ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2024
M. Skouras and H. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 8

A Multi-layer Solver for XPBD

A. Mercier-Aubin P. G. Kry

McGill University, Canada

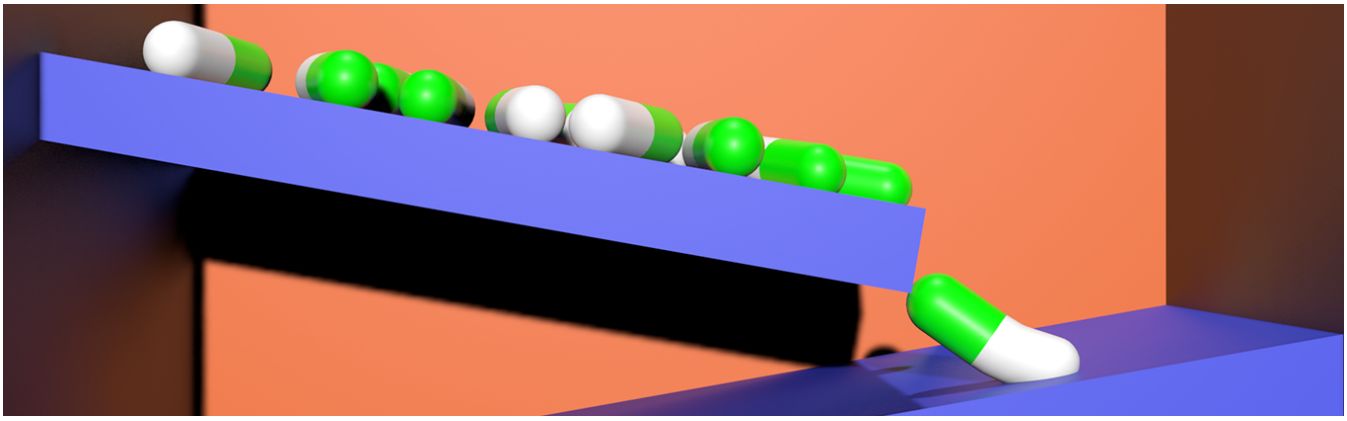


Figure 1: *The pills have green parts are stiff and white parts that are soft. Due to independent rigid motions of different sliding pills, and mixed rigid-elastic contacts, we obtain an efficient and stable simulation of this contact heavy scene.*

Abstract

We present a novel multi-layer method for extended position-based dynamics that exploits a sequence of reduced models consisting of rigid and elastic parts to speed up convergence. Taking inspiration from concepts like adaptive rigidification and long-range constraints, we automatically generate different rigid bodies at each layer based on the current strain rate. During the solve, the rigid bodies provide coupling between progressively less distant vertices during layer iterations, and therefore the fully elastic iterations at the final layer start from a lower residual error. Our layered approach likewise helps with the treatment of contact, where the mixed solves of both rigid and elastic in the layers permit fast propagation of impacts. We show several experiments that guide the selection of parameters of the solver, including the number of layers, the iterations per layers, as well as the choice of rigid patterns. Overall, our results show lower compute times for achieving a desired residual reduction across a variety of simulation models and scenarios.

CCS Concepts

• ***Computing methodologies*** → ***Interactive simulation;*** *Simulation by animation;* ***Real-time simulation;***

Keywords: XPBD, rigid bodies, soft bodies, multigrid, contact

1. Introduction

Real-time simulations of soft bodies is an important component in many interactive applications, such as training simulators and videos games. In these applications, the compute resources are often limited, and must be shared between rendering, application logic, and the computation of physics. Real-time applications typically requires steady frame rates for the rendering of intricate geometry often leaving tight budgets for the computation of physics. In this setting, deformation computations must be interactive, stable, and accurate for optimal immersion.

Step and project methods like XPBD are popular for the simulation of soft or rigid bodies as they allow fast stable simulations. However, the convergence of the solver is slow due to local propagation of information through Jacobi or Gauss-Seidel iterations. Hence, while these methods offer fast computation for time stepping, the slow convergence can lead to results that poorly approximate the expected physical behaviors of a system. To improve propagation, long-range attachments create additional

constraints to couple distant elements. Identifying where extra constraints are needed can be challenging or specific to a given scenario, and overall, the extra constraints increase the complexity of the system and the cost of the solve.

We introduce a multi-layer method for soft body simulation. Drawing inspiration from multigrid methods, the concepts of long-range constraints, and adaptive rigidification, we automatically generate various resolutions as mixes of rigid and elastic components with varying degrees of freedom. Our work derives from an opportunity to reduce models based on the current simulation state rather than the geometry or representation of our models. Through a coarse-to-fine sequence of solver iterations using these reduced models, the different couplings of elastic and rigid parts propagate distant information, improving the convergence of the XPBD solve. We handle contacts as both rigid and elastic throughout a time step, allowing efficient propagation of impulses. Our method has been tested across a range of scenarios, including contact-rich scenarios, purely rigid motions, and global deformation challenges. The results demonstrate improved convergence in all of our examples. We further explore the performance of different input parameters such as rigid patterns, number of iterations per layer, and the number of layers to provide deeper insights into the behavior of our method.

2. Related Work

One of the most important contributions to physics-based animation with respect to efficient time integration is unequivocally position-based dynamics (PBD) [MHHR07]. The method proposes a Gauss-Seidel-like solve for the motions of constrained particles by projecting linearized constraints one at a time. Bender et al. [BKCW14] demonstrate the applicability of PBD continuum elasticity simulation by modeling constraints as per-element elastic potentials. The extended position based dynamics (XPBD) [MMC16] framework addresses one of the early, yet major shortcomings of traditional PBD. Without the compliance term, elastic models are infinitely stiff and too many iterations causes them to behave as rigid bodies. In contrast, when rigid bodies are desired, XPBD can still be applied, noting that most of the displacement occurs in the fast symplectic stepping prior to the constraint solve, which instead deals with environmental interactions, such as integration of joints, mixed bodies and more [MMC*20].

Multigrid solvers find solutions to systems of equations by first eliminating, i.e., smoothing, high-frequency errors in the solution iterates. The remaining low-frequency error is then eliminated by solving a reduced version of the original problem, where such low frequencies become high frequencies, revealing its recursive structure. Applied to linear systems, a multigrid solver typically features multiple resolutions, i.e., levels, of the full space problem (e.g., constructed by simplification [LZBJ21]), with the intention of doing a few Jacobi or Gauss-Seidel iterations at each level as error smoothing agents. The exponential reduction in problem size between the hierarchy's levels yields fast convergence to the solution.

Xian et al. [XTL19] present a linear multigrid solver for physics-based animation based on projective dynamics [BML*14] using Newton-Raphson iterations, where resolutions are created from furthest point samplings of the high-resolution simulated mesh. Coarser level point samples act as linear blend skinning (LBS) handles for the immediate finer level points, with simulation mesh vertices as the finest level. By clamping the LBS weights to discrete binary values, they obtain restriction and prolongation operators which encourage linear system sparsity at coarser levels, yielding impressive computational efficiency. Unlike PBD and XPBD [MEM*20], the projected dynamics approach is tailored to the primal formulation.

In contrast, Müller [Mül08] proposes a non-linear multigrid solver for PBD where each layer is similarly a coarse point sampling of the original particle system. Here, the prolongation operator consists of weighted averaging of coarse grid solutions, with weights inversely proportional to approximate geodesic distance between coarse *parent* particles and their fine *children* particles. This multigrid solver goes solely from coarse to fine, approximately solving a reduced set of constraints at each level. The proposed scheme lends itself well to mass spring PBD deformable models, where coarsening occurs via edge collapses. Unfortunately, it remains unclear how to generalize the approach to the preferred continuum constraints [TKA23].

While XPBD is a fast method for real time simulation, the local nature of the constraint solve prevents efficient global propagation of deformation computations, which hinders convergence. To address this issue, long range attachements [KCM12] and long-range constraints [MCMJ17] have been proposed. Such approaches successfully create constraints between distant elements to explicitly enforce the desired propagation of local elastic effects, at the cost of added constraints. Nonetheless, the improved convergence dominates the additional per-iteration overhead. We take inspiration from this approach by using rigid bodies to improve propagation of information across long distances. The new constraints, while helping propagation, change the original optimization problem, impacting the final solution. Hence, these constraints need to be removed during the solve to obtain a ground truth simulation and ensure correct convergence. Unlike the previous work, the rigid bodies in our approach serve as approximate long range constraints that accelerate the convergence across solver iterations on a sequence of approximate systems, and these rigid bodies are absent in the final fully elastic solve. In contrast, long range attachments and constraints [KCM12; MCMJ17] create additional constraints that must be solved in addition to those of the fully elastic system.

We are also inspired by the concept of adaptive rigidification [MKWL22], which proposes that non-deforming parts of the simulation can be automatically discovered and adaptively transformed into rigid bodies on the fly. This technique is specifically tailored to Newton-Raphson solvers for standard finite element simulation, where the involved global linear systems of equations are significantly reduced by substituting elastic degrees of freedom with low-dimensional rigid motions. Unfortunately, the original adaptive rigidification algorithm is unsuitable to the XPBD framework, which explicitly ignores the problem's Hessian matrix needed for the oracle. As the assembly of such matrix is costly, we instead

propose an assembly-less method benefiting from many advantages of adaptive rigidification, without the need for an oracle.

Instead of approximating a simulation using coarsening methods like Delaunay remeshing [AD99], our proposed method is more similar to the work of Müller [Mül08]. Our resolutions are based on the current state of the simulation and its interactions with the environment, which is likewise similar to adaptive rigidification [MKWL22] but without the need of an oracle. This allows us to use a single unified model for all of our resolutions without resorting to the more involved geometric, algebraic, and functional hierarchical constructions of existing multigrid methodologies [MZS*11; Gui93; SVJ15]. Because we propose a fully-fledged iterative solver instead of approximating motions as rigid, we avoid contact handling problems like those described by Mercier-Aubin et al. [MK23] that would otherwise surface from the use of an oracle. Our approach instead speeds up the XPBD method by efficiently creating long-range propagation through a variety of rigid patterns, and without the need for domain knowledge. Much like the work of Barbié et al. [BRL15], our systems of different resolutions will be generated automatically. Rather than using refinement, our layers are simplifications of a fine model without any change to the geometry. As such, our method is not subject to the problems that would otherwise surface in subdivision methods.

3. Standard XPBD

We briefly review the original XPBD method for the relevant information and refer to the original paper for the details. Following Macklin et al. [MMC16], we start from the discretized formulation of Newton's equations of motion

$$M\left(\frac{\boldsymbol{x}^{t+1}-2\boldsymbol{x}^{t}+\boldsymbol{x}^{t-1}}{h^2}\right)=-\nabla U^T(\boldsymbol{x}^{t+1}), \tag{1}$$

where $U(\boldsymbol{x})$ is an energy potential, $\boldsymbol{x}$ is a vector of positions with superscript denoting the time step, M is the lumped mass matrix, and h a time step size. From a vector of constraints C and compliance block diagonal matrix α, we obtain the forces

$$-\nabla_{\boldsymbol{x}} U^T(\boldsymbol{x})=-\nabla C^T(\boldsymbol{x})\alpha^{-1}C(\boldsymbol{x}), \tag{2}$$

of our system. In typical XPBD fashion, this is solved using the approximate linearized constraint formulation of the system.

In XPBD, we first step the vertices in time using the symplectic Euler method. We start by updating the velocities, and then the positions

$$\dot{\boldsymbol{x}} \leftarrow \dot{\boldsymbol{x}}+hM^{-1}\boldsymbol{f}, \tag{3}$$
$$\boldsymbol{x} \leftarrow \boldsymbol{x}+h\dot{\boldsymbol{x}}, \tag{4}$$

of each elastic particle due to force $\boldsymbol{f}$. The solver uses the standard XPBD Gauss-Seidel-like updates, which includes a compliance term $\overline{\alpha}=\frac{\alpha}{h^2}$. With the Lagrange multipliers λ first initialized to zero, we iteratively solve for incremental updates

$$\Delta\lambda_j=\frac{-C_j(\boldsymbol{x})-\overline{\alpha}_j\lambda_j}{\nabla C_j(\boldsymbol{x})M^{-1}\nabla C_j^T(\boldsymbol{x})+\overline{\alpha}_j}, \tag{5}$$

for constraint j. We then convert the $\Delta\lambda_j$ impulse updates into position updates

$$\Delta\boldsymbol{x}=M^{-1}\nabla C(\boldsymbol{x})^T\Delta\lambda_j. \tag{6}$$

We use the Saint Venant-Kirchoff constitutive model expressed with Voigt notation [Cet23; SLM06] as our elastic potential. As such, our constraints are the lower triangular entries of the strain tensor $E=\frac{1}{2}(F^TF-I)$, thus, a vector of 6 constraints in 3D (or a vector of 3 constraints in 2D). We define the per-element blocks of the compliance matrix as

$$\alpha_e=\begin{bmatrix}\zeta+2\mu & \zeta & \zeta & 0 & 0 & 0\\ \zeta & \zeta+2\mu & \zeta & 0 & 0 & 0\\ \zeta & \zeta & \zeta+2\mu & 0 & 0 & 0\\ 0 & 0 & 0 & \mu & 0 & 0\\ 0 & 0 & 0 & 0 & \mu & 0\\ 0 & 0 & 0 & 0 & 0 & \mu\end{bmatrix}^{-1}, \tag{7}$$

where ζ and μ are the first and second Lamé parameters. We must preserve the coupling of constraints due to the off diagonals of the upper-left of α_e. This requires a solve such that the denominator of Equation 5 forms the left-hand side matrix and the numerator becomes the right-hand side vector. For each element, we solve the set s of coupled constraints as a 3-by-3 system (or 2-by-2 for 2D elements),

$$\left(\nabla C_s(\boldsymbol{x})M^{-1}\nabla C_s^T(\boldsymbol{x})+\overline{\alpha}_{ss}\right)\Delta\lambda_s=-C_s(\boldsymbol{x})-\overline{\alpha}_{ss}\lambda_s. \tag{8}$$

For the uncoupled constraints, i.e., corresponding to the lower right block of coefficients of Equation 7, we simply use the standard XPBD update from Equation 5.

For rigid bodies, the steps are similar [MMC*20]. We use symplectic Euler to step each rigid body's linear velocities and center of mass with Equation 3 and Equation 4. We also need to update each rigid body's torques τ, angular velocities $\omega\in\mathbb{R}^3$, and rigid body rotation $R\in SO(3)$ as

$$\tau \leftarrow r\times f, \tag{9}$$
$$\omega \leftarrow \omega+hI^{-1}(\tau-\omega\times I\omega), \tag{10}$$
$$R \leftarrow e^{h\widehat{\omega}}R, \tag{11}$$

where $I\in\mathbb{R}^{3\times3}$ is the inertia tensor and $\widehat{\omega}$ is the skew-symmetric cross product matrix. The matrix exponential provides the rotation update from the angular velocity vector and time step size, and is computed using the Rodrigues' formula [MSZ94].

Thus, the position of each vertex $\boldsymbol{x}_i$ making up rigid body r has position

$$\boldsymbol{x}_i=R_r\boldsymbol{r}_i+\boldsymbol{x}_r, \tag{12}$$

that is, they can be computed from the properties of rigid body r, with rotation R_r, center of mass $\boldsymbol{x}_r$, where $\boldsymbol{r}_i$ specifies the location of vertex $\boldsymbol{x}_i$ in the local frame of the rigid body.

3.1. Graph Coloring

We find independent constraints using graph coloring. While minimal graph coloring is a difficult problem, it is often reasonable to precompute a greedy graph coloring [FP15] by simply assigning each element a color unassigned to shared degrees of freedom from a set of colors. In Figure 2, we show an example of mesh graph coloring. We run constraints of the same color in parallel.

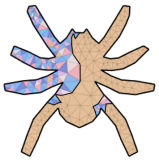


Figure 2: *Graph coloring example.*

When using partially rigid layers, the rigid bodies create distant coupling, hindering parallelization, however we note that each rigid body is independent from the others, otherwise they would be merged together. Therefore, it is reasonable to solve the rigid body constraints in parallel for the different rigid bodies. Because the constraints on a single rigid body are coupled, we solve them in a Jacobi style [MMCK14], while the rest of the constraints are solved in a Gauss-Seidel like fashion similar to standard XPBD.

3.2. XPBD Elastic and Rigid Coupling

In our rigid-elastic XPBD implementation, the coupling is not implicit. The elastic element update does not take into account the rigid bodies within the mesh. This creates a discrepancy between the rigid body and elastic body views of the position of vertices that are on the boundaries between rigid and elastic regions. We couple both views via an equality constraint similar to that of Müller et al. [MMC*20], that is, for vertex $\boldsymbol{x}_i$ on the boundary with rigid body r we have

$$C_i(\boldsymbol{x}) = \|\boldsymbol{x}_i - (R_r \boldsymbol{r}_i + \boldsymbol{x}_r)\|^2, \tag{13}$$

$$\Delta\lambda_i = \frac{-C_i(\boldsymbol{x}) - \bar{\alpha}_i \lambda_i}{w_i + \frac{1}{m_i} + \bar{\alpha}_i}, \tag{14}$$

where m_i is the mass of the elastic vertex, w_i is the generalized inverse mass of the vertex in the rigid body, and $\bar{\alpha}_i = 0$ to specify a hard constraint and preserve the rigid body boundary. We compute the generalized inverse mass as

$$w_i = \frac{n_C}{m_r} + (\boldsymbol{r}_i^{\mathrm{g}} \times \boldsymbol{d})^T I^{-1} (\boldsymbol{r}_i^{\mathrm{g}} \times \boldsymbol{d}), \tag{15}$$

where the numerator of the first term accounts for mass splitting [TBV12] with n_C being the number of constraints affecting the rigid body. Here, the vector $\boldsymbol{r}_i^{\mathrm{g}} = R_r \boldsymbol{r}_i$ points from the rigid body center of mass to the constrained vertex in the global frame, and $\boldsymbol{d}$ is the normalized direction of vector $\boldsymbol{x}_i - (R_r \boldsymbol{r}_i + \boldsymbol{x}_r)$, i.e., the correction direction for the equality constraint.

We get a valid coupling when the boundary vertices match the rigid body surface. We use a modified position update similar to that of Müller et al. [MMC*20] except that instead of working with two rigid bodies we update the elastic particle and a rigid body,

$$\boldsymbol{p} = \Delta\lambda \boldsymbol{d}, \tag{16}$$

$$\Delta\boldsymbol{x}_i = \frac{\boldsymbol{p}}{m_i}, \tag{17}$$

$$\Delta\boldsymbol{x}_r = -\frac{\boldsymbol{p}}{m_r}, \tag{18}$$

$$\Delta\omega_r = -\frac{1}{h} I^{-1} (\boldsymbol{r}_i^{\mathrm{g}} \times \boldsymbol{p}). \tag{19}$$

Some iterations feature strongly coupled constraints, but quaternion multiplications are not commutative, hindering constraint parallelization. So typical XPBD simulations approximate quaternion multiplications as commutative sums with the assumption of infinitesimal time steps (e.g., as done in Kalman filters [MYB*01]). In Equation 19, we instead use a Jacobi style parallel accumulation of angular velocities for dependent constraints on rigid bodies [BYM05], which offers the same benefits. Commutativity simplifies our constraint solves by allowing the computation of all rigid body boundary constraints simultaneously.

After solving all the boundary constraints of a rigid body in parallel, we update the rotation using Equation 11 from the accumulated change of angular velocity. We then update the relevant particle positions of a rigid body with respect to the new rotation and center of mass. We solve the rigid body constraints last, therefore we only update their vertex positions once per iteration.

3.3. Multi-Layer Method For XPBD

Our method has the nice property of allowing natural generation of multiple resolutions on the fly without remeshing by using adaptive rigidification concepts. Because the adaptive layers preserve the mesh vertices, the coarsening and refinement operations are intuitive. In the context of this multi-layer solver, we define coarse as a geometric model with more elements simulated as rigid, and fine as a model with more elastic elements. We first need a sorting process to determine the priority of element rigidification, e.g., sorting elements by strain rate. Then we gather elements into rigid bodies incrementally by inserting elements in the given order.

3.3.1. Rigidification

In the work of Mercier-Aubin et al. [MKWL22], the rigidification process consists of monitoring the strain rate computed as a finite difference

$$\dot{E} = \frac{E^t - E^{t-1}}{h}, \tag{20}$$

to detect non-deforming elements at each time step. This is compatible with any per-element strain measure E like the Green strain. While the adaptive rigidification method uses the strain rate to determine non-deformation, we use it to generate problems of different sizes to approximate the solution of the fully elastic model.

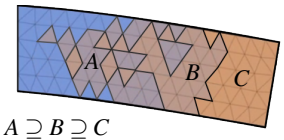
3.3.2. Hierarchy generation

We create the resolutions of a multi-layer solver using strain-rate dependent rigidification patterns. We build the hierarchies prior to the solve, from fine to coarse, by inserting the elements to be

Algorithm 1: INCREMENTALMERGING

```
input : Sorted elements grouped by layer, fine to coarse
output: Connectivity of layers
Initialize union-find structure u[e] = −1 for all elements e
Initialize claimed vertices v.claimed = −1 for all vertices v
foreach layer l do
    foreach uninserted element e of layer l do
        u[e] ← e
        foreach vertex v of element e do
            // find returns -1 when the input has no root
            s ← u.find(v.claimed)        // with path compression
            if e ≠ s ∧ s ≠ −1 then
                u[e] ← s
            end
            v.claimed ← e
        end
    end
    L[l] ← u        // snapshot of u provides connectivity of layer l
end
```

treated as rigid, increasing in strain-rate order, into a disjoint set. This is efficient due to path compression and the use of contiguous memory [TvL84]. Construction of each coarser layer includes the previously rigid elements from the past layer (continuing with the same disjoint set) as shown in Figure 3.



$A \supseteq B \supseteq C$

Figure 3: *Different incremental rigid patterns created from strain-rate insertion.*

Initially the vector containing our disjoint set is initialized with all entries set to -1. This vector has a size equivalent to the number of elements. On element insertion, a new set is created at the index of the inserted element, pointing to itself as its root. We then verify if the vertices of the element are already part of a set. When a rigidifying element is vertex-adjacent to existing rigid sets, we merge the sets. We do path compression when the disjoint-set function find is called. Because we are only ever inserting one new element at a time, the computation time to find the connectivity remains linear. Therefore the algorithm has a computation upper bound tied to the sorting algorithm rather than the creation of components. We do a single pass over the elements to generate the hierarchies as shown in Algorithm 1. Unlike adaptive rigidification, we do not need to handle hinges as the partially rigid layers only serve as useful intermediate models. As such, at a given layer, if two rigid bodies share a single vertex we merge them into a single rigid body.

This type of insertion allows us to incrementally update the connected components that will form rigid bodies in each resolution. While inserting elements, we keep a copy of the connected components for each slice of a predetermined percentage of all elements. After building these models of the system at different resolutions, we can switch layers freely, using refinement and coarsening operations (see Figure 4). Here, *coarsening* consists of removing the elastic constraints of the elements that are rigid in the coarse layer, and adding extra coupling constraints on the new rigid boundary. In contrast, *refinement* queries all the current vertex positions from rigid body properties, updates boundary constraints, handles residual velocities, and reintroduces elasticity constraints to newly elastic elements. We introduce residual velocities in Section 3.4.

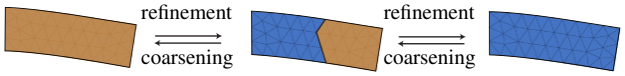


Figure 4: *Example of a sequence of layers with precomputed rigidification patterns. The solver iterates through the different layered resolutions using refinement operations to add elasticity during the solve.*

3.4. Iterating Through Layers

Doing a symplectic step before projecting the constraints would lead to inflated elements because the particles of an element move on straight lines before the first rigid layer on rotational motions. This would lead to inaccurate shapes for rigid bodies and hinder convergence. Instead we progressively step the velocities by depleting them over many layers using residual velocities.

Residual velocities are leftover internal elastic velocities inside of rigid bodies. Initially the residual velocities replace the velocity update for the entire system. Rigid body angular and linear velocities are computed from the per-particle residual velocities $\boldsymbol{v}_i$. On layer switch, we must compute the new rigid body properties from the vertices of rigid body r,

$$m_r = \sum_{i \in r} m_i, \tag{21}$$

$$\boldsymbol{x}_r = \frac{1}{m_r} \sum_{i \in r} m_i \boldsymbol{x}_i, \tag{22}$$

$$\dot{\boldsymbol{x}}_r = \frac{1}{m_r} \sum_{i \in r} m_i \boldsymbol{v}_i, \tag{23}$$

$$\boldsymbol{r}_i = \boldsymbol{x}_i - \boldsymbol{x}_r, \tag{24}$$

$$I_r = \sum_{i \in r} m_i \hat{\boldsymbol{r}}_i^T \hat{\boldsymbol{r}}_i, \tag{25}$$

$$\omega_r = I_r^{-1} \sum_{i \in r} m_i \boldsymbol{r}_i \times (\boldsymbol{v}_i - \dot{\boldsymbol{x}}_r). \tag{26}$$

These values remain fixed throughout the solver iterations of the layer. Hence, we compute the relative positions $\boldsymbol{r}_i$ once per layer. Throughout the solve, We let $\boldsymbol{x}_i^{t+1}$ be the current approximation of the position vertex i at the next time step (it is initialized to $\boldsymbol{x}_i^t$), and $\boldsymbol{x}_i$ we use to denote the position of the vertex at the beginning of the solve of the current layer. For vertices that are part of rigid bodies, the layer solve steps their positions with rigid motion. That is, we obtain a rotation update R_r by stepping the rigid bodies, and vertices that are part of the body have their positions computed based on the rigid body state (Equation 12). At this point, the vertex velocities for this rigid motion can be computed as $\frac{1}{h}(\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i)$, but there can still be non rigid velocities in the residual velocity $\boldsymbol{v}_i$. Thus, we rotate the current residual and subtract the current

velocity to update the residuals, i.e.,

$$\boldsymbol{v}_i \leftarrow R_r \boldsymbol{v}_i - \frac{1}{h}(\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i). \tag{27}$$

The process of updating residuals throughout the layers continues until the elements are solved as elastic in the final iterations. Thus each layer has less residual velocities to rigidly step as they are depleted through rigid body motions. To be clear, it is only vertices that are part of rigid bodies that have residual velocities, while residual velocities are zero (depleted) for vertices that are stepped elastically.

Note that the residual velocities must be rotated in Equation 27 because they are unstepped velocities at time t while the finite difference of particle positions are rigidly stepped velocities at time $t+1$. In the case of a purely rigid motions of a spinning elastic body in equilibrium, these velocities exactly cancel out. This is because the initial residual velocities are exactly a rigid motion (the previous step velocities, which are computed using finite differences at the end of each simulation step in XPBD).

Because rigid bodies move during the symplectic steps, we rotate the residual velocities to match the rigid body frames and preserve the inner elastic velocities. The constraint solves also change the rigid body positions and orientations, which requires a rotation update to the residual velocities before changing layers where $\Delta\omega_r$ is the change in angular velocities due to constraint solves.

This modification to XPBD requires only minor changes around the solver, making it compatible with most existing frameworks. This can be seen in Algorithm 2.

3.5. Contact Handling

To handle contacts, we use penalty constraints

$$C_c(\boldsymbol{x}) = \min(0, \boldsymbol{d}_c \cdot (\boldsymbol{p}_c - \boldsymbol{x}_c)), \tag{28}$$

for each vertex $\boldsymbol{x}_c$ in contact, and correct the interpenetration at contact $\boldsymbol{p}_c$ with normal $\boldsymbol{d}_c$. We set the compliance parameter α to 10^{-4}, which is the value recommended in the original XPBD work. The constraint is treated in a consistent way, regardless if the vertex $\boldsymbol{x}_c$ is part of a rigid body or an elastic element.

For a given layer, all contacts on elastic vertices are independent and solved in parallel. We find the constraint Lagrangian update using Equation 5. All contacts on rigid vertices are solved using the same procedure as Section 3.2, but with the elastic particle in this case being replaced with an infinite mass contact position $\boldsymbol{p}_c$, and the constraint direction being the contact normal. This means that contacts are often solved as both rigid and elastic, during a single time step.

As proposed by Müller et al. [MMC*20], we handle restitution after the solve with a velocity update

$$\Delta\dot{\boldsymbol{x}}_c = \boldsymbol{n}_c(\min(-\epsilon \boldsymbol{n}_c \cdot \dot{\boldsymbol{x}}_c, 0) - \boldsymbol{n}_c \cdot \dot{\boldsymbol{x}}_c), \tag{29}$$

where ϵ is a restitution parameter, and $\boldsymbol{n}_c$ is the contact normal. As this is a post-solve operation, we consider the soft body contacts as fully elastic and run this operation in parallel as independent contacts.

Algorithm 2: MULTI-LAYERXPBD

input : Position vector $\boldsymbol{x}_t$
Velocity vector $\dot{\boldsymbol{x}}_t$
Step size h
External forces vector $\boldsymbol{f}_{ext}$
output: Positions $\boldsymbol{x}^{t+1}$ and velocities $\dot{\boldsymbol{x}}^{t+1}$ after stepping

$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t$
$\boldsymbol{v} \leftarrow \dot{\boldsymbol{x}}^t + hM^{-1}\boldsymbol{f}_{\text{ext}}$ // residual velocity initialization
$L \leftarrow$ INCREMENTALMERGING // Algorithm 1
foreach layer $l \in L$ **do**
  foreach newly elastic vertex i of layer l **do**
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i^{t+1}$ // vertex at the start of the layer
    $\boldsymbol{x}_i^{t+1} \leftarrow \boldsymbol{x}_i^{t+1} + h\boldsymbol{v}_i$ // step with residual velocity
  end
  foreach rigid body r in layer l **do**
    $\omega_r, \dot{\boldsymbol{x}}_r, I_r, M_r \leftarrow$ RIGIDPROPERTIES$(l, \boldsymbol{v})$ // Section 3.4
    $R_r, \boldsymbol{x}_r \leftarrow$ STEPRIGIDBODIES$(\omega_r, \dot{\boldsymbol{x}}_r)$ // Section 3
    foreach vertex i of each rigid body r **do**
      $\boldsymbol{x}_i^{t+1} \leftarrow R_r \boldsymbol{r}_i + \boldsymbol{x}_r$ // Equation 12
      $\boldsymbol{v}_i \leftarrow R_r \boldsymbol{v}_i - \frac{1}{h}(\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i)$ // Equation 27
    end
  end
  foreach iteration for layer l **do**
    $\Delta\omega_r, \boldsymbol{x}^{t+1} \leftarrow$ SOLVECONSTRAINTS
  end
  foreach rigid vertex i of layer l **do**
    $\boldsymbol{v}_i \leftarrow e^{h\Delta\widehat{\omega}_r}\boldsymbol{v}_i$ // rotate residual velocities
  end
end
$\dot{\boldsymbol{x}}^{t+1} \leftarrow \frac{1}{h}(\boldsymbol{x}^{t+1} - \boldsymbol{x}_t)$
$\dot{\boldsymbol{x}}^{t+1} \leftarrow$ RESTITUTIONUPDATE$(\dot{\boldsymbol{x}}^{t+1})$ // Equation 29

3.6. Layer-Stop Criterion

Solving constraints in a partially rigidified mesh can lead to stagnation if the remaining error is located inside of rigidified elements. Implementing a stop criterion based on error improvement can enhance speed, at the cost of the constant-time property of our solver. Because the first few steps of XPBD solvers often lead to an increase in residual error, we activate this feature only after observing the first decrease in constraint residual error

$$\|C + \overline{\alpha}\lambda\|. \tag{30}$$

When the change in constraint residual is near zero (below 1e-8) for an iteration, we switch directly to the next layer in the sequence. This allows the solver to quickly switch to the fully elastic layer.

4. Results

We evaluate our method on different fronts to develop the intuition on how to tune the parameters of our multi-layer solver and to validate our various hypotheses on the behaviour of the solver.

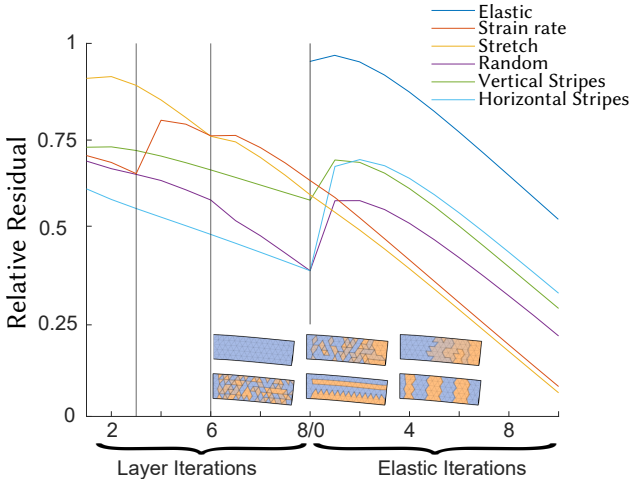


Figure 5: *Comparison of convergence per coarsening pattern: On the left the plot shows convergence including the reduced iterations, while on the right, we see the convergence on the fully elastic layer. The vertical lines represent layer switches.*

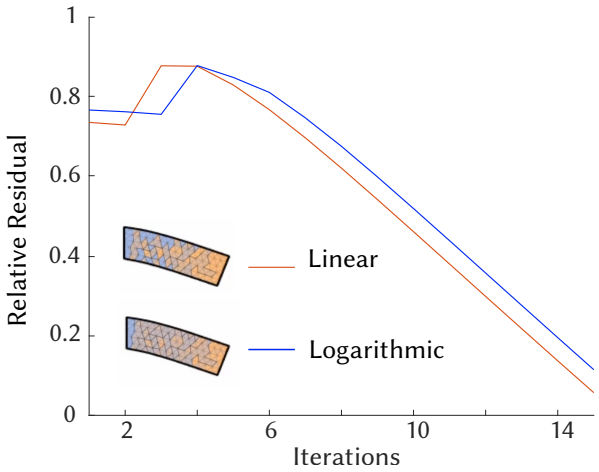


Figure 6: *Linear or logarithmic: choosing linear changes between the size of layer groups lead to better convergence compared to logarithmic halving group size for this cantilever example.*

4.1. Choice of Pattern

The choice of an adequate rigid pattern is critical to efficiently propagate motions. In Figure 5 we compare orders of insertion into the disjoint set, leading to different rigid patterns. Our tests include insertions in random order, strain-rate based, stretch-based, and vertical or horizontal stripes. We note that the stretch-based ordering using the eigenvalues of $M^{-1}K$ is simply here for comparison purposes as it is an expensive measure due to the assembly of the stiffness matrix and eigenvalue decomposition. The strain-based approach provides similar convergence rates to the stretch-based approach, albeit at a much cheaper cost. The random patterns sometimes lead to good convergence, but are unreliable and just as often lead to worse performance. The stripe patterns correspond to cases where an animator has domain knowledge for how a complex mesh would deform. That is, prior specification of layers may potentially be useful in niche applications. Hence, we suggest using strain-rate based ordering for the insertion of elements in the disjoint set to generate the layers. We note that the error sometimes increases on layer switch, which is not unexpected because of the stepping of residual velocities in the system on layer switch.

4.2. Choice of Layer Group Sizes

Because large deformations are more likely to impact global simulation than tiny elastic vibrations, we suggest that starting with more aggressively rigidified layers first is more likely to lead to faster convergence than coarsening after spending iterations on a fine resolution solve. Hence, we always start our test from rigid to elastic. Likewise, there are different ways to select the change in rigidification group sizes per layer. In Figure 6 we compare different types of layer selection for a standard cantilever example. We break the elements into groups with an equal number of elements in each such that the number of rigid elements across the different layers changes linearly. We try starting rigid and increasingly elastifying elements with constant size jumps of 25% rigidification. We also test halving the number of elements rigid for each layer, starting at 100% rigidification and ending at around 12% before going fully elastic. The logarithmic approach underperforms due to the change in group sizes being too steep initially, missing the opportunity to propagate important rigid motions.

We note that steps with purely rigid motions can instantly terminate early as the symplectic stepping of rigid bodies generate low error solutions before solving the elastic constraints. We show this phenomenon in Figure 9a.

4.3. Number of Layers

For each layer we must build the current rigid body and compute the properties like the rotation and center of mass. As such, there is a small linear overhead over rigid bodies on layer switch. Switching between many small rigid layer group sizes with a low number of iterations could hinder performances. Hence, choosing an appropriate number of layers is important.

A case-by-case selection can allow optimal performances. For instance, if we know that 30% of the elements are stiffer, like the rim of a wheel, then that proportion could be included in the layers. A single initial 100% layer is only a good start for scenes without pinned vertices to instantly propagate global rigid motions (otherwise, with pinned vertices, the fully rigid layer does not move and provides no benefit). In Figure 7 we show the cantilever test with different numbers of layers. Because the cantilever is pinned, we start after the first reduction in percentages. We also double the number of iterations done per layer. For instance, the 50% plot does one iteration in the layer 50%, while the 33% plot does 1 iteration in the layer 66% and 2 in the layer with 33% rigidification. Because the plot for the 10% decrease in group size has too many layers to double the iteration count, we evenly distribute the computation using 2 iterations per layer.

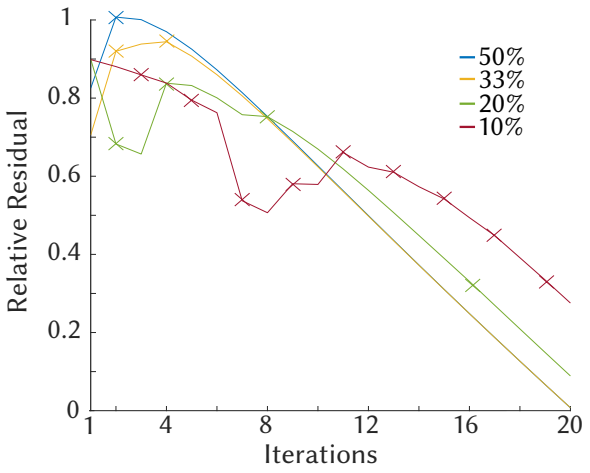


Figure 7: *Test of the impact of the layer size in the cantilever scene for linearly distributed layers. The cross markers show the iterations where layer switches happen.*

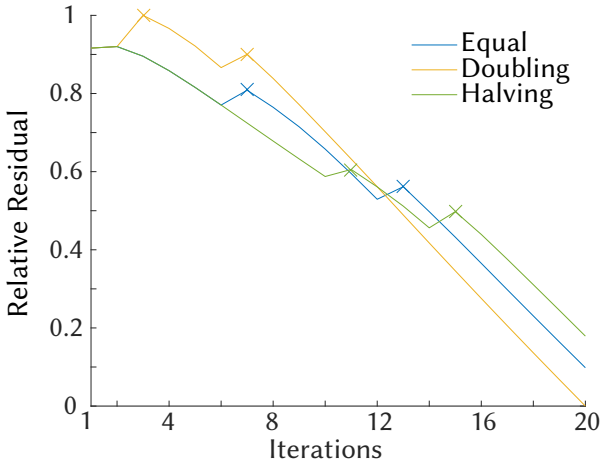


Figure 8: *For the cantilever example, increasing the number of iterations as we elastify the model yields the best performances. The cross markers show layer switches.*

4.4. Number of Iterations per Layers

For more aggressively rigidified layers, it is reasonable to assume that fewer iterations are needed, because the problem is effectively smaller. Likewise, doing too many iterations in any partially rigidified layer can lead to convergence plateaus. In Figure 8, we explore different iteration numbers per layer to validate our hypothesis. In our test, we use equal number of iterations (6 per rigid layer), increasing iterations by doubling the number of iterations per layer, and decreasing iterations by halving the number of iterations per layer. We notice how the doubling approach outperforms the other by doing less iterations in aggressive layers, while spending more time in finer layers.

4.5. Example Simulations

We designed challenging test scenes to evaluate the performances of our method, featuring varying degrees of deformations and contacts. Figure 9 shows our example models and compares the run times of our approach to XPBD with histograms of the solve times necessary to achieve a desired amount of error for each time step in a typical simulation trajectory.

We run our experiments on CPU, with efforts made for parallelization of the code using vectorized MATLAB code. Our tests use relative measures so as to provide a fair evaluation of the performances. While our current implementation could be ported to GPU, modifications are necessary to adapt the dynamic nature (e.g., problem sizes) of the method. A compiled language could also provide performance improvement especially for the computation of constraint gradients.

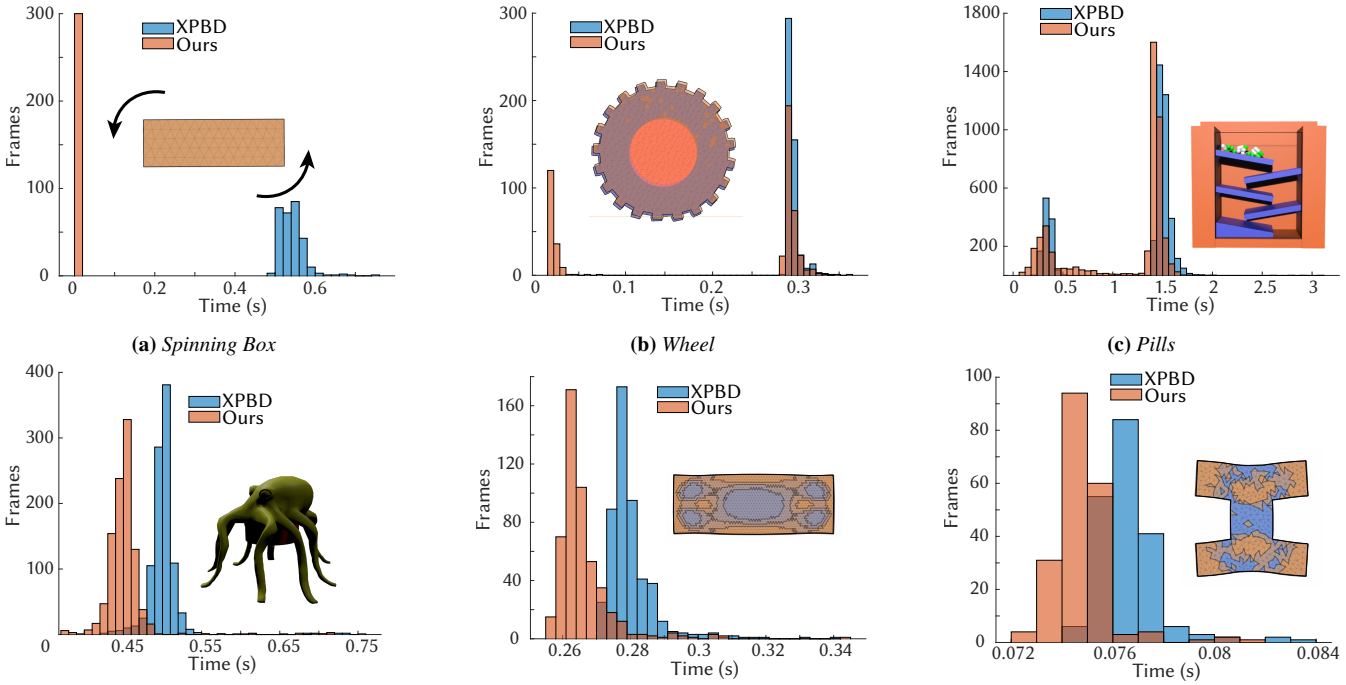
While our solver offers ground truth solutions, the difference in convergence and constraint error distribution can lead to different behaviors when compared to a standard XPBD solve. We note that the standard XPBD method can introduce error due to the linear discretization of impulses, leading to inflation during rigid motions. This is not the case in our solver when using a fully rigid first layer, as demonstrated by the spinning box shown in Figure 9a. The simulation of a spinning box instantly terminates because the rigid motions are solved accurately in the symplectic step of the first layer.

In Figure 9b we notice two different distributions due to the fast solve of purely rigid motions. Because of the imprecision of elastic simulations, we can see a divergence in the simulations. Our solver preserves the purely rigid motions in this wheel example much like it does in the spinning box example.

Our solver can speed up contact heavy scenes like the pills machine of Figure 9c. Due to improved propagation, our method consistently yield better performances even in a contact heavy scene. We note that performances could be further improved by analyzing the residual error locally instead of globally, allowing early stop for various independent regions like individual pills.

Even in active scenes like Figure 9d, motions can often be represented as a set of rigid body motions, leading to a handful of nearly free time steps and overall cheaper solves. A simpler example with global deformation is shown in Figure 9e where a box is stretched and released, providing no undeforming region. Even in this fully deforming scene, the residual velocity solver outperforms the elastic simulation at all time. Likewise, in Figure 9f, global deformation in wildly diverging directions lead to adequate rigid body formation, and ultimately an enhancement in performances. In both cases, the simulation benefits from the rigid motions of the first layers, propagating information to distant elements similar to long-range constraints.

Much like adaptive rigidification, our algorithm shows increased benefits when used with finer meshes. For instance, while simulating a 3 by 1 box stretch horizontally by 10% before being released, we see that refining the model increases the proportion of computation time saved. In Table 1, we show the percentages of improvement for different resolutions of this scene. We also monitor the proportion of time that the sort and the connected



(a) *Spinning Box* **(b)** *Wheel* **(c)** *Pills*

(d) *Octopus* **(e)** *Inflated Box* **(f)** *I-Shaped*

Figure 9: *Histograms of wall clock time to simulate a frame with a stop criterion of 90% improvement in residual error for the frame compared to the initial residual of the elastic simulation. In all these various examples, we see improvements in performance.*

Table 1: *Comparison across resolutions. For the simulation of a box stretched horizontally by 10%, the proportion of time taken sorting and computing connected components is minimal, while the overall performance improvement in comparison to the full elastic solve becomes large at higher resolution.*

Elements	651	1550	3304
Sort and connected components	0.01%	0.017%	0.02%
Performance improvement	0.1%	5.8%	11.93%

components occupy in the solve. We note that while it is not currently a bottleneck, scenes featuring very large numbers of elements would benefit from parallelization of this algorithm.

Using the octopus, we also evaluate scalability across stop criteria in Figure 10. We see that the benefits of using the multi-layer method becomes more apparent for higher residual-reduction stop thresholds. Obviously solving the system more precisely requires more time, but we note that the performance gain increases as we raise the improvement percentage threshold.

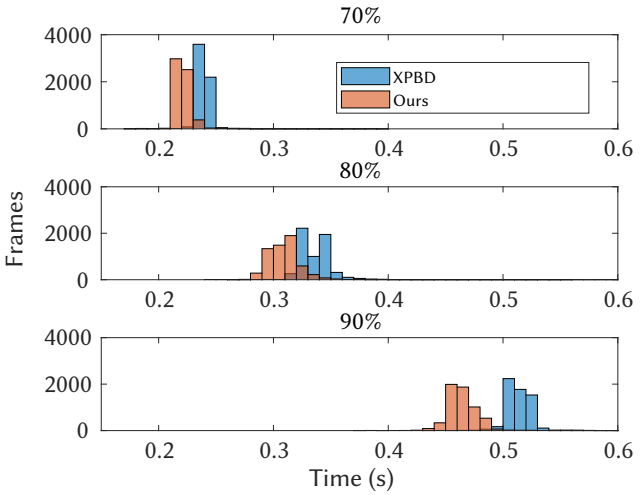


Figure 10: *In the octopus scene, the gap between the multi-layer and XPBD solvers in terms of computation time increases proportionally to precision.*

5. Discussion and Limitations

Our implementation comes with the same drawbacks as its overarching method. Based on XPBD, it suffers from some of the same convergence issues. While our method does improve convergence throughout our examples, the worst case scenario remains in the same order of magnitude as the original XPBD method. This happens when the solver hits a plateau for many iterations without using an early stop for the layer. However, our method also makes the generation of long range constraints trivial by not requiring any domain knowledge, and adapting itself to the current environmental

inputs. Likewise, our method can propagate contacts efficiently by treating them as both elastic and rigid while we iterate through the layers. For rigid motions, our solver instantly reduces the error up to relevant numerical accuracy making the simulations efficient, while retaining ground truth accuracy.

Our method uses concepts similar to adaptive rigidification for the choice of layers, but without an oracle. As such the last layer must always be elastic to generate good strain rates at the next time step. Future work could create an oracle for XPBD simulations that would render this method compatible with the approximate simulations of adaptive rigidification, further improving the speedups when accuracy is not critical. Because XPBD benefits from not assembling the system's Hessian matrix, the original oracle of adaptive rigidification is incompatible with such framework. We theorize that an alternative oracle could be to do elastic iterations first, then proceed with rigidified parts, but this would likely be subject to localized propagation issues.

Likewise, the current implementation does not include co-dimensional shell simulations. Such simulation requires extra care for the bending components [MK23]. It is not obvious that locking bending angles would yield good first steps in the partially rigid layers. A potential alternative could be to only use multi-layers on the 2D projection of the shells, leaving the bending components fully simulated.

There are potential opportunities to parallelize the union-find connected component builder [GPP*14; CNS*18]. Likewise, we use a standard sorting algorithm, which could be parallelized using forms of radix parallel sorts. Furthermore, we do not need a full sort. That is, the grouped elements at a fine layer do not need to be sorted and are only expected to have lower strain rate than those elements in the group at the next coarser layer. Nevertheless, the sort and union-find are not the current bottlenecks of our implementation.

While early stops allow fast simulations of deformables, setting a constant number of layers and iterations can have potential uses for real time applications where constant time solves are important. With our simulator, it is easy to allocate a constant amount of resources for elasticity and still benefit from improved convergence. This is an important feature for real-time application as resources are often limited.

Our method shares similarities with multigrid methods, which would make the introduction of a v-cycle pattern intuitive, starting elastic to rigid and back to elastic. However, such pattern lose the benefits of residual velocity layers, and the possibility to instantly terminate on rigid motions during the first iteration.

While we use strain-based energies for our elasticity, the overarching multi-layer solver is independent of such constraint and could be used with other popular formulations like the stable Neo-Hookean from Macklin et al. [MM21].

6. Conclusions

We present a multi-layer approach for the simulation of soft bodies with XPBD. Our method converges faster than standard XPBD by automatically generating cheap coupling similar to long-range constraints through the use of rigid bodies. Without using a stop criterion for layers, our solver provides an iterative solution that offers steady performances across frames.

While iterating through coarse layers, the solver provides an efficient propagation of information to distant elements with a significantly reduced number of constraints to solve. We hope that this new method will inspire the community to build upon our work and generate novel multi-layer solvers working outside of the typical algebraic or geometric multigrid approaches.

7. Acknowledgment

This research was funded by the FRQNT Doctoral scholarship 332127. We are grateful to the Fonds de recherche du Québec for their resources. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) via the Discovery grant program and Alliance grant ALLRP-570702-21 with Symgery. We likewise thank S. Andrews and Q. M. Ton-That for their invaluable insights and feedback throughout the development of this project.

References

[AD99] ADAMS, M. and DEMMEL, J. "Parallel Multigrid Solver for 3D Unstructured Finite Element Problems". *SC '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*. 1999, 27–27. DOI: 10.1145/331532.331559 3.

[BKCW14] BENDER, JAN, KOSCHIER, DAN, CHARRIER, PATRICK, and WEBER, DANIEL. "Position-based simulation of continuous materials". *Computers & Graphics* 44 (2014), 1–10. ISSN: 0097-8493. DOI: 10.1016/j.cag.2014.07.004 2.

[BML*14] BOUAZIZ, SOFIEN, MARTIN, SEBASTIAN, LIU, TIANTIAN, et al. "Projective Dynamics: Fusing Constraint Projections for Fast Simulation". *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601116 2.

[BRL15] BARBIÉ, L., RAMIÈRE, I., and LEBON, F. "An automatic multilevel refinement technique based on nested local meshes for nonlinear mechanics". *Computers & Structures* 147 (2015). CIVIL-COMP, 14–25. ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2014.10.008 3.

[BYM05] BELL, NATHAN, YU, YIZHOU, and MUCHA, PETER J. "Particle-based simulation of granular materials". *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '05. Los Angeles, California: Association for Computing Machinery, 2005, 77–86. ISBN: 1595931988. DOI: 10.1145/1073368.1073379 4.

[Cet23] CETINASLAN, OZAN. "ESBD: Exponential Strain-based Dynamics using XPBD algorithm". *Computers & Graphics* 116 (2023), 500–512. ISSN: 0097-8493. DOI: 10.1016/j.cag.2023.09.014 3.

[CNS*18] CHEN, JUN, NONAKA, KEISUKE, SANKOH, HIROSHI, et al. "Efficient Parallel Connected Component Labeling With a Coarse-to-Fine Strategy". *IEEE Access* 6 (2018), 55731–55740. DOI: 10.1109/ACCESS.2018.2872452 10.

[FP15] FRATARCANGELI, M. and PELLACINI, F. "Scalable Partitioning for Parallel Position Based Dynamics". *Computer Graphics Forum* 34.2 (2015), 405–413. DOI: 10.1111/cgf.12570 4.

[GPP*14] GUPTA, SIDDHARTH, PALSETIA, DIANA, PATWARY, MD. MOSTOFA ALI, et al. "A New Parallel Algorithm for Two-Pass Connected Component Labeling". *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. 2014, 1355–1362. DOI: 10.1109/IPDPSW.2014.152 10.

[Gui93] GUILLARD, HERVÉ. *Node-nested multi-grid method with Delaunay coarsening*. Research Report RR-1898. INRIA, 1993 3.

[KCM12] KIM, TAE-YONG, CHENTANEZ, NUTTAPONG, and MÜLLER-FISCHER, MATTHIAS. "Long range attachments - a method to simulate inextensible clothing in computer games". *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '12. 2012, 305–310. DOI: 10.2312/SCA/SCA12/305-310 2.

[LZBJ21] LIU, HSUEH-TI DEREK, ZHANG, JIAYI ERIS, BEN-CHEN, MIRELA, and JACOBSON, ALEC. "Surface Multigrid via Intrinsic Prolongation". *ACM Trans. Graph.* 40.4 (2021) 2.

[MCMJ17] MÜLLER, MATTHIAS, CHENTANEZ, NUTTAPONG, MACKLIN, MILES, and JESCHKE, STEFAN. "Long Range Constraints for Rigid Body Simulations". *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '17. 2017. ISBN: 9781450350914. DOI: 10.1145/3099564.3099574 2.

[MEM*20] MACKLIN, M., ERLEBEN, K., MÜLLER, M., et al. "Primal/-Dual Descent Methods for Dynamics". *Computer Graphics Forum* 39.8 (2020), 89–100. DOI: 10.1111/cgf.14104 2.

[MHHR07] MÜLLER, MATTHIAS, HEIDELBERGER, BRUNO, HENNIX, MARCUS, and RATCLIFF, JOHN. "Position based dynamics". *Journal of Visual Communication and Image Representation* 18.2 (2007), 109–118. ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2007.01.005 2.

[MK23] MERCIER-AUBIN, ALEXANDRE and KRY, PAUL G. "Adaptive Rigidification of Discrete Shells". *Proc. ACM Comput. Graph. Interact. Tech.* 6.3 (Aug. 2023). DOI: 10.1145/3606932 3, 10.

[MKWL22] MERCIER-AUBIN, ALEXANDRE, KRY, PAUL G., WINTER, ALEXANDRE, and LEVIN, DAVID I. W. "Adaptive Rigidification of Elastic Solids". *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530124 2–4.

[MM21] MACKLIN, MILES and MULLER, MATTHIAS. "A Constraint-based Formulation of Stable Neo-Hookean Materials". *Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games*. MIG '21. Virtual Event, Switzerland: Association for Computing Machinery, 2021. ISBN: 9781450391313. DOI: 10.1145/3487983.3488289 10.

[MMC*20] MÜLLER, MATTHIAS, MACKLIN, MILES, CHENTANEZ, NUTTAPONG, et al. "Detailed Rigid Body Simulation with Extended Position Based Dynamics". *Computer Graphics Forum* 39.8 (2020), 101–112. DOI: 10.1111/cgf.14105 2–4, 6.

[MMC16] MACKLIN, MILES, MÜLLER, MATTHIAS, and CHENTANEZ, NUTTAPONG. "XPBD: Position-Based Simulation of Compliant Constrained Dynamics". *Proceedings of the 9th International Conference on Motion in Games*. MIG '16. 2016, 49–54. ISBN: 9781450345927. DOI: 10.1145/2994258.2994272 2, 3.

[MMCK14] MACKLIN, MILES, MÜLLER, MATTHIAS, CHENTANEZ, NUTTAPONG, and KIM, TAE-YONG. "Unified particle physics for real-time applications". *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601152 4.

[MSZ94] MURRAY, RICHARD M., SASTRY, S. SHANKAR, and ZEXIANG, LI. *A Mathematical Introduction to Robotic Manipulation*. 1st. USA: CRC Press, Inc., 1994. ISBN: 0849379814 3.

[Mül08] MÜLLER, MATTHIAS. "Hierarchical Position Based Dynamics." Vol. 8. Jan. 2008, 1–10. DOI: 10.2312/PE/vriphys/vriphys08/001-010 2, 3.

[MYB*01] MARINS, J.L., YUN, XIAOPING, BACHMANN, E.R., et al. "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors". *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 4. 2001, 2003–2011 vol.4. DOI: 10.1109/IROS.2001.976367 4.

[MZS*11] MCADAMS, ALEKA, ZHU, YONGNING, SELLE, ANDREW, et al. "Efficient elasticity for character skinning with contact and collisions". *ACM Trans. Graph.* 30.4 (July 2011). ISSN: 0730-0301. DOI: 10.1145/2010324.1964932 3.

[SLM06] SERVIN, MARTIN, LACOURSIÈRE, CLAUDE, and MELIN, NIKLAS. "Interactive Simulation of Elastic Deformable Materials". *Proceedings of SIGRAD*. Jan. 2006, 22–32. ISBN: 91-85643-17-3 3.

[SVJ15] SACHT, LEONARDO, VOUGA, ETIENNE, and JACOBSON, ALEC. "Nested cages". *ACM Trans. Graph.* 34.6 (Nov. 2015). ISSN: 0730-0301. DOI: 10.1145/2816795.2818093 3.

[TBV12] TONGE, RICHARD, BENEVOLENSKI, FEODOR, and VOROSHILOV, ANDREY. "Mass Splitting for Jitter-Free Parallel Rigid Body Simulation". *ACM Trans. Graph.* 31.4 (July 2012). ISSN: 0730-0301. DOI: 10.1145/2185520.2185601 4.

[TKA23] TON-THAT, QUOC-MINH, KRY, PAUL G., and ANDREWS, SHELDON. "Parallel block Neo-Hookean XPBD using graph clustering". *Computers & Graphics* 110 (2023), 1–10. ISSN: 0097-8493. DOI: 10.1016/j.cag.2022.10.009 2.

[TvL84] TARJAN, ROBERT E. and van LEEUWEN, JAN. "Worst-Case Analysis of Set Union Algorithms". *J. ACM* 31.2 (Mar. 1984), 245–281. ISSN: 0004-5411. DOI: 10.1145/62.2160 5.

[XTL19] XIAN, ZANGYUEYANG, TONG, XIN, and LIU, TIANTIAN. "A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects". *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356486 2.