DOI: 10.1111/cgf.12346

COMPUTER GRAPHICS *forum*
Volume 00 (2014), number 0 pp. 1–24

A Survey on Position-Based Simulation Methods in Computer Graphics

Jan Bender[1], Matthias Müller[2], Miguel A. Otaduy[3], Matthias Teschner[4] and Miles Macklin[2]

[1]Graduate School CE, TU Darmstadt, Darmstadt, Germany
bender@gsc.tu-darmstadt.de
[2]NVIDIA PhysX Research, Zurich, Switzerland
{matthiasm, mmacklin}@nvidia.com
[3]URJC Madrid, Madrid, Spain
maotaduy@gmail.com
[4]Department of Computer Science, University of Freiburg, Freiburg, Germany
teschner@informatik.uni-freiburg.de

Abstract
The dynamic simulation of mechanical effects has a long history in computer graphics. The classical methods in this field discretize Newton's second law in a variety of Lagrangian or Eulerian ways, and formulate forces appropriate for each mechanical effect: joints for rigid bodies; stretching, shearing or bending for deformable bodies and pressure, or viscosity for fluids, to mention just a few. In the last years, the class of position-based methods has become popular in the graphics community. These kinds of methods are fast, stable and controllable which make them well-suited for use in interactive environments. Position-based methods are not as accurate as force-based methods in general but they provide visual plausibility. Therefore, the main application areas of these approaches are virtual reality, computer games and special effects in movies. This state-of-the-art report covers the large variety of position-based methods that were developed in the field of physically based simulation. We will introduce the concept of position-based dynamics, present dynamic simulation based on shape matching and discuss data-driven upsampling approaches. Furthermore, we will present several applications for these methods.

Keywords: position-based simulation, shape matching, data-driven upsampling, deformable solids, fluids

ACM CCS: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Animation

1. Introduction

The simulation of solid objects such as rigid bodies, soft bodies or cloth has been an important and active research topic in computer graphics for more than 30 years. The field was introduced to graphics by Terzopoulos and his colleagues in the late 1980s [TPBF87a]. Since then, a large body of work has been published and the list is growing rapidly. There exists a variety of survey papers [GM97], [MTV05], [NMK*06], [MSJT08], [BETC12] which document this development. However, due to the shear number of papers, it has become difficult to cover the entire field in one survey.

In this state-of-the-art report, we focus on a special class of simulation methods, namely position-based approaches. These methods were originally developed for the simulation of solids. However, some recent works demonstrated that the position-based concepts can even be used to simulate fluids. Classical dynamics simulation methods formulate the change of momentum of a system as a function of applied forces, and evolve positions through numerical integration of accelerations and velocities. Position-based approaches, instead, compute positions directly, based on the solution to a quasi-static problem.

Physical simulation is a well-studied problem in the computational sciences and therefore, many of the well-established methods have been adopted in graphics such as the finite element method (FEM) [OH99], the finite differences method [TPBF87b], the finite volume method [TBHF03], the boundary element method [JP99] or particle-based approaches [DSB99], [THMG04]. The main goal of computer simulations in computational physics and chemistry is to replace real-world experiments and thus, to be as accurate as possible. In contrast, the main applications of physically based simulation methods in computer graphics are special effects in movies and commercials and more recently, computer games and other interactive systems. Here, speed and controllability are the most important factors and all that is required in terms of accuracy is visual plausibility. This is especially true for real-time applications.

Position-based methods are tailored particularly for use in interactive environments. They provide a high level of control and are stable even when simple and fast explicit time integration schemes are used. Due to their simplicity, robustness and speed, these approaches have recently become very popular in computer graphics and in the game industry.

In this state-of-the-art report, we focus the discussion on several geometrically motivated methods, in particular those that directly manipulate the positions of the simulated bodies. We start with a description of position-based dynamics methods, which compute equilibrium positions by iteratively resolving geometric constraints. Then, we cover shape matching methods, and we conclude with data-driven upsampling methods, which compute positions as a function of data from pre-captured deformation examples.

Collision detection is an important part of any simulation system. However, an adequate discussion of this topic is beyond the scope of this report. Therefore, we refer the reader to the surveys of Lin and Gottschalk [LG98] and the one of Teschner *et al.* [TKH*05].

2. Background

The most popular approaches for the simulation of dynamic systems in computer graphics are force based. Internal and external forces are accumulated from which accelerations are computed based on Newton's second law of motion. A time integration method is then used to update the velocities and finally the positions of the object. A few simulation methods (most rigid body simulators) use impulse-based dynamics and directly manipulate velocities. In contrast, geometry-based methods omit the velocity layer as well and immediately work on the positions. The main advantage of a position-based approach is its controllability. Overshooting problems of explicit integration schemes in force-based systems can be avoided. In addition, collision constraints can be handled easily and penetrations can be resolved completely by projecting points to valid locations.

Among the force-based approaches, one of the simplest methods is to represent and simulate solids with mass-spring networks. A mass-spring system consists of a set of point masses that are connected by springs. The physics of such a system is straightforward and a simulator is easy to implement. However, there are some significant drawbacks of the simple method.

- The behaviour of the object depends on the way the spring network is set up.
- It can be difficult to tune the spring constants to get the desired behaviour.
- Mass-spring networks cannot capture volumetric effects directly such as volume conservation or prevention of volume inversions.

The FEM solves all of the above problems because it considers the entire volume of a solid instead of replacing it with a finite number of point masses. Here, the object is discretized by splitting the volume into a number of elements with finite size. This discretization yields a mesh as in the mass-spring approach in which the vertices play the role of the mass points and the elements, typically tetrahedra, can be viewed as generalized springs acting on multiple points at the same time. In both cases, forces at the mass points or mesh vertices are computed due to their velocities and the actual deformation of the mesh.

3. Position-Based Dynamics

In this section, we present position-based dynamics, an approach which omits the velocity and acceleration layer and immediately works on the positions [MHHR07]. We will first describe the basic idea and the simulation algorithm of position-based dynamics (PBD). Then, we will focus specifically on how to solve the system of constraints that describe the object to be simulated. After this, we will provide a list of constraints that can be used to simulate a variety of materials such as soft bodies, cloth or even fluids with PBD. We will also discuss the relation of PBD to strain limiting methods, how to include damping and ways to parallelize the simulation algorithm.

3.1. Overview

The objects to be simulated are represented by a set of N particles and a set of M constraints. Each particle i has three attributes, namely

m_i	Mass
$\mathbf{x}_i$	Position
$\mathbf{v}_i$	Velocity

A constraint j is defined by the five attributes

n_j	Cardinality
$C_j : \mathbb{R}^{3n_j} \to \mathbb{R}$	Scalar constraint function
$\{i_1, \dots i_{n_j}\}, i_k \in [1, \dots N]$	Set of indices
$k_j \in [0 \dots 1]$	Stiffness parameter
unilateral or *bilateral*	Type

Constraint j with type *bilateral* is satisfied if $C_j(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_{n_j}}) = 0$. If its type is *unilateral* then it is satisfied if $C_j(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_{n_j}}) \geq 0$. The stiffness parameter k_j defines the strength of the constraint in a range from zero to one.

Algorithm 1 Position-based dynamics

```
1:  for all vertices $i$ do
2:      initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
3:  end for
4:  loop
5:      for all vertices $i$ do $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{\text{ext}}(\mathbf{x}_i)$
6:      for all vertices $i$ do $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
7:      for all vertices $i$ do genCollConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
8:      loop solverIteration times
9:          projectConstraints($C_1, \dots, C_{M+M_{\text{Coll}}}, \mathbf{p}_1, \dots, \mathbf{p}_N$)
10:     end loop
11:     for all vertices $i$ do
12:         $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
13:         $\mathbf{x}_i \leftarrow \mathbf{p}_i$
14:     end for
15:     velocityUpdate($\mathbf{v}_1, \dots, \mathbf{v}_N$)
16: end loop
```

Given these data and a time step Δt, the simulation proceeds as described by Algorithm 1. Since the algorithm simulates a system which is second order in time, both the positions and the velocities of the particles need to be specified in (1)–(3) before the simulation loop starts. Lines (5)–(6) perform a simple explicit forward Euler integration step on the velocities and the positions. The new locations $\mathbf{p}_i$ are not assigned to the positions directly but are only used as predictions. Non-permanent external constraints such as collision constraints are generated at the beginning of each time step from scratch in line (7). Here, the original and the predicted positions are used in order to perform continuous collision detection. The solver (8)–(10) then iteratively corrects the predicted positions such that they satisfy the M_{Coll} external as well as the M internal constraints. Finally, the corrected positions $\mathbf{p}_i$ are used to update the positions and the velocities. It is essential here to update the velocities along with the positions. If this is not done, the simulation does not produce the correct behaviour of a second-order system. As you can see, the integration scheme used here is very similar to the Verlet method. It is also closely related to Jos Stam's *Nucleus* solver [Sta09] which also uses a set of constraints to describe the objects to be simulated. The main difference is that *Nucleus* solves the constraints for velocities, not positions.

3.2. The System to be Solved

The goal of the solver steps (8)–(10) is to correct the predicted positions $\mathbf{p}_i$ of the particles such that they satisfy all constraints. The problem that needs to be solved comprises of a set of M equations for the $3N$ unknown position components, where M is now the total number of constraints. This system does not need to be symmetric. If $M > 3N$ ($M < 3N$) the system is overdetermined (underdetermined). In addition to the asymmetry, the equations are in general non-linear. The function of a simple distance constraint $C(\mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2|^2 - d^2$ yields a non-linear equation. What complicates things even further is the fact that collisions produce inequalities rather than equalities. Solving a non-symmetric, non-linear system with equalities and inequalities is a tough problem.

Let $\mathbf{p}$ be the concatenation $[\mathbf{p}_1^T, \dots, \mathbf{p}_N^T]^T$ and let all the constraint functions C_j take the concatenated vector $\mathbf{p}$ as input while only using the subset of coordinates they are defined for. We can now write the system to be solved as

$$\begin{aligned} C_1(\mathbf{p}) &\succ 0 \\ &\dots \\ C_M(\mathbf{p}) &\succ 0, \end{aligned}$$

where the symbol $\succ$ denotes either $=$ or $\geq$. Newton–Raphson iteration is a method to solve non-linear symmetric systems with equalities only. The process starts with a first guess of a solution. Each constraint function is then linearized in the neighbourhood of the current solution using

$$C(\mathbf{p} + \Delta\mathbf{p}) = C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} + O(|\Delta\mathbf{p}|^2) = 0.$$

This yields a *linear* system for the global correction vector $\Delta\mathbf{p}$

$$\begin{aligned} \nabla_{\mathbf{p}} C_1(\mathbf{p}) \cdot \Delta\mathbf{p} &= -C_1(\mathbf{p}) \\ &\dots \\ \nabla_{\mathbf{p}} C_M(\mathbf{p}) \cdot \Delta\mathbf{p} &= -C_M(\mathbf{p}), \end{aligned}$$

where $\nabla_{\mathbf{p}} C_j(\mathbf{p})$ is the $1 \times N$ dimensional vector containing the derivatives of the function C_j with respect to all its parameters, i.e. the N components of $\mathbf{p}$. It is also the jth row of the linear system. Both, the rows $\nabla_{\mathbf{p}} C_j(\mathbf{p})$ and the right-hand side scalars $-C_j(\mathbf{p})$ are constant because they are *evaluated* at the location $\mathbf{p}$ before the system is solved. When $M = 3N$ and only equalities are present, the system can be solved by any linear solver, e.g. a preconditioned conjugate gradient method. Once it is solved for $\Delta\mathbf{p}$ the current solution is updated as $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. A new linear system is generated by evaluating $\nabla_{\mathbf{p}} C_j(\mathbf{p})$ and $-C_j(\mathbf{p})$ at the new location after which the process repeats.

If $M \neq 3N$ the resulting matrix of the linear system is non-symmetric and not invertible. Goldenthal *et al.* [GHF*07] solve this problem by using the pseudo-inverse of the system matrix which yields the best solution in the least-squares sense. Still, handling inequalities is not possible directly.

3.3. The Non-Linear Gauss–Seidel Solver

In the position-based dynamics approach, non-linear Gauss–Seidel is used. It solves each constraint equation separately. Each constraint yields a single scalar equation $C(\mathbf{p}) \succ 0$ for all the particle positions associated with it. The subsystem is therefore highly underdetermined. PBD solves this problem as follows. Again, given $\mathbf{p}$ we want to find a correction $\Delta\mathbf{p}$ such that $C(\mathbf{p} + \Delta\mathbf{p}) \succ 0$. It is important to notice that PBD also linearizes the constraint function but individually for each constraint. The constraint equation is approximated by

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} \succ 0. \tag{1}$$

The problem of the system being underdetermined is solved by restricting $\Delta\mathbf{p}$ to be in the direction of $\nabla_{\mathbf{p}} C$ which conserves the linear and angular momenta. This means that only one scalar λ—a Lagrange multiplier—has to be found such that the correction

$$\Delta\mathbf{p} = \lambda \nabla_{\mathbf{p}} C(\mathbf{p})$$

solves Equation (1). This yields the following formula for the correction vector of a single particle i

$$\Delta\mathbf{p}_i = -s\, w_i \nabla_{\mathbf{p}_i} C(\mathbf{p}), \tag{2}$$

where

$$s = \frac{C(\mathbf{p})}{\sum_j w_j |\nabla_{\mathbf{p}_j} C(\mathbf{p})|^2} \tag{3}$$

and $w_i = 1/m_i$.

As mentioned above, this solver linearizes the constraint functions. However, in contrast to the Newton–Raphson method, the linearization happens individually *per constraint*. Solving the linearized constraint function of a single distance constraint for instance yields the correct result in a single step. Because the positions are immediately updated after a constraint is processed, these updates will influence the linearization of the next constraint because the linearization depends on the actual positions. Asymmetry does not pose a problem because each constraint produces one scalar equation for one unknown Lagrange multiplier λ. Inequalities are handled trivially by first checking whether $C(\mathbf{p}) \geq 0$. If this is the case, the constraint is simply skipped.

The fact that each constraint is linearized individually before its projection makes the solver more stable than a global approach in which the linearizations are kept fixed for the entire global solve of a Newton iteration. Liu *et al.* [LBOK13] pointed out that PBD can be interpreted as a heuristic variant of the variational implicit Euler method taking the inertial term out of the solver and into the integration step of the simulation.

We have not considered the stiffness k of the constraint so far. There are several ways to incorporate it. The simplest variant is to multiply the corrections $\Delta\mathbf{p}$ by $k \in [0\ldots1]$. However, for multiple iteration loops of the solver, the effect of k is non-linear. The remaining error for a single distance constraint after n_s solver iterations is $\Delta\mathbf{p}(1-k)^{n_s}$. To get a linear relationship, we multiply the corrections not by k directly but by $k' = 1-(1-k)^{1/n_s}$. With this transformation, the error becomes $\Delta\mathbf{p}(1-k')^{n_s} = \Delta\mathbf{p}(1-k)$ and, thus, becomes linearly dependent on k and independent of n_s as desired. However, the resulting material stiffness is still dependent on the time step of the simulation. Real-time environments typically use fixed time steps in which case this dependency is not problematic.

3.4. Constraint Examples

In the following, we will introduce different constraint examples. For better readability, we define $\mathbf{p}_{i,j} = \mathbf{p}_i - \mathbf{p}_j$.

3.4.1. *Stretching*

To give an example, let us consider the distance constraint function $C(\mathbf{p}_1,\mathbf{p}_2) = |\mathbf{p}_{1,2}| - d$. The derivatives with respect to the points are $\nabla_{\mathbf{p}_1} C(\mathbf{p}_1,\mathbf{p}_2) = \mathbf{n}$ and $\nabla_{\mathbf{p}_2} C(\mathbf{p}_1,\mathbf{p}_2) = -\mathbf{n}$ with $\mathbf{n} = \frac{\mathbf{p}_{1,2}}{|\mathbf{p}_{1,2}|}$. The scaling factor s is, thus, $s = \frac{|\mathbf{p}_{1,2}|-d}{1+1}$ and the final corrections

$$\begin{aligned}\Delta\mathbf{p}_1 &= -\tfrac{w_1}{w_1+w_2}(|\mathbf{p}_{1,2}|-d)\tfrac{\mathbf{p}_{1,2}}{|\mathbf{p}_{1,2}|}\\ \Delta\mathbf{p}_2 &= +\tfrac{w_2}{w_1+w_2}(|\mathbf{p}_{1,2}|-d)\tfrac{\mathbf{p}_{1,2}}{|\mathbf{p}_{1,2}|},\end{aligned}$$

which are the formulas proposed in [Jak01] for the projection of distance constraints (see Figure 1). They pop up as a special case of the general constraint projection method.

3.4.2. *Bending*

In cloth simulation, it is important to simulate bending in addition to stretching resistance. To this end, for each pair of adjacent triangles

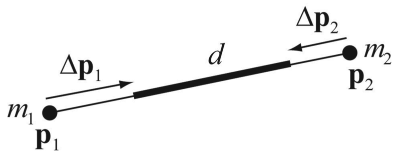

Figure 1: *Projection of the constraint $C(\mathbf{p}_1,\mathbf{p}_2) = |\mathbf{p}_{1,2}| - d$. The corrections $\Delta\mathbf{p}_i$ are weighted according to the inverse masses $w_i = 1/m_i$.*

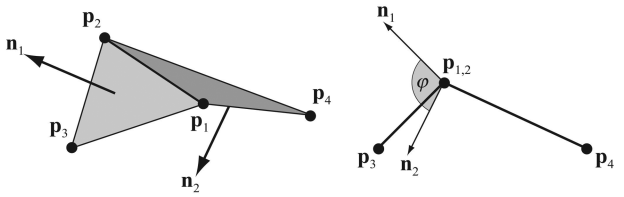

Figure 2: *For bending resistance, the constraint function $C(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4) = \arccos(\mathbf{n}_1\cdot\mathbf{n}_2) - \varphi_0$ is used. The actual dihedral angle φ is measured as the angle between the normals of the two triangles.*

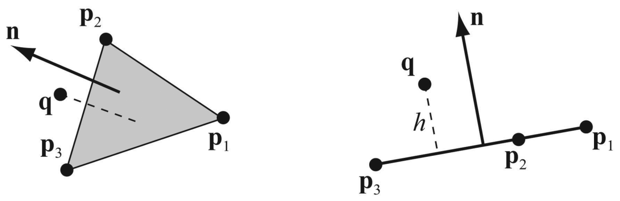

Figure 3: *Constraint function $C(\mathbf{q},\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3) = (\mathbf{q}-\mathbf{p}_1)\cdot\mathbf{n} - h$ makes sure that $\mathbf{q}$ stays above the triangle $\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$ by the cloth thickness h.*

$(\mathbf{p}_1,\mathbf{p}_3,\mathbf{p}_2)$ and $(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_4)$ a bilateral bending constraint is added with constraint function

$$C_{\text{bend}}(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4) = \\ \operatorname{acos}\left(\frac{\mathbf{p}_{2,1}\times\mathbf{p}_{3,1}}{|\mathbf{p}_{2,1}\times\mathbf{p}_{3,1}|}\cdot\frac{\mathbf{p}_{2,1}\times\mathbf{p}_{4,1}}{|\mathbf{p}_{2,1}\times\mathbf{p}_{4,1}|}\right) - \varphi_0$$

and stiffness k_{bend}. The scalar φ_0 is the initial dihedral angle between the two triangles and k_{bend} is a global user parameter defining the bending stiffness of the cloth (see Figure 2). The advantage of this bending term over adding a distance constraint between points $\mathbf{p}_3$ and $\mathbf{p}_4$ is that it is *independent of stretching*. This is because the term is independent of edge lengths. In Figure 4 we show how bending and stretching resistance can be tuned independently.

3.4.3. *Triangle collisions*

Self-collisions within cloth can be handled by additional unilateral constraints. For vertex $\mathbf{q}$ moving through a triangle $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, the constraint function reads

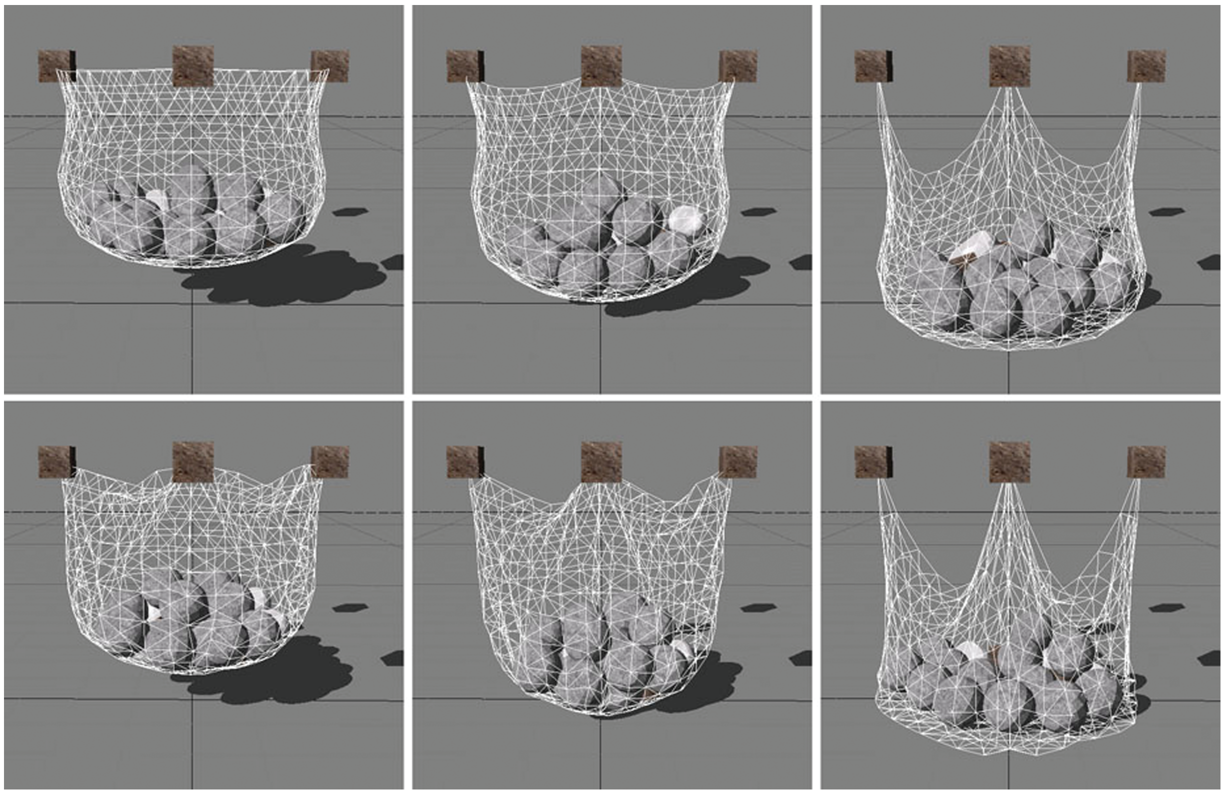

Figure 4: *The image shows a mesh that is simulated using stretching and bending constraints. The top row shows* $(k_{stretching}, k_{bending}) = (1, 1)$, $(\frac{1}{2}, 1)$ *and* $(\frac{1}{100}, 1)$. *The bottom row shows* $(k_{stretching}, k_{bending}) = (1, 0)$, $(\frac{1}{2}, 0)$ *and* $(\frac{1}{100}, 0)$.

$$C(\mathbf{q}, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = (\mathbf{q} - \mathbf{p}_1) \cdot \frac{\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}}{|\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}|} - h,$$

where h is the cloth thickness (see Figure 3). If the vertex enters from below with respect to the triangle normal, the constraint function has to be

$$C(\mathbf{q}, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = (\mathbf{q} - \mathbf{p}_1) \cdot \frac{\mathbf{p}_{3,1} \times \mathbf{p}_{2,1}}{|\mathbf{p}_{3,1} \times \mathbf{p}_{2,1}|} - h.$$

3.4.4. *Volume conservation*

For tetrahedral meshes, it is useful to have a constraint that conserves the volume of single tetrahedron. Such a constraint has the form

$$C(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) = \frac{1}{6}\left(\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}\right) \cdot \mathbf{p}_{4,1} - V_0,$$

where $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_4$ are the four corners of the tetrahedron and V_0 is its rest volume. In a similar way, the area of a triangle can be kept constant by introducing

$$C(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = \frac{1}{2}\left|\mathbf{p}_{2,1} \times \mathbf{p}_{3,1}\right| - A_0.$$

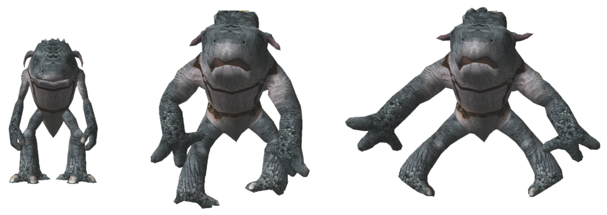

Figure 5: *Simulation of overpressure inside a character.*

3.4.5. *Cloth balloons*

For closed triangle meshes, overpressure inside the mesh as shown in Figure 5 can easily be modelled with an *equality* constraint concerning all N vertices of the mesh:

$$C(\mathbf{p}_1, \ldots, \mathbf{p}_N) = \left(\sum_{i=1}^{n_{triangles}} (\mathbf{p}_{t_1^i} \times \mathbf{p}_{t_2^i}) \cdot \mathbf{p}_{t_3^i}\right) - k_{\text{pressure}} V_0.$$

Here, t_1^i, t_2^i and t_3^i are the three indices of the vertices belonging to triangle i. The sum computes the actual volume of the closed mesh.

It is compared against the original volume V_0 times the overpressure factor k_{pressure}. This constraint function yields the gradients

$$\nabla_{\mathbf{p}_i} C = \sum_{j:t_1^j=i} (\mathbf{p}_{t_2^j} \times \mathbf{p}_{t_3^j}) + \sum_{j:t_2^j=i} (\mathbf{p}_{t_3^j} \times \mathbf{p}_{t_1^j}) + \sum_{j:t_3^j=i} (\mathbf{p}_{t_1^j} \times \mathbf{p}_{t_2^j}).$$

These gradients have to be scaled by the scaling factor given in Equation (3) and weighted by the masses according to Equation (2) to get the final projection offsets $\Delta\mathbf{p}_i$.

3.4.6. *Fluids*

It is also possible to simulate fluids in the PBD framework even though it has been used almost exclusively for the simulation of deformable objects. We mention fluids simply as an item in the list of possible constraints because all that is needed to simulate liquids and gases is a specialized constraint.

A straightforward approach would be to model the fluid as a system of particles constrained to maintain a minimum distance from each other, however this leads to granular-like behaviour and will typically fail to reach hydrostatic equilibrium when coming to rest. An alternative method is presented by Macklin and Müller [MM13] where fluid incompressibility is enforced using density constraints. Borrowing the concept of a density estimator from smoothed particle hydrodynamics (SPH) [Mon94, Mon92], a density constraint is constructed for each particle i in the system as follows:

$$C_i(\mathbf{p}_1, ..., \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1, \tag{4}$$

where ρ_0 is the fluid rest density and ρ_i is the density at a particle, defined as the sum of smooth kernels [MCG03] centred at the particle's neighbour positions

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h).$$

Note that here each particle's mass is assumed to be one, and the rest density adjusted accordingly. In order to solve these density constraints using position-based dynamics, the derivative of the constraint function (4) with respect to each particle's position is required. This can be calculated using the gradient of SPH kernels

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = i \\ -\nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = j. \end{cases}$$

Then, by taking advantage of symmetry in the SPH smoothing kernel W, the corrective change in position due to the particle's own density constraint, and the density constraints of its neighbours is given by

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0} \sum_j \left(\lambda_i + \lambda_j\right) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h),$$

where λ is the per-constraint scaling factor (see Section 3.3). Figure 6 shows a real-time water simulation using this method.

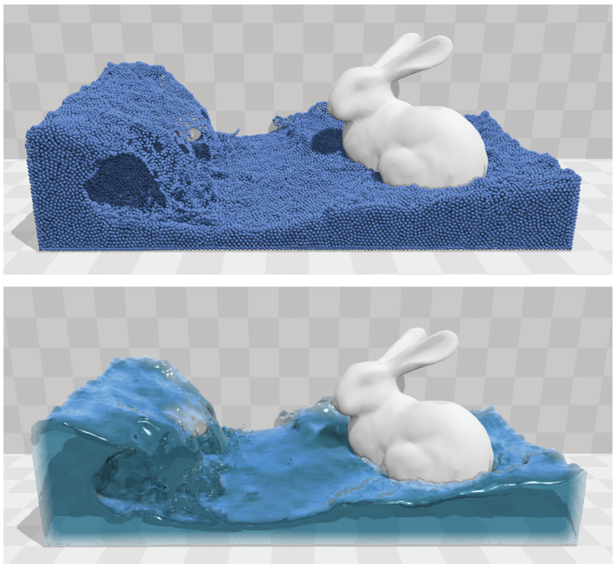

Figure 6: *A wave pool scene consisting of 128K fluid particles simulated in 10 ms/frame on the GPU. Incompressibility is enforced using density constraints solved using position-based dynamics.*

3.5. Strain Limiting

Strain limiting is an important topic in the field of cloth simulation. The reason is that the low solver iteration counts used in real-time applications yield stretchy cloth. Since most cloth types are perceived by the human eye as completely inextensible, it is important to make simulated cloth inextensible in order to avoid disturbing visual artefacts [GHF*07, BB08].

A strain limiting method makes sure that the overall stretch of the cloth stays below a certain threshold. In force-based simulations, strain limiting is a separate pass which is executed before or after the regular cloth solver. In most cases, this pass moves the positions of vertices directly, even in force-based simulations. Therefore, most strain limiting methods fall under the category of position-based methods.

A straightforward way of limiting strain is to iterate through all edges of a cloth mesh and project the adjacent particles of overstretched edges as shown in Figure 1 so that the stretch of the edge does not exceed the stretch limit. Provot [Pro95] was among the first to use this method in the context of cloth simulation. He performs a single iteration through all cloth edges after a force-based solver. Desbrun *et al.* [DSB99] and Bridson *et al.* [BMF03] later used the same post-solver strain limiter but with multiple iterations through all edges. Due to its simplicity, this method is still one of the most popular strain limiting methods used in cloth simulations .

The method is very similar to position-based cloth simulation. The main difference is that the strain limiting pass described above does not influence the velocities. These are updated by the force-based solver. In contrast, position-based cloth simulation derives the new velocities from the projections, making an additional solver pass obsolete. Therefore, every position-based strain limiting method used in force-based simulations can directly be used in a PBD solver.

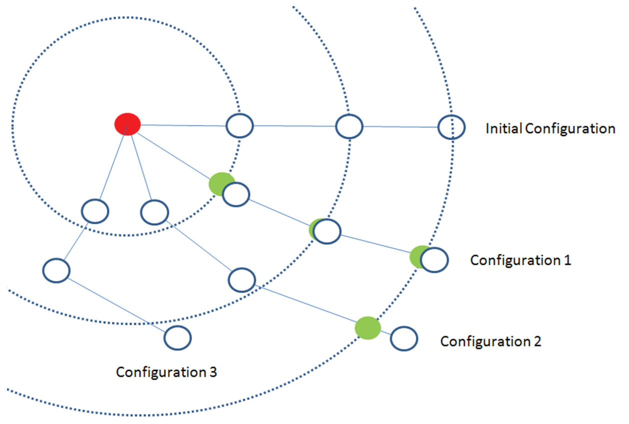

Figure 7: *The Long-Range Attachments (LRA) method used to simulate an inextensible rope attached at one end. Each particle is constrained or remain inside a sphere centred at the attachment point (red) whose radius is the initial distance from the particle to the attachment. For each configuration, target positions are shown in green when particles need to be projected. Particles inside the constraint spheres are allowed to move freely.*

The result of projecting along edges depends on the structure of the mesh. To reduce this artefact, Wang *et al.* [WOR10] propose to limit the principal strains of the 2D deformation field within each triangle. The 2D deformation field can be determined by considering the 2D coordinates of the vertices of a triangle within the planes of the rest and current triangle configurations. Wang *et al.* compute the principal strains of the 2D deformation gradient, clamp them and construct a new 2D transformation using the clamped strains. With this new transformation, they correct the current positions of the triangle vertices. As before, to limit strain globally, they iterate through all triangles multiple times in a Gauss–Seidel fashion.

Due to the relatively slow convergence rate of a Gauss–Seidel solver, high iteration counts are necessary to limit the strain globally which slows down the simulation. The two main methods to improve the convergence rate are the use of a global Newton–Raphson solver as proposed by Goldenthal *et al.* [GHF*07] or to perform Gauss–Seidel iterations on a hierarchy of meshes as proposed in [Mül08], [WOR10] and [SKBK13]. However, these methods complicate the implementation and even though their convergence rate is higher, a single iteration can be significantly more expensive than a simple Gauss–Seidel iteration.

Recently, Kim *et al.* [KCM12] found a surprisingly simple and robust technique they call *Long-Range Attachments* (LRA) to prevent cloth from getting stretched globally with low iteration counts. Their method exploits the fact that stretching artefacts almost always appear when cloth is attached. In this case, instead of only applying attachment constraints to the subset of the vertices near the region where the cloth is attached and relying on error propagation of the solver for all other vertices, they apply unilateral attachment constraints to all the vertices by attaching each vertex to one or more attachment point directly. The rest lengths of these LRA can either be set to the Euclidian distance in the rest state or via measuring geodesic lengths along the cloth. Figure 7 demonstrates the method

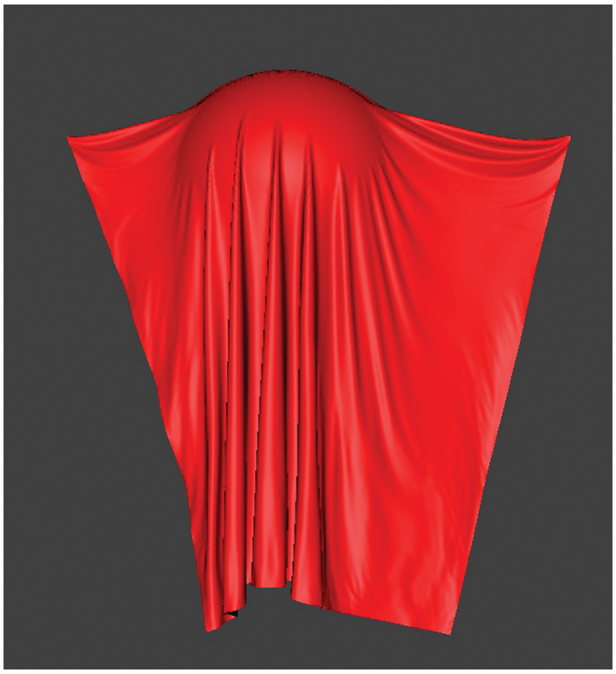

Figure 8: *Simulation of a piece of cloth with 90K vertices at 20 fps on a GPU using LRA.*

on a single rope attached at one end. The method allows the simulation of a piece of cloth with 90K vertices at interactive rates as shown in Figure 8.

A similar approach was recently proposed by Müller *et al.* [MKC12] to guarantee zero stretch in a single pass for the case of attached ropes. This approach allows the simulation of thousands of hair strands in real time (Figure 9). Figure 10 visualizes the basic idea. Particle $\mathbf{x}_1$ is attached. To satisfy the first distance constraint, particle $\mathbf{x}_2$ is moved towards $\mathbf{x}_1$ such that their mutual distance is l_0. Particle $\mathbf{x}_3$ is then moved towards the *new* position of $\mathbf{x}_2$ and similarly along the chain until the last particle is reached. After this single pass, all the distance constraints are satisfied. This method is called *Follow The Leader (FTL)*. While LRA guarantees zero stretch of all the particles with respect to the attachment points, the constraint between consecutive particles can still remain overstretched. On the other hand, in contrast to LRA which is momentum conserving, FTL introduces unphysical behaviour. Not projecting distance constraints symmetrically means that a system is simulated for which each particle has infinitely more mass than its successor. To compensate for this behaviour, the authors replace the PBD velocity update $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$ by

$$\mathbf{v}_i \leftarrow \frac{\mathbf{p}_i - \mathbf{x}_i}{\Delta t} + s_{\text{damping}} \frac{-\mathbf{d}_{i+1}}{\Delta t},$$

where $\mathbf{d}_{i+1}$ is the position correction applied to particle $i+1$ and $s_{\text{damping}} \in [0,1]$ a scaling factor do influence damping. While this modification of DFTL (dynamic FTL) hides the unphysical behaviour of FTL, it introduces a certain amount of damping which is

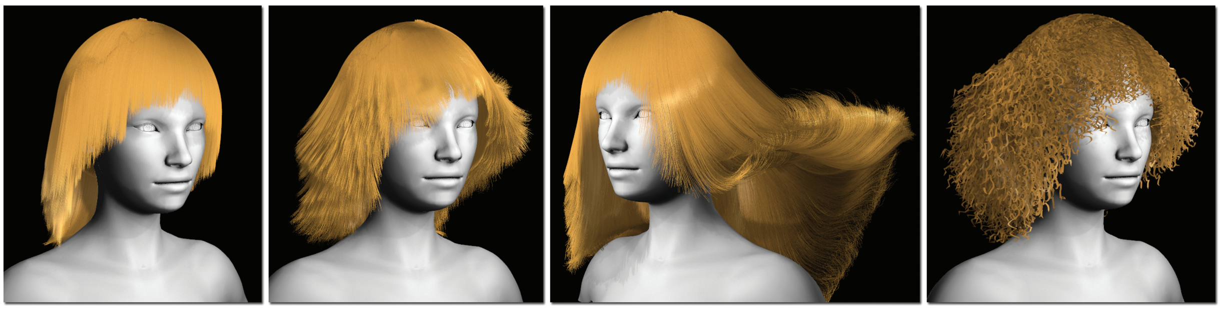

Figure 9: *Dynamic FTL allows the simulation of every hair strand in real time. From left to right: 47K hair strands simulated at 25 fps including rendering and hair–hair repulsion. Long hair composed of 1.9-m particles at 8 fps. Curly hair using visualization post-processing.*

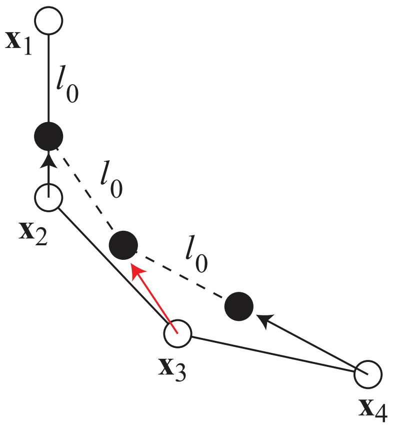

Figure 10: *Follow The Leader (FTL) projection. Starting from the attachment down, each particle is moved directly towards its predecessor such that their mutual distance constraint is satisfied.*

acceptable for the simulation of hair and fur as the author's results show.

3.6. Wrinkle Meshes

In cloth simulations, reducing the mesh resolution not only reduces the cost of a single solver iteration but also the number of iterations required to get visually pleasing results. In [MC10], the authors proposed a way to reduce the resolution of the dynamic mesh without losing too much visual detail. The most significant detail in cloth simulations are small wrinkles. The method is based on the observation that global dynamic behaviour of the cloth and wrinkle formation can be separated. Therefore, expensive dynamic simulation including collision handling is performed on a low-resolution mesh. The wrinkle formation is handled on a high-resolution mesh that is attached to the dynamic mesh (see Figures 11 and 12). Since wrinkles do not oscillate, it is sufficient to use a static solver with a low iteration count on the high-resolution mesh.

Figure 13 shows the constraints defined on the high-resolution mesh to make it form wrinkles and follow the dynamic mesh. The attachment constraints makes sure that the vertices of the wrinkle mesh stay close to their attachment points on the dynamic mesh. If the dynamic mesh has outside/inside information, a one-sided constraint can be used which makes sure that the wrinkle vertices stay on the outside of the dynamic mesh, thus avoiding penetrations with other objects. The stretching and bending constraints are responsible for wrinkle formation.

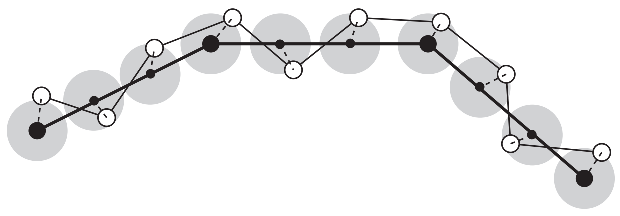

Figure 11: *Basic idea of wrinkle meshes. The high resolution wrinkle mesh (white vertices) follows the low-resolution dynamic mesh (black vertices) by restricting the white vertices to remain within a certain distance (grey discs) to the dynamic mesh.*

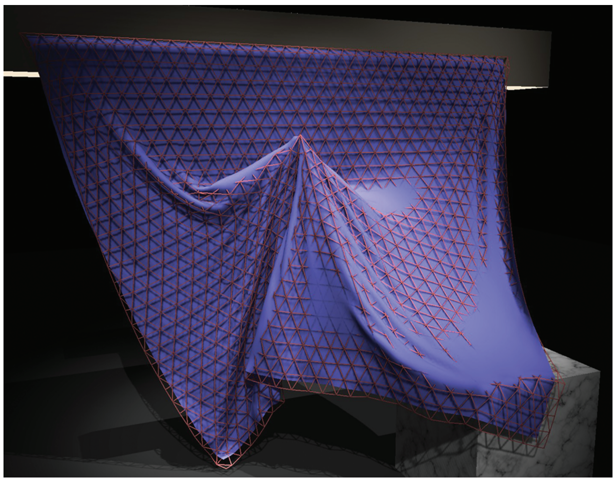

Figure 12: *Visualization of the wrinkle mesh (solid) and the underlying dynamic mesh (wireframe).*

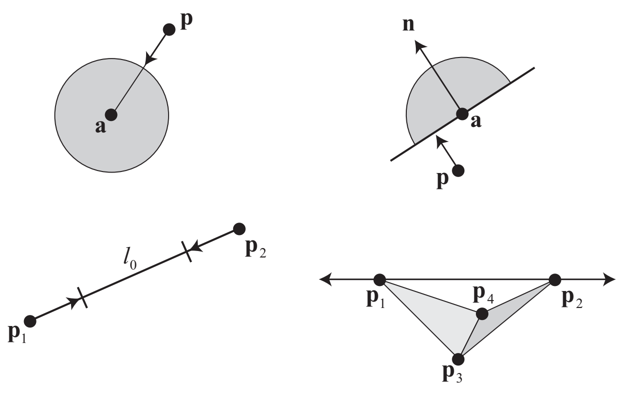

Figure 13: *Static constraints on a wrinkle mesh: Attachment constraint (top left), one-sided attachment constraint (top right), stretch constraint (bottom left) and bending constraint (bottom right).*

3.7. Damping

The quality of dynamic simulations can generally be improved by the incorporation of an appropriate damping scheme. As a positive effect, damping can improve the stability by reducing temporal oscillations of the point positions of an object. This enables the use of larger time steps which increases the performance of a dynamic simulation. On the other hand, damping changes the dynamic motion of the simulated objects. The resulting effects can be either desired, e.g. reduced oscillations of a deformable solid, or disturbing, e.g. changes of the linear or angular momentum of the entire object.

Generally, a damping term $\mathbf{C}\dot{\mathbf{X}}$ can be incorporated into the motion equation of an object where $\dot{\mathbf{X}}$ denotes the vector of all first time derivatives of positions. If the user-defined matrix $\mathbf{C}$ is diagonal, absolute velocities of the points are damped, which sometimes is referred to as point damping. If appropriately computed, such point damping forces result in an improved numerical stability by reducing the acceleration of a point. Such characteristics are desired in some settings, e.g. in the context of friction. In the general case, however, the overall slow-down of an object, caused by point damping forces, is not desired. Point damping forces are, e.g. used in [TF88] or in [PB88], where point damping is used for dynamic simulations with geometric constraints such as point-to-nail.

In order to preserve linear and angular momentum of deformable objects, symmetric damping forces, usually referred to as spring damping forces, can be used. Such forces can be represented by non-diagonal entries in the matrix $\mathbf{C}$. Damping forces are, e.g. described by Baraff and Witkin [BW98] or Nealen *et al.* [NMK*06]. These forces can also be applied to position-based methods. However, as the approaches of Baraff and Witkin and Nealen *et al.* rely on topological information of the object geometry, they cannot be applied to meshless techniques such as shape matching.

Point and spring damping can be used to reduce current velocities or relative velocities. However, it is generally more appropriate to consider predicted velocities or relative velocities for the next time step.

An interesting damping alternative has been presented in [SGT09]. Here, the idea of symmetric, momentum-conserving forces is extended to meshless representations. Global symmetric damping forces are computed with respect to the centre of mass of an object. While such forces conserve the linear momentum, the preservation of the angular momentum is guaranteed by force projection onto relative positions or by torque elimination using Linear Programming. The approach presented in [SGT09] iteratively computes damping forces. The paper, however, also shows the convergence of the iterative process and how the solution can be computed directly without performing iterations. Therefore, the approach is an efficient alternative to compute damping forces for arbitrary position-based deformation models with or without connectivity information. The approach can be used to damp oscillations globally or locally for user-defined clusters.

3.8. Parallelization

Parallelization of the PBD approach is an important topic since multi-core systems and massively parallel GPUs are ubiquitous today.

In a single CPU implementation, the solver processes the constraints one by one in a Gauss–Seidel-type fashion. Thereby, after each constraint projection, the positions of affected particles are immediately updated. In a parallel implementation, the constraints are processed in parallel by multiple threads. If two constraints affecting the same particle are handled by two different threads simultaneously, they are not allowed to immediately update the particle's position because writing to the same position simultaneously leads to race conditions making the process unpredictable. A solution to circumvent this problem is to use atomic operations. Such operations are guaranteed not to be interrupted. However, atomics can slow down parallel execution significantly.

To avoid these issues, a parallel implementation of PBD needs to split the constraints into groups or phases. In each phase, none of the constraints are allowed to share a common particle. With this restriction, the constraints in the first phase can be processed in parallel without conflicts. Then, after a global synchronization, the next phase can be processed. This cycle is repeated until all constraints are processed.

As an example, if N particles are connected in a serial chain, the constraints 1–2, 3–4, 5–6, 7–8, ... can be processed in phase 1 and the constraints 2–3,4–5,6–7, ... in phase 2. This specific example corresponds to the Red–Black Gauss Seidel scheme, where there are two sets (colours) of constraints. For more general types of constraint such as the stretch, shear and bending constraints of cloth, more phases are needed. In this general case, splitting constraints into phases corresponds to the graph colouring problem, where each constraint corresponds to a node of the graph and two constraints are connected by an edge if they affect one or more common particles. The minimum number of colours determines how many phases are needed in the parallel execution of PBD. Keeping the number of phases small is not the only optimization criterion. The sets also need to have similar sizes for good load balancing.

3.9. Discussion

Position-based dynamics is fast, easy to implement and controllable. Furthermore, it avoids the overshooting problems of force-based

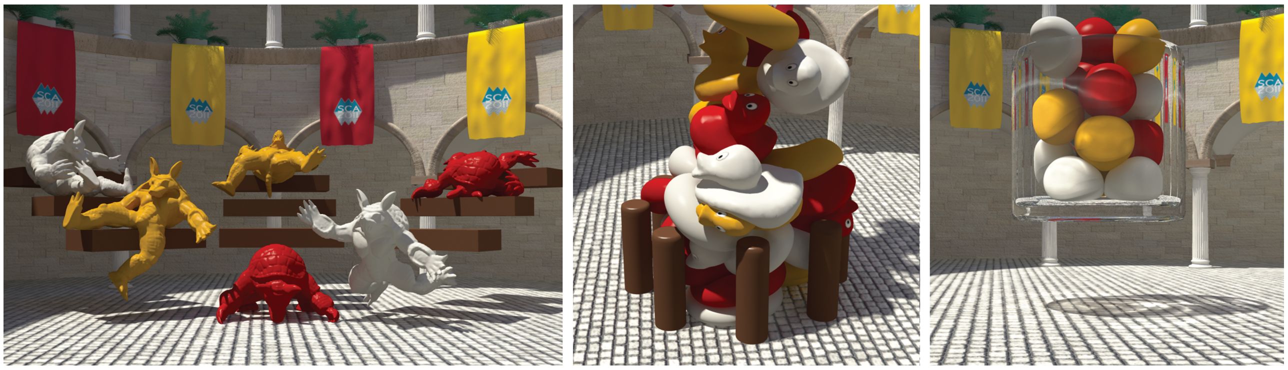

Figure 14: *Robust and volume-conserving deformations using shape matching. Armadillos (32 442 particles total), 20 ducks and 20 tori (21 280 particles total) and 20 balls (7640 particles total) were simulated in real-time on a GPU.*

simulation models when using an explicit time integration scheme. The method can handle arbitrary bilateral and unilateral constraints as long as the gradient of the constraint function can be determined. Therefore, this method is very flexible and has already been used to simulate cloth, deformable solids and fluids.

However, position-based dynamics also has some disadvantages. The stiffness of the model does not only depend on the user-defined stiffness parameter but also on the time step size and the number of solver iterations. Although the dependency can be reduced as described in Section 3.3, it cannot be completely removed. Therefore, it is difficult to adjust parameters independently. Decoupling these parameters as well as adaptive time stepping are open problems and important topics for future work. Another drawback is that position-based dynamics is not convergent, i.e. the simulation does not converge to a certain solution with mesh refinement. Hence, the usage of adaptive meshes is another open problem.

4. Shape Matching

The geometrically motivated concept of shape matching to simulate deformable objects was introduced by Müller *et al.* [MHTG05]. Shape matching is a meshless approach which is able to simulate visually plausible elastic and plastic deformations (see Figure 14). This approach is easy to implement, very efficient and unconditionally stable.

The basic idea of simulating elastic behaviour with shape matching is shown in Figure 15. For the simulation, the initial configuration of the deformable object must be stored. Since no connectivity information is needed, this configuration is defined by the initial positions $\bar{\mathbf{x}}_i$. In each time step, the positions and velocities of the particles are updated without considering any internal constraints between the particles. Only external forces and collision response are taken into account. Instead of using internal constraints, goal positions are determined by matching the initial shape with the deformed configuration. Then, each particle is pulled towards its goal position.

In the following, we first describe how the goal positions are determined. Then we show how large deformations can be simulated using region-based shape matching and introduce fast summation techniques for this approach. In the end, the concept of oriented particles and different extensions of the shape matching method are presented.

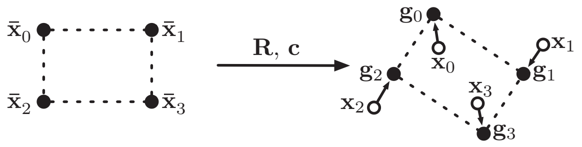

Figure 15: *The initial shape with the vertex positions $\bar{\mathbf{x}}_i$ is matched to the deformed configuration $\mathbf{x}_i$ to obtain goal positions $\mathbf{g}_i$. The deformed shape is pulled towards these goal positions to simulate elastic behaviour.*

4.1. Goal Positions

In order to obtain goal positions for the deformed shape, the best rigid transformation is determined which matches the set of initial positions $\bar{\mathbf{x}}$ and the set of deformed positions $\mathbf{x}$. The corresponding rotation matrix $\mathbf{R}$ and the translational vectors $\mathbf{c}$ and $\bar{\mathbf{c}}$ are determined by minimizing

$$\sum_i w_i \left(\mathbf{R}\left(\bar{\mathbf{x}}_i - \bar{\mathbf{c}}\right) + \mathbf{c} - \mathbf{x}_i\right)^2,$$

where w_i are the weights of the individual points. The optimal translation vectors are given by the centre of mass of the initial shape and the centre of mass of the deformed shape:

$$\bar{\mathbf{c}} = \frac{1}{M}\sum_i m_i \bar{\mathbf{x}}_i, \quad \mathbf{c} = \frac{1}{M}\sum_i m_i \mathbf{x}_i, \quad M = \sum_i m_i. \qquad (5)$$

If we minimize the term $\sum_i (\mathbf{A}\bar{\mathbf{r}}_i - \mathbf{r}_i)^2$ with $\mathbf{r}_i = \mathbf{x}_i - \mathbf{c}$ and $\bar{\mathbf{r}}_i = \bar{\mathbf{x}}_i - \bar{\mathbf{c}}$, we get the optimal linear transformation $\mathbf{A}$

of the initial and the deformed shape. This transformation is determined by

$$\mathbf{A} = \left(\sum_i m_i \mathbf{r}_i \bar{\mathbf{r}}_i^T\right)\left(\sum_i m_i \bar{\mathbf{r}}_i \bar{\mathbf{r}}_i^T\right)^{-1} = \mathbf{A}_r \mathbf{A}_s. \quad (6)$$

In our case, we are only interested in the rotational part of this transformation. Since $\mathbf{A}_s$ is symmetric, it contains no rotation. Therefore, we only need to extract the rotational part of $\mathbf{A}_r$ to get the optimal rotation $\mathbf{R}$ for shape matching. This can be done by a polar decomposition $\mathbf{A}_r = \mathbf{RS}$ of the transformation matrix where $\mathbf{S}$ is a symmetric matrix.

Finally, the goal positions are determined by

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

where $\mathbf{T} = \begin{bmatrix}\mathbf{R} & (\mathbf{c} - \mathbf{R}\bar{\mathbf{c}})\end{bmatrix}$. These goal positions are used to compute position corrections:

$$\Delta\mathbf{x}_i = \alpha\left(\mathbf{g}_i(t) - \mathbf{x}_i(t)\right),$$

where $\alpha \in [0, 1]$ is a user-defined stiffness parameter which defines how far the particles are pulled to their goal positions.

Shape matching can be seen as a form of constraint projection which can directly be integrated in the position-based dynamics algorithm. By performing shape matching in line (9) of Algorithm 1, it can be easily combined with other position-based constraint.

4.2. Region-Based Shape Matching

The shape matching algorithm described above allows only for small deviations from the initial shape. For the simulation of large deformations, the concept of region-based shape matching became popular, see, e.g. [MHTG05, RJ07, DBB11]. The idea is to perform shape matching on several overlapping regions of the original shape. In each region, we can have a small deviation from the corresponding part of the initial shape which results in a large deformation over all regions.

Diziol *et al.* [DBB11] propose to define a region for each particle of the model where the ith region contains all particles in the ω-ring of the ith particle in the original mesh of the model. Shape matching is a meshless method but Diziol *et al.* require a mesh to define the shape matching regions. Rivers and James [RJ07] use a regular lattice instead to define their regions. No matter which kind of regions are used, the stiffness of the model depends on the size of the overlapping regions (see Figure 16). Enlarging the regions results in a more global shape matching and therefore the stiffness of the simulated model is increased.

In region-based shape matching, a particle is part of multiple regions. In the following, we denote the set of regions to which a particle i belongs by $\Re_i$. Since particles can belong to more than one region, Rivers and James [RJ07] proposed to use modified particle masses $\tilde{m}_i = m_i/|\Re_i|$ for shape matching. This ensures that a particle which is part of many regions has not more influence than others. The optimal translation vectors for a region i are determined by

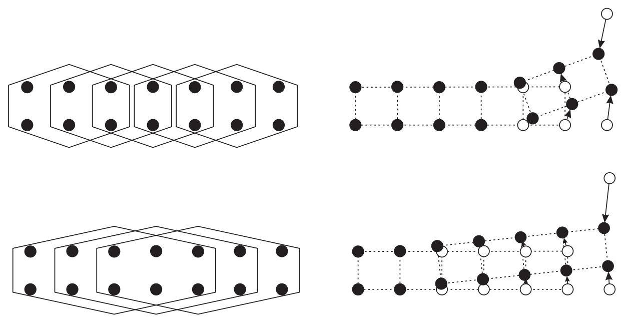

Figure 16: *The stiffness of the model depends on the region size. Smaller regions (top) allow larger deformations than larger regions (bottom). The hexagons in the left images represent the overlapping regions of the model. The right images show the goal positions after one particle is moved away.*

$$\bar{\mathbf{c}}_i = \frac{1}{\tilde{M}_i}\sum_{j\in\Re_i}\tilde{m}_j\bar{\mathbf{x}}_j, \qquad \mathbf{c}_i = \frac{1}{\tilde{M}_i}\sum_{j\in\Re_i}\tilde{m}_j\mathbf{x}_j, \quad (7)$$

where $\tilde{M}_i = \sum_{j\in\Re_i}\tilde{m}_j$ is the effective region mass which can be precomputed. The optimal rotation matrix $\mathbf{R}$ is computed by extracting the rotational part of the following matrix:

$$\mathbf{A}_{r,i} = \sum_{j\in\Re_i}\tilde{m}_j\mathbf{x}_j\bar{\mathbf{x}}_j^T - \tilde{M}_i\mathbf{c}_i\bar{\mathbf{c}}_i^T. \quad (8)$$

In this form, the first term depends on the particles j of the region while the second term depends on the region i. This isolation of the dependencies is required for fast summation techniques (see below).

After performing shape matching for all regions, we get multiple goal positions for each particle. The final goal position for a particle is determined by blending the goal positions of the corresponding regions:

$$\mathbf{g}_i = \frac{1}{|\Re_i|}\sum_{j\in\Re_i}\mathbf{T}_j\begin{bmatrix}\bar{\mathbf{x}}_i \\ 1\end{bmatrix}.$$

4.3. Fast Summation Techniques

In the case of region-based shape matching, the stiffness increases with growing region size ω. However, at the same time the computation of the optimal translation $\mathbf{c}$ and the transformation matrix $\mathbf{A}_r$ becomes a bottleneck since large sums have to be computed for each region. For a mesh with the dimension d and n regions, $O(\omega^d n)$ operations are required with the naive approach.

4.3.1. *Regular lattices*

Rivers and James demonstrated in [RJ07] how the number of operations for computing the sums can be reduced to $O(n)$ for regular lattices ($d = 3$). Their optimization is closely related to the concept of summed-area tables [Cro84]. In their approach, they compute the summation for a set of particles just once and reuse it for all regions that contain this set. This reduces redundant computations significantly for a system with large overlapping regions. The fast summation of Rivers and James is based on the usage of cubical regions. These cubical regions can be subdivided in 2D plate regions which can again be subdivided in one-dimensional bar regions. The region summation is performed in three passes. In the first pass, the sum for each bar is determined. The results are used to compute the sums for the plates which are again used to obtain the final region sum. Each pass requires $O(\omega)$ operations. However, the region sum can even be determined in constant time if we take into account that the sum of two neighbouring bars, plates or cubes only differs by one element. Lattice shape matching can be performed in linear time if the sums in Equations (7) and (8) are evaluated using the fast summation technique described above.

The FastLSM method of Rivers and James has several limitations. To handle regions where the lattice is not regular, e.g. on the boundary, several sums are defined in a pre-processing step for the corresponding node. In the case of fracturing, the definition of these sums must be performed at runtime which is expensive to compute. Small features need a fine sampling to obtain realistic results. Since a regular lattice is used, a fine sampling yields an explosion of the computational costs. FastLSM does not support a varying region size to simulate inhomogeneous material.

4.3.2. *Adaptive lattices*

Steinemann *et al.* [SOG08] introduce an adaptive shape matching method, which is based on lattice shape matching to overcome these limitations. A fast summation is realized by an octree-based sampling and an interval-based definition of the shape matching regions. The hierarchical simulation model is created by starting with a coarse cubic lattice and then performing an octree subdivision. The subdivision process can be controlled by a user-defined criterion. At the end of the process, a simulation node is placed at the centre of each leaf cell and a virtual node at the centre of each non-leaf cell. A virtual node stores the sum of all its descendant simulation nodes.

The fast summation for the hierarchical model is performed by an interval-based method which requires $O(1)$ operations per region. For each simulation node n_i, a shape matching region is defined by a region width ω_i. To perform a fast summation, all summation nodes of the region i are determined in a pre-processing step. First, for each node n_j of the octree the interval of minimum and maximum distances of all descendant leaves of n_j to n_i are determined. Then, during a top–down traversal each node n_j where the maximum distance is smaller than the region width is added to region i. If the descendant leaf nodes are contained only partially in region i, the current node must be refined. Only in this case the traversal continues.

The top–down traversal assigns $O(1)$ summation nodes to each region. A fast summation can now be performed in two steps. In the first step, the sums of all nodes in the hierarchy are determined. This is done by first computing the sums for the simulation nodes which are the leaf nodes of the hierarchy, and then updating the sums of the virtual nodes in a bottom–up fashion. The second step sums up the values of the summation nodes for each region. For a roughly balanced octree, the computation of the sums takes $O(n)$ time where n is the number of simulation nodes. Hence, the adaptive shape matching method requires linear time when using the described fast summation technique to evaluate Equations (7) and (8).

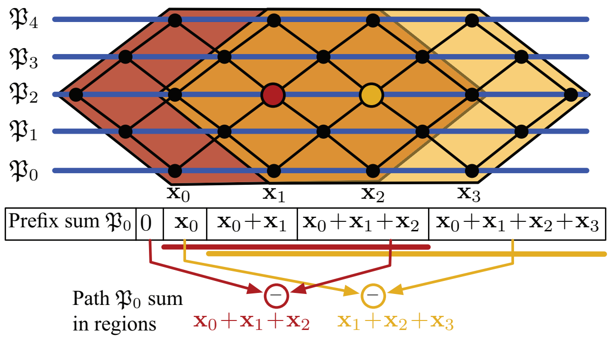

Figure 17: *Fast summation technique for arbitrary triangle meshes [DBB11]. First, the prefix sums for the disjoint paths are determined. Then the region sum is computed by adding the difference of the intersection interval for each path.*

4.3.3. *Triangle meshes*

In contrast to Rivers and James, Diziol *et al.* [DBB11] only use the surface mesh of a volumetric model to simulate its deformation. Therefore, no interior elements are required for the simulation which reduces the computational costs. Diziol *et al.* introduce a fast summation technique for arbitrary triangle meshes ($d = 2$) to compute the large sums of the region-based approach efficiently. This technique only requires $O(\omega n)$ operations instead of $O(\omega^2 n)$ and can be performed very efficiently in parallel.

The fast summation technique of Diziol *et al.* is based on a subdivision of all particles of the mesh in disjoint paths. A path i is a set of vertices $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_n}$ which are connected by edges. The paths are determined in a pre-computation step. The goal of the path construction algorithm is that each region is intersected by a minimum number of paths. To determine the optimal path layout is computationally expensive. Therefore, a heuristic is used to find a good path layout. Starting with a single vertex, adjacent vertices are added to a path until the path length exceeds a maximum size or cannot be extended any further. The heuristic tries to avoid gaps by choosing vertices which have neighbours that are already part of a path. To obtain paths which are as parallel as possible we add the vertex which is closest to a plane passing through the starting vertex of the current path, e.g. the xy-plane.

The fast summation is split in two phases (see Figure 17). In the first phase, the prefix sum for each path i is computed with $j \in [1, n_i]$:

$$\mathbf{c}^p_{i_j} = \sum_{k=1}^{j} \tilde{m}_{i_k} \mathbf{x}_{i_k}, \qquad \mathbf{A}^p_{i_j} = \sum_{k=1}^{j} \tilde{m}_{i_k} \mathbf{x}_{i_k} \bar{\mathbf{x}}^T_{i_k}.$$

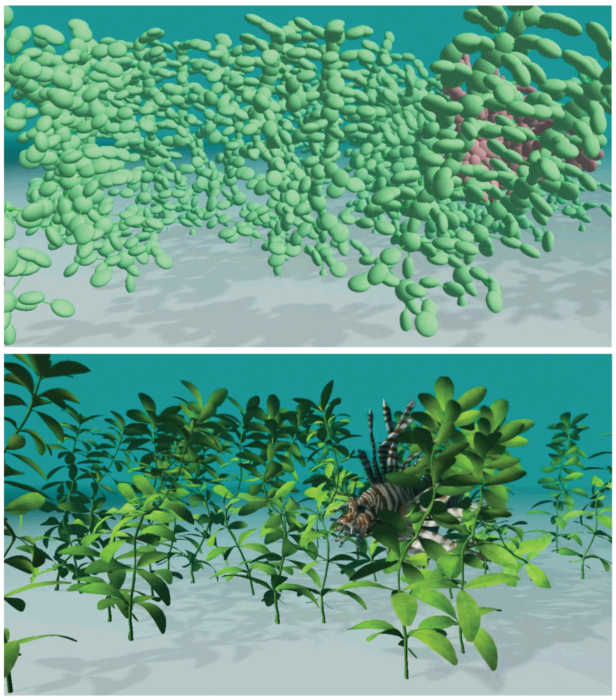
Figure 18: *This underwater scene demonstrates the ability of the oriented particle approach to handle sparse meshes such as the one-dimensional branches of the plants or the fins of the lion fish.*

Since the prefix sums for all paths are independent of each other, they can be computed in parallel. The sums for a region r are computed by first setting $\mathbf{c}_r := \mathbf{0}$ and $\mathbf{A}_r := \mathbf{0}$. Then, for each path i, which intersects the region in the interval $[i_k, \ldots, i_l]$, the following terms are added:

$$\mathbf{c}_r := \mathbf{c}_r + \mathbf{c}^p_{i_l} - \mathbf{c}^p_{i_{k-1}}, \qquad \mathbf{A}_r := \mathbf{A}_r + \mathbf{A}^p_{i_l} - \mathbf{A}^p_{i_{k-1}}. \tag{9}$$

The final translational vector and the affine matrix are determined by $\mathbf{c}_r := (1/\tilde{M}_r)\mathbf{c}_r$ and $\mathbf{A}_r := \mathbf{A}_r - \tilde{M}_r\mathbf{c}_r\bar{\mathbf{c}}_r^T$, respectively.

4.4. Oriented Particles

For a small number of particles or particles that are close to colinear or coplanar (as in Figure 18), the matrix $\mathbf{A}_r$ in Equation (6) becomes ill-conditioned and the polar decomposition needed to obtain the optimal rotation tends to be numerically unstable.

To solve this problem, Müller *et al.* [MC11] proposed to use oriented particles. By adding orientation information to particles, the polar decomposition becomes stable even for single particles. The moment matrix of a single spherical particle with orientation $\mathbf{R} \in \mathbb{R}^{3\times 3}$ and finite radius r at the origin is well-defined and can be computed via an integral over its volume as

$$\begin{aligned}\mathbf{A}_{\text{sphere}} &= \int_{V_r} \rho(\mathbf{Rx})\mathbf{x}^T \mathrm{d}V = \rho\mathbf{R}\int_{V_r} \mathbf{x}\mathbf{x}^T \mathrm{d}V \\ &= \tfrac{4}{15}\pi r^5 \rho \mathbf{R} = \tfrac{4}{15}\pi r^5 \tfrac{m}{V_r}\mathbf{R} \\ &= \tfrac{1}{5} m r^2 \mathbf{R},\end{aligned}$$

where V_r is the volume of a sphere of radius r. Since $\mathbf{R}$ is an orthonormal matrix, $\mathbf{A}_i$ always has full rank and an optimal condition number of 1. For an ellipsoid with radii a, b and c, we get

$$\mathbf{A}_{\text{ellipsoid}} = \frac{1}{5}m\begin{bmatrix} a^2 & 0 & 0 \\ 0 & b^2 & 0 \\ 0 & 0 & c^2 \end{bmatrix}\mathbf{R}.$$

However, the moment matrices of the individual particles cannot simply be added because each one is computed relative to the origin. We need the moment matrix of particle i relative to the position $\mathbf{x}_i - \mathbf{c}$.

Fortunately, this problem can be fixed easily. As we saw above, the equation for computing the moment matrix

$$\mathbf{A} = \sum_i m_i(\mathbf{x}_i - \mathbf{c})(\bar{\mathbf{x}}_i - \bar{\mathbf{c}})^T \tag{10}$$

can be rewritten as

$$\mathbf{A} = \sum_i m_i \mathbf{x}_i \bar{\mathbf{x}}_i^{\mathrm{T}} - M\mathbf{c}\bar{\mathbf{c}}^{\mathrm{T}},$$

where $\bar{\mathbf{c}}$ and $\mathbf{c}$ are the centres of mass of the initial and the deformed shape, respectively (see Equation (5)).

Therefore, shifting the evaluation from the origin to the position $\mathbf{x}_i - \mathbf{c}$ yields

$$\mathbf{A}_i^{\text{global}} = \mathbf{A}_i + m_i \mathbf{x}_i \bar{\mathbf{x}}_i^{\mathrm{T}} - m_i \mathbf{c}\bar{\mathbf{c}}^{\mathrm{T}}.$$

Equation (10) now generalizes to

$$\begin{aligned}\mathbf{A} &= \textstyle\sum_i \left(\mathbf{A}_i + m_i \mathbf{x}_i \bar{\mathbf{x}}_i^{\mathrm{T}}\right) - M\mathbf{c}\bar{\mathbf{c}}^{\mathrm{T}} \\ &= \textstyle\sum_i \left(\mathbf{A}_i + m_i(\mathbf{x}_i - \mathbf{c})(\bar{\mathbf{x}}_i - \bar{\mathbf{c}})^{\mathrm{T}}\right).\end{aligned}$$

As you can see, the last form looks like Equation (10) but with all the individual particle moment matrices added in the sum.

In addition to position $\mathbf{x}$ and velocity $\mathbf{v}$, oriented particles carry a rotation which can be defined as an orthonormal matrix $\mathbf{R}$ as above or a unit quaternion $\mathbf{q}$. They also carry the angular velocity ω. In the prediction step of position-based dynamics, these two quantities have to be integrated as well:

$$\begin{aligned}\mathbf{x}_p &\leftarrow \mathbf{x} + \mathbf{v}\Delta t \\ \mathbf{q}_p &\leftarrow \left[\frac{\omega}{|\omega|}\sin\left(\frac{|\omega|\Delta t}{2}\right), \cos\left(\frac{|\omega|\Delta t}{2}\right)\right]\mathbf{q}.\end{aligned}$$

For stability reasons, $\mathbf{q}_p$ should directly be set to $\mathbf{q}$ if $|\omega| < \epsilon$.

After the prediction step, the solver iterates multiple times through all shape match constraints in a Gauss-Seidel type fashion as before. To simulate objects represented by a mesh of linked particles, Müller and Chentanez [MC11] define one shape matching group per particle. A group contains the corresponding particle and all the particles connected to it via a single edge. The positions of the particles in a group are updated as in regular shape matching by pulling

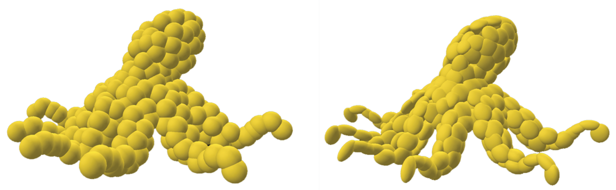

Figure 19: *The rotation information of oriented particles cannot only be used to stabilize shape matching, it also allows the use of ellipsoids as collision primitives. The figure shows how the same mesh is approximated much more accurately with ellipsoids (right) than with the same number of spheres (left).*

them towards the goal positions while the orientation of the centre particle only is replaced by the optimal rotation of shape matching.

After the solver has modified the predicted state $(\mathbf{x}_p, \mathbf{q}_p)$, the current state is updated using the integration scheme

$$\begin{aligned}
\mathbf{v} &\leftarrow (\mathbf{x}_p - \mathbf{x})/\Delta t\\
\mathbf{x} &\leftarrow \mathbf{x}_p\\
\omega &\leftarrow \text{axis}\,(\mathbf{q}_p\mathbf{q}^{-1}) \cdot \text{angle}\,(\mathbf{q}_p\mathbf{q}^{-1})/\Delta t\\
\mathbf{q} &\leftarrow \mathbf{q}_p,
\end{aligned}$$

where axis () returns the normalized direction of a quaternion and angle () its angle. Again, for stability reasons, ω should be set to zero directly if $|\text{angle}\,(\mathbf{q}_p\mathbf{q}^{-1})| < \epsilon$. There are two rotations, $\mathbf{r} = \mathbf{q}_p\mathbf{q}^{-1}$ and $-\mathbf{r}$ transforming $\mathbf{q}$ into $\mathbf{q}_p$. It is important to always choose the shorter one, i.e. if $\mathbf{r}_w < 0$ use $-\mathbf{r}$, where $\mathbf{r}_w$ is the real part of the quaternion. As in traditional PBD for translation, changing the rotational quantity $\mathbf{q}_p$ in the solver also affects its time derivate ω through the integration step creating the required second-order effect.

The orientation information of particles cannot only be used to stabilize shape matching but also to move a visual mesh along with the physical mesh. With position and orientation, each particle defines a full rigid transformation at every point in time. This allows the use of traditional linear blend skinning with particles replacing skeletal bones.

An additional advantage of having orientation information is that ellipsoids can be used as collision volumes for particles. This allows a more accurate approximation of the object geometry than with the same number of spherical primitives (see Figure 19).

4.5. Extensions

There exist several extensions for shape matching. In the following, we will introduce volume conservation and plastic deformation.

4.5.1. *Volume conservation*

The conservation of volume plays an important role in the dynamic simulation of deformable bodies [HJCW06], [ISF07], [DBB09]. Since most soft biological tissues are incompressible, this is an essential extension in the field of medical simulation. However, it is also used in the field of shape modelling [vFTS06] since volume conserving deformations appear more realistic.

In the following, we introduce the position-based approach for volume conservation of Diziol *et al.* [DBB11]. This method considers only the surface of a simulated object and does not require interior particles which reduces the computational effort. The volume V of a volumetric 3D shape $\mathcal{V}$ can be determined by using the divergence theorem as proposed in [Mir96] and [HJCW06]:

$$\iiint_{\mathcal{V}} \nabla \cdot \mathbf{x}\,\mathrm{d}\mathbf{x} = \iint_{\partial\mathcal{V}} \mathbf{x}^{\mathrm{T}}\mathbf{n}\,\mathrm{d}\mathbf{x} = 3V, \tag{11}$$

where $\partial\mathcal{V}$ is the boundary of the shape and $\mathbf{n}$ is the surface normal. If the boundary is given as triangle mesh, the integral can be written as sum over all triangles i:

$$V(\mathbf{X}) := \frac{1}{3}\iint_{\partial\mathcal{V}} \mathbf{x}^{\mathrm{T}}\mathbf{n}\,\mathrm{d}\mathbf{x} = \frac{1}{9}\sum_i A_i(\mathbf{x}_{i_1} + \mathbf{x}_{i_2} + \mathbf{x}_{i_3})^{\mathrm{T}}\mathbf{n}_i, \tag{12}$$

where A_i is the area and i_1, i_2 and i_3 are the vertex indices of the ith triangle. Now we can define a volume constraint $C := V(\mathbf{X}) - V(\bar{\mathbf{X}}) = 0$ and compute a corresponding position correction (see Section 3):

$$\Delta\mathbf{x}_i^V = -\frac{w_i C(\mathbf{X})}{\sum_j w_j \|\nabla_{\mathbf{x}_j} C(\mathbf{X})\|^2}\nabla_{\mathbf{x}_i} C(\mathbf{X}). \tag{13}$$

The weights w_i are used to realize a local volume conservation (see below). The gradient can be approximated by

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

where $\bar{\mathbf{n}}_i = \sum A_j\mathbf{n}_j$ is the sum of the area weighted normals of all triangles which contain particle i.

The weights in Equation (13) are chosen as follows:

$$w_i = (1-\alpha)w_i^l + \alpha w_i^g, \qquad w_i^l = \frac{\|\Delta\mathbf{x}_i\|}{\sum_j \|\Delta\mathbf{x}_j\|}, \qquad w_i^g = \frac{1}{n},$$

where w_i^l and w_i^g are the weights for local and global volume conservation, respectively, and the user-defined value $\alpha \in [0, 1]$ is used to blend between both. The vector $\Delta\mathbf{x}_i$ contains the position change of the ith particle in the shape matching step. Hence, strongly deformed particles participate more in volume correction. The weight of a colliding particle is set to zero in order to ensure that a collision constraint is not violated during the position correction for the volume conservation. Finally, the weights are smoothed by a Laplacian filter.

Diziol *et al.* also propose another definition for the local weights w_i^l. To propagate volume changes through the object, they first determine pairs of opposing particles in a pre-processing step by

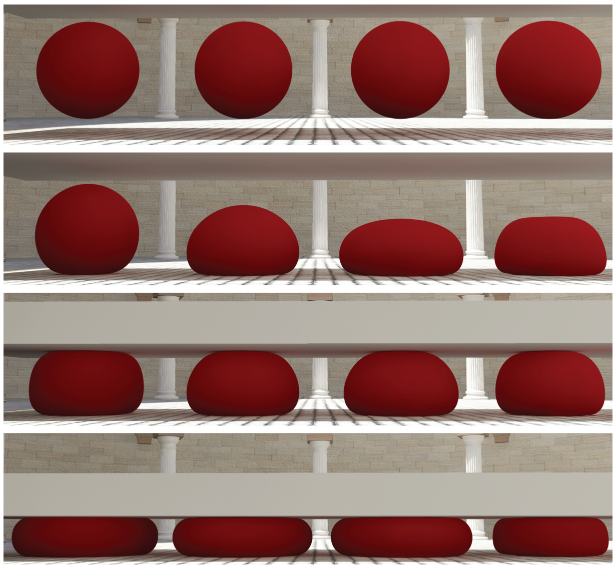

Figure 20: *Four spheres with different volume conservation squeezed by a plate. Left to right: global conservation, local conservation with distance constraints, local conservation without distance constraints and no volume conservation. The maximum volume loss was* 0.6%, 0.7%, 0.7% *and* 40%, *respectively.*

intersecting the geometry with multiple rays. For each particle i, one particle k on the opposite side of the volumetric body is stored. Then they choose a local weight which does not only depend on the position change $\Delta\mathbf{x}_i$ of a particle but also on the distance changes Δd_i of the corresponding particle pairs:

$$w_i^l = \frac{\beta s_i \Delta d_i + (1-\beta)\|\Delta\mathbf{x}_i\|}{\sum_j \left(\beta s_j \Delta d_j + (1-\beta)\|\Delta\mathbf{x}_j\|\right)},$$

where s_i is a user-defined stiffness parameter and $\beta \in [0, 1]$ is used to define the influence of the distance changes.

Analogous to the positions correction, we perform a velocity correction to fulfil the constraint $\partial C/\partial t = 0$. This leads to a divergence-free velocity field.

In Figure 20, different configurations for the presented volume conservation method are compared with each other.

4.5.2. *Plastic deformation*

Shape matching can be extended in order to simulate plastic deformations [MHTG05]. If we perform a polar decomposition $\mathbf{A}_r = \mathbf{RS}$ for the linear transformation matrix $\mathbf{A}_r$ (see Equation 6), we get a rotational part $\mathbf{R}$ and a symmetric part $\mathbf{S} = \mathbf{R}^T\mathbf{A}_r$. The matrix $\mathbf{S}$ represents a deformation in the unrotated reference frame. Hence, for each region we can store the plastic deformation state in a matrix $\mathbf{S}^p$ which is initialized with the identity matrix $\mathbf{I}$. As proposed by Goktekin *et al.* [GBO04], we use two parameters c_{yield} and c_{creep} to control the plastic behaviour of the material. If the condition $\|\mathbf{S}-\mathbf{I}\|_2 > c_{\text{yield}}$ is fulfilled for the deformation matrix $\mathbf{S}$ of the current time step, the plastic deformation state is updated as follows:

$$\mathbf{S}^p \leftarrow \left[\mathbf{I} + \Delta t c_{\text{creep}}(\mathbf{S}-\mathbf{I})\right]\mathbf{S}^p.$$

After this update, $\mathbf{S}^p$ is divided by $\sqrt[3]{\det(\mathbf{S}^p)}$ in order to conserve the volume. The plastic state $\mathbf{S}^p$ is integrated in the shape matching process by deforming the reference shape in Equation (6). This is done by replacing the definition of $\bar{\mathbf{r}}_i$ (see Section 4.1) with

$$\bar{\mathbf{r}}_i = \mathbf{S}^p(\bar{\mathbf{x}}_i - \bar{\mathbf{c}}).$$

Note that the plasticity can be bound by the condition $\|\mathbf{S}^p - \mathbf{I}\|_2 > c_{\max}$ where $c_{\max}$ is the threshold for the maximum plastic deformation. If this condition is fulfilled, we use $\mathbf{S}^p \leftarrow \mathbf{I} + c_{\max}(\mathbf{S}^p - \mathbf{I})/\|\mathbf{S}^p - \mathbf{I}\|_2$.

4.6. Cloth Simulation

Stumpp *et al.* [SSBT08] present a region-based shape matching approach for the simulation of cloth. In their work, they define a region for each triangle in the model. But instead of using the triangles directly as regions for shape matching, overlapping regions are defined. The region of a triangle is defined by the outer corners of its adjacent triangles. These overlapping regions enable the bending resistance of the cloth model. Since the model of Stumpp *et al.* uses regions with only three vertices, the stiffness of high-resolution models is too low for realistic results. Therefore, they introduce so-called fibre clusters to increase the stretching stiffness. These one-dimensional regions are determined in a pre-processing step by subdividing the mesh into multiple edge strips. During the simulation, each strip is traversed in both directions to obtain additional goal positions. The resulting displacements are translated so that they sum up to zero to preserve the momentum of the model. The final goal positions are blended with the goal positions of the triangular regions.

The usage of fibre clusters increases the stiffness of the cloth model. However, this effect is limited and for high-resolution models the stiffness is still too low to achieve a realistic cloth behaviour. Bender *et al.* [BWD13] solve this problem by the introduction of multi-resolution shape matching (see Figure 21) which is based on the idea of multi-grid solvers [Hac85]. A shape matching region is defined for each edge and each triangle in a cloth model. To increase the influence of these simple regions and therefore the stretching and shearing stiffness of the model, shape matching is performed on different resolution levels. Multi-resolution shape matching enables the robust simulation of stiff cloth models in linear time.

In the following, we first describe 2D shape matching for triangular regions and then introduce multi-resolution shape matching.

For a cloth simulation with triangular regions, shape matching is performed per triangle in the 2D space of the triangle plane. First, the optimal translation vectors of the regions are computed by

Figure 21: *A stiff cloth model with 32 467 triangles is simulated using multi-resolution shape matching with five hierarchy levels.*

evaluating Equation (7). Then, for each triangle with the vertices $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ and the normal $\mathbf{n}$ a projection matrix is determined:

$$\mathbf{P} = \begin{pmatrix} \mathbf{a}_x^T \\ \mathbf{a}_y^T \end{pmatrix} \in \mathbb{R}^{2\times 3}$$

with

$$\mathbf{a}_x = \frac{\mathbf{x}_2 - \mathbf{x}_1}{\|\mathbf{x}_2 - \mathbf{x}_1\|}, \quad \mathbf{a}_y = \frac{\mathbf{n} \times \mathbf{a}_x}{\|\mathbf{n} \times \mathbf{a}_x\|}.$$

The matrix $\mathbf{P}$ is used to project the vectors $\mathbf{r}$ and $\bar{\mathbf{r}}$ in Equation (6) to get a 2D version of the matrix $\mathbf{A}_r$:

$$\bar{\mathbf{r}}'_i = \bar{\mathbf{P}}(\bar{\mathbf{x}}_i - \bar{\mathbf{c}}), \quad \mathbf{r}'_i = \mathbf{P}(\mathbf{x}_i - \mathbf{c}),$$

where $\bar{\mathbf{r}}'_i \in \mathbb{R}^2$ can be pre-computed. The optimal rotation for shape matching is obtained by performing a 2D polar decomposition [SD92] for the resulting matrix $\mathbf{A}'_r \in \mathbb{R}^{2\times 2}$. This rotation matrix is used to compute 2D goal positions $\mathbf{g}'_i$ for the particles and the corresponding 2D position changes $\Delta\mathbf{x}'_i$:

$$\mathbf{g}'_i = \mathbf{R}'\bar{\mathbf{r}}'_i, \quad \Delta\mathbf{x}'_i = \alpha\frac{1}{|\Re_i|}(\mathbf{g}'_i - \mathbf{x}'_i).$$

Finally, the vectors $\Delta\mathbf{x}'_i$ are transformed to world space by $\Delta\mathbf{x}_i = \mathbf{P}^T\Delta\mathbf{x}'_i$ and the particle positions are updated. This process is shown in Figure 22.

In a simulation with multi-resolution shape matching [BWD13], two intergrid transfer operators are required to couple the different meshes in the multi-resolution hierarchy. The restriction operator $\mathbf{I}^l_{l+1}$ transfers values from level $l+1$ to the next coarser level l and the prolongation operator $\mathbf{I}^{l+1}_l$ transfers values in the opposite direction. These operators can be defined by barycentric coordinates [GW06]. In each simulation step, first the positions of the finest mesh are updated by time integration. For non-nested models, the positions of the coarser meshes are interpolated using the restriction operator. Then multi-resolution shape matching is performed in a V-cycle as described by Algorithm 2.

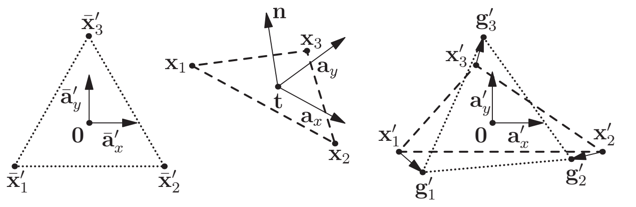

Figure 22: *2D shape matching. The initial configuration of a triangle in 2D (left) is matched to the deformed configuration (middle) by projecting the deformed triangle into 2D and computing the optimal translation and rotation to get goal positions (right).*

Algorithm 2 Multi-resolution shape matching

```
 1: for $l = l_{max}$ to 1 do
 2:     Store current positions: $\hat{\mathbf{x}}^l \leftarrow \mathbf{x}^l$
 3:     Perform shape matching
 4:     $\mathbf{x}^{l-1} := \mathbf{x}^{l-1} + \mathbf{I}^{l-1}_l(\mathbf{x}^l - \hat{\mathbf{x}}^l)$
 5: end for
 6: for $l = 0$ to $l_{max}$ do
 7:     Store current positions: $\hat{\mathbf{x}}^l \leftarrow \mathbf{x}^l$
 8:     Perform shape matching
 9:     if $l \neq l_{max}$ then
10:         $\mathbf{x}^{l+1} := \mathbf{x}^{l+1} + \mathbf{I}^{l+1}_l(\mathbf{x}^l - \hat{\mathbf{x}}^l)$
11:     end if
12: end for
```

In the restriction phase, the hierarchy is traversed from the finest to the coarsest mesh performing a shape matching step on each level and projecting the resulting position differences $\mathbf{x}^l - \hat{\mathbf{x}}^l$ to the next coarser level with the restriction operator. In the prolongation phase, the hierarchy is traversed in the opposite direction. On each level, a shape matching step is performed and the position differences are interpolated and added to the next finer level. Since only position differences are propagated between the levels, fine details are conserved on finer levels. However, fine details could get lost if the original shape matching method is used on the coarse levels of the hierarchy. Wrinkles on a fine resolution cause a compression of elements on a coarser level. Shape matching reduces this compression and thus eliminates fine details. Therefore, Bender *et al.* [BWD13] propose a modified computation of the goal positions on the coarse levels of the hierarchy so that shape matching only prevents stretching on these levels but not a compression.

4.7. Parallelization

In Section 4.3, we presented different fast summation techniques for shape matching. The one of Diziol *et al.* [DBB11] is best suited for a parallel implementation on the GPU. In the following, the GPU implementation of this technique with CUDA is described in detail. For such an implementation, memory access and memory layouts play an important role as well as the number of kernel calls.

Since each kernel call introduces a computational overhead, the particles of all objects in a simulation are packed into one single array. This array is ordered according to the path layout which is used for the fast summation (see Section 4.3). Since the array contains the paths one after another, a segmented prefix sum [SHZO07] can be used to determine the prefix sums of all paths at once. To avoid numerical problems due to the 32-bit floating-point arithmetic on the GPU, the path length is limited to 512. The resulting prefix sums are stored in texture memory to benefit from the texture cache when the translational vectors and the affine matrices are determined (see Equation 9).

The volume conservation introduced in Section 4.5.1 can also be performed efficiently on the GPU. This is done by evaluating the volume integral (see Equation 11), the integral which is required to obtain a divergence free velocity field and the weights for the local volume conservation in parallel. Both integrals can be written as sums over the vertices (see Equation 12). Hence, the integrals as well as the weights can be computed by a segmented sum reduction. Finally, the smoothing of the weights by a Laplacian filter can be performed in parallel using the fast summation technique as described above.

The multi-resolution approach described in Section 4.6 can be implemented on the GPU as follows. Shape matching on each level of the hierarchy is performed by computing the goal positions per element in parallel in a first step. The results are stored for each element. In a second step, shape matching is completed by summing up the contributions of all elements containing a vertex to get a final goal position for the vertex. The restriction and the prolongation of the results can be performed efficiently using the sparse matrix data structure of Weber *et al.* [WBS*13]. This implementation allows to simulate the deformation of a cloth model with more than 200K triangles on the finest level in 22 ms/step on a GeForce GTX 470.

4.8. Discussion

Shape matching is efficient, unconditionally stable and easy to implement. However, the shape matching algorithm can only handle small deformations. Therefore, the concept of region-based shape matching was introduced as well as corresponding fast summation techniques. Although this method is a meshless approach, many implementations use a mesh to define the required overlapping regions. Shape matching can even handle sparse structures by the usage of oriented particles.

We also want to discuss the drawbacks of shape matching. Being a geometric approach, only visually plausible results can be obtained. The stiffness of the model does not only depend on the user-defined stiffness parameter α but also on the time step size and the region sizes or, in case of multi-resolution shape matching, the number of hierarchy levels. Moreover, the physical behaviour depends on the mesh geometry and does not converge to a certain solution as the mesh is refined. Hence, adaptive time stepping and the application of level-of-detail methods are open problems for further research.

5. Data-Driven Upsampling Methods

The behaviour of solid objects can be accurately described using well-known mechanical models, but real-world materials display other inherent sources of complexity that largely limit the results of traditional models in computer animation. Complexity is produced, for example, by non-linear or anisotropic behaviours, by heterogeneous properties, or by a wide frequency spectrum. These sources of complexity are typically addressed by designing complex non-linear constitutive models to describe the mechanical behaviour of diverse materials. However, these models require computationally expensive simulation algorithms, and their parameters are difficult and tedious to tune, particularly if the properties are heterogeneous. All in all, the animation of solid objects is limited by the domain of effects captured by the underlying physical models, but also by their parametrization accuracy.

Data-driven methods offer an alternative to complex constitutive models, as they turn the modelling metaphor into the knowledge of a system's response under several example conditions. This section describes geometric data-driven methods in computer animation. It formulates a two-scale representation of geometry and dynamics, describes the computation of detailed geometry as the interpolation of example data, and discusses several successful examples.

5.1. Two-Scale Geometry and Dynamics

Let us consider the surface of a solid object (e.g. the cloth in Figure 23), with vertex positions $\mathbf{x} \in \mathbb{R}^3$. These positions can be decomposed into a low-resolution position $\mathbf{x}_0$ and a fine-scale displacement $\Delta\mathbf{x}$, expressed in a local reference system for each vertex (i.e. with orientation $\mathbf{R}$):

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{R}\,\Delta\mathbf{x}.$$

This definition of vertex positions essentially decomposes large-scale geometry (i.e. the overall shape of a solid object) from the small-scale deformation (i.e. wrinkles).

The large-scale and fine-scale geometry can be represented at different resolutions, connected through subdivision schemes. As shown in Figure 23 for a cloth object, a low-resolution *feature mesh* defines the large-scale deformation. A high-resolution *smooth mesh* is obtained by subdividing the feature mesh a user-defined number of times, and can be regarded as an upsampled version of the feature mesh. Finally, a high-resolution *detail mesh* is obtained by adding local displacements onto the smooth mesh. The smooth and detail meshes fully share the connectivity, thus trivially defining their correspondence.

Once large-scale and fine-scale geometry are separated, they can be computed using different models. The choice of models can be made based on the following observations. First, for many objects such as the face or cloth, the most salient dynamic effects can be captured at a large scale. For example, Bickel *et al.* [BLB*08] and Ma *et al.* [MJC*08] compute the large-scale face geometry from an actor's performance, using a single face scan and sparse mocap markers as input, and a linear deformation model. Wang *et al.* [WHRO10], Zurdo *et al.* [ZBO13] and Kavan *et al.* [KGBS11] compute the large-scale geometry of cloth using a low-resolution dynamics model with contact handling.

A second important observation is that plausible high-resolution wrinkles can often be defined as a quasi-static function in a reduced

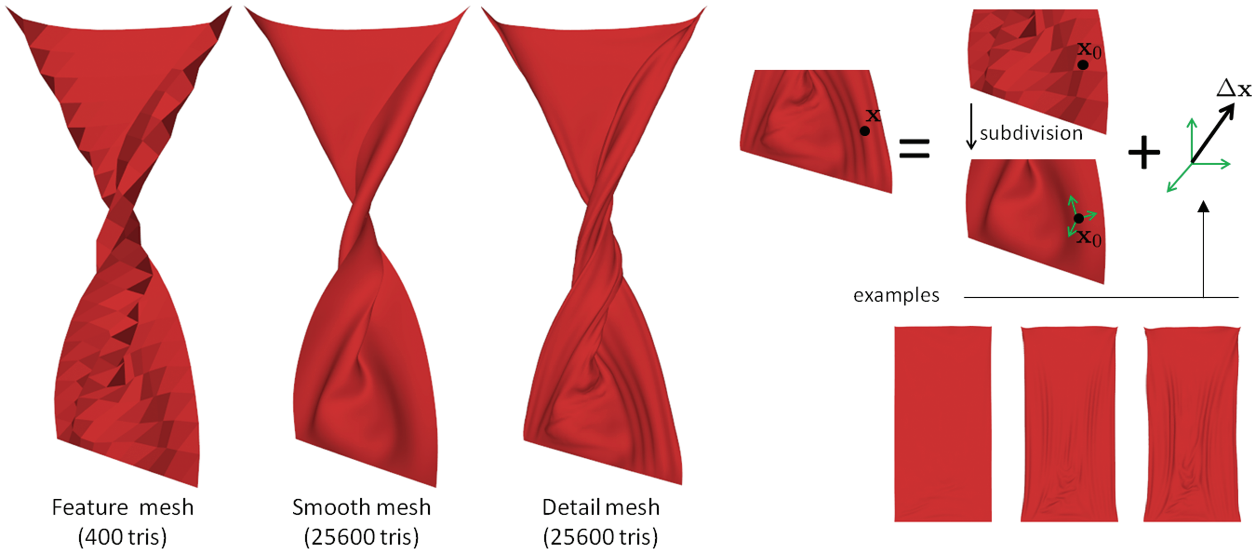

Figure 23: *Data-driven animation of cloth wrinkles. On the left, a low-resolution feature mesh, a smooth mesh obtained by subdivision and a high-resolution detail mesh. On the right, a schematic depiction of the data-driven wrinkle computation. The feature mesh is subdivided to obtain the smooth mesh, and then detailed wrinkles are added in a local reference frame by combining detail from examples.*

low-resolution domain $\mathbf{u}$. All authors mentioned in the previous paragraph follow this observation to some extent. For the face, and due to the repetitive nature of facial expressions, tissue becomes weaker at certain locations, and expressive wrinkles appear in a deterministic fashion as a function of large-scale deformation, and indirectly as a function of muscle activations and facial bone configurations. For cloth, even though real wrinkles require a large number of degrees of freedom to capture their true diversity, plausible wrinkles can be defined in the reduced domain of large-scale deformation. We can formally write the dependency between high-resolution detail $\Delta\mathbf{x}$ and the low-resolution configuration $\mathbf{u}$ as $\Delta\mathbf{x} = f(\mathbf{u})$, where the function f is a position-based model.

The reduced-domain definition presents some limitations, which should be mentioned upfront. Fine-scale wrinkle dynamics cannot be captured, as wrinkles are defined quasi-statically. And the model captures only a limited set out of all the possible wrinkles that a solid object might present. However, the power of data-driven methods is that the generic function f makes use of data from real deformation examples; therefore, data-driven wrinkles preserve natural characteristics such as length, width and consistency over time, and they appear plausible despite their limitations.

5.2. Data-Driven Geometric Detail

At this point, we have a suitable setting to define a position-based data-driven model. For data collection, we need to record example deformations (denoted by the subscript i), with vertex displacements $\{\Delta\mathbf{x}_i\}$ and low-resolution configurations $\{\mathbf{u}_i\}$ in correspondence. Then, we apply learning methods to design a data-driven approximation of the function f, which can be formally defined as $\Delta\mathbf{x} \approx \hat{f}(\mathbf{u}, \{\Delta\mathbf{x}_i\}, \{\mathbf{u}_i\})$.

A general and successful approach to define a data-driven approximation $\hat{f}$ is through a linear combination of example-based basis functions,

$$\Delta\mathbf{x} = \sum_j w_j(\mathbf{u})\,\mathbf{b}_j \qquad (14)$$

with weights w_j computed as a function of the low-resolution configuration $\mathbf{u}$. In the expression above, $\mathbf{b}_j \in \mathbb{R}^3$ constitutes an example-based detail displacement, and represents the values associated with one vertex in the jth basis function.

In the rest of this section, we discuss several examples of position-based data-driven deformation models, which differ in terms of the choice of low-resolution configuration, basis functions, or interpolation method.

5.2.1. *Weighted pose-space deformation (WPSD)*

Bickel *et al.* [BLB*08] proposed a data-driven method to compute facial wrinkles as a function of large-scale deformation and a small set of example deformations, and later Zurdo *et al.* [ZBO13] followed a similar strategy for cloth animation. Figure 24 shows an example facial animation by Bickel *et al.*, and Figure 25 an example cloth animation by Zurdo *et al.*

In their methods, the basis function combination in Equation (14) follows the WPSD approach [KM04]. In a nutshell, the basis functions constitute local vertex displacements in example deformations, called *poses*, and the weights w are convex weights for each of the poses. The weights of the poses are computed using scattered-data interpolation based on *radial basis functions* (RBFs) in *pose space*. In this case, the pose space is given by the large-scale deformation $\mathbf{u}$.

As shown by Zurdo *et al.*, pose weights can be computed through WPSD on the sparse vertices of the feature mesh, and then simply interpolated to the vertices of the detail mesh using subdivision weights. For the computation of pose weights on a given vertex,

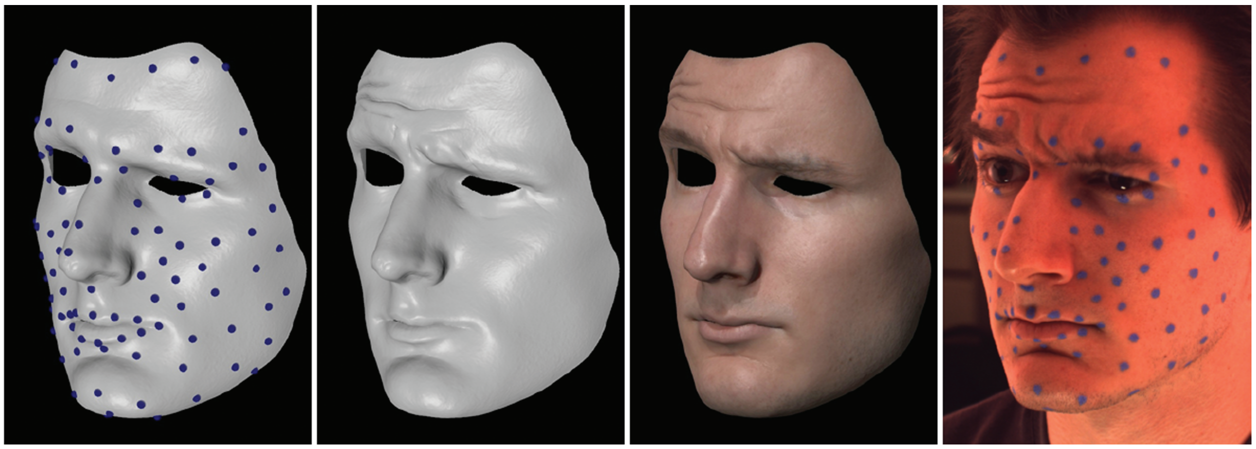

Figure 24: *Example of facial animation with the data-driven method of Bickel et al. [BLB*08]. From left to right: large-scale deformation interpolating mocap markers, full result data-driven detail computation, the same result with full shading and comparison to the real actor's face.*

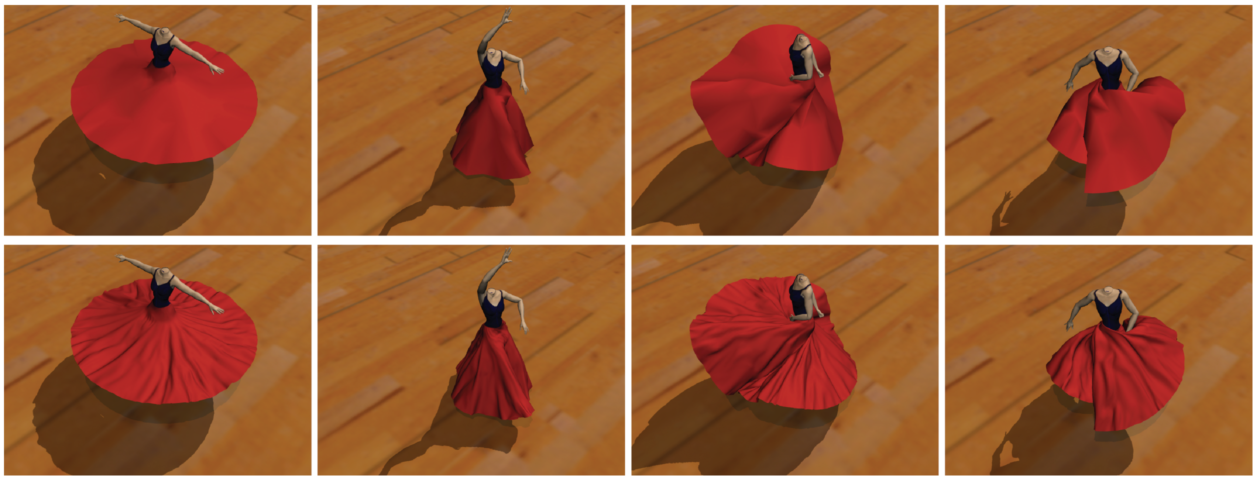

Figure 25: *Top row: a low-resolution model of a dress, with only* 381 *triangles, defines the dynamics, the large-scale deformation, and response to contact. The dress is then subdivided to 24 384 triangles for rendering. Bottom row: high-resolution wrinkles are interpolated from six example poses based on the large-scale deformation of the dress, using the data-driven method of Zurdo et al. [ZBO13].*

WPSD requires a local definition of the pose space **u**. Both Bickel *et al.* and Zurdo *et al.* use a metric of local low-resolution strain, formed by concatenating the deformation of the 16 closest edges of the feature mesh, multiplied by fast-decaying weights. With RBF interpolation, the weight of each pose is computed as

$$w_j = \sum_i \omega_{i,j}\, \phi(\|\mathbf{u} - \mathbf{u}_i\|).$$

The RBF weights $\omega_{k,j}$ are pre-computed such that pose weights fulfil the Kronecker delta for the database of poses, i.e. $w_j(\mathbf{u}_i)$ is 0 if $j \neq i$ and 1 if $j = i$, for poses j and i in the database. The function ϕ represents a specific type of RBF. Both Bickel *et al.* and Zurdo *et al.* use RBFs with global support $\phi(r) = r$, as they avoid complex tuning of support radii for unevenly sampled data [CBC*01].

The selection of poses starts with the generation of training data, which requires a database of synchronized feature and detail meshes. Zurdo *et al.* employ the TRACKS method [BMWG07] to pre-compute cloth simulations where the detail mesh tracks the motion of the feature mesh. In the training simulations, contact is solved both on the feature mesh and the detail mesh, and then the high-resolution information in the poses captures the response to contact. From all mesh pairs in the database, Zurdo *et al.* select only a small number of poses (typically 6), until the L^2 error between synthesized animations and the training data set is below a certain threshold. The set of poses is grown in a greedy manner, adding each time the mesh pair with largest L^2 error between the training and synthesized configurations.

5.2.2. *Polynomial displacement maps (PDM)*

Ma *et al.* [MJC*08] designed PDM, a method for data-driven computation of wrinkles in facial animation, following the computational strategy of polynomial texture maps [MGW01]. Despite the computational differences, the method shares many similarities with the WPSD approach of Bickel *et al.* [BLB*08].

In the general data-driven framework described by Equation (14), the low-resolution configuration of PDM is defined as a 2D local strain metric of the feature mesh, $\mathbf{u} = (u_1, u_2)$. In particular, this metric is obtained through principal component analysis of a five-dimensional vector formed by the local vertex offset and the in-plane strain of the feature mesh. Based on this low-resolution configuration, the weights in Equation (14) are defined based on biquadratic polynomials, $\mathbf{w}(\mathbf{u}) = (u_1^2 \; u_2^2 \; u_1 u_2 \; u_1 \; u_2 \; 1)$. Finally, the detail position of each vertex in the detail mesh, $\Delta\mathbf{x}$, is defined as a scalar displacement in the local normal direction.

Due to the choice of biquadratic PDMs, the per-vertex displacements are computed through linear combination of six basis functions, which are stored in texture maps. Ma *et al.* compute the coefficients in these texture maps as the result of a least-squares problem that minimizes the fitting error over the training data.

5.2.3. *Physics-inspired upsampling (PIU)*

Kavan *et al.* [KGBS11] proposed a data-driven method to compute cloth animations with detailed wrinkles at video game frame rates. In their approach, high-resolution cloth positions are computed following a position-based data-driven method, but the data are learned from dynamic simulations with tracking of high-resolution and low-resolution geometry, similar to the approach by Zurdo *et al.* [ZBO13] described earlier.

In PIU, the weights w in Equation (14) correspond directly to the vertex positions of the feature mesh. The basis functions coefficients **b** extend subdivision schemes, and are learned from a database of example deformations.

For the description of the method in more detail, let us define the low-resolution deformation **u** as a vector that concatenates the vertex positions of the low-resolution feature mesh. We also define a matrix **B** of upsampling basis functions, being each row of **B** the basis function for one vertex coordinate. Finally, we define a vector **X** that concatenates all vertex positions of the high-resolution detail mesh. Note that the method of Kavan *et al.* computes high-resolution positions directly, not local displacements as discussed earlier. With these definitions, the data-driven detail from Equation (14) can be rewritten as $\mathbf{X} = \mathbf{B}\,\mathbf{u}$.

Given the data captured in training simulations, expressed as example deformations of the feature mesh, $\{\mathbf{u}_i\}$, and the detail mesh, $\{\mathbf{X}_i\}$, Kavan *et al.* compute basis functions through an optimization problem

$$\mathbf{B} = \arg\min \sum_i \|\mathbf{B}\,\mathbf{u}_i - \mathbf{X}_i\|.$$

The optimization must satisfy additional constraints. In particular, the basis functions **B** must constitute a partition of unity, and the objective function includes a regularization term to prevent overfitting.

As mentioned earlier, one of the limitations of pure position-based data-driven methods is that wrinkles are quasi-static functions of low-resolution deformations. In PIU, Kavan *et al.* add oscillatory modes to allow the simulation of travelling waves.

5.3. Parallelization

One of the most attractive features of position-based data-driven methods is that they can be massively parallelized on GPUs. Taking as reference Equation (14), the multiplication of basis weights times the basis values follows the same procedure at all vertices of the detail mesh, but it can be executed in a completely independent manner on each vertex. Therefore, this multiplication is very well suited for parallel implementation on GPUs. In fact, it can be handled as a dense matrix–vector product.

Moreover, the evaluation of weights w can also be done independently for each basis. In the PIU method described above, the weights are trivially defined as the positions of vertices in the feature mesh, hence they do not need further evaluation. In the WPSD and PDM methods, on the other hand, these weights are computed as a function of low-resolution deformation. As shown by Zurdo *et al.* [ZBO13], the non-linear weights at feature vertices can be evaluated easily on the CPU, and then interpolated to detail vertices using subdivision weights. This interpolation step is trivial to parallelize on single instruction, multiple data (SIMD) architectures.

Thanks to the highly parallel nature of data-driven algorithms, existing implementations exhibit very high performance. For example, Zurdo *et al.* [ZBO13] simulate cloth with 25.6K triangles at 125 fps including contact handling on the feature mesh, Kavan *et al.* [KGBS11] simulate a cape with 10K vertices at 1 kHz, and Bickel *et al.* [BLB*08] animate a face with more than 1M triangles at 30 fps.

5.4. Discussion

The particular data-driven methods discussed here present different pros and cons. Among all methods, PIU is probably the fastest method, as it relies on simple matrix-vector multiplication. However, the result is limited to a linear operator, and wrinkles may appear smoothed. WPSD and PDM, on the other hand, allow non-linear operators, and the combination of the input data is decided in a local manner.

With WPSD and PDM, the representation is more compact, thanks to the choice of low-resolution strain as interpolation domain. Low-resolution strain also has been used as a parameter space for cloth wrinkles in procedural models [RPC*10], or in the Wrinkle Meshes method [MC10]. In the position-based Wrinkles Meshes method, described in Section 3.6, the low-resolution strain affects wrinkle modelling indirectly, as it influences the deformation of the high-resolution cloth. Seiler *et al.* [SSH12] propose a different interpolation domain based on local contact data, as their work targets the animation of high-resolution effects due to contact interactions.

One limitation of most data-driven methods is that wrinkles are defined as a quasi-static function of the feature mesh, hence they exhibit no dynamics. Kavan *et al.* [KGBS11] somewhat alleviate this problem by adding oscillatory modes, but the solution does not support all dynamic effects. The work of de Aguiar *et al.* [dASTH10] offers an interesting approach to data-driven simulation of dynamic effects, but it models low-resolution motion, not high-resolution details.

The data-driven cloth upsampling method of Feng *et al.* [FYK10] departs from those presented here. It not only defines fine-scale details in a data-driven manner, but also a mid-scale transformation from a coarse cloth simulation. Its two deformation transformers, for fine-scale and mid-scale deformations, are learned using regression with rotation-invariant data. The method of Feng *et al.* produces high-quality results if the training sequence captures the dynamics that will appear at runtime, but it will suffer larger errors otherwise.

This section has covered position-based data-driven methods for animating high-resolution effects on solids, but recent advances open the possibility to extend this paradigm also to fluids. The recent work of Kim *et al.* [KTT13] introduces a decomposition of liquid surface geometry into a low-resolution surface plus high-resolution ripples, just like the one defined for solid surfaces in Section 5.1 above. However, for liquids, a quasi-static definition of high-resolution details is a particularly severe limitation. Kim *et al.* propose a model that decouples low-resolution and high-resolution dynamics, and they use the *i Wave* model [Tes04] to synthesize high-resolution liquid ripples. We envision that such ripples could also be synthesized from pre-computed data using a data-driven model.

6. Applications

In this section, we introduce different application areas of position-based methods. These methods are mainly used in interactive applications where performance, controllability and stability are more important than accuracy, like, e.g. in [SGdA*10], [DB13]. But there exist also other works which use a position-based approach for stabilization.

One application area for position-based methods is interactive surgical simulation. In this area, Wang *et al.* [WXX*06] introduce a mass-spring model based on a surface mesh to simulate deformable bodies in real-time. Since such a model can neither preserve its volume nor resume its rest shape in the absence of external forces, the authors propose to couple the surface model with a rigid core by using spring forces. This rigid core is simulated using shape matching [MHTG05] which results in a fast and stable simulation. Kubiak *et al.* [KPGF07] present a simulation method for surgical threads which is based on the position-based dynamics approach of Müller *et al.* [MHHR07]. Their method simulates the stiffness, bending and torsion of a thread and also provides feedback for a haptic device. For the simulation, Kubiak *et al.* define distance constraints for stiffness and bending, torsion constraints, contact constraints and friction constraints. The presented method allows for an interactive and robust simulation of knots.

The simulation of complex hairstyles using a shape matching approach is presented by Rungjiratananon *et al.* [RKN10]. Their approach is based on Lattice Shape Matching which was originally introduced by Rivers and James [RJ07]. For the simulation, each hair strand is represented by a chain of particles which is subdivided in overlapping chain regions. After shape matching an additional position-based strain limiting is applied to each strand which moves the particles in the direction of their root. Different hair styles are realized by using appropriate initial configurations and by modifying the region sizes of a chain.

O'Brien *et al.* [ODC11] use position-based dynamics for the physically plausible adaptation of motion-captured animations. In their work, they use a vertex-based character skeleton and different constraints to preserve the skeleton structure, to define joint limits and to implement a centre of mass control. In addition to the kinematic constraints, they define a couple of dynamics constraints which consider vertices in multiple frames. Dynamics constraints are used to enforce smooth acceleration and dynamical correctness.

Fierz *et al.* [FSAH12] introduce a position-based approach to stabilize a finite element simulation. When using an explicit time integration for a finite element simulation, the time step size is typically limited by the stiffness of the model and its spatial discretization. In each simulation step, Fierz *et al.* use the Courant–Friedrichs–Lewy (CFL) condition to determine the maximum allowed time step size for each tetrahedral element in their volumetric simulation model. However, instead of using the time step size given by the CFL condition to perform a stable simulation step with an explicit integration scheme, they use a fixed size and mark all elements where the condition is not met. The marked elements are then simulated using a shape matching approach while for all other elements a linear FEM is used for the simulation.

7. Conclusion

In this survey, we focused on a popular and practically relevant subset of approaches for dynamically deforming solids, namely on position-based approaches. Such geometrically motivated techniques are not force-driven and are particularly appropriate in interactive applications due to their versatility, robustness, controllability and efficiency. We explained general ideas of position-based methods, shape-matching approaches and data-driven techniques. Various deformation aspects for 2D and 3D solids and efficient solution strategies were discussed with a particular focus on the benefits of position-based approaches compared to force-driven techniques.

Acknowledgments

The work of Jan Bender was supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School CE at TU Darmstadt. Miguel A. Otaduy was partially funded by the Spanish Ministry of Economy (grant TIN2012-35840) and the European Research Council (ERC-2011-StG-280135 Animetrics).

References

[BB08] Bender J., Bayer D.: Parallel simulation of inextensible cloth. In *VRIPHYS 08: Proceedings of Fifth Workshop in Virtual Reality Interactions and Physical Simulations* (2008), pp. 47–56.

[BETC12] Bender J., Erleben K., Trinkle J., Coumans E.: Interactive simulation of rigid body dynamics in computer graphics. In *EG 2012 - State of the Art Reports* (2012), Cani and F. Ganovelli, (Eds.), Eurographics Association, pp. 95–134.

[BLB*08] Bickel B., Lang M., Botsch M., Otaduy M. A., Gross M.: Pose-space animation and transfer of facial details. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), pp. 57–66.

[BMF03] Bridson R., Marino S., Fedkiw R.: Simulation of clothing with folds and wrinkles. In *Proceedings of the ACM/Eurographics Symposium on Computer Animation* (2003), pp. 28–36.

[BMWG07] Bergou M., Mathur S., Wardetzky M., Grinspun E.: TRACKS: Toward directable thin shells. In *Proceedings of ACM SIGGRAPH* (2007).

[BW98] Baraff D., Witkin A.: Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of Computer Graphics and Interactive Techniques* (1998), ACM, pp. 43–54.

[BWD13] Bender J., Weber D., Diziol R.: Fast and stable cloth simulation based on multi-resolution shape matching. *Computers & Graphics* (2013).

[CBC*01] Carr J. C., Beatson R. K., Cherrie J. B., Mitchell T. J., Fright W. R., McCallum B. C., Evans T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of ACM SIGGRAPH* (2001), pp. 67–76.

[Cro84] Crow F. C.: Summed-area tables for texture mapping. *SIGGRAPH Computer Graphics 18*, 3 (Jan. 1984), 207–212.

[dASTH10] de Aguiar E., Sigal L., Treuille A., Hodgins J. K.: Stable spaces for real-time clothing. *ACM Transactions on Graphics 29*, 4 (July 2010), 106:1–106:9.

[DB13] Deul C., Bender J.: Physically-based character skinning. In *VRIPHYS 13: Proceedings of 10th Workshop on Virtual Reality Interactions and Physical Simulations* (Lille, France, 2013), Eurographics Association, pp. 25–34.

[DBB09] Diziol R., Bender J., Bayer D.: Volume conserving simulation of deformable bodies. In *Short Paper Proceedings of Eurographics* (Mar. 2009).

[DBB11] Diziol R., Bender J., Bayer D.: Robust real-time deformation of incompressible surface meshes. In *SCA '11: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), Eurographics Association.

[DSB99] Desbrun M., Schröder P., Barr A.: Interactive animation of structured deformable objects. In *Proceedings of SIGGRAPH 99* (1999), ACM, pp. 1–8.

[FSAH12] Fierz B., Spillmann J., Aguinaga I., Harders M.: Maintaining large time steps in explicit finite element simulations using shape matching. *IEEE Transactions on Visualization and Computer Graphics 18,* 5 (May 2012), 717 –728.

[FYK10] Feng W.-W., Yu Y., Kim B.-U.: A deformation transformer for real-time cloth animation. *ACM Transactions on Graphics 29*, 4 (July 2010), 108:1–108:9.

[GBO04] Goktekin T. G., Bargteil A. W., O'Brien J. F.: A method for animating viscoelastic fluids. *ACM Transactions on Graphics 23*, 3 (August 2004), 463–468.

[GHF*07] Goldenthal R., Harmon D., Fattal R., Bercovier M., Grinspun E.: Efficient simulation of inextensible cloth. *ACM Trans. Graph. 26*, 3 (2007), 49.

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

[GW06] Georgii J., Westermann R.: A multigrid framework for real-time simulation of deformable bodies. *Computer & Graphics 30* (2006), 408–415.

[Hac85] Hackbusch W.: *Multi-Grid Methods and Applications*, vol. *4* of *Springer Series in Computational Mathematics*. Springer, 1985.

[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.

[ISF07] Irving G., Schroeder C., Fedkiw R.: Volume conserving finite element simulations of deformable models. *ACM Transactions on Graphics 26*, 3 (July 2007), 13:1–13:6.

[Jak01] Jakobsen T.: Advanced character physics. In *Proceedings, Game Developer's Conference 2001* (2001).

[JP99] James D. L., Pai D. K.: Artdefo: Accurate real time deformable objects. In *Proceedings of SIGGRAPH 99* (1999), ACM, pp. 65–72.

[KCM12] Kim T.-Y., Chentanez N., Müller M.: Long range attachments — A method to simulate inextensible clothing in computer games. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2012), J. Lee, and P. Kry, (Eds.), Eurographics Association, pp. 305–310.

[KGBS11] Kavan L., Gerszewski D., Bargteil A. W., Sloan P.-P.: Physics-inspired upsampling for cloth simulation in games. In *Proceedings of ACM SIGGRAPH* (2011).

[KM04] Kurihara T., Miyata N.: Modeling deformable human hands from medical images. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 357–366.

[KPGF07] Kubiak B., Pietroni N., Ganovelli F., Fratarcangeli M.: A robust method for real-time thread simulation. In *VRST '07: Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology* (2007), ACM, pp. 85–88.

[KTT13] Kim T., Tessendorf J., Thürey N.: Closest point turbulence for liquid surfaces. *ACM Transactions on Graphics 32*, 2 (April 2013), 15:1–15:13.

[LBOK13] Liu T., Bargteil A. W., O'Brien J. F., Kavan L.: Fast simulation of mass-spring systems. In *Proceedings of ACM SIGGRAPH Asia* (Hong Kong, 2013). *ACM Transactions on Graphics 32*, 6 (Nov. 2013), 209:1–7.

[LG98] Lin M. C., Gottschalk S.: Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces* (1998), pp. 37–56.

[MC10] Müller M., Chentanez N.: Wrinkle meshes. In *SCA'10: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010), Eurographics Association, pp. 85–92.

[MC11] Müller M., Chentanez N.: Solid simulation with oriented particles. *ACM Trans. Graph. 30*, 4 (July 2011), 92:1–92:10.

[MCG03] Müller M., Charypar D., Gross M.: Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), Eurographics Association, pp. 154–159.

[MGW01] Malzbender T., Gelb D., Wolters H.: Polynomial texture maps. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), Computer Graphics Proceedings, Annual Conference Series, pp. 519–528.

[MHHR07] Müller M., Heidelberger B., Hennix M., Ratcliff J.: Position based dynamics. *Journal of Visual Communication and Image Representation 18*, 2 (2007), 109–118.

[MHTG05] Müller M., Heidelberger B., Teschner M., Gross M.: Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478.

[Mir96] Mirtich B.: Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools 1*, 2 (February 1996), 31–50.

[MJC*08] Ma W.-C., Jones A., Chiang J.-Y., Hawkins T., Frederiksen S., Peers P., Vukovic M., Ouhyoung M., Debevec P.: Facial performance synthesis using deformation-driven polynomial displacement maps. In *Proceedings of ACM SIGGRAPH Asia, ACM Transactions on Graphics 27*, 5 (2008).

[MKC12] Müller M., Kim T.-Y., Chentanez N.: Fast simulation of inextensible hair and fur. In *VRIPHYS 12: Proceedings of 9th Workshop on Virtual Reality Interactions and Physical Simulations* (2012), Eurographics Association.

[MM13] Macklin M., Müller M.: Position based fluids. *ACM Transactions on Graphics 32*, 4 (July 2013), 104:1–104:12.

[Mon92] Monaghan J. J.: Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 1 (1992), 543–574.

[Mon94] Monaghan J. J.: Simulating free surface flows with SPH. Journal of. *Computational Physics 110*, 2 (February 1994), 399–406.

[MSJT08] Müller M., Stam J., James D., Thürey N.: Real time physics: Class notes. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes* (2008), ACM, pp. 88:1–88:90.

[MTV05] Magnenat-Thalmann N., Volino P.: From early draping to haute couture models: 20 years of research. *The Visual Computer 21* (2005), 506–519.

[Mül08] Müller M.: Hierarchical Position Based Dynamics. In *VRIPHYS 08: Proceedings of Fifth Workshop in Virtual Reality Interactions and Physical Simulations* (2008), F. Faure and M. Teschner (Eds.), Eurographics Association, pp. 1–10.

[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.

[ODC11] O'Brien C., Dingliana J., Collins S.: Spacetime vertex constraints for dynamically-based adaptation of motion-captured animation. In *SCA '11: Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), ACM, pp. 277–286.

[OH99] O'Brien J. F., Hodgins J. K.: Graphical modeling and animation of brittle fracture. 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. 137–146.

[PB88] Platt J. C., Barr A. H.: Constraints methods for flexible objects. In *SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques* (1988), ACM, pp. 279–288.

[Pro95] Provot X.: Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proceedings of Graphics Interface* (1995), W. A. Davis and P. Prusinkiewicz (Eds.), Canadian Human-Computer Communications Society, pp. 147–154.

[RJ07] Rivers A. R., James D. L.: FastLSM: fast lattice shape matching for robust real-time deformation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (2007), ACM, p. 82.

[RKN10] Rungjiratananon W., Kanamori Y., Nishita T.: Chain shape matching for simulating complex hairstyles. *Computer Graphics Forum 29*, 8 (2010), 2438–2446.

[RPC*10] Rohmer D., Popa T., Cani M.-P., Hahmann S., Sheffer A.: Animation wrinkling: Augmenting coarse cloth simulations with realistic-looking wrinkles. *ACM Transactions on Graphics 29*, 5 (2010), 157:1–157:8.

[SD92] Shoemake K., Duff T.: Matrix animation and polar decomposition. In *Proceedings of the Conference on Graphics Interface '92* (1992), Morgan Kaufmann Publishers Inc., pp. 258–264.

[SGdA*10] Stoll C., Gall J., de Aguiar E., Thrun S., Theobalt C.: Video-based reconstruction of animatable human characters. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 139:1–139:10.

[SGT09] Schmedding R., Gissler M., Teschner M.: Optimized damping for dynamic simulations. In *Proceedings of Spring Conference on Computer Graphics* (2009), pp. 205–212.

[SHZO07] Sengupta S., Harris M., Zhang Y., Owens J. D.: Scan primitives for GPU computing. In *Proceedings of the 22nd ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware* (2007), Eurographics Association, pp. 97–106.

[SKBK13] Schmitt N., Knuth M., Bender J., Kuijper A.: Multilevel Cloth Simulation using GPU Surface Sampling. In *VRIPHYS 13: Proceedings of 10th Workshop on Virtual Reality Interactions and Physical Simulations* (Lille, France, 2013), Eurographics Association, pp. 1–10.

[SOG08] Steinemann D., Otaduy M. A., Gross M.: Fast adaptive shape matching deformations. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), Eurographics Association, pp. 87–94.

[SSBT08] Stumpp T., Spillmann J., Becker M., Teschner M.: A Geometric Deformation Model for Stable Cloth Simulation. In *VRIPHYS 08: Proceedings of Fifth Workshop in Virtual Reality Interactions and Physical Simulations* (2008), F. Faure and M. Teschner (Eds.), Eurographics Association, pp. 39–46.

[SSH12] Seiler M., Spillmann J., Harders M.: Enriching coarse interactive elastic objects with high-resolution data-driven deformations. In *SCA '12: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), Eurographics Association, pp. 9–17.

[Sta09] Stam J.: Nucleus: Towards a unified dynamics solver for computer graphics. In *Proceedings of the IEEE International Conference on Computer-Aided Design and Computer Graphics* (2009), pp. 1–11.

[TBHF03] Teran J., Blemker S., Hing V. N. T., Fedkiw R.: Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), Eurographics Association, pp. 68–74.

[Tes04] Tessendorf J.: Simulating ocean water. *ACM SIGGRAPH 2004 Course Notes* (2004).

[TF88] Terzopoulos D., Fleischer K.: Deformable models. *The Visual Computer 4* (1988), 306–331.

[THMG04] Teschner M., Heidelberger B., Muller M., Gross M.: A versatile and robust model for geometrically complex deformable solids. In *CGI '04: Proceedings of the Computer Graphics International* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 312–319.

[TKH*05] Teschner M., Kimmerle S., Heidelberger B., Zachmann G., Raghupathi L., Fuhrmann A., Cani M.-P., Faure F., Magnenat-Thalmann N., Strasser W., Volino P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (Mar. 2005), 61–81.

[TPBF87a] 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* (1987), ACM, pp. 205–214.

[TPBF87b] Terzopoulos D., Platt J., Barr A., Fleischer K.: Elastically deformable models. In *Proceedings of SIGGRAPH 87, Computer Graphics 21* (1987), 205–214.

[vFTS06] von Funck W., Theisel H., Seidel H.-P.: Vector field based shape deformations. *ACM Transsactions on Graphics 25*, 3 (July 2006), 1118–1125.

[WBS*13] Weber D., Bender J., Schnoes M., Stork A., Fellner D.: Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26.

[WHRO10] Wang H., Hecht F., Ramamoorthi R., O'Brien J.: Example-based wrinkle synthesis for clothing animation. *ACM Transactions on Graphics 29*, 4 (July 2010), 107:1–107:8.

[WOR10] Wang H., O'Brien J., Ramamoorthi R.: Multi-resolution isotropic strain limiting. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 156:1–156:10.

[WXX*06] Wang Y., Xiong Y., Xu K., Tan K., Guo G.: A mass-spring model for surface mesh deformation based on shape matching. In *GRAPHITE '06: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia* (2006), ACM, pp. 375–380.

[ZBO13] Zurdo J. S., Brito J. P., Otaduy M. A.: Animating wrinkles by example on non-skinned cloth. *IEEE Transactions on Visualization and Computer Graphics 19*, 1 (2013).