Pacific Graphics 2013
B. Levy, X. Tong, and K. Yin
(Guest Editors)

Volume 32 (2013), Number 7

Implicit Integration for Particle-based Simulation of Elasto-Plastic Solids

Yahan Zhou, Zhaoliang Lun, Evangelos Kalogerakis, Rui Wang

University of Massachusetts, Amherst

Abstract

We present a novel particle-based method for stable simulation of elasto-plastic materials. The main contribution of our method is an implicit numerical integrator, using a physically-based model, for computing particles that undergo both elastic and plastic deformations. The main advantage of our implicit integrator is that it allows the use of large time steps while still preserving stable and physically plausible simulation results. As a key component of our algorithm, at each time step we compute the particle positions and velocities based on a sparse linear system, which we solve efficiently on the graphics hardware. Compared to existing techniques, our method allows for a much wider range of stiffness and plasticity settings. In addition, our method can significantly reduce the computation cost for certain range of material types. We demonstrate fast and stable simulations for a variety of elasto-plastic materials, ranging from highly stiff elastic materials to highly plastic ones.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-dimensional graphics and Realism—Animation; Simulation and Modeling [I.3.7]: Types of Simulation—Animation;

1. Introduction

A fundamental challenge in computer animation is to produce fast, realistic, and reliable simulations of solids with a variety of different material types. To achieve this goal, particle-based approaches have become increasingly popular in computer graphics, due to their efficient computation cost and the ability to simulate a wide range of materials (e.g. from fluids to fractured solids) using the same meshless representation.

Over the last years, particle-based methods have also been applied to handle simulations of deformable solids with material types ranging from elastic to plastic ones. However, among existing particle-based methods for elasto-plastic materials, an important limitation is that they require using small time steps in order to maintain stable simulation results. This is because they calculate particle positions and velocities by relying on *explicit numerical integration*. To keep the simulation error bounded and to ensure numerical stability, it is well-known that explicit integrators require small time steps, especially when the solid material is stiff. Thus, they are generally slow to compute and often unsuitable for scenes containing a variety of different solid materials.

A solution to this problem is to enable *implicit numerical integration* for particle-based methods. In this paper, we present a novel formulation of particle-based simulation for elasto-plastic materials with implicit numerical integration. The main advantage of implicit integration is that it allows much larger time steps while still providing stable and physically plausible simulation results. A key component of our algorithm is to formulate the deformation computation into a sparse linear system, which is both efficient to represent and can be solved quickly on the graphics hardware by exploiting the particle-level parallelism. As a result, in addition to preserving the stability, our method is memory-efficient and greatly speeds up the computation time in several cases (particularly with highly plastic and/or stiff materials).

Another advantage is that our method can handle solids with much wider range of material properties compared to existing techniques. For example, the method presented in [BIT09] can only handle elastic materials, while the method presented in [GBB09] is unstable when the object undergoes large elastic deformations. Using our method, we demonstrate stable results for both very stiff and very soft materials, and also for objects that undergo either large elastic or large plastic deformations.

Contribution. To summarize, our main contribution is a new mathematical formulation that enables implicit integration for particle-based simulation of elasto-plastic materials based on a physically plausible model. Our method preserves numerical stability even at large time steps. At each

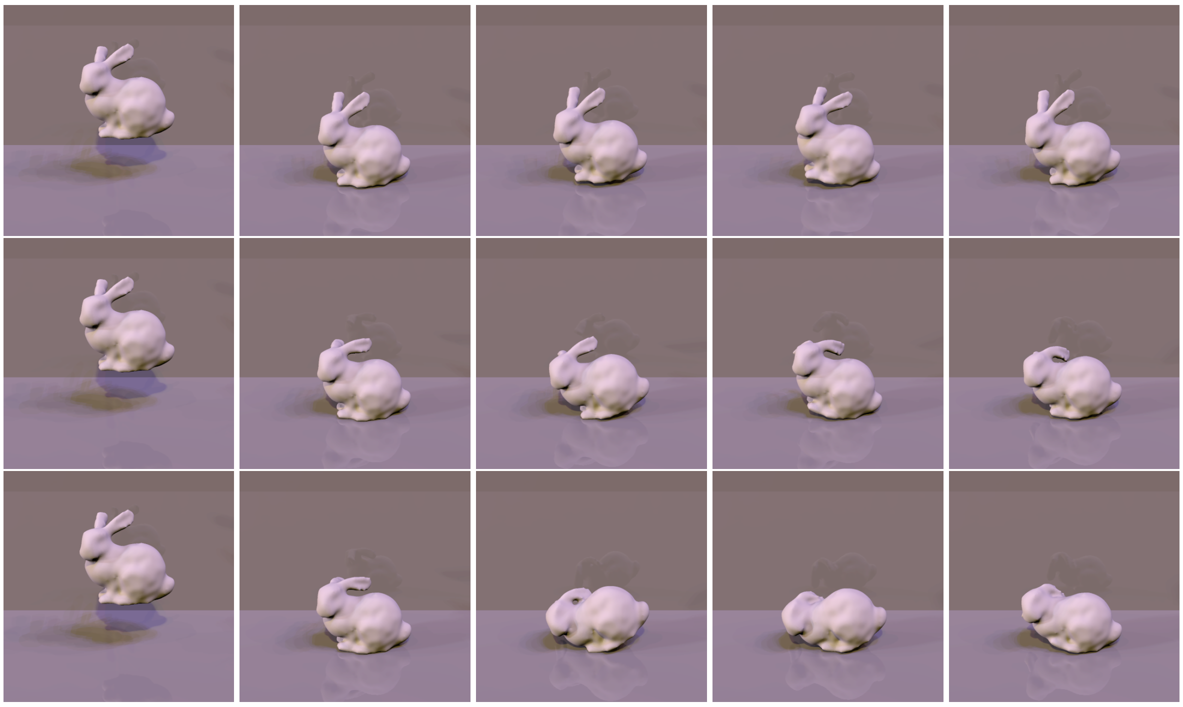

Figure 1: *A scene with a bunny falling on the ground. From top to bottom, each row corresponds to a different set of parameters, and shows selected frames of the resulting animation. We refer the readers to the paper video for the complete animations.*

time step we compute the particle positions and velocities based on a sparse linear system, which we solve efficiently on the graphics hardware. In addition, our method can significantly reduce the computation cost for certain range of material types. Experimental results show that our method can handle a variety of solid materials, from highly stiff elastic materials to highly plastic ones.

2. Related Work

Finite-element elasto-plastic simulation. Our work is related to previous methods that perform physically-based simulation of elasto-plastic materials. In the pioneer work of [TF88], Terzopoulos et al. introduced deformation models for viscoplastic materials to computer graphics. O'Brien et al. [OBH02] incorporated a simple additive model of plasticity into a finite element simulation that preserves volume more realistically. However, the method tends to become unstable for substantial elastic and plastic deformations due to poorly conditioned and tangled elements. Irving et al. [ITF04] proposed a multiplicative model of plasticity such that larger plastic deformations can be handled. However, when the basis matrices of the elements are ill-conditioned, the finite element simulation will still become unstable. To avoid this problem, remeshing approaches have been proposed to maintain well-shaped tetrahedral elements [BWHT07, WT08, WTGT09]. Unfortunately, the remeshing steps can introduce numerical errors due to resampling and smoothing of the physical properties of the material. A conservative local remeshing algorithm was suggested in [WRK*10] to limit the accumulative errors during simulation.

Particle-based elasto-plastic simulation. The above methods all handle elasto-plastic materials by using finite-element simulation. Our paper builds upon the same multiplicative plasticity and deformation gradient formulation presented in [ITF04, BWHT07]. However, instead of using finite elements, we use a particle-based method, which is meshless and does not require any remeshing steps. Our approach is mostly related to prior work in particle-based simulation of elasto-plastic materials. A complete survey of such simulation methods can be found in [GP07].

Particle-based methods are particularly useful as a unified approach for solid simulations. Müller et al. [MKN*04] first proposed such unified particle-based approach to model all of elastic, plastic, and melting behavior of objects. Their approach uses the Moving Least Squares (MLS) to derive a deformation field from particle positions between the resting state and deformed state. Plastic deformations are handled by updating a plastic strain measure based on a simple additive model of plasticity. However, as discussed in [ITF04] and [BWHT07], this model loses its physical meaning for finite strains. Later, Keiser et al. [KAG*05] incorporated the Navier-Stokes equations into the same plastic deformation regime of Müller et al. to simulate fluids with varying viscosity and the transitions from elasto-plastic materials to fluids and vice versa. Instead of using Moving Least

Squares to approximate the deformation field, Solenthaler et al. [SSP07] and Becker et al. [BIT09] use particles in conjunction with Smoothed Particle Hydrodynamics-based forces, which has the advantage of better handling coarsely spaced particle configurations. In particular, Becker et al.'s method also better handles rigid body rotations by using the Singular Value Decomposition to factor out the rotation component from the deformation gradient for each particle. Gerszewski et al. [GBB09] uses a multiplicative model of plasticity in the context of particle-based elasto-plastic simulation that is more physically plausible and accurate. It also avoids storing the initial resting state of the object by storing only the elastic part of the deformation, and then computing deformation gradients from particle positions only between successive timesteps. As a result, their method can handle large plastic deformations. More recently, a few other methods for handling ill-conditioned co-planar or co-linear configurations of particles have been suggested in the approach of elastons [MKB*10] and oriented particles [MC11].

Implicit integration in particle-based methods. All the above methods use explicit numerical integration to simulate the particle positions and velocities for elasto-plastic materials, with the exception of Müller et al. [MKN*04] and Martin et al. [MKB*10]. Müller et al. included both an explicit and implicit integration scheme in their formulation, while Martin et al. demonstrated elaston-based simulation with a semi-implicit integration scheme. Both approaches use a simple additive plasticity model to incorporate plastic strains in their simulations.

Our implicit integration formulation builds upon the multiplicative plasticity and deformation gradient model suggested in [GBB09], which (compared to the additive model) is more physically correct and can better handle large plastic deformations. While [GBB09] uses an explicit numerical integration scheme, it is well-known that explicit numerical integration causes stability issues when the timesteps are not sufficiently small. These stability issues can be avoided with implicit numerical integration [BW98, DSB99, MDM*02, MKN*04, MTG04]. As we demonstrate in our results (Section 5), our method yields stable simulations even for large time steps and is able to handle large elastic and plastic deformations very well. Furthermore, our method does not suffer from drifting issues (for example, see Figure 5). Even if there is more computation involved at each time step, our method has comparable performance or is even much faster (particularly with highly plastic and/or stiff materials - see Table 1), since it enables much larger timesteps. At each timestep our method solves a Laplacian linear system, which is sparse (thus, also memory-efficient). In addition, the system can be efficiently solved on graphics hardware, as we explain in Section 4.

3. Algorithms

Overview. This section describes our algorithm details. While implicit integration has the advantage in producing stable simulation results even at large time step values, conventionally it has been expensive to compute, due to the nonlinear formulation of force computations in previous particle-based solid simulators [MSJT08]. The key of our algorithm is an elasto-plastic model which can be easily linearized. As the result, we turn implicit integration into a sparse linear system which can be efficiently solved, including on the graphics hardware.

This section is organized as followed. Section 3.1 describes the background of elastic deformation models, including the definition of the strain energy density and the derivation of the elastic force. Section 3.2 describes how to extend the method to handle plastic deformations. Then in Section 3.3 we introduce our implicit integration method.

3.1. Elastic Deformation Model

This subsection describes the elastic deformation model based on [GBB09]. To begin, we assume the geometry to be animated is represented by a set of particles, and we use $\mathbf{F}$ to denote an affine transformation matrix that represents the deformation. For each particle i, we denote its transformation matrix as $\mathbf{F}_i$, which is defined as:

$$\mathbf{F}_i = \arg\min_{\mathbf{F}} \sum_{j \in N(i)} w_{i,j} \parallel \mathbf{F}(\mathbf{p}_j - \mathbf{p}_i) - (\mathbf{q}_j - \mathbf{q}_i) \parallel, \tag{1}$$

where $\mathbf{p}$ denotes a particle's resting position (i.e., material space coordinate, approximated from the previous timestep, which will be explained in Section 3.2), $\mathbf{q}$ denotes its new deformed position, i is the index of the current particle we are computing, and j is the index of a particle in the neighborhood $N(i)$ of the particle i. The neighborhood size is determined by a radius of a euclidean ball (i.e. particles within a certain distance to i are considered neighbors of i, see Section 4). The weight kernel $w_{i,j} = w(\mathbf{p}_i - \mathbf{p}_j)$ is computed from the cosine-based weighting kernel described in [SSP07]. Note that the weight kernel is defined over the approximated particle's resting position. For convenience we define the following matrices:

$$\begin{aligned}
\mathbf{P}_i &= [(\mathbf{p}_1 - \mathbf{p}_i), (\mathbf{p}_2 - \mathbf{p}_i), \ldots (\mathbf{p}_r - \mathbf{p}_i)], &(2)\\
\mathbf{Q}_i &= [(\mathbf{q}_1 - \mathbf{q}_i), (\mathbf{q}_2 - \mathbf{q}_i), \ldots (\mathbf{q}_r - \mathbf{q}_i)], &(3)\\
\mathbf{W}_i &= Diag(w_{i,1}, w_{i,2}, \ldots w_{i,r}). &(4)
\end{aligned}$$

where r is the number of all neighboring particles of i. Then following [GBB09], the transformation matrix $\mathbf{F}_i$ can be solved as:

$$\mathbf{F}_i = \mathbf{Q}_i \mathbf{W}_i \mathbf{P}_i^T \left(\mathbf{P}_i \mathbf{W}_i \mathbf{P}_i^T \right)^{-T} \tag{5}$$

where T denotes matrix transpose.

Following [ITF04], to remove the rotation factors, we decompose $\mathbf{F}_i$ into $\mathbf{U}_i \hat{\mathbf{F}}_i \mathbf{V}_i^T$ using the Singular Value Decomposition, where $\mathbf{U}_i$ and $\mathbf{V}_i$ are two rotation matrices, and $\hat{\mathbf{F}}_i$ is a diagonal matrix. Then the linear Cauchy-Green strain tensor can be computed as:

$$\varepsilon_i = \frac{1}{2}(\hat{\mathbf{F}}_i + \hat{\mathbf{F}}_i^T) - \mathbf{I} = \hat{\mathbf{F}}_i - \mathbf{I}. \tag{6}$$

As described in [KAG*05], the strain energy density is:

$$U_i = \frac{1}{2}(\varepsilon_i \cdot \sigma_i). \tag{7}$$

where $\sigma_i = \mathbf{C}\,\varepsilon_i$ following Hooke's law, and $\mathbf{C}$ is the material property which depends only on Young's modulus (i.e., the stiffness of the material) and Poisson's Ratio for isotropic materials (i.e., the amount of deformation in other dimensions when the material is stretched in one dimension).

The force $\mathbf{f}_i$ on particle i is computed as the derivative of the strain energy over $\mathbf{q}_i$, summed over all neighboring particles j:

$$\begin{aligned}\mathbf{f}_i &= -\sum_j \frac{\partial U_j}{\partial \mathbf{q}_i} \\ &= -\sum_j \left(\frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_i} \mathbf{C}\,(\hat{\mathbf{F}}_j - \mathbf{I}) \right). \end{aligned} \tag{8}$$

Assuming that the rotation matrices $\mathbf{U}_j$ and $\mathbf{V}_j$ are constant within each timestep, then $\frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_i}$ can be computed as:

$$\frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_{i,d}} = \begin{cases} w_{i,j}\,\mathbf{U}_j^T\,\mathbf{e}_d\,(\mathbf{p}_j - \mathbf{p}_i)^T \left(\mathbf{P}_j \mathbf{W}_j \mathbf{P}_j^T\right)^{-T} \mathbf{V}_j, & \text{(if } i \neq j) \\ -\mathbf{U}_i^T\,\mathbf{e}_d \sum_{k \neq i} w_{i,k}(\mathbf{p}_k - \mathbf{p}_i)^T \left(\mathbf{P}_i \mathbf{W}_i \mathbf{P}_i^T\right)^{-T} \mathbf{V}_i. & \text{(if } i = j) \end{cases} \tag{9}$$

Here $d \in \{1,2,3\}$ refers to the x, y, z coordinates of $\mathbf{q}_i$, k indexes a neighboring particle of i, $\mathbf{e}_d$ is the d-th column of the identity matrix. $\frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_{i,d}}$ is a 3x3 matrix which can be encoded into a 9-element vector. In practice, however, because of the stress model we use, we only need to store the symmetric matrix $\frac{1}{2}(\frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_{i,d}} + \frac{\partial \hat{\mathbf{F}}_j^T}{\partial \mathbf{q}_{i,d}})$, which can be encoded into a 6-element vector.

If we used explicit integration, the update rule at time step t would be:

$$\mu_i^{t+1} = \mu_i^t + \frac{\Delta t}{m_i}(\mathbf{f}_{i,ext}^t + \mathbf{f}_i^t), \tag{10}$$

$$\mathbf{q}_i^{t+1} = \mathbf{q}_i^t + \Delta t\,\mu_i^{t+1} \tag{11}$$

where μ_i^t and $\mathbf{q}_i^t$ are the particle velocity and position at the current time step t, Δt is the time step value, m_i is the particle mass, $\mathbf{f}_{i,ext}^t$ is the external force (e.g. gravity), and $\mathbf{f}_i^t$ is the force computed from the current particle position $\mathbf{q}_i^t$ as described in Equation 8.

3.2. Plastic Deformation Model

In contrast to the elastic deformation, the plastic deformation is non-linear with respect to the particle positions, therefore directly formulating it will lead to nonlinear equations. Instead, we assume within each time step the model undergoes only elastic deformations, and then at the end of the time step we compute the plastic deformations. In practice, this assumption provides plausible plastic deformation effects and does not affect the stability or efficiency of the algorithm.

Following the work in [BWHT07], we consider the eigenvalue of deformation gradient as the multiplication of an elastic deformation component and a plastic one:

$$\hat{\mathbf{F}}_i = \hat{\mathbf{F}}_{e,i}\hat{\mathbf{F}}_{p,i}. \tag{12}$$

where the plastic component $\hat{\mathbf{F}}_{p,i}$ is computed using the method presented in [BWHT07]. Following their work, we also define elastic deformation gradient $\mathbf{F}_{e,i} = \mathbf{U}\hat{\mathbf{F}}_{e,i}\mathbf{V}^T$ and plastic deformation gradient $\mathbf{F}_{p,i} = \mathbf{V}\hat{\mathbf{F}}_{p,i}\mathbf{V}^T$. This also implies that:

$$\mathbf{F}_i = \mathbf{F}_{e,i}\mathbf{F}_{p,i}. \tag{13}$$

We consider the plastic deformation as the local deformation on the resting state of the previous timestep, and the deformed resting state is $\mathbf{F}_{p,i}\mathbf{P}_i$. Because of the definition of $\mathbf{F}_i$ in Equation 13 we have:

$$\mathbf{Q}_i \approx \mathbf{F}_i\mathbf{P}_i = \mathbf{F}_{e,i}\mathbf{F}_{p,i}\mathbf{P}_i. \tag{14}$$

Thus we have:

$$\mathbf{F}_{p,i}\mathbf{P}_i \approx \mathbf{F}_{e,i}^{-1}\mathbf{Q}_i. \tag{15}$$

Here $\mathbf{F}_{e,i}^{-1}$ is the pseudo-inverse of $\mathbf{F}_{e,i}$ if it is ill-conditioned. The above equation implies that accommodating plastic deformations requires replacing $\mathbf{P}_i$ with $(\mathbf{F}_{e,i}^t)^{-1}\mathbf{Q}_i^t$ where $(\mathbf{F}_{e,i}^t)^{-1}$ is the (pseudo) inverse of the elastic deformation of the last time step. By doing so we are implicitly multiplying plastic deformations of all time steps together.

3.3. Implicit Integration

The update rule of implicit integration is similar to equations 10 and 11, except that we use $\mathbf{f}_i^{t+1}$ (i.e., the force at the next time step) instead of $\mathbf{f}_i^t$. Since $\mathbf{f}_i^{t+1}$ is unknown, the solution requires solving an implicit equation. Specifically, by substituting equation 10 into 11 and replacing $\mathbf{f}_i^t$ with $\mathbf{f}_i^{t+1}$, we have the following equation:

$$\mathbf{q}_i^{t+1} = \mathbf{q}_i^t + \Delta t\,\mu_i^t + \frac{\Delta t^2}{m_i}(\mathbf{f}_{i,ext}^{t+1} + \mathbf{f}_i^{t+1}), \tag{16}$$

or in vector form:

$$\mathbf{Q}^{t+1} = \mathbf{Q}^t + \Delta t\,\mathbf{M}^t + \Delta t^2\,\mathbf{m}^{-1}(\mathbf{f}_{ext}^{t+1} + \mathbf{f}^{t+1}). \tag{17}$$

It turns out that this produces a linear system. To see so, first, note that $\hat{\mathbf{F}}_i$ can be expressed in first-order approximation as:

$$\hat{\mathbf{F}}_i = \sum_k \frac{\partial \hat{\mathbf{F}}_i}{\partial \mathbf{q}_k}\mathbf{q}_k, \tag{18}$$

Not surprisingly, from the definition in Equation 5, $\mathbf{F}_i$ (and hence $\hat{\mathbf{F}}_i$) is linear with respect to the $\mathbf{q}$'s, therefore the first-order approximation is exact (i.e. all second and higher order terms are zero).

To simplify the expression we can define:

$$L_{j,i} = \left[\frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_{i,1}}, \frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_{i,2}}, \frac{\partial \hat{\mathbf{F}}_j}{\partial \mathbf{q}_{i,3}} \right] \tag{19}$$

which is a 6x3 matrix that describes the gradient of deformation matrix over all nearby particles. We write all $L_{j,i}$'s together into $\mathbf{L}$, which is a $n \times n \times 6 \times 3$ tensor, where n is the total number of particles. We can see that each $n \times n$ submatrix of $\mathbf{L}$ is actually a Laplacian matrix because Equation 9 and 19 implies $\sum_i L_{j,i} = 0$. In our implementation, since $\mathbf{L}$ is very sparse, we fit it into the GPU memory by storing it in compressed sparse row format. In addition, note that $\mathbf{L}$ is independent of all particle positions in the current timestep under the assumption that the rotation matrices $\mathbf{U}$ and $\mathbf{V}$ are constant within each timestep.

Equation 18 can now be rewritten as:

$$\hat{\mathbf{F}}_i = \sum_k \frac{\partial \hat{\mathbf{F}}_i}{\partial \mathbf{q}_k} \mathbf{q}_k = L_{i,*}\mathbf{Q}. \tag{20}$$

where $L_{i,*}$ is the i^{th} row vector of the Laplacian tensor $\mathbf{L}$ (in which each element is a 6x3 matrix, refer to Equation 19) and $\mathbf{Q}$ stores all particle positions (each element of $\mathbf{Q}$ is a 3-dimensional position vector). Given this, equation 8 can now be rewritten in the form:

$$\mathbf{f}_i = -L^T_{*,i}\,\mathbf{C}\,\mathbf{L}\mathbf{Q} + \sum_j L^T_{j,i}\,\mathbf{C}\,\mathbf{I}, \tag{21}$$

or in matrix form:

$$\mathbf{f} = -\mathbf{L}^T\,\mathbf{C}\,\mathbf{L}\mathbf{Q} + \sum_j L^T_{j,*}\,\mathbf{C}\,\mathbf{I}. \tag{22}$$

In the above equation, C is the 6x6 material property matrix and $\mathbf{I}$ encodes the identity matrices. By replacing $\mathbf{Q}$ with $\mathbf{Q}^{t+1}$ and substituting equation 22 into equation 17 and rearranging the term, we will have the following linear system:

$$\mathbf{A}\mathbf{Q}^{t+1} = \mathbf{b}, \tag{23}$$

where:

$$\begin{aligned} \mathbf{A} &= \Delta t^2 \mathbf{m}^{-1}\mathbf{L}^{\mathbf{T}}\mathbf{C}\mathbf{L} + \mathbf{I}, &(24)\\ \mathbf{b} &= \mathbf{Q}^t + \Delta t \mathbf{V}^t + \Delta t^2 \mathbf{m}^{-1}\mathbf{f}^{t+1}_{ext} \\ &\quad + \Delta t^2 \mathbf{m}^{-1} \sum_j L^T_{j,*}\, W\mathbf{C}\mathbf{I}. &(25) \end{aligned}$$

By solving the linear system we can compute $\mathbf{Q}^{t+1}$. The velocity $\mathbf{M}^{t+1}$ will be computed from $\mathbf{Q}^{t+1}$ and $\mathbf{Q}^t$ for later use.

In practice, we do not explicitly compute $\mathbf{A}$ in order to reduce memory cost and computation overhead. Instead, we store the much sparser tensor $\mathbf{L}$. When a vector is multiplied by $\mathbf{A}$, we compute the multiplication by expanding Equation 24.

4. Implementation

In this section we describe implementation details. Source code and executable demos will all be published on our project webpage. Starting from the initial state of the input objects (which are represented as particles), the algorithm performs the following steps:

1. Estimate the particle positions $\mathbf{Q}_0^{t+1}$ for the next timestep by assuming the force is the same as the last timestep (i.e., $\mathbf{f}^{t+1} = \mathbf{f}^t$), and performing one step of the simulation using Equation 17.
2. Compute the rotation matrices $\mathbf{U}_i$, $\mathbf{V}_i$ based on the estimated $\mathbf{Q}_0^{t+1}$ using SVD.
3. Compute the Laplacian matrix $\mathbf{L}$ (Equation 19); also compute $\mathbf{b}$ from Equations 25.
4. Solve the sparse linear system from Equation 23.
5. Update $\mathbf{Q}^{t+1}$. Calculate velocity as $(\mathbf{Q}^{t+1} - \mathbf{Q}^t)/\Delta t$
6. Separate elastic deformation from plastic deformation (Equation 13). The elastic deformation $\mathbf{F}_{e,i}$ will be stored for estimating $\mathbf{P}_i$ in the next timestep from Equation 15.

We have implemented the entire pipeline on the GPU to parallelize the above steps as much as possible. We use one GPU thread per particle. We make use of the open-source fluid simulation package published by [KE12]. For step 2, we have implemented a simple SVD algorithm for 3x3 matrices which runs per-thread. Steps 3, 5, 6 are implemented based on our provided equations. Step 4 requires a sparse linear system solver, for which we make use of the GPU-based Conjugate Gradient solver from the CUSP library [Cus12] to handle tensors. The main computational benefit of the GPU solver is that it parallelizes the sparse matrix-vector multiplications required in the main loop of the Conjugate Gradient method [BG09]. To speed up the convergence, we use the $\mathbf{Q}_0^{t+1}$ from step 1 as the initial guess in the Conjugate Gradient solver. In our experiments, the relative tolerance of the solver is set to 10^{-4}.

Steps 2 and 3 above involve searching all particles within the neighborhood size. To speed up the neighborhood search step, we build a uniform grid structure for all particles. At any point in space, when looking for neighboring particles, we enumerate all particles in the neighboring grids and test if they are inside the neighborhood implied by the weighting kernel [SSP07]. We process all particles within this neighborhood, which is more accurate than the method of [GBB09] where they only pick the k-nearest neighbors. In our experiments, we use a fix kernel size of 2.5 times the minimum particle distance in the resting state. We noticed that choosing a smaller neighborhood size, such as 1.5 times the minimum particle distance in the resting state, caused instabilities in places with thin features.

In our experiments, we chose a time step of 2ms. In theory, our method could use a larger time step as long as the rotation and plastic deformations do not significantly change within each time step. In practice, a larger timestep would slow down the convergence of the linear solver (thus, increasing total simulation time).

In all experiments, we assume the mass of each particle is equal and remains constant. Also, we aim to demonstrate our system on materials with only solid properties, thus we do not include any fluid properties such as SPH pressure forces and viscosity in our experiments. This is different from the method of [GBB09], which relies on adding SPH pressure forces and viscosity terms to improve stability. Topology changes are handled implicitly when the neighborhood of particles changes.

Finally, to render the animation results, we reconstruct a smooth surface from the particles at each timestep. The surface reconstruction is achieved by running the level-set method described in [BGB11] as a post-process. The timing results in the next section do not include the surface reconstruction and rendering time.

5. Results

Our method can simulate solid objects with a wide range of material properties. Figure 1 and 3 show examples with different stiffness parameters on the bunny model and armadillo model respectively. For each parameter setting we show selected frames from the animation result. Refer to the paper video for the complete set of frames. As the figures show, our model handles a range of materials starting from very low stiffness value to very high stiffness value.

We also demonstrate examples of materials with different elastic and plastic properties in Figure 2. Here the scene is a cube falling on the ground. From the left to right, we show materials that are purely elastic, medium plastic, and highly plastic. Note the differences in the deformations, and how the deformation is permanent on plastic materials.

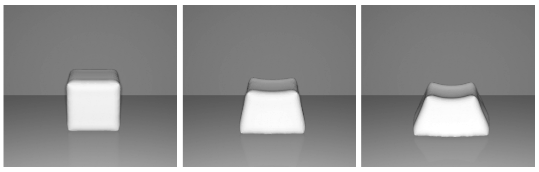
Figure 2: *Simulation of different elastic and plastic materials.* ***Left:*** *purely elastic.* ***Middle:*** *medium plastic.* ***Right:*** *highly plastic.*

Qualitative comparisons with [GBB09]. Experimental results show that our method is generally more stable than [GBB09] in several aspects. First of all, as discussed in their paper, the method of [GBB09] becomes unstable when a solid object (without any added fluid force) undergoes large elastic deformation. In contrast, our method can successfully compute a stable solution for such cases without any added fluid force. Figure 4 demonstrates an example of a falling armadillo where our method successfully simulates the large elastic deformation of soft objects while the method of [GBB09] suffers from poor stability. When using the explicit integration method by [GBB09], the particles quickly diverge as the animation proceeds, resulting in non-reconstructable surfaces. In the supplementary video, one of the Armadillo hands flickers during freefall with the explicit integration method. This is the result of instability caused by numerical error accumulation. It could be solved by clamping the stress to 0, when it has very small values e.g., below $1e-5$. On the other hand, our method does not suffer from any of these stability issues and does not need such ad-hoc corrections. In addition, experiments show that our model suffers much less from drifting issues when simulating stiff objects, as shown in Figure 5. Here the scene is a cube falling on the ground. After the cube bounces from the ground, our method preserves the correct rotation angles of the cube, while the method by [GBB09] results in drifting and incorrect rotations of the cube as the animation proceeds.

Performance comparisons with [GBB09]. For a fair comparison of performance, we implemented both our method and [GBB09] on the GPU, and optimized the performance of each as best as we can. To make [GBB09] suitable for the GPU, when performing neighbor search, we search all particles in a uniform grid structure. This is slightly different from the original implementation in [GBB09] (which uses kd-tree), but this change does not result in any noticeable difference in accuracy throughout our experiments. In addition, we set this technique to use the maximum possible time step ensuring stable simulation. The performance results are shown in Table 1. The results are obtained on a Intel i7 3.40GHz CPU and NVIDIA GeForce GTX 480 GPU. In some experiments the method of [GBB09] is unstable when the neighborhood size is small, thus we increase the neighborhood size to make it stable (see rows marked with *, ** in the table). Our GPU method is faster than the GPU implementation of [GBB09] in several cases, while it is more than an order of magnitude faster than their original CPU-based implementation.

6. Discussion

In this paper we have presented a new method for particle-based simulation of elasto-plastic materials based on implicit numerical integration. Our method is more stable than previous techniques and in many cases faster. It can also handle solid objects with a wide range of material properties, ranging from very soft to very stiff materials without having the artifacts that previous techniques suffer from. It can also handle objects undergoing highly elastic deformation and highly plastic deformation in a unified manner.

Limitations. One drawback of our algorithm lies in cases where neighboring particles are co-planar or co-linear. In this case the deformation matrix will become ill-conditioned, resulting in poor stability. We expect that combining our formulation with the GMLS technique of elastons [MKB*10] or oriented particles [MC11] can potentially solve this problem. Another drawback of our algorithm is that it keeps the rotation matrices **U** and **V** constant within each timestep. The formulations in [CPSS10, MZS*11] are useful to tackle this problem. Also, our model is limited to linear stress-strain relationships with co-rotated strain.

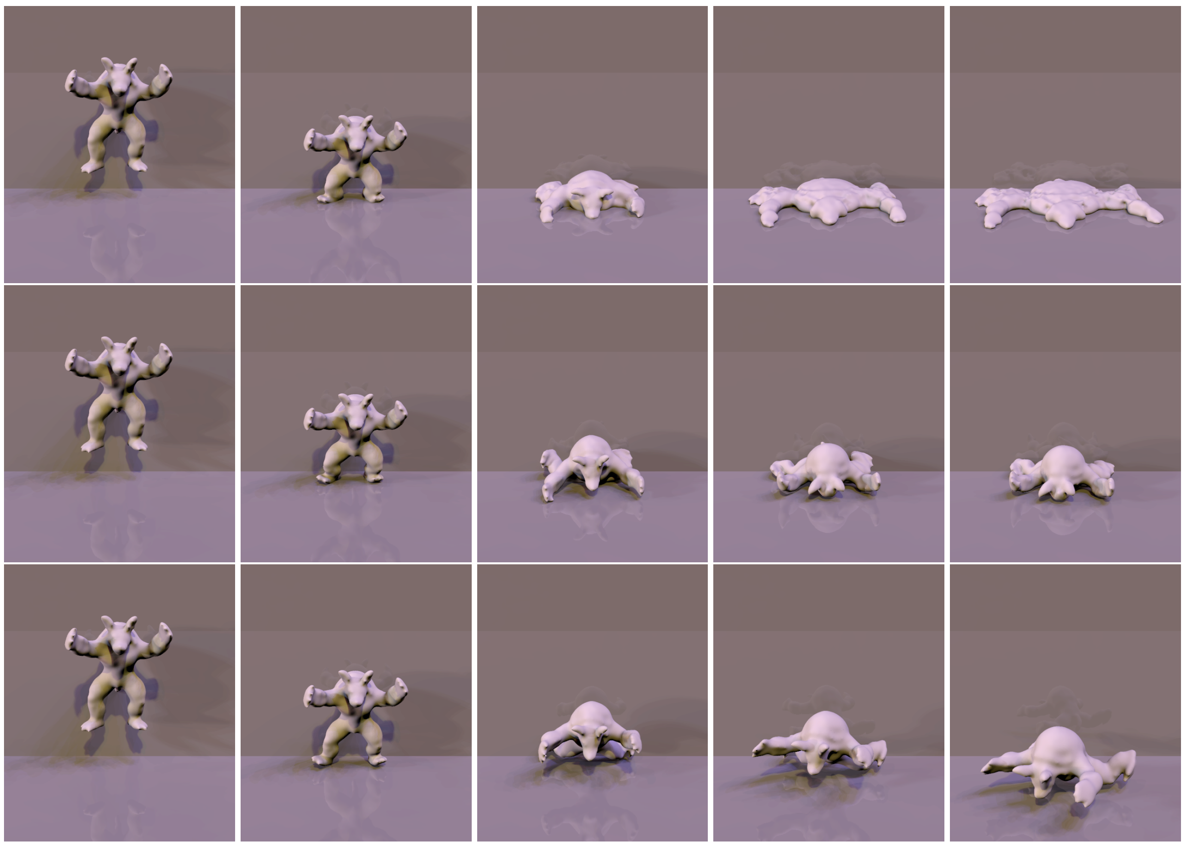

Figure 3: *Comparison of materials with different stiffness settings applied to the Armadillo model falling on the floor. Each row corresponds to a different parameter setting, and shows selected frames of the resulting animation. Our algorithm is able to successfully simulate objects with materials ranging from very soft, liquid-like ones (top row) to medium stiff (middle row), and to very stiff, nearly rigid ones (bottom row).*

		Our implicit method		Our explicit method		[GBB09]		
Scene	Number of particles	Time step (s)	Total Time (s) (GPU)	Time step (s)	Total Time (s) (GPU)	Time step (s)	Total Time (s) (GPU)	Total Time (s) (CPU)
Armadillo (top)**	19866	0.002	59	0.00005	467	0.0001	623	20280
Armadillo (middle)*	19866	0.002	94	0.00005	496	0.0006	74	2448
Armadillo (bottom)*	19866	0.002	149	0.00002	1244	0.0001	443	14526
Cube (Fig.5)	2744	0.002	52	0.0001	90	0.0002	29	1075
Angry Birds (see video) *	12974	0.002	158	0.00005	1153	0.00005	629	-

Table 1: *Performance comparison between our algorithm and [GBB09]. In the rows marked with *, the algorithm of [GBB09] uses a larger neighborhood size (with radius equal to 4.5 times the minimum particle distance in the resting state) and in the CPU version a larger number of neighboring particles (from 30 to 200) to avoid instabilities. In the row marked with ** (Armadillo top example), we use even larger neighborhood size (5.5 times the minimum particle distance) for that method again to avoid instabilities. The timesteps chosen for [GBB09] are the largest possible we found after exhaustive search before the simulation becomes unstable in each example.*

Finally, we believe that pre-conditioning techniques developed in the context of finite element simulation (e.g. [HLSO12]) or other pre-conditioning techniques proposed recently (e.g. [KFS13]) can be used in conjunction with our Laplacian linear system formulation, potentially offering even larger performance gains in future implementations of our method.

References

[BG09] Bell N., Garland M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (2009), pp. 18:1–18:11. 5

[BGB11] Bhatacharya H., Gao Y., Bargteil A.: A level-set method for skinning animated particle data. In *Proceedings*

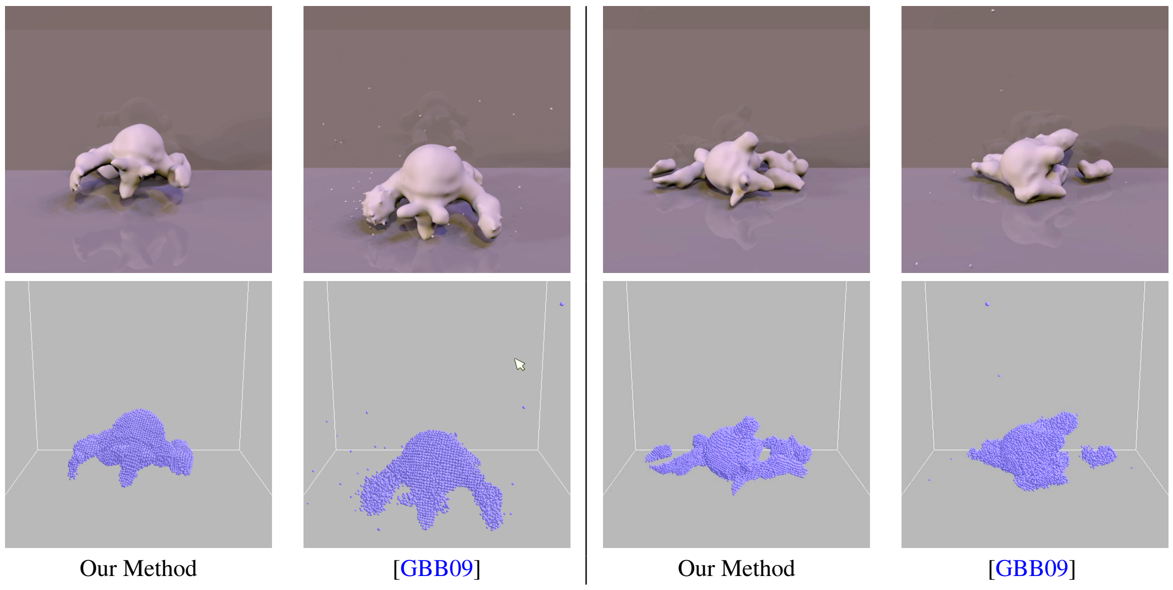

Figure 4: *Comparison between our method and [GBB09] with objects undergoing large elastic deformations. While our method successfully handles these cases, the explicit integration technique in [GBB09] can have stability issues (notice the "exploding" particles in the left and right scene).* ***Top row:*** *reconstructed surfaces of the simulated objects.* ***Bottom row:*** *visualization of the underlying particles of the simulation.*

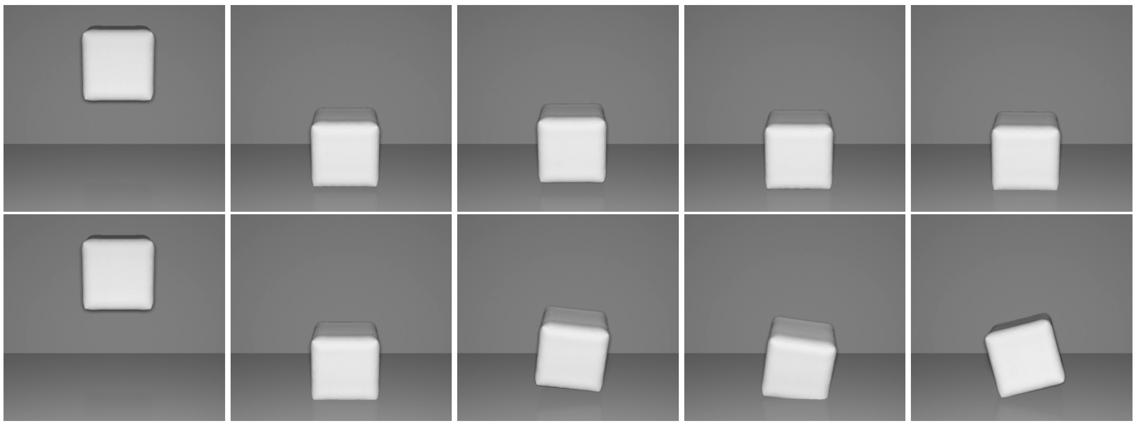

Figure 5: *A demonstration of the drifting problem caused by the explicit integration technique in [GBB09]. Here the scene is a cube dropping straight down onto the ground.* ***Top row:*** *Our method does not suffer from any drifting issues.* ***Bottom row:*** *using [GBB09], the cube starts to drift after falling on the ground.*

of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2011), pp. 17–24. 6

[BIT09] Becker M., Ihmsen M., Teschner M.: Corotated sph for deformable solids. In *Proc. Eurographics conference on Natural Phenomena* (2009), pp. 27–34. 1, 2

[BW98] Baraff D., Witkin A.: Large steps in cloth simulation. In *Proc. SIGGRAPH* (1998), pp. 43–54. 3

[BWHT07] Bargteil A. W., Wojtan C., Hodgins J. K., Turk G.: A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. 26*, 3 (2007). 2, 4

[CPSS10] Chao I., Pinkall U., Sanan P., Schröder P.: A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (2010), 38:1–38:6. 6

[Cus12] Cusp library: *Cusp*. 2012. `http://cusplibrary.github.io/`. 5

[DSB99] Desbrun M., Schröder P., Barr A.: Interactive animation of structured deformable objects. In *Proc. Graphics Interface* (1999), pp. 1–8. 3

[GBB09] Gerszewski D., Bhattacharya H., Bargteil A. W.: A point-based method for animating elastoplastic solids. In *Proc. Symposium on Computer Animation* (2009), pp. 133–138. 1, 3, 5, 6, 7, 8

[GP07] Gross M., Pfister H.: *Point-Based Graphics*. Morgan Kaufmann Publishers Inc., 2007. 2

[HLSO12] Hecht F., Lee Y. J., Shewchuk J. R., O'Brien J. F.: Updated sparse cholesky factors for corotational elastodynamics. *ACM Trans. Graph. 31*, 5 (2012), 123:1–123:13. 6

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proc. Symposium on Computer animation* (2004), pp. 131–140. 2, 3

[KAG*05] KEISER R., ADAMS B., GASSER D., BAZZI P., DUTRÉ P., GROSS M.: A unified lagrangian approach to solid-fluid animation. In *Proc. Conference on Point-Based Graphics* (2005), pp. 125–133. 2, 3

[KE12] KROG O. E., ELSTER A. C.: Fast gpu-based fluid simulations using sph. In *Proc. Conference on Applied Parallel and Scientific Computing - Volume 2* (2012), pp. 98–109. 5

[KFS13] KRISHNAN D., FATTAL R., SZELISKI R.: Efficient preconditioning of laplacian matrices for computer graphics. *ACM Trans. Graph. 32* (2013), To Appear. 6

[MC11] MÜLLER M., CHENTANEZ N.: Solid simulation with oriented particles. *ACM Trans. Graph. 30*, 4 (2011), 92:1–92:10. 3, 6

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. In *Proc. SCA* (2002), pp. 49–54. 3

[MKB*10] MARTIN S., KAUFMANN P., BOTSCH M., GRINSPUN E., GROSS M.: Unified simulation of elastic rods, shells, and solids. *ACM Trans. Graph. 29*, 4 (2010). 3, 6

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *Proc. Symposium on Computer animation* (2004), pp. 141–151. 2, 3

[MSJT08] MÜLLER M., STAM J., JAMES D., THÜREY N.: Real time physics: class notes. In *Proc. ACM SIGGRAPH 2008 classes* (2008), pp. 88:1–88:90. 3

[MTG04] MULLER M., TESCHNER M., GROSS M.: Physically-based simulation of objects represented by surface meshes. In *Proc. CGI* (2004), pp. 26–33. 3

[MZS*11] MCADAMS A., ZHU Y., SELLE A., EMPEY M., TAMSTORF R., TERAN J., SIFAKIS E.: Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30*, 4 (2011), 37:1–37:12. 6

[OBH02] O'BRIEN J. F., BARGTEIL A. W., HODGINS J. K.: Graphical modeling and animation of ductile fracture. In *Proc. SIGGRAPH* (2002), pp. 291–294. 2

[SSP07] SOLENTHALER B., SCHLÄFLI J., PAJAROLA R.: A unified particle model for fluid-solid interactions: Research articles. *Computer Animation and Virtual Worlds 18*, 1 (2007), 69–82. 2, 3, 5

[TF88] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *Proc. SIGGRAPH* (1988), pp. 269–278. 2

[WRK*10] WICKE M., RITCHIE D., KLINGNER B. M., BURKE S., SHEWCHUK J. R., O'BRIEN J. F.: Dynamic local remeshing for elastoplastic simulation. vol. 29, pp. 49:1–49:11. 2

[WT08] WOJTAN C., TURK G.: Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27*, 3 (2008), 47:1–47:8. 2

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. vol. 28, pp. 76:1–76:10. 2