COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1667

RESEARCH ARTICLE

Interactive constrained dynamics for rigid and deformable objects

Luca Vezzaro*, Davide Zerbato and Paolo Fiorini

Department of Computer Science, University of Verona Ca' Vignal 2, Strada Le Grazie, 15, 37134 Verona, Italy

ABSTRACT

Following the continuous increase in computational power of consumer hardware, interactive virtual environments have been recently enriched with more and more complex deformable objects. However, many physics engines are still very limited in the way they handle interacting rigid and deformable objects. This paper proposes a constraint-based approach to real-time simulation of coupled rigid and deformable objects capable of providing two-way interactions. Similar techniques have seen widespread usage for either rigid or deformable objects, but not for the simultaneous simulation of both. By extending such approaches, we show not only how interaction is possible but also how it can be performed at real-time rates. We address contact response and also show how to implement typical constraints to enforce limitations in the degrees of freedom and to enhance the dynamical properties of deformable objects. The method is easily integrated into existing physics engines that use similar constraint solvers and is independent on the kind of deformable object paradigm chosen. The provided simulation results show that the method is fast and effective in handling contacts between rigid and deformable objects and in simulating friction and other kinds of constraints. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS

constraints; deformable objects; real time

Supporting information may be found in the online version of this article at the publisher's web site.

***Correspondence**

Luca Vezzaro, Department of Computer Science, University of Verona Ca' Vignal 2, Strada Le Grazie, 15, 37134 Verona, Italy.
E-mail: elvezzaro@gmail.com

1. INTRODUCTION

The simulation of interacting rigid objects is often performed using velocity-based (also called impulse-based) dynamics and collision response schemes. Velocity-based approaches are simple to implement, are efficient, and can work well even when using large time steps. Non-rigid objects, in contrast, are often specified at the position (e.g., shape matching) or acceleration level (e.g., finite elements). Therefore, it is difficult to develop a general-purpose system that can handle two-way interaction between rigid and non-rigid objects, as the physical quantities involved are not homogeneous and it is not straightforward how they should interact.

Robust, accurate, and realistic collision response is the subject of most work in the area of cloth and deformable object simulation [1,2]. These approaches usually focus on accurate self-collision and one-way interaction with rigid objects, and while succeeding in doing so, they require great computational time. Full two-way coupling is also usually not modeled. One of the few interactive examples of two-way coupling is given in [3], while a non-interactive example is given in [4].

With the goal of achieving real-time simulation of *constrained* two-way interactions between rigid and deformable objects, we aimed at developing a fast and simple approach that could provide plausible deformable object behavior, without worrying too much about achieving absolute realism. The result is a fast and effective dynamical system capable of handling any number of interacting rigid and deformable objects. We ensure real-time performance by solving for constraints separately, in an iterative projected Gauss–Seidel (PGS) fashion, and achieve two-way interaction by employing a staggered [5] approach, alternating rigid and deformable object constraint solvers.

We use a robust and freely available physics engine [6] for collision detection and constraint solving of rigid objects and introduce a separate deformable object constraint solver implemented using the techniques described

Copyright © 2015 John Wiley & Sons, Ltd.

in~[7]. We specify deformable object constraints at the velocity level, not at the position level (as is usually the case [8]), observing how this formulation can also be very effective and how it makes for easy integration into existing velocity-based constraint solvers.

Because it requires very limited knowledge about the underlying deformable object paradigm, our approach can easily be made to support a multitude of force-based or position-based methods. We only assume deformable objects to be discrete and described by an explicit surface representation, which are reasonable requirements considering our targets are interactive simulations. We have evaluated the effectiveness of our approach by using mass–spring–damper meshes [9], co-rotational finite element models, position-based dynamics [3], and linear shape matching [10]. Results show how we are able to interactively tackle typical rigid–deformable contact scenarios (with friction), all the while providing plausible two-way interaction between objects.

2. RELATED WORK

The field of deformable objects simulation has reached a great level of maturity in the last few years, as summarized by the recent state-of-the-art report [8]. Undoubtedly, position-based dynamics is becoming the *de facto* standard in interactive simulation, owing to its simplicity, efficiency, and unconditional stability.

Force-based or implicit approaches [9,11] are still useful in some applications. A recent example is [12], where the authors proposed the use of mass–spring models for hair simulation: by using a sophisticated integration scheme, and thanks to strain limiting [2], stable simulation of stiff chains was made possible.

Early position-based approaches were introduced to model inextensible cloth [13,14]. These approaches, based on *relaxation* of constraints, were later extended in [3,15] to solve arbitrary constraints. Relaxation happens at the position level, thus producing discontinuous velocities and making interaction with rigid objects and friction modeling difficult. A purely velocity-based approach to inextensible cloth simulation that does not suffer from such limitations is given in [16]. Each pair of connected particles is kept at a constant distance by iteratively applying impulses to them.

None of the works discussed so far explicitly addressed two-way interaction between rigid and deformable objects. One of the first attempts at modeling two-way interaction is reported in [17]: by building deformable objects as aggregates of rigid components and using constrained rigid-body dynamics, the authors were able to approximate two-way interactions. More recently, Sifakis *et al*. [1] showed the so-called bindings to model deformable object interaction with rigid objects, but their approach does not achieve interactive rates, lacks a friction model, and can only handle anchoring constraints. Lenoir and Fonteneau [18] used an approach based on Lagrange multipliers to model two-way interaction, but it is limited to analytical deformable objects.

Recently, Deul *et al*. [19] extended the position-based dynamics approach to rigid objects. The need for a velocity solver to model friction and the use of a peculiar approach to collision response make for a quite involved framework, which also requires additional work to support two-way interaction.

All serious attempts at modeling the problem of two-way interaction involve constraints [8,14,20] of some form. In the simplest cases, the presence of equality constraints only leads to a system of linear equations that is easily solved using well-known methods [21,22]. Because non-penetration and friction cannot be modeled using equality constraints, the linearization of systems including these leads to what is called a linear complementarity problem (LCP) [7,23]. LCPs are commonly solved using PGS or projected Jacobi methods.

Reducing the scale at which the physical simulation is performed always produces an increase in accuracy and realism. Unfortunately, this also leads to non-interactive performance, so it is not always desirable or possible to do that. Macklin *et al*. [24] and Harada [25] show how performance can be greatly increased by exploiting the massively parallel architecture of graphics processing units (GPUs). While this is probably the future of physical simulation, GPU-based implementation is still difficult to implement and is less flexible, as all algorithms must be parallelized and all data structures have to be made into arrays and matrices.

3. METHOD OVERVIEW

We discretize the dynamical properties of a volume or a thin sheet of deformable material by tracking the physical state of a finite number of particles, which we call *nodes*. The state of the i-th node is given by its position $\tilde{\boldsymbol{x}}_i$, velocity $\tilde{\boldsymbol{v}}_i$, and acting force $\tilde{\boldsymbol{f}}_i$, all $\in \mathbb{R}^3$. We assume lumped masses; thus, we associate a mass $\tilde{m}_i$ to each node, which, in most practical cases, is uniformly distributed. For a rigid object j, we track the following quantities: $\boldsymbol{x}_j \in \mathbb{R}^3$, the location of its center of gravity; $\boldsymbol{q}_j \in \mathbb{H}$, a unit quaternion specifying its orientation in space; $\boldsymbol{v}_j \in \mathbb{R}^3$, the linear velocity of the object; $\boldsymbol{\omega}_j \in \mathbb{R}^3$, the vector-valued angular velocity; m_j, its mass; and $\boldsymbol{Q}_j$, the world-space inertia tensor. We advance the simulation in discrete time steps of duration h.

In the case of force-based deformable objects, we start by computing per-node internal forces. In all cases, external forces are added to the $\tilde{\boldsymbol{f}}_i$ variables. Position-based methods are supported by "converting" their output to velocities. Assume $\tilde{\boldsymbol{g}}_i$ to be the goal obtained by a position-based method from $\tilde{\boldsymbol{x}}_i^{t+h}$:

$$\tilde{\boldsymbol{x}}_i^{t+h} = \tilde{\boldsymbol{x}}_i + h\left(\tilde{\boldsymbol{v}}_i + h\tilde{m}_i^{-1}\tilde{\boldsymbol{f}}_i\right) \tag{1}$$

If we are using shape matching [10], we compute the updated velocity as

$$\tilde{\boldsymbol{v}}_i^{SM} = (1-\gamma)\left(\tilde{\boldsymbol{v}}_i + k\frac{\tilde{\boldsymbol{g}}_i - \tilde{\boldsymbol{x}}_i^{t+h}}{h}\right) \tag{2}$$

whereas, if we are using position-based dynamics [3], the updated velocity is

$$\tilde{\boldsymbol{v}}_i^{PD} = \gamma\tilde{\boldsymbol{v}}_i + (1-\gamma)\frac{\tilde{\boldsymbol{g}}_i - \tilde{\boldsymbol{x}}_i}{h} \tag{3}$$

where γ is a damping coefficient and k is the material stiffness. Positions are not explicitly changed; only node velocities are. This allows the velocity-based solver to compute the correct impulses accounting for contacts, friction, and other constraints.

For collision detection and rigid object dynamics, we choose to exploit the Bullet Physics [6] engine. Thanks to the iterative nature of the constraint-solving process, we are able to achieve two-way coupling by alternating a single iteration of Bullet's rigid object solver with an iteration of our solver.

A constraint is typically formulated as a function of positions and orientations:

$$\begin{aligned} C\left(\boldsymbol{x}_0, \boldsymbol{q}_0, \ldots, \boldsymbol{x}_{|R|-1}, \boldsymbol{q}_{|R|-1}, \tilde{\boldsymbol{x}}_0, \ldots, \tilde{\boldsymbol{x}}_{|N|-1}\right) = \\ C\left(X^t\right) = 0 \end{aligned} \tag{4}$$

where R is the set of all rigid objects and N the set of all deformable object nodes. X is a column vector containing all node and rigid object positions. The t superscript on X indicates the values are taken at time t.

Similar to X^t, we define the velocity of the system using the vector V^t, which is simply the time derivative of X^t for positions, whereas in place of the time derivative of quaternions, it contains the angular velocities.

Using this notation, the time derivative of a constraint function becomes

$$\dot{C}\left(X^t, V^t\right) = 0 \tag{5}$$

If we stack the s constraints in the system as a column of scalar-valued functions $\boldsymbol{C}$ and take a first-order time derivative, we obtain (from the chain rule)

$$\dot{\boldsymbol{C}} = JV^t = \boldsymbol{0} \tag{6}$$

where J is the s-by-$6n$ matrix of constraint Jacobians. Intuitively, (6) tells us that the only allowed velocities are orthogonal to all Jacobian rows. So the rows of J represent forbidden directions in the velocity space, and the constraint forces $\boldsymbol{F_c}$ must act along these forbidden direction to ensure (6) is always satisfied:

$$\boldsymbol{F_c} = J^T\boldsymbol{\lambda} \tag{7}$$

where $\boldsymbol{\lambda}$ is an (unknown) vector of impulsive constraint forces.

We illustrate our method with an example. If we want to constrain a single deformable object node i to a fixed location in space $\boldsymbol{y}$, we formulate the constraint as

$$C_{ex} = \frac{1}{2}\left\|\tilde{\boldsymbol{x}}_i - \boldsymbol{y}\right\|^2 = 0 \tag{8}$$

Because its derivative is $\dot{C}_{ex} = (\tilde{\boldsymbol{x}}_i - \boldsymbol{y})\cdot\tilde{\boldsymbol{v}}_i$, we only need $\boldsymbol{J}_{ex} = (\tilde{\boldsymbol{x}}_i - \boldsymbol{y})^T$, the Jacobian component for this constraint. Then, the constrained equations of motion for node i become

$$\tilde{m}_i\frac{d}{dt}\tilde{\boldsymbol{v}}_i = \boldsymbol{J}_{ex}^T\lambda_{ex} + \tilde{\boldsymbol{f}}_i \tag{9}$$

$$\boldsymbol{J}_{ex}\tilde{\boldsymbol{v}}_i = 0 \tag{10}$$

where the first expression comes from Newton's second law of motion (we used (7) for the right-hand side of (9), and the second expression comes from (6)). Using an explicit integration step for the velocities, we approximate the velocity's time derivative by $\frac{d}{dt}\tilde{\boldsymbol{v}}_i$ and by substituting the following in (9):

$$\tilde{m}_i\frac{\tilde{\boldsymbol{v}}_i^{t+h} - \tilde{\boldsymbol{v}}_i^t}{h} = \boldsymbol{J}_{ex}^T\lambda_{ex} + \tilde{\boldsymbol{f}}_i \tag{11}$$

where the superscript indicates velocity at a given time. We multiply both sides by $\boldsymbol{J}_{ex}$ and enforce the constraints to be satisfied at time $t+h$ using (10) to obtain

$$\frac{\boldsymbol{J}_{ex}\boldsymbol{J}_{ex}^T\lambda}{\tilde{m}_i} = -\boldsymbol{J}_{ex}\left(\frac{\tilde{\boldsymbol{v}}_i^t}{h} + \frac{\tilde{\boldsymbol{f}}_i}{\tilde{m}_i}\right) \tag{12}$$

which can be solved for λ. Note that position error has disappeared from the equation and we can constrain only the velocity of the node not to change along $\boldsymbol{J}_{ex}$, regardless of the value of C_{ex}. This causes the so-called constraint drift, which is dealt with by using the Baumgarte scheme [26], introducing a constraint *bias* into the system. This term is directly proportional to constraint error and is added into (6), obtaining

$$\dot{C} = JV^t = -\beta C \tag{13}$$

where β is chosen experimentally and is subject to the stability condition $\beta \leq 1/h$ [27]. In our experiments, we have used a value of $\beta = 0.1/h$.

We also need to limit the range of λ in (12) to accurately model friction and non-contact constraints, obtaining the following:

$$\begin{aligned} \frac{\boldsymbol{J}_{ex}\boldsymbol{J}_{ex}^T\lambda}{\tilde{m}_i} = \frac{\zeta_{ex}}{h} - \boldsymbol{J}_{ex}\left(\frac{\tilde{\boldsymbol{v}}_i^t}{h} + \frac{\tilde{\boldsymbol{f}}_i}{\tilde{m}_i}\right) \\ \lambda^- \leq \lambda \leq \lambda^+ \end{aligned} \tag{14}$$

where

$$\zeta_{ex} = -\beta\left\|\tilde{\boldsymbol{x}}_i - \boldsymbol{y}\right\| \tag{15}$$

The set of all s constraints in the system can be formalized as a linear complementarity problem [7]:

$$A\boldsymbol{\lambda} + \boldsymbol{b} \geq \mathbf{0}$$
$$\boldsymbol{\lambda}^- \leq \boldsymbol{\lambda} \leq \boldsymbol{\lambda}^+ \tag{16}$$

where

$$\begin{aligned} A &= hJM^{-1}J^T \\ \boldsymbol{b} &= -\boldsymbol{\zeta} + J\left(V^t + hM^{-1}\boldsymbol{F}_{ext}\right) \end{aligned} \tag{17}$$

and $\boldsymbol{\lambda}$ is the vector of unknown impulsive constraint forces. J is a (sparse) matrix of Jacobians, with a row for each constraint and non-zero entries only in correspondence to velocities of objects or nodes involved in that constraint. $\boldsymbol{\zeta}$ is the vector of constraint biases. F is the vector of forces and torques acting in the system. M is a (block-diagonal) matrix containing the node and rigid object's masses and world-space inertia tensors:

$$M = \begin{bmatrix} m_0 I & & & & & \\ & \boldsymbol{Q}_0 & & & & \\ & & \ddots & & & \\ & & & \tilde{m}_0 I & & \\ & & & & \ddots & \\ & & & & & \tilde{m}_{|N|-1} I \end{bmatrix} \tag{18}$$

where I is a 3-by-3 identity matrix.

To ensure real-time performance, our solver uses an iterative PGS approach. Convergence is not guaranteed with this solver, but in most practical cases, each iteration of the solver becomes closer to the optimal solution [7]. By tuning the iteration number, the simulation can be made more interactive or more accurate (in the limits of the integration method used).

The following sections will discuss the details of constraint specifications and solutions. We will not perform an in-depth analysis of our collision detection system, as any algorithm could be used, provided it can supply a contact point location, penetration amount, and contact normal. In our case, we represent the objects' surface using triangle meshes and keep an axis-aligned bounding box tree of faces, using the Gilbert–Johnson–Keerthi algorithm [28] for contact point generation.

4. CONSTRAINTS

We exploit the sparse structure of J and M to speed up the constraint-solving process, as described in [7]. We write a generic constraint as

$$\dot{C} = \sum_{rb \in S_r} J_{rb} \begin{bmatrix} \boldsymbol{v}_{rb} \\ \boldsymbol{\omega}_{rb} \end{bmatrix} + \sum_{def \in S_d} \tilde{J}_{def} \tilde{v}_{def} \tag{19}$$

where S_r and S_d are fields of the *Constraint* data structure. In C-like syntax,

```
struct Constraint {
 Set<SolverDeformable> S_d
 Set<SolverRigid> S_r
 float ζ, λ^0, λ, λ^-, λ^+
}
struct SolverDeformable {
 DeformableObject d
 int i // the node
 Vector3 J̃
}
struct SolverRigid {
 RigidObject r
 float m^-1 // inverse mass
 Vector3 J_x, J_q
 Vector3 t̂ // torque axis
}
```

We set $\lambda = \lambda^0 = 0$ during initialization. We arrange the data required for each object involved in a constraint into separate data structures, as different constraints handle a different amount of rigid and deformable object nodes. The solver is very flexible, so it makes sense to specify all constraints in a uniform manner so as to keep the solver simple and compact and simplify the addition of new kinds of constraints.

We are aware some of the following constraints do not represent the state of the art, but we think they are the most effective for introducing the solver. It will be easy to integrate other constraints into the system once the basic principles involved are understood.

4.1. Anchor Constraints

To anchor a node $\tilde{\boldsymbol{x}}_i$ to a fixed world-space position $\boldsymbol{y}$, we use the previously defined constraint (8):

$$C_{fix} = \frac{1}{2} \|\tilde{\boldsymbol{x}}_i - \boldsymbol{y}\|^2 = 0 \tag{20}$$

which gives us Jacobians and bias: $\tilde{J}_i = (\tilde{\boldsymbol{x}}_i - \boldsymbol{y})^T$, $\zeta = -\beta \|\tilde{\boldsymbol{x}}_i - \boldsymbol{y}\|$. Constraint forces need not be limited in this case, so $\lambda^+ = \infty$, $\lambda^- = -\infty$.

Things become more complicated if we replace $\boldsymbol{y}$ with a point $\boldsymbol{r}$ fixed in a rigid object's local coordinate system:

$$C_{fixrigid} = \frac{1}{2} \left\|\boldsymbol{x}_j + \boldsymbol{q}_j * \boldsymbol{r}_j - \tilde{\boldsymbol{x}}_i\right\|^2 = \frac{1}{2} \|\boldsymbol{d}\|^2 = 0 \tag{21}$$

We use $\boldsymbol{q}_j * \boldsymbol{r}_j$ to represent application of the transformation stored in $\boldsymbol{q}_j$ to $\boldsymbol{r}_j$.

From (21), we derive Jacobians $\tilde{J} = -\boldsymbol{d}^T$, $J_{\boldsymbol{x}} = \boldsymbol{d}^T$, and $J_{\boldsymbol{q}} = ((\boldsymbol{q}_j * \boldsymbol{r}_j) \times \boldsymbol{d})^T$. This time, the Constraint data structure holds a SolverRigid and a SolverDeformable

instance. Again, $\lambda^+ = \infty$, $\lambda^- = -\infty$. The torque axis $\hat{\boldsymbol{t}}$ is given by

$$\hat{\boldsymbol{t}} = Q_j^{-1}\left(\boldsymbol{d} \times \left(\boldsymbol{q}_j * \boldsymbol{r}_j\right)\right) \tag{22}$$

If there are multiple anchors acting on a rigid object, multiple impulses will be computed. We apply the mean impulse to the object to ensure the number of anchors does not influence the strength of the overall constraint force.

4.2. Inextensibility Constraints

Instead of computing ad hoc impulses to constrain two nodes to stay at a given distance [16], we can model the problem as a constraint. We want to preserve a rest distance L between two nodes:

$$C_{inext} = \frac{1}{2}\left(\left\|\tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i\right\|^2 - L^2\right) = 0 \tag{23}$$

The Jacobians are $\tilde{J}_i = -(\tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i)^T$, $\tilde{J}_j = (\tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i)^T$. The rest length becomes part of the bias: $\zeta = -\beta(L - \|\tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i\|)$.

Theoretically, the impulsive constraint force λ in this case should not need to be limited. In practice, by tuning the λ^- and λ^+ parameters, it is possible to obtain various kinds of interesting behaviors. For example, we have the following:

- $\lambda^+ = \lambda_{\max}$, $\lambda^- = -\lambda_{\max}$: inextensible material, with limited stretch resistance. Higher values of $\lambda_{\max}$ increase the resistance to stretching, but numerical stability might suffer.
- $\lambda^+ = \lambda_{\max}$, $\lambda^- = 0$: material that resists compression but not stretching, useful for enforcing volume conservation properties.
- $\lambda^+ = 0$, $\lambda^- = -\lambda_{\max}$: material that resists stretching but not compression, useful for damping excessive deformations in non-stiff objects.

4.3. Non-penetration Constraints

A fast and simple approach to collision response is to introduce non-penetration constraints that push objects away from each other to eliminate the penetration detected during the collision detection step. This approach allows reuse of a wealth of existing implementations for collision detection and rigid object-only collision response, as we did in our case. The downside is that tunneling artifacts may happen and that correct and robust collision response is difficult if not impossible in extreme scenarios. We choose to allow for some tunneling in order to keep collision detection time manageable. As discussed earlier, our constraint solver only needs to handle contacts between rigid and deformable (deformable–rigid) objects or between deformable (deformable–deformable) objects.

For a deformable–rigid contact involving rigid object i, we have $\boldsymbol{x}_{cp}$, the contact point; $\boldsymbol{n}_{cp}$, the contact normal, pointing from the rigid to soft objects; and ϵ, the penetration amount (which is negative in case of inter-penetration and zero for sliding contacts). Before being able to formulate the non-penetration constraint, we need to determine on which face of the deformable object $\boldsymbol{x}_{cp}$ lies. This information is usually provided by the collision detection algorithm. Let us call this face $f = (a, b, c)$. We determine the barycentric coordinates of $\boldsymbol{x}_{cp}$ with respect to f and use these to compute the interpolated velocity and force at the contact point [2]. The mass at the contact point is not interpolated: instead, we split the total mass of the deformable object across contact points. We use these quantities to create a *dummy* node for each contact point (we refer to the node using the subscript notation, e.g., $\tilde{\boldsymbol{x}}_{dummy}$). We keep dummy nodes at the end of our list of nodes and discard them once done with the constraint-solving process. The non-penetration constraint function for a deformable–rigid contact is

$$C_{pen} = \left(\tilde{\boldsymbol{x}}_{dummy} - \boldsymbol{x}_i - \boldsymbol{q}_i * \boldsymbol{r}_i\right) \cdot \boldsymbol{n}_{cp} = 0 \tag{24}$$

where $\boldsymbol{r}_i$ is the contact point in the rigid object's coordinate frame. The time derivative of the constraint function is

$$\dot{C}_{pen} = \left(\tilde{\boldsymbol{v}}_{dummy} - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times (\boldsymbol{q}_i * \boldsymbol{r}_i)\right) \cdot \boldsymbol{n}_{cp} = 0 \tag{25}$$

which gives Jacobians $\tilde{J} = \boldsymbol{n}_{cp}^T$, $J_{\boldsymbol{x}} = -\boldsymbol{n}_{cp}^T$, and $J_{\boldsymbol{q}} = -((\boldsymbol{q}_i * \boldsymbol{r}_i) \times \boldsymbol{n}_{cp})^T$. The bias is directly proportional to penetration, $\zeta = \beta\epsilon$, and we want the impulsive constraint force only to push the bodies apart, so $\lambda^+ = \infty$ and $\lambda^- = 0$. The torque axis $\hat{\boldsymbol{t}}$ is given by

$$\hat{\boldsymbol{t}} = Q_i^{-1}\left(\boldsymbol{n}_{cp} \times (\boldsymbol{q}_i * \boldsymbol{r}_i)\right) \tag{26}$$

The Constraint structure can then be filled with a SolverRigid and a SolverDeformable instance containing the values we have just computed.

Deformable–deformable contact is handled similarly, but two dummy nodes need to be added. Consider two deformable objects A and B and respective dummy nodes $\tilde{\boldsymbol{x}}_{dummy_A}$ and $\tilde{\boldsymbol{x}}_{dummy_B}$. The Jacobians are $\tilde{J}_A = -\boldsymbol{n}_{cp}^T$ and $\tilde{J}_B = \boldsymbol{n}_{cp}^T$. ζ is the same as before, as are λ^+ and λ^-. We just need to construct a Constraint structure by building the two SolverDeformable instances. Self-collision can be handled similarly.

Once the constraint solver has finished, a λ value for the contact constraint will be provided, but that is relative to one or more dummy nodes. To obtain λ values that can be applied to the object's nodes, we simply consider a rigid motion of the face containing the dummy node, so we accumulate the λ values from each dummy node to the corresponding face vertices (a, b, c) and average them. The averaging step is important as usually there are multiple contact constraints acting on a single node. Moreover, the overall impulsive force for a time step should be their

mean, not their sum; otherwise, constraint forces will be proportional to the amount of contact points. Of course, fixed nodes should be ignored by the accumulation process.

Similarly, impulses are averaged for rigid objects to ensure the number of contact points does not influence the intensity of the applied impulses.

4.4. Friction Constraints

We can exploit the velocity-based nature of the constraint solver to develop a simplified friction model, which is effective although not completely accurate (as explained in [27]).

Friction acts on a plane that is orthogonal to the contact normal, so we build an orthonormal basis $(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{n}_{cp})$, where $\boldsymbol{u}$ and $\boldsymbol{w}$ are any two orthonormal vectors specifying the plane where friction is acting. We derive these from the relative velocity at the contact point $\Delta\boldsymbol{v}$:

$$\boldsymbol{u} = \frac{\Delta\boldsymbol{v} \times \boldsymbol{n}_{cp}}{\left\| \Delta\boldsymbol{v} \times \boldsymbol{n}_{cp} \right\|} \tag{27}$$

$$\boldsymbol{w} = \frac{\boldsymbol{u} \times \boldsymbol{n}_{cp}}{\left\| \boldsymbol{u} \times \boldsymbol{n}_{cp} \right\|} \tag{28}$$

To minimize numerical errors, we check the vector lengths before performing the normalization in (27), and if it is close to zero, no friction constraints are created, as the relative velocity would be close to zero and/or orthogonal to the friction plane.

With the same information used in contact response, we can directly specify the two friction constraints at the velocity level. For the rigid–deformable case, we have

$$\begin{aligned} \dot{C}_u &= \Delta\boldsymbol{v} \cdot \boldsymbol{u} = 0 \\ \dot{C}_w &= \Delta\boldsymbol{v} \cdot \boldsymbol{w} = 0 \\ \Delta\boldsymbol{v} &= \tilde{\boldsymbol{v}}_{dummy} - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times (\boldsymbol{q}_i * \boldsymbol{r}_i) \end{aligned} \tag{29}$$

giving us Jacobians $\tilde{J} = \boldsymbol{u}^T$, $J_{\boldsymbol{x}} = -\boldsymbol{u}^T$, and $J_{\boldsymbol{q}} = (\boldsymbol{u} \times \boldsymbol{r}_i)^T$, for $\dot{C}_u$. Values for $\dot{C}_w$ are derived similarly. The torque axes $\hat{\boldsymbol{t}}_{\boldsymbol{u}}$ and $\hat{\boldsymbol{t}}_{\boldsymbol{w}}$ are given by

$$\begin{aligned} \hat{\boldsymbol{t}}_{\boldsymbol{u}} &= Q_i^{-1} \left(\boldsymbol{u} \times \left(\boldsymbol{x}_{cp} - \boldsymbol{x}_i \right) \right) \\ \hat{\boldsymbol{t}}_{\boldsymbol{w}} &= Q_i^{-1} \left(\boldsymbol{w} \times \left(\boldsymbol{x}_{cp} - \boldsymbol{x}_i \right) \right) \end{aligned} \tag{30}$$

Two Constraint structures can then be filled with SolverRigid and SolverDeformable instances containing the values we have just computed.

Likewise, deformable–deformable contact is given by

$$\begin{aligned} \dot{C}_u &= \Delta\boldsymbol{v} \cdot \boldsymbol{u} = 0 \\ \dot{C}_w &= \Delta\boldsymbol{v} \cdot \boldsymbol{w} = 0 \\ \Delta\boldsymbol{v} &= \tilde{\boldsymbol{v}}_{dummy_B} - \tilde{\boldsymbol{v}}_{dummy_A} \end{aligned} \tag{31}$$

giving us Jacobians $\tilde{J}_A = -\boldsymbol{u}^T$ and $\tilde{J}_B = \boldsymbol{u}^T$, for $\dot{C}_u$. Values for $\dot{C}_w$ are derived similarly. The friction constraint functions have no bias, so $\zeta = 0$.

We approximate the pressure acting on the contact point using

$$\rho = m_{cp} \left| \boldsymbol{g} \cdot \boldsymbol{n}_{cp} \right| \tag{32}$$

where m_{cp} is the contact point "mass" and $\boldsymbol{g}$ is the gravity acceleration. $m_{cp} = \tilde{m}_{dummy} + m_i / N_i$ for deformable–rigid contact (N_i is the amount of contact points between the current deformable object and rigid object i), while $m_{cp} = \tilde{m}_{dummy_A} + \tilde{m}_{dummy_B}$ for deformable–deformable contact. By setting $\lambda^+ = -\lambda^- = \mu\rho$, the maximum allowed friction force is directly proportional to pressure, improving realism. μ can be seen as the dynamic friction coefficient. Because interacting materials may be heterogeneous, we combine friction coefficients from different materials by multiplying them, as that is the approach used in Bullet.

The λ values computed by the solver for friction constraints are then distributed and averaged in the same way we do for non-penetration constraints.

4.5. Bending Constraints

Bending constraints [3,29] are usually applied to cloth simulations to increase the smoothness of its behavior and avoid excessive folding. The model by Kelager *et al*. [29] is easily integrated, as their position-based constraint function can be differentiated in time to obtain a constraint on velocities. Unfortunately, numerical concerns arise when adjacent faces have similar normals (which is often the case).

We use instead an angle-based constraint: given an edge $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$ and the two adjacent faces $(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_L)$ and $(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_R)$ with initial normals $\boldsymbol{n}_L$ and $\boldsymbol{n}_R$, respectively, our goal is to keep the angle α between the two normals close to a rest angle α^0. For our approach to work, we assume the angle before constraint application to be close to the rest angle; otherwise, the constraint may enforce a "reversed" state.

With this assumption in mind, we can build a simple and clearly approximated constraint function that works pretty well in practice:

$$C_{bend} = (\tilde{\boldsymbol{x}}_L - \tilde{\boldsymbol{x}}_C) \cdot (\tilde{\boldsymbol{x}}_R - \tilde{\boldsymbol{x}}_C) - C^0_{bend} = 0 \tag{33}$$

where $\tilde{\boldsymbol{x}}_C = (\tilde{\boldsymbol{x}}_i + \tilde{\boldsymbol{x}}_j)/2$ and C^0_{bend} is the initial value of the dot product in C_{bend}, with the object at rest. Setting $\tilde{\boldsymbol{x}}_D = (\tilde{\boldsymbol{x}}_L + \tilde{\boldsymbol{x}}_R)/2$ and differentiating with respect to time, we obtain

$$\begin{aligned} \dot{C}_{bend} = {} & \tilde{\boldsymbol{v}}_L \cdot (\tilde{\boldsymbol{x}}_R - \tilde{\boldsymbol{x}}_C) + \tilde{\boldsymbol{v}}_R \cdot (\tilde{\boldsymbol{x}}_L - \tilde{\boldsymbol{x}}_C) \\ & - \tilde{\boldsymbol{v}}_i \cdot (\tilde{\boldsymbol{x}}_D - \tilde{\boldsymbol{x}}_C) - \tilde{\boldsymbol{v}}_j \cdot (\tilde{\boldsymbol{x}}_D - \tilde{\boldsymbol{x}}_C) = 0 \end{aligned} \tag{34}$$

The Jacobians are trivially determined in this case: $\tilde{J}_L = (\tilde{\boldsymbol{x}}_R - \tilde{\boldsymbol{x}}_C)$, $\tilde{J}_R = (\tilde{\boldsymbol{x}}_L - \tilde{\boldsymbol{x}}_C)$, $\tilde{J}_i = -(\tilde{\boldsymbol{x}}_D - \tilde{\boldsymbol{x}}_C)$, and $\tilde{J}_j = -(\tilde{\boldsymbol{x}}_D - \tilde{\boldsymbol{x}}_C)$. For the bias, we can use the position-based error $\zeta = -\beta \left| \boldsymbol{n}_L \cdot \boldsymbol{n}_R - \boldsymbol{n}^0_L \cdot \boldsymbol{n}^0_R \right|$, where

$$\begin{aligned} \boldsymbol{n}_R &= \left(\tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i \right) \times (\tilde{\boldsymbol{x}}_R - \tilde{\boldsymbol{x}}_i) \\ \boldsymbol{n}_L &= (\tilde{\boldsymbol{x}}_L - \tilde{\boldsymbol{x}}_i) \times \left(\tilde{\boldsymbol{x}}_j - \tilde{\boldsymbol{x}}_i \right) \end{aligned} \tag{35}$$

and $\boldsymbol{n}_L^0$ and $\boldsymbol{n}_R^0$ are their initial value at rest. Here is where the reversed state issue [29] should be addressed, but this is outside the focus of this work. In general, $\lambda^- = -\lambda^+$, and this value determines the strength of the constraint. Each bending constraint can be created by filling out four SolverDeformable instances, one for each node involved.

4.6. Solver

The impulsive forces applied to nodes and rigid objects by the constraint solver are stored separately from the forces computed by the underlying deformable object model. For this reason, we store impulsive forces applied to the i-th node inside $\tilde{\boldsymbol{I}}_i$ and impulsive forces and torques applied to a rigid object into $\boldsymbol{I_x}$ and $\boldsymbol{I_q}$, respectively. The solver is initialized by creating dummy nodes required for contact and friction processing, by calculating constraint biases, and eventually by warm-starting constraints. Warm starting is performed by initializing the constraint's λ using the value computed in the previous time step, λ^0. If this is performed, the impulse accumulation variables need to be initialized by adding the contribution of each constraint. Otherwise, they start at zero. Warm starting usually results in the need for less iterations of the solver but can also result in an excess of energy in the system, thus causing instabilities. The contribution of a single SolverDeformable entry is given by

$$\Delta\tilde{\boldsymbol{I}} = h\lambda^0\tilde{m}^{-1}\tilde{\boldsymbol{J}}^T \tag{36}$$

while the contribution of a SolverRigid entry is

$$\begin{aligned}\Delta\boldsymbol{I_x} &= h\lambda^0 m^{-1}\boldsymbol{J_x}\\ \Delta\boldsymbol{I_q} &= h\lambda^0\hat{\boldsymbol{t}}\end{aligned} \tag{37}$$

The solver in Algorithm 1 uses these equations too. For the most part, the PGS algorithm is the same used in [7]. Once the constraint solver is done iterating, the dummy nodes' impulses are distributed and averaged across faces, and external and internal forces' contributions are applied to the deformable objects.

4.7. Time Integration

Once the constraint solver is done, we advance the state of the system using a semi-implicit Euler scheme (also called symplectic Euler). For a deformable object,

$$\begin{aligned}\tilde{\boldsymbol{v}}_i^{t+h} &= \tilde{\boldsymbol{v}}_i^t + \left(\tilde{\boldsymbol{I}}_i + h\tilde{m}^{-1}\tilde{\boldsymbol{f}}_i\right)\\ \tilde{\boldsymbol{x}}_i^{t+h} &= \tilde{\boldsymbol{x}}_i + h\tilde{\boldsymbol{v}}_i{}^{t+h}\end{aligned} \tag{38}$$

Algorithm 1 A single iteration of the constraint solver

Input: C: set of constraints
for each $c \in C$ **do**
  $d \leftarrow 0$
  for each $rb \in c.S_r$ **do**
    $d \leftarrow d + rb.m^{-1} rb.\boldsymbol{J_x} \cdot rb.\boldsymbol{J_x}^T$
    $d \leftarrow d + rb.\boldsymbol{J_q} \cdot Q_{rb.r}{}^{-1} rb.\boldsymbol{J_q}^T$
  end for
  for each $def \in c.S_d$ **do**
    $d \leftarrow d + def.m^{-1} def.\tilde{\boldsymbol{J}} \cdot def.\tilde{\boldsymbol{J}}^T$
  end for
  $d \leftarrow 1/d$
  $\Delta\lambda \leftarrow dc.\zeta$
  for each $rb \in c.S_r$ **do**
    $\Delta\lambda \leftarrow \Delta\lambda - \frac{d}{h}\left(rb.\boldsymbol{J_x} \cdot \boldsymbol{I_{x_{rb.r}}}\right)$
    $\Delta\lambda \leftarrow \Delta\lambda - \frac{d}{h}\left(rb.\boldsymbol{J_q} \cdot \boldsymbol{I_{q_{rb.r}}}\right)$
  end for
  for each $def \in c.S_d$ **do**
    $\Delta\lambda \leftarrow \Delta\lambda - \frac{d}{h}\left(rb.\tilde{\boldsymbol{J}} \cdot \tilde{\boldsymbol{I}}_{\boldsymbol{def.d_{def.i}}}\right)$
  end for
  $c.\lambda^0 \leftarrow c.\lambda$
  $c.\lambda \leftarrow c.\lambda^0 + \Delta\lambda$
  $c.\lambda \leftarrow clamp\left(c.\lambda, c.\lambda^-, c.\lambda^+\right)$
  $\Delta\lambda \leftarrow c.\lambda - c.\lambda^0$
  for each $rb \in c.S_r$ **do**
    $\boldsymbol{I_{x_{rb.r}}} \leftarrow \boldsymbol{I_{x_{rb.r}}} + h\Delta\lambda rb.m^{-1} rb.\boldsymbol{J_x}$
    $\boldsymbol{I_{q_{rb.r}}} \leftarrow \boldsymbol{I_{q_{rb.r}}} + h\Delta\lambda rb.\hat{\boldsymbol{t}}$
  end for
  for each $def \in c.S_d$ **do**
    $obj \leftarrow def.d$
    $node \leftarrow def.i$
    $\tilde{\boldsymbol{I}}_{\boldsymbol{obj_{node}}} \leftarrow \tilde{\boldsymbol{I}}_{\boldsymbol{obj_{node}}} + h\Delta\lambda\tilde{m}^{-1}_{obj_{node}} def.\tilde{\boldsymbol{J}}$
  end for
end for

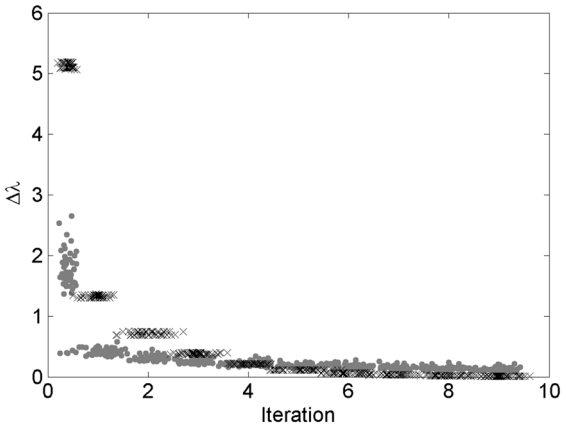

Figure 1. Evolution of $\Delta\boldsymbol{\lambda}$ (Root-Mean-Square) as a function of iteration. Values have been slightly perturbed along the horizontal axis to improve their visibility. The dots are the root mean square values of $\Delta\boldsymbol{\lambda}$ for a cloth interacting with a sphere (Figure 3), while the crosses are for a jelly–chocolate–dish interaction (Figure 2).

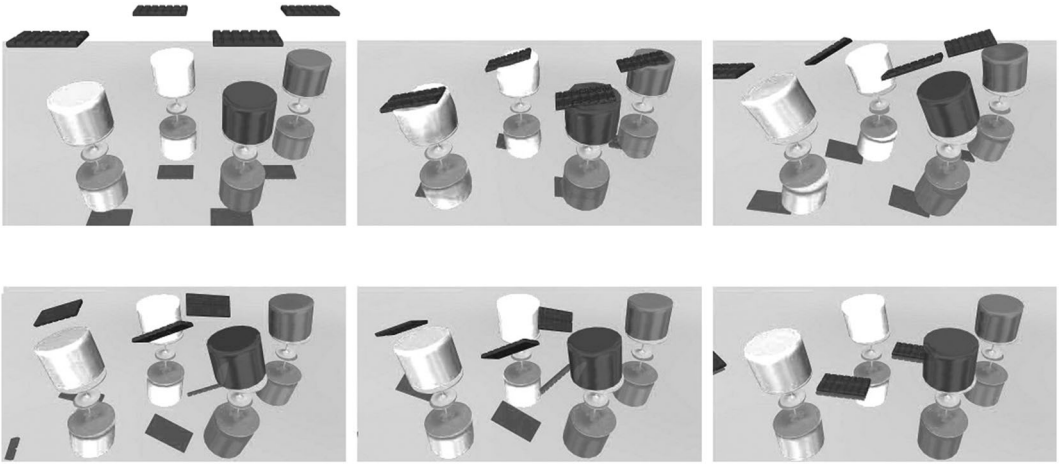

Figure 2. Two-way interaction between solid jelly cylinders, a rigid chocolate bar, and a rigid dish. The lower left jelly is mass–spring based, the lower right is based on co-rotational finite element methods, the upper left is with the use of position-based dynamics and the upper right is with the use of shape matching. The simulation consists of 1720 nodes, 2296 faces, 6784 springs, and 1701 tetrahedra. Constraint count reached7595, of which 3115 were inextensibility constraints and 861 bending constraints. The time required for stepping by 20 milliseconds took 6 milliseconds on average and 14 milliseconds in the worst case.

For rigid objects, a typical approach is given by

$$\begin{aligned} \boldsymbol{v}^{t+h} &= \boldsymbol{v}^t + \boldsymbol{I}_{\boldsymbol{x}} \\ \boldsymbol{\omega}^{t+h} &= \omega^t + \boldsymbol{I}_{\boldsymbol{q}} \\ \boldsymbol{x}^{t+h} &= \boldsymbol{x} + h\boldsymbol{v}_i{}^{t+h} \\ \hat{\boldsymbol{q}}^{t+h} &= \boldsymbol{q}^t + \begin{bmatrix} 0 \\ \boldsymbol{\omega}^{t+h} \end{bmatrix}^T \boldsymbol{q}^t \frac{h}{2} \end{aligned} \tag{39}$$

where $\hat{\boldsymbol{q}}^{t+h}$ in (39) denotes a non-unit quaternion, which should be normalized. This integration scheme is easy to implement and allows for straightforward linearization of the constraint-solving problem, but note how it is not unconditionally stable, especially when applied to most force-based methods for simulating deformable objects.

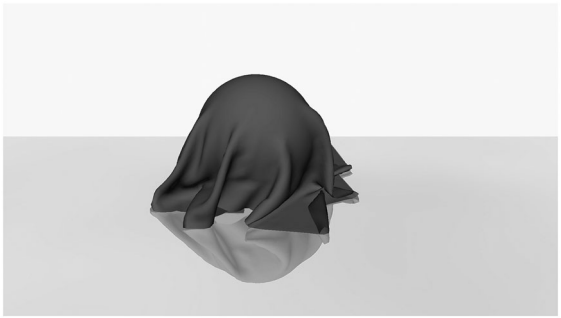

Figure 3. Real-time interaction between a mass–spring cloth and a rigid sphere. A similar scenario took 2 minutes per frame to simulate a decade ago [2]. The cloth is built of 51 × 51 nodes, with 5000 inextensibility and 7400 bending constraints. The maximum constraint count reached in this scenario was 20 492, with collision detection taking about 45% of simulation time.

5. RESULTS

A detailed analysis on stability, convergence, and performance of the solver is already available in [7]. Figure 1, along with satisfying visual results should serve as proof of the correct implementation of the solver. It is interesting to notice the convergence rate, which is very high in the first iterations.

Figure 2 depicts a two-way interaction between solid jelly cylinders, a rigid chocolate bar, and a rigid dish. The lower left jelly is mass–spring based, the lower right is based on co-rotational FEMs, the upper left is using position-based dynamics, and the upper right is using shape matching. Very similar visual results were obtained, in spite of the substantial differences between the deformable object paradigms used. Also observe how the two-way interaction with the dish and the chocolate bar is present in all four cases.

The impact of using the proposed deformable object-specific constraint is clearly demonstrated in Figure 3. Notice how bending constraints give a feeling of stiffness and thickness to the cloth, improving its realism.

Anchor constraints are showcased in Figure 4, along with other interesting effects. Two teapots of different masses (the one on the right is heavier) interact with each other through the rope they are attached to using anchor constraints. Friction between the rope and the constrained wheel is causing it to spin in the expected

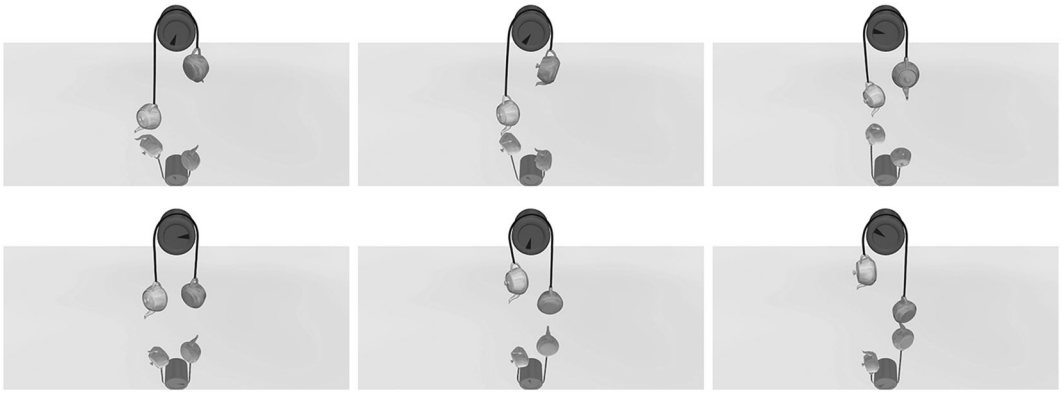

Figure 4. Anchor constraints at work: two teapots of different masses (the one on the right is heavier) interact through a constraint-based rope. The friction model makes sure the rope's movement is causing the rigid wheel to spin in the expected way. The rope is modeled as a "strip" composed of 50 nodes and 48 faces. One hundred and twenty-one inextensibility constraints and 47 bending constraints are used to model the rope's dynamics.

direction; meanwhile, the heavier teapot tries to "drag" the other one. Performance evaluations were performed in all cases. Real-time simulation was possible up to about 30 000 constraints, with collision detection taking almost half of CPU time in the more demanding scenarios. Figures 5 and 6 show constraint-solving time as a function of constraint count. As already pointed out in several sources, an iteration of this constraint solver has a linear cost with respect to constraint count.

All simulations were run with $h = 20$ milliseconds, using two iterations for the rigid object solver and two iterations for the deformable object solver. Timing and performance evaluation were performed using a 3.4-GHz Core i7. Our solver and collision detection code is single threaded at the moment, and therefore, the multi-core architecture is not exploited.

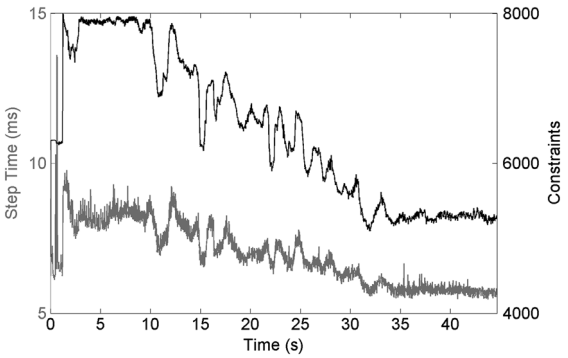

Figure 5. Constraint-solving time as a function of constraint count for the cloth simulation depicted in Figure 3.

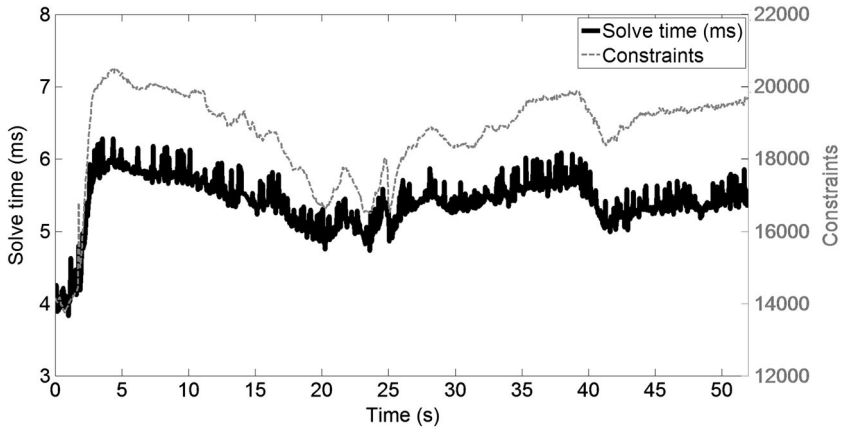

Figure 6. Constraint-solving time as a function of constraint count for a jelly–chocolate–dish interaction as depicted in Figure 2. Overall, the mean breakdown of a time step is 43% force and position dynamics computations, 32% collision detection, and 24% constraint solving. Time integration and rigid-body simulation took negligible time.

6. CONCLUSIONS AND FUTURE WORK

We have shown how to obtain two-way coupling between rigid and deformable objects by running a stand-alone constraint solver for deformable objects alongside an impulse-based physics engine. Our approach makes very few assumptions about the kind of deformable objects used, and the result is a system that can work with a multitude of deformable object simulation paradigms.

By using a generic constraint solver to achieve two-way coupling, we are able to simulate not only free deformable objects but also constrained ones. Experimental results show how a number of different scenarios involving multiple rigid and deformable objects could be simulated at interactive rates.

The contact model is undoubtedly one of the major weaknesses of our current implementation, as tunneling and jittering may happen when the time step is increased and/or objects are subject to strong forces. For these reasons, we are exploring the possibility of integrating a more advanced contact model such as the ones described in [30,31]. We are also considering techniques for contact decimation and caching as to increase performance and robustness in contact handling.

Widespread usage of deformable objects in real-time simulation and games is still limited by strict performance requirements, and the proposed method may not be fast enough for such applications. While experts in code optimization will surely be able to surpass the performance figures of our implementation, it is hard to judge by how much. Performance increases may be obtained by using level-of-detail/multi-resolution techniques [32], at the cost of accuracy. Future work will consider such techniques, but we point out how parallel implementations of the PGS solver, similar to the one proposed in [23], may save the trouble of developing involved data structures and algorithms.

ACKNOWLEDGEMENTS

This work was supported by the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement 248960 (Patient Safety in Robotic Surgery).

REFERENCES

1. Sifakis E, Der KG, Fedkiw R. Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation,* SCA'07. Eurographics Association, Aire-la-Ville, 2007; 73–80.
2. Bridson R, Fedkiw R, Anderson J. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics* 2002; **21**(3): 594–603.
3. Müller M, Heidelberger B, Hennix M, Ratcliff J. Position based dynamics. *Journal of Visual Communication and Image Representation* 2007; **18**(2): 109–118.
4. Shinar T, Schroeder C, Fedkiw R. Two-way coupling of rigid and deformable bodies. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation,* SCA'08. Eurographics Association, Aire-la-Ville, 2008; 95–103.
5. Baraff D, Witkin A. Partitioned dynamics. *Technical Report*. The Robotics Institute, Carnegie Mellon University, 1997.
6. Coumans E. Bullet Physics. http://bulletphysics.org [3 February 2015].
7. Erleben K. Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics* 2007; **26**(2). DOI: 10.1145/1243980.1243986.
8. Bender J, Müller M, Otaduy MA, Teschner M. Position-based methods for the simulation of solid objects in computer graphics. In *Eurographics 2013 State of the Art Reports*. Eurographics Association, Girona, 2012; 1–22.
9. Gibson SF. 3D ChainMail: a fast algorithm for deforming volumetric objects. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* I3D '97. ACM, New York, NY, USA, 1997; 149.
10. Müller M, Heidelberger B, Teschner M, Gross M. Meshless deformations based on shape matching. *ACM Transactions on Graphics* 2005; **24**(3): 471–478.
11. Nealen A, Müller M, Keiser R, Boxerman E, Carlson M. Physically based deformable models in computer graphics. *Computer Graphics Forum* 2006; **25**(4): 809–836.
12. Selle A, Lentine M, Fedkiw R. A mass spring model for hair simulation. *ACM Transactions on Graphics* 2008; **27**(3): 64:1–64:11.
13. Gibson S, Mirtich B. A survey of deformable models in computer graphics. *Technical Reports TR-97-19*, MERL, Cambridge, MA, 1997.
14. Provot X. Deformation constraints in a mass–spring model to describe rigid cloth behavior. In *Graphics Interface*, Quebec, Quebec, Canada, 1995; 147–154.
15. Jakobsen T. Advanced character physics. In *Game Developers Conference Proceedings*. CMP Media, Inc., San Jose Convention Center, CA, USA, 2001; 383–401.
16. Bender J, Bayer D. Impulse-based simulation of inextensible cloth. In *IADIS International Conference Computer Graphics and Visualization 2008 (part of MCCSIS 2008)*, Yingcai Xiao, Eleonore ten T (eds). Amsterdam: The Netherlands, 2008; 202–205.

17. Jansson J, Vergeest JS. Combining deformable and rigid-body mechanics simulation. *The Visual Computer* 2003; **19**(5): 280–290.
18. Lenoir J, Fonteneau S. Mixing deformable and rigid-body mechanics simulation. In *Proceedings Computer Graphics International, 2004*, Crete, Greece, June 2004; 327–334.
19. Deul C, Charrier P, Bender J. Position-based rigid body dynamics. *Computer Animation and Virtual Worlds* 2014. DOI: 10.1002/cav.1614.
20. Goldenthal R, Harmon D, Fattal R, Bercovier M, Grinspun E. Efficient simulation of inextensible cloth. *ACM Transactions on Graphics* 2007; **26**(3). DOI: 10.1145/1276377.1276438.
21. Bender J, Schmitt A. Fast dynamic simulation of multi-body systems using impulses. In *Proceedings of VRIPHYS '06*, Madrid, Spain, 2006; 81–90.
22. Stam J. Nucleus: towards a unified dynamics solver for computer graphics. In *11th IEEE International Conference on Computer-aided Design and Computer Graphics, CAD/Graphics '09*, Huangshan, China, 2009; 1–11.
23. Tonge R, Benevolenski F, Voroshilov A. Mass splitting for jitter-free parallel rigid body simulation. *ACM Transactions on Graphics* 2012; **31**(4): 105:1–105:8.
24. Macklin M, Müller M, Chentanez N, Kim T-Y. Unified particle physics for real-time applications. *ACM Transactions on Graphics* 2014; **33**(4): 153:1–153:12.
25. Harada T. Heterogeneous particle-based simulation. In *SIGGRAPH Asia 2011 Sketches,* SA '11. ACM, New York, NY, USA, 2011; 19:1–19:2.
26. Baumgarte J. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering* 1972; **1**(1): 1–16.
27. Catto E. Iterative dynamics with temporal coherence. In *Game Developers Conference Proceedings*, San Francisco, CA, USA, 2005; 1–24.
28. Gilbert EG, Johnson DW, Keerthi SS. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation* 1988; **4**(2): 193–203.
29. Kelager M, Niebe S, Erleben K. A triangle bending constraint model for position-based dynamics. In *Proceedings of VRIPHYS '10*, Copenhagen, Denmark, 2010; 31–37.
30. Smith B, Kaufman DM, Vouga E, Tamstorf R, Grinspun E. Reflections on simultaneous impact. *ACM Transactions on Graphics* 2012; **31**(4): 106:1–106:12.
31. Otaduy MA, Tamstorf R, Steinemann D, Gross M. Implicit contact handling for deformable objects. *Computer Graphics Forum* 2009; **28**(2): 559–568.
32. Otaduy MA, Germann D, Redon S, Gross M. Adaptive deformations with fast tight bounds. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation,* SCA'07. Eurographics Association, Aire-la-Ville, 2007; 181–190.

SUPPORTING INFORMATION

Supporting information may be found in the online version of this article.

AUTHORS' BIOGRAPHIES

Luca Vezzaro received his MSc in Intelligent and Multimedia Systems from the University of Verona (Italy) in 2010. He then started working at the ALTAIR Robotics Laboratory of the University of Verona, where he did research and development on simulation technologies to be used for training in robotic surgery. Since 2013, he has worked as a game programmer at Gameloft Iberica S.A.U. (Barcelona, Spain), the studio responsible for Asphalt 8 and Despicable Me: Minion Rush.

Davide Zerbato received his PhD in Computer Science from the University of Verona (Italy) in 2010 with a thesis on the physics simulation of frictional contact between deformable bodies. His research ranges from physics simulation of complex environment for interactive applications to parallel computation on graphics processing units and to assistive technologies for surgery planning and execution. Since 2006, he has been a Research Assistant at the ALTAIR Robotics Laboratory of the University of Verona and has been involved in several European Unions projects (Accurate Robot Assistant, Patient Safety in Robotics Surgery, and Intelligent Surgical Robotics).

Paolo Fiorini received a laurea degree in Electronic Engineering from the University of Padova (Italy), an MSEE from the University of California at Irvine (USA), and a PhD in ME from UCLA (USA). From 1985 to 2000, he was with the NASA Jet Propulsion Laboratory, California Institute of Technology, where he worked on. telerobotic and teloperated systems for space exploration From 2000 to 2010, he was an Associate Professor of Control Systems at the School of Science of the University of Verona (Italy) where he founded the ALTAIR Robotics Laboratory with his students. He is currently a Full Professor of Computer Science at the University of Verona. His research focuses on teleoperation for surgery, service, and exploration robotics and is funded by several European and Italian Projects. He is an IEEE fellow (2009).

DOI: 10.1002/cav