3 \quad \text{for} \quad r_o^2 = \mathbf{b}^2 \tag{11}$$

$$\nabla\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \mathbf{0}_{3,3} \;,\; \nabla\nabla\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \mathbf{0}_{3,3,3} \quad \text{for} \quad r_o^2 = \mathbf{b}^2 \tag{12}$$

$$\text{div}(\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}) \equiv 0 \tag{13}$$

where $\mathbf{0}_3$, $\mathbf{0}_{3,3}$, $\mathbf{0}_{3,3,3}$ are the 3D zero tensors of order $1,2,3$, respectively. (11) and (12) mean that for $r_o^2 = \mathbf{b}^2$, $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$ and its first and second order partials vanish. Now we can define $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ as

$$\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t) = \begin{cases} \mathbf{w}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},t) & \text{for} \quad \mathbf{b}^2 \le r_o^2 \\ \mathbf{0}_3 & \text{else} \end{cases} \tag{14}$$

and $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ as the result of a path line integration over the time interval $[0,1]$:

$$\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x}) = \mathbf{x} + \int_0^1 \mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x}(s),s)\,ds. \tag{15}$$

Then (11), (12) and (14) give that $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ is C^2 continuous, which implies (5). (10) yields (3). (4) follows from the piecewise definition of $\mathbf{v}_{\mathbf{p},\mathbf{q},r_0}$ in (14). (6) and (7) follow directly from (13) [vFTS06]. Note that $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ is a piecewise polynomial vector field of degree 8. Figure 2 illustrates $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},0)$, $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},\frac{1}{2})$, $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}(\mathbf{x},1)$ for $r_o = \frac{1}{2}\|\mathbf{q}-\mathbf{p}\|$ using illuminated stream lines [ZSH96]. Movie 2 illustrates $\mathbf{d}_{\mathbf{p},\mathbf{q},r_o}$ as path line integration of $\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$.

Comparison with [vFTS06]:
The idea of defining volume-preserving deformations by integrating divergence-free vector fields was recently introduced in [vFTS06]. The framework presented there was used for the definition of interactive primary deformations (see section 2 for a short description). Note that the vector field $\mathbf{v}$ constructed by (1)– (2) as used in [vFTS06] is a piecewise C^1 continuous vector field of degree 16 (if r describes the the Euclidean distance to a center point). The vector field

$\mathbf{v}_{\mathbf{p},\mathbf{q},r_o}$ defined by (8)–(14) appears to be significantly simpler and smoother: C^2 continuous and of degree 8. This enhancement is achieved by letting the inner ring of the deformation collapse to a point, making the expensive (in terms of polynomial degree) C^1 joint between inner and intermediate region unnecessary. In fact, $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$ was constructed similar to [vFTS06] as

$$r(\mathbf{x}) = (\mathbf{x}-\mathbf{m})^2 \quad , \quad r_i = 0 \tag{16}$$

and $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o} = \|\mathbf{q}-\mathbf{p}\| \cdot (\nabla p \times \nabla q)$ with $p = (1-b)\cdot e + b\cdot 0$ and $q = (1-b)\cdot f + b\cdot 0$ where e, f are linear scalar fields with $(\nabla e)^2 = (\nabla f)^2 = 1, \nabla e \cdot \nabla f = \nabla e \cdot \mathbf{a} = \nabla f \cdot \mathbf{a} = 0$, and $e(\mathbf{m}) = f(\mathbf{m}) = 0$. Since this way the inner region vanishes, the blending function can be simplified to

$$b(r) = \sum_{i=0}^{2} w_i \, B_i^2\left(\frac{r-r_i}{r_o-r_i}\right). \tag{17}$$

with $(w_0,\ldots,w_2) = (0,1,1)$. This explains the reduction in the polynomial degree in $\mathbf{w}_{\mathbf{p},\mathbf{q},r_o}$. To explain the improved continuity, we note that $\mathbf{v}$ constructed by (1)–(2) is actually C^1 at the joint between inner and intermediate region, and it is C^2 between intermediate and outer region. Since we make the inner region disappear, the remaining global continuity is C^2.

3.3. Composition of vector field integration

In general, more than one mass-spring set is independently used to get the desired elastic deformation. Since any linear combination or time-concatenation of divergence-free vector fields is divergence-free as well, there are two simple options to compose two or more mass-spring sets. Let two mass spring sets be given by $\mathbf{p}_1, \mathbf{q}_1, r_{o1}$ and $\mathbf{p}_2, \mathbf{q}_2, r_{o2}$ respectively. Then we use the following compositions:

1. Adding the vector fields: the deformation is obtained by a path line integration of $\mathbf{v}(\mathbf{x},t) = \mathbf{v}_{\mathbf{p}_1,\mathbf{q}_1,r_{o1}} + \mathbf{v}_{\mathbf{p}_2,\mathbf{q}_2,r_{o2}}$ over the time interval $[0,1]$.
2. Concatenating the vector field: here
$\mathbf{v}(\mathbf{x},t) = \begin{cases} \mathbf{v}_{\mathbf{p}_1,\mathbf{q}_1,r_{o1}}(\mathbf{x},t) & \text{for} \quad 0 \le t < 1 \\ \mathbf{v}_{\mathbf{p}_2,\mathbf{q}_2,r_{o2}}(\mathbf{x},t-1) & \text{for} \quad 1 \le t \le 2 \end{cases}$
is integrated over the time interval $[0,2]$.

Note that concatenating the vector fields corresponds to a concatenation of $\mathbf{d}_{\mathbf{p}_i,\mathbf{q}_i,r_{oi}}$: the integration of $\mathbf{v}(\mathbf{x},t)$ leads to $\mathbf{d}_{\mathbf{p}_2,\mathbf{q}_2,r_{o2}}(\mathbf{d}_{\mathbf{p}_1,\mathbf{q}_1,r_{o1}}(\mathbf{x}))$.

In order to deform smaller details of the shape correctly, they are deformed by first using an addition of vector fields with small influences. Then the shape is further deformed by concatenating an addition of vector fields with larger influences. Figure 3 illustrates this and Section 5.1 gives more details about our implementation of this approach. Movie 3 (part 3) shows the composition of deformations coming from 7 mass-spring sets for the cow model.

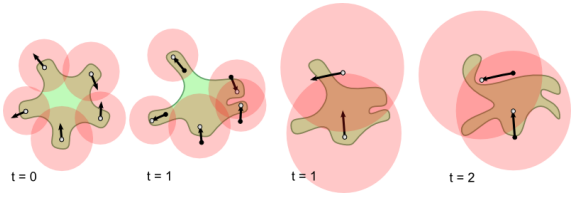

Figure 3: *For $0 \le t < 1$, small details are deformed using an addition of vector fields. Afterward, for $1 \le t \le 2$, larger areas are deformed using vector fields with larger influence.*

3.4. Deformation skinning

Being a space deformation, the described deformation technique may not produce desirable results if independent parts of the shape are spatially close to each other. For instance, suppose a mass-spring set is placed in the right foot of the camel model (figure 4 left). When the spring is elongated, the left leg can enter the influence region and will be deformed as well. In order to prevent such situations, we need a mechanism to constrain the deformation to a specific segment of the shape. We use an extension to a standard technique called matrix palette skinning (indexed skinning), which is both GPU-friendly and straightforward to implement [LKM01]. It works by assigning a set of usually four index-weight pairs to each vertex. Each index points to an element of a matrix palette, which is an array of 4x4 matrices defining affine transformations. A vertex is deformed by computing its transformations for all four indices and computing the weighted sum of them. Usually, an animated character is segmented into its different body parts, and a transformation is assigned to each segment. The weight for this transformation is 1 for most vertices of the segment – only at the joints, where two or more segments meet, the weights are chosen such that a smooth blending between the different transformations is achieved. Usually the weights are specified together with a skeleton (which defines the affine transformations of the segments) by an animator in a 3D authoring tool. We extend this by additionally assigning a list of mass-spring sets to each segment. Each list of mass-spring sets defines a composed vector field. This means that instead of a palette of matrices, we have a palette of matrices plus vector fields. The extended skinning algorithm works basically the same as before: the vertex is deformed by transforming it using matrix multiplication and by afterward deforming it using vector field integration, for each of the four palette indices. The resulting vertex is again the weighted sum of these four deformed positions. Section 5.2 describes the implementation in more detail. Figure 4 and movie 4 show how a mass-spring set deforms a leg of the camel model without affecting other body parts by using deformation skinning. Note that this surface-based skinning does not prevent global self-intersections and does not exactly preserve volume at the parts where different deforma-

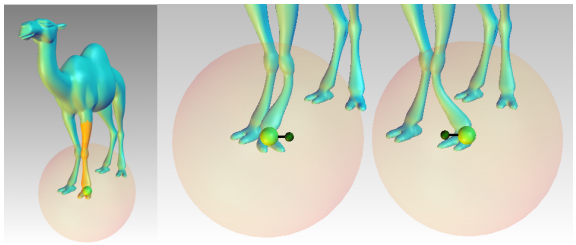

Figure 4: *Using deformation skinning, the deformation described by the mass-spring set can be constrained to one leg of the camel.*

tions are blended. However, global self-intersections can be avoided by performing an additional collision detection between shape segments. The areas where multiple deformations are blended (e.g. at the joints of a character) are usually small, so the effect of non-exact volume-preservation is negligible.

3.5. Setting the mass-spring sets

The plausibility of our approach strongly depends on number, location and parametrization of the mass-spring sets. Since it turns out that a rather low number of mass-spring sets already gives pleasing results and that the location of "good" sets follows the shape intuitively, we left the placing of the mass-spring sets to the user. However, in our experiments we realized a number of rules of thumbs to get pleasing secondary deformations.

Mass-spring sets should be placed close to the center of large homogeneous areas. If only one mass-spring set is used, it should be placed close to the center of gravity of the object. Furthermore, the area of influence should approximately reflect the size of the object. In addition, further mass-spring sets with smaller area of influence can be placed into distant parts of the shape like head, legs or ears. Furthermore, $m_{\mathbf{q}}, k, c$ have to be set for each mass-spring set. Throughout this paper we have chosen $m_{\mathbf{q}}$ proportional to the area of influence while k is constant for all springs. That way, mass-spring sets with larger influence move more slowly than others, resulting in a composition of motions with different frequencies. Movie 3 (part 1) shows how 7 mass-spring sets are set and parametrized for the cow model.

3.6. Level of detail

Motivated by the fact that mass-spring sets which are far away from the viewer and/or have a small influence radius are visually not significant, we can take advantage of a simple level of detail technique in order to increase performance. Let $\mathbf{p}$ be the anchor point and r_o be the radius of influence of a mass-spring set. Given the distance d between $\mathbf{p}$ and the viewer, we use this mass-spring for deformation

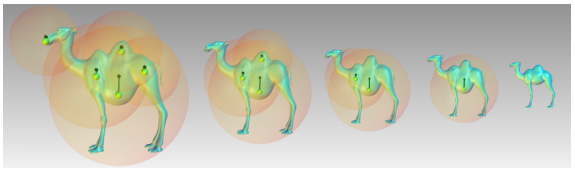

Figure 5: *Level of detail: with increasing distance, more and more mass-spring sets can be omitted.*

only if $r_o > t \cdot d$, where t is a user defined threshold. That way, only mass-spring sets that are close enough and whose influence is large enough are considered during deformation. Figure 5 illustrates this.

4. Collision handling

Realistic secondary deformations should be able to react properly on collisions of the shape coming from the primary animation. In this section we show that our approach does so by simply concatenating an additional vector field to the integration. This enables us to combine our technique with a rigid body simulation. The idea is to allow objects to penetrate during the rigid body simulation and adding a repelling force which is proportional to the depth of penetration. This way we can simulate elastic collisions. Movie 5 (part 1) shows an example. In order to remove the penetration, we deform the soft object by integrating a new concatenated vector field $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$. Movie 5 (part 2) illustrates this.

4.1. Definition of $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}$

The construction of $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}$ follows [vFTS06] in the following way: a center point $\mathbf{c}$ should be translated along a direction vector $\mathbf{d}$ to $\mathbf{c}+\mathbf{d}$ by integrating over a time interval of 1; r_i and r_o denote the inner and outer radius of the deformation. We apply (1)–(2) with the choices $r(\mathbf{x},t) = \| \mathbf{c}+t\,\mathbf{d}-\mathbf{x} \|$ and $e(\mathbf{x},t)$, $f(\mathbf{x},t)$ are linear scalar fields with $(\nabla e)^2 = (\nabla f)^2 = 1$, $\nabla e \cdot \nabla f = \nabla e \cdot \mathbf{d} = \nabla f \cdot \mathbf{d} = 0$, and $e(\mathbf{c}+t\,\mathbf{d},t) = f(\mathbf{c}+t\,\mathbf{d},t) = 0$. Then we get $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t) = \|\mathbf{d}\| \cdot (\nabla p(\mathbf{x},t) \times \nabla q(\mathbf{x},t))$.

4.2. Placing the vector field

After defining $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$, we need to place it such that it deforms the shape realistically under collision. More precisely, we have to choose $\mathbf{c}$, $\mathbf{d}$, r_i, r_o such that interpenetrations between the elastic body and the collision geometry are canceled. We assume that the collision geometry (i.e. the objects that the body can collide with) is decomposed into convex hulls. While automatic methods exist [LA04], we decomposed our scenes manually. As described in Section 3.4, the mesh of the elastic object may be decomposed

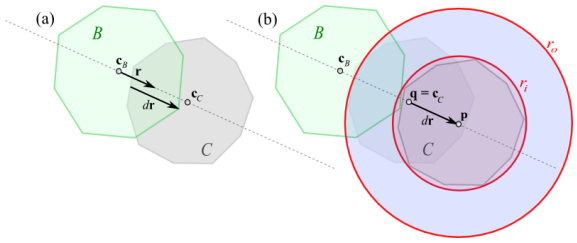

Figure 6: *(a) Body B penetrates collision geometry C with penetration depth d. (b) The vector field is placed by setting parameters* $\mathbf{p}, \mathbf{q}, r_i, r_o$ *appropriately.*

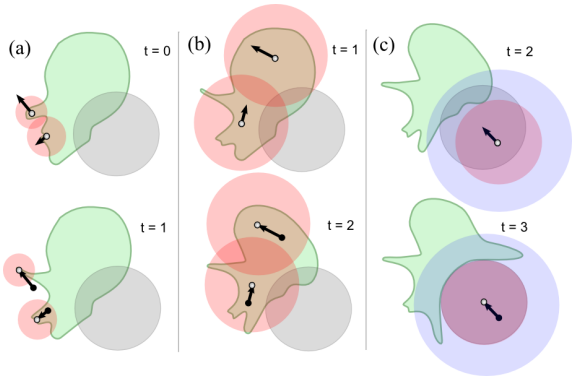

Figure 7: *(a) A body (green) is deformed locally using* $\mathbf{v}_{local}$*. (b) Afterward, it is deformed more globally by* $\mathbf{v}_{global}$ *which is defined by mass-spring sets with larger influence. (c) Finally the penetration into the collision object (grey) is reversed using* $\mathbf{v}_{collision}$*.*

into different segments. From now on, we treat each segment independently as an *elastic body*. Let B be an elastic body penetrating a convex part C of the collision geometry. Furthermore let $\mathbf{c}_B$ be the center of gravity of B and $\mathbf{c}_C$ be the center of gravity of C. Then, for r_i we choose the maximum distance between $\mathbf{c}_C$ and all points in C, i.e. $r_i = \max\{\|\mathbf{x} - \mathbf{c}_C\| : \mathbf{x} \in C\}$. That way, we make sure that C lies completely in the inner region with the result – as we will see later – that no point of B ever enters C. The parameter r_o basically determines the (visual) softness of the material: the smaller r_o, the softer appears the material. We found that setting $r_o - r_i$ proportional to the size of the body gives pleasing results. To do that, we use the maximum distance of all points of B from $\mathbf{c}_B$ as a measure for the size of B and compute $r_o = r_i + s \cdot \max\{\|\mathbf{x} - \mathbf{c}_B\| : \mathbf{x} \in B\}$, where s is a user defined softness factor. In our tests we used values between 1 and 2. Of course, r_i and r_o can be precomputed for better performance.

To simplify matters, we allow only translations along the vector $\mathbf{c}_B - \mathbf{c}_C$. In order to determine $\mathbf{c}$ and $\mathbf{d}$, we first compute the maximum penetration depth d of B in C along the vector $\mathbf{r} = \frac{\mathbf{c}_C - \mathbf{c}_B}{\|\mathbf{c}_C - \mathbf{c}_B\|}$. Figure 6 illustrates this. Then we can set $\mathbf{c} = \mathbf{c}_C + d\mathbf{r}$
and $\mathbf{d} = \mathbf{c}_C - \mathbf{c}$. If we now center C at $\mathbf{c}$ instead of $\mathbf{c}_C$, C and B would touch each other but not interpenetrate. Furthermore, since all points of C are completely in the inner region, where we have a constant vector field, no point of B can ever enter C during integration. Altogether, the deformation defined by $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$ cancels the penetration of B into C by pushing the penetrating parts of B out of C.

In the case of multiple collisions, for each convex collision geometry C that is penetrated by B, we compute the corresponding vector field $\mathbf{u}_{\mathbf{p},\mathbf{q},r_i,r_o}(\mathbf{x},t)$ as described above. These fields are summed up and finally concatenated with the vector field computed from the mass-spring sets (Section 3.2).

5. Implementation

The presented deformation technique is totally independent of any mesh connectivity information or control lattices like grids or volumetric meshes. It is based on an integration of vector fields defined by a relatively small number of mass-spring sets. This fact allows for a direct GPU implementation of the shape deformation. While the simulation of the mass-spring movements is still carried out on the CPU, the resulting spring positions are sent to the GPU which is responsible for deforming and rendering the mesh. Since the deformation always starts from the original, undeformed mesh, the original mesh can be stored as a static vertex buffer (together with buffers for normals, indices, weights etc.) on the GPU, i.e. the vertex positions don't have to be updated. Practically this means that the mesh is deformed by rendering it. This is a contrast to the GPU implementation of [vFTS06], which relies on a read-back of vertex positions from the GPU, which is a performance bottleneck.

5.1. Composition

As mentioned in Section 3.3, we can compose vector fields in two ways: either by addition or by concatenation. In order to get the best results, we use a combination of both. We construct three vector fields $\mathbf{v}_{local}, \mathbf{v}_{global}, \mathbf{v}_{collision}$ and concatenate them. $\mathbf{v}_{local}$ is the summation of all vector fields defined by mass-spring sets whose radius is smaller than some user defined threshold. This means that $\mathbf{v}_{local}$ gives a rather local deformation. $\mathbf{v}_{global}$ is a summation of all vector fields defined by mass-spring sets whose radius is larger than the threshold. That way, it deforms the shape more globally. Finally $\mathbf{v}_{collision}$ is constructed as described in Section 4 and concatenated to the other fields. Figure 7 illustrates this.

5.2. Skinning and integration

As explained in Section 3.4, we combine the vector field integration with matrix palette skinning in order to constrain

the deformation to specific segments and to be able to emulate elasticity for animated objects and characters. A vertex $\mathbf{x}$ is deformed using the following formula:

$$\mathbf{x}' = \sum_{k=1}^{4} w_k \mathbf{d}_{i_k}(\mathbf{M}_{i_k}\mathbf{x}). \tag{18}$$

Here, i_k, w_k are the index-weight pairs for this vertex, $\mathbf{M}_{i_k}$ is the i_kth matrix from the matrix palette, and $\mathbf{d}_{i_k}(.)$ is the i_kth deformation from the vector field palette.

The algorithm can be optimized by skipping index-weight pairs whose weight is zero. The numerical integration needed for $\mathbf{d}_{i_k}$ is carried out using a standard Euler integration of the composed vector fields from Section 5.1. It turns out that even a small number of integration steps gives pleasing results. In our implementation we used twelve steps.

5.3. Normal computation

In order to get a correct lighting of the deformed shape, we need to compute the deformed normals as well. Since we want to avoid read-backs from the GPU, we cannot employ standard methods considering for instance the 1-ring neighborhood of a vertex. Our solution works as follows: Instead of deforming only the vertex $\mathbf{x}$, two additional points $\mathbf{x}_1, \mathbf{x}_2$ in the tangent plane of $\mathbf{x}$ and close to $\mathbf{x}$ are deformed using the same indices and weights as $\mathbf{x}$. Then the deformed normal can be computed as the normalized of $(\mathbf{x}_2' - \mathbf{x}') \times (\mathbf{x}_1' - \mathbf{x}')$, where $\mathbf{x}', \mathbf{x}_1', \mathbf{x}_2'$ are the deformed vertices of $\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2$ respectively.

5.4. GPU implementation

The necessary parameters like spring positions, elongations and bone matrices are passed to the vertex shader as uniform variables. As mentioned above, the vertices, indices, weights, normals etc. are stored as static buffers in video memory. In order to prevent vertices from being deformed multiple times, we exploit the vertex cache by computing cache-optimized triangle strips of our models. For that purpose, we used the NvTriStrip library (*http://developer.nvidia.com/object/nvtristrip_library.html*).

6. Applications

In this section we apply our technique to different kinds of primary animations.

6.1. Keyframe animation

In interactive applications like games, character animation is usually performed via skeletal animation. That means that for every part of the body a rigid transformation is defined to get different poses of the character. Being rigid, these transformations cannot describe elastic secondary deformations resulting from jiggling muscles or fat. In order to demonstrate our method, we have built a simple keyframe animation system based on a linear interpolation between different character poses. Figure 8 and movie 6 show the animation of a boxer with and without our secondary deformations. Here, the animator has placed 13 masses with different influences in the character. During the animation, the corresponding mass-spring sets are simulated based on the movement of the body parts. Then the shape is deformed and rendered by the GPU (see Section 5). As the comparison in the video shows, the secondary deformations introduced by vector field integration enhance the visual appearance and make the animation more lifelike.

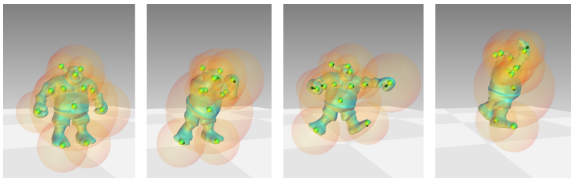

Figure 8: *Skeletal animations become more lifelike by adding secondary animations of muscles and fat.*

6.2. Interactive moving and deformation

Even simple operations like moving or rotating objects as well as applying standard deformation techniques can be visually enhanced by adding elastic secondary deformations to the object. In Figure 9 and movie 7, the cow model is deformed and moved interactively. In addition, the mass-spring sets which are placed throughout the body are simulated based on the resulting motion. By choosing different influence radii, masses and spring parameters, realistic looking motions of fat and muscle can be emulated. Usually, mass-spring sets with low influence and mass are placed at small details (like the ears of the cow model) to get fast vibrations there, while large and plump regions (like the belly of the cow model) are deformed by mass-spring sets with large mass and influence.

6.3. Rigid body simulation

In order to get realistically moving and colliding elastic bodies, we apply our secondary deformation to rigid body simulation methods. That means that an object is represented by one or more rigid bodies which are interconnected by joints. These rigid bodies are simulated using a standard rigid body dynamics system. For our implementation, we used the *Newton Game Dynamics* (*http://www.newtondynamics.com*) library. Using deformation skinning (Section 3.4), we get a ragdoll simulation of the object. In order to make the object appear more elastic, each rigid body is rotated towards its rest pose during the simulation. This basically means that the bodies are interconnected by elastic joints. Furthermore,

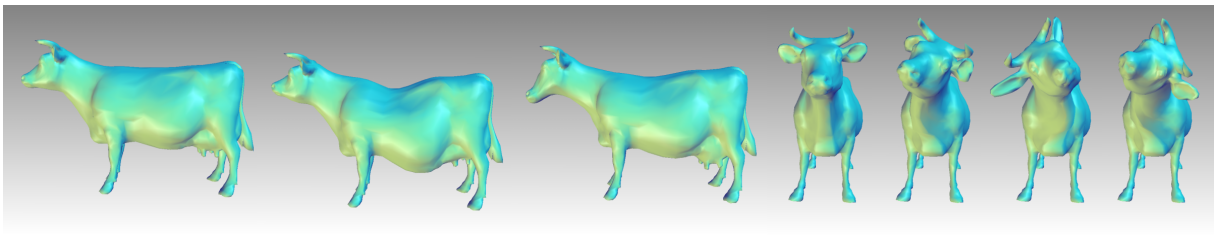

Figure 9: *During interactive movement and deformation of the cow model, different parts of the body jiggle elastically with different frequencies.*

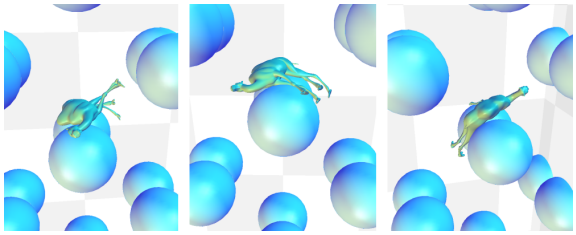

Figure 10: *A camel ragdoll falls through a tower with obstacles.*

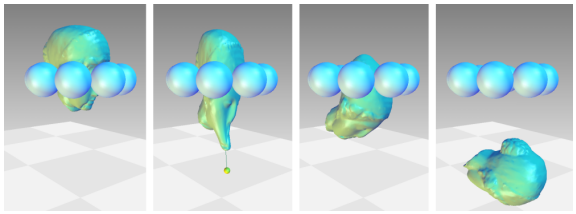

Figure 11: *While the model is pulled through the ring, its volume is preserved and intersections with the ring geometry are avoided.*

we have to allow the object to penetrate the collision environment, such that we can apply the collision handling algorithm from Section 4. During penetration, we apply an impulse to the object which is proportional to the penetration depth. That way, we get an elastic collision.

Figure 10 and movie 8.1 show a scene of a camel model falling through a scene composed of walls and bars. The model is composed of 7 different rigid bodies for legs, body, neck and head, and 5 mass-spring sets are distributed in the model. In the scene shown in Figure 11 and movie 8.2, a head model simulated by one rigid body and five mass-spring sets is dragged through a ring. Here we can see how the volume of the shape is preserved and intersections with the collision geometry are avoided. Figure 12 shows how the shape is deformed by multiple draggers. Each dragger contributes a new vector field to the integration.

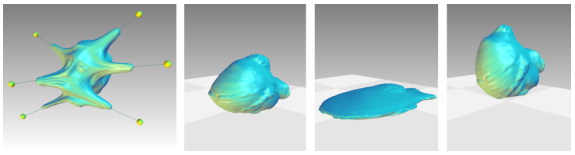

Figure 12: *Using vector field integration, multiple draggers can be used to deform the model. Even after extreme deformations, the model returns to its original shape.*

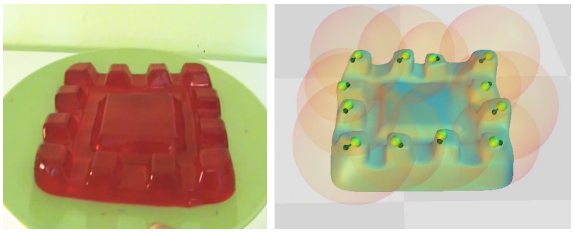

Figure 13: *In order to evaluate the visual plausibility of our approach, we compared a real jelly (left) with a virtual jelly whose secondary motion is emulated by our system (right).*

7. Evaluation and Comparison

In order to get a formal evaluation of the approach, physically exact ground truth deformations of real models are necessary. Since our approach is heuristic in nature, we prefer to use the visual plausibility as one parameter of our evaluation. The examples in the figures and the movie show that the secondary deformations look rather realistic even though only a low number of steering parameters are used. In order to get a comparison between a real elastic incompressible material and our method, we recorded a video of a jelly. In Figure 13 and movie 9.1 we see a real jelly which is shaked in order to obtain inertial motion. As a result, the small knobs on the jelly jiggle with different frequencies. We modeled this jelly using our approach, which is shown in Figure 13 and movie 9.2. Here we placed mass-spring sets with different masses and influence radii in the knobs, which took us less than one minute. When the virtual jelly is shaked in our system, the resulting deformation looks quite similar to the real one.

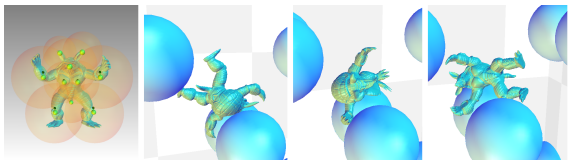

Figure 14: *Also high-resolution models consisting of several hundred thousands of triangles can be deformed in real-time.*

Another point of evaluation is the choice and parametrization of the mass-spring sets. Without doing a formal user study, we presented the system to different people (mainly students). It turned out that even when using the system for the first time, it took them only a few minutes of "playing around" with the mass-spring sets to get realistic secondary deformations similar to the ones shown in the figures.

Furthermore, a number of "hard" evaluations are possible concerning the following properties: due to its nature, our deformations are volume preserving, C^2 continuous for mass-spring simulations (but only C^1 if collision treatment is involved), and without any self-intersections. We believe that these conditions are strong enough to produce visually plausible deformations even though no explicit physical model is involved. Thanks to the avoidance of complex physical simulations, the method is stable. Figure 12 shows a shape that undergoes an extreme deformation during collision and still returns to its rest state afterward.

Due to the fast vector field integration and the GPU implementation, the algorithm allows to emulate high resolution models at interactive rates. For instance, the model shown in figure 14 and movie 8.4, consisting of 345,944 triangles, 7 segments and 10 mass-spring sets, can be deformed and simulated at 10-14 frames per second. The following table shows detailed benchmark results for different scenes, measured on an 2.6 GHz CPU with a GeForce 7800 GTX graphics card.

Fig.	Movie	*#t*	*#s*	*#m*	*fps*
9	7	5,804	9	8	374 – 376
8	6	15,596	8	13	133 – 135
10	8.1	19,536	7	5	101 – 125
11	8.2	16,532	1	4	55 – 135
9	8.3	5,804	9	8	302 – 355
14	8.4	345,944	7	10	10 – 14

Here, *#t* is the number of triangles, *#s* the number of skinning segments, *#m* the number of mass-spring sets and *fps* the number of frames per second (worst and best).

Comparison to existing approaches.
In order to evaluate our contributions, we point out the most important differences to previous approaches of deformable object modeling.

In contrast to mass-spring systems, our mass-spring sets are not interconnected. This makes the simulation straightforward to implement, intuitive, fast and stable. Furthermore, since the actual deformation is performed by an integration of divergence-free vector fields, the volume of the shape is preserved and a low number of mass-spring sets suffices to give plausible deformations.

While Finite Difference Methods, Finite Element Methods and Finite Volume Methods require control structures like grids or volumetric meshes, our approach is mesh-free. This allows for a fast and intuitive placement of mass-spring sets and requires no preprocessing.

Existing mesh-free methods like point based animation usually require a larger amount of particles in the shape than our approach needs mass-spring sets. This is because of the fact that the underlying physical simulation requires a certain particle density in order to be accurate. In contrast to this, our method gives plausible results even for a small number of mass-spring sets, which is due to its inherent volume-preserving nature.

The geometrically motivated approach by [MHTG05] resembles our approach in the sense that it is mesh-free, requires a small number of particles and exchanges physical accuracy for interactivity and stability. However, this approach does not preserve the volume of the shape, which is especially noticeable for large deformations. While this approach needs to embed the shape into regularly placed cubical regions to get more detailed deformations, our method requires mass-spring sets with appropriate influence radii and positions.

Limitations.
Like most related approaches, the presented work has some limitations. We did not consider collisions between multiple elastic bodies. Here, a convex decomposition of the colliding shapes and a similar algorithm as presented in Section 4 might work out. Furthermore, the vector fields described by the mass-spring sets perform only translations in their inner regions. That way, bending or twisting deformations are not directly possible. Here, rotational fields as presented in [vFTS06] might be a solution. Also a resampling of the deformed mesh as used in [vFTS06] is not applicable to our approach, because it is GPU based without readbacks. Although highly controllable compared to related approaches, steering the deformation exactly is difficult due to the simple, implicitly defined influence of the mass-spring sets and their corresponding vector fields. Using deformation skinning, prevention of global self-intersections and exact preservation of volume are not guaranteed, as depicted in Figure 4. However, we believe that deformation skinning adds to the visual plausiblity and furthermore integrates well with existing skeleton-based animation frameworks.

8. Conclusions

In this paper, we made the following contributions:

- We introduced the construction and integration of

divergence-free vector fields to get elastic secondary deformations.
- In comparison to [vFTS06], we enhanced the used vector fields both in polynomial degree and continuity.
- Contrary to [vFTS06], the GPU implementation does not perform any read-back operations during the integration. This allows real-time deformations even for fairly large shapes.
- We have shown that vector field integration can also be used to avoid unwanted intersections and penetrations of soft objects in rigid body deformations. It turns out that the combination of rigid body simulation as primary animation and our secondary deformation based on vector field integration gives a realistic emulation of elastically deforming models. Furthermore, it can be built on top of an existing rigid body system, which makes the implementation more unified and robust.

Our deformations are stable, smooth, and, without deformation skinning, volume preserving and free of self-intersections. They allow to apply LOD approaches. Furthermore, they can be used on top of arbitrary primary animations.

9. Acknowledgments

This research has partially been funded by the Max Planck Center for Visual Computing and Communication (MPC-VCC).

References

[CBC*05] CAPELL S., BURKHART M., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Physically based rigging for deformable characters. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), ACM Press, pp. 301–310. 2

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 586–593. 2

[CHP89] CHADWICK J. E., HAUMANN D. R., PARENT R. E.: Layered construction for deformable animated characters. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1989), ACM Press, pp. 243–252. 2

[CK05] CHOI M. G., KO H.-S.: Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics 11*, 1 (2005), 91–101. 2

[DC96] DESBRUN M., CANI M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)* (Aug 1996), Boulic R., Hegron G., (Eds.), Springer-Verlag, pp. 61–76. Published under the name Marie-Paule Gascuel. 2

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic real-time deformations using space & time adaptive sampling. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM Press, pp. 31–36. 2

[DSB99] DESBRUN M., SCHRÖDER P., BARR A.: Interactive animation of structured deformable objects. In *Proceedings of the 1999 conference on Graphics interface '99* (San Francisco, CA, USA, 1999), Morgan Kaufmann Publishers Inc., pp. 1–8. 2

[GEW05] GEORGII J., ECHTLER F., WESTERMANN R.: Interactive simulation of deformable bodies on gpus. In *Simulation and Visualisation 2005* (2005). 2

[GKS02] GRINSPUN E., KRYSL P., SCHRÖDER P.: Charms: a simple framework for adaptive simulation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 281–290. 2

[JP99] JAMES D. L., PAI D. K.: Artdefo: accurate real time deformable objects. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 65–72. 2

[JP02] JAMES D. L., PAI D. K.: Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 582–585. 2

[LA04] LIEN J.-M., AMATO N. M.: Approximate convex decomposition. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry* (New York, NY, USA, 2004), ACM Press, pp. 457–458. 5

[LKM01] LINDHOLM E., KLIGARD M. J., MORETON H.: A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM Press, pp. 149–158. 4

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), ACM Press, pp. 49–54. 2

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *GI '04: Proceedings of the 2004 conference on Graphics interface* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), Canadian Human-Computer Communications Society, pp. 239–246. 2

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM Press, pp. 471–478. 2, 9

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2004), ACM Press, pp. 141–151. 2

[NMK*06] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4 (December 2006), 809–836. 3

[NT98] NEDEL L. P., THALMANN D.: Real time muscle deformations using mass-spring systems. In *CGI '98: Proceedings of the Computer Graphics International 1998* (Washington, DC, USA, 1998), IEEE Computer Society, p. 156. 2

[OBH02] O'BRIEN J. F., BARGTEIL A. W., HODGINS J. K.: Graphical modeling and animation of ductile fracture. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 291–294. 2

[OZH00] O'BRIEN J., ZORDAN V., HODGINS J.: Combining active and passive motions for secondary motion. *IEEE Computer Graphics and Applications 1*, 1 (2000), 86–96. 1

[PB81] PLATT S. M., BADLER N. I.: Animating facial expressions. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1981), ACM Press, pp. 245–252. 2

[PH06] PARK S., HODGINS J.: Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics (SIGGRAPH 2006) 25*, 3 (Aug. 2006). 1

[TBHF03] TERAN J., BLEMKER S., HING V. N. T., FEDKIW R.: Finite volume methods for the simulation of skeletal muscle. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 68–74. 2

[Ton98] TONNESEN D. L.: *Dynamically coupled particle systems for geometric modeling, reconstruction, and animation*. PhD thesis, 1998. Adviser-Demetri Terzopoulos. 2

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 205–214. 2

[TW90] TERZOPOULOS D., WATERS K.: Physically-based facial modeling, analysis and animation. *Journal of Visualization and Computer Animation 1*, 1 (1990), 73–80. 2

[TW91] TERZOPOULOS D., WATERS K.: Modeling animated faces using scanned data. *Journal of Visualization and Computer Animation 2*, 2 (1991), 123–128. 2

[vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM Press, pp. 1118–1125. 2, 3, 4, 5, 6, 9, 10

[ZG05] ZHENG GUO K. C. W.: Skinning with deformable chunks. *Computer Graphics Forum 24*, 3 (2005), 373–382. 2

[ZSH96] ZÖCKLER M., STALLING D., HEGE H.: Interactive visualization of 3D-vector fields using illuminated stream lines. In *Proc. IEEE Visualization '96* (1996), pp. 107–113. 3