Augmented Vertex Block Descent

CHRIS GILES, Roblox, USA
ELIE DIAZ, University of Utah, USA
CEM YUKSEL, University of Utah, USA

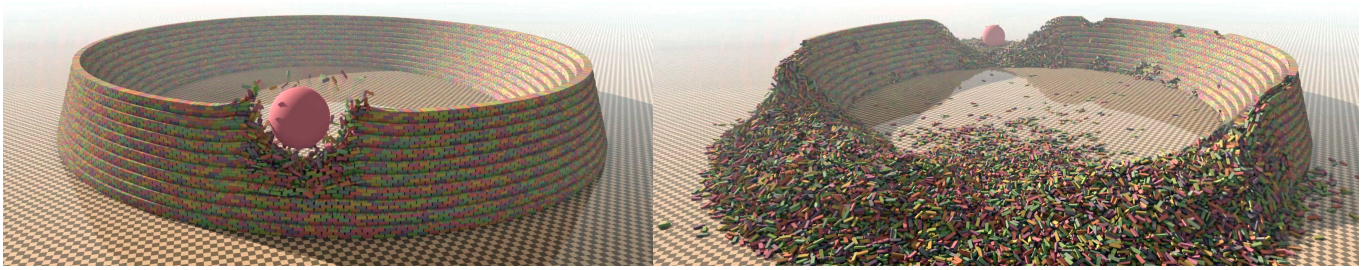


Fig. 1. *Our augmented vertex block descent method allows simulating hard constraints, which are critical for handling contacts and stacking, shown here simulating a pile of 110,000 blocks smashed by a sphere using 4 iterations, taking 3.5 ms (9.8 ms including collision detection) per frame on an NVIDIA RTX 4090 GPU.*

Vertex Block Descent is a fast physics-based simulation method that is unconditionally stable, highly parallelizable, and capable of converging to the implicit Euler solution. We extend it using an augmented Lagrangian formulation to address some of its fundamental limitations. First, we introduce a mechanism to handle hard constraints with infinite stiffness without introducing numerical instabilities. Second, we substantially improve the convergence in the presence of high stiffness ratios. These changes we introduce allow simulating complex contact scenarios involving rigid bodies with stacking and friction, articulated bodies connected with hard constraints, including joints with limited degrees of freedom, and stiff systems interacting with soft bodies. We present evaluations using a parallel GPU implementation that can deliver real-time performance and stable simulations with low iteration counts for millions of objects interacting via collisions, various joint/attachment constraints, and springs of various stiffness. Our results show superior performance, convergence, and stability compared to the state-of-the-art alternatives.

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: Physics-based animation, simulation

ACM Reference Format:
Chris Giles, Elie Diaz, and Cem Yuksel. 2025. Augmented Vertex Block Descent. *ACM Trans. Graph.* 44, 4, Article 90 (August 2025), 12 pages. https://doi.org/10.1145/3731195

1 Introduction

Physics-based simulation is the cornerstone of many graphics applications and various simulation methods have been developed over the years for different systems/problems. Recently, the *vertex block descent* (VBD) [Chen et al. 2024a] method presented superior convergence behavior over prior techniques for soft body dynamics, delivering higher performance, improved parallelism, unconditional stability, and the ability to converge to the implicit Euler solution with the desired accuracy. In addition, VBD was shown to handle other simulation problems, such as particle systems and rigid body dynamics. Because VBD is fast and remains stable with few iterations, it is particularly useful for real-time graphics applications with strict computation budgets. Since VBD is a *primal* method [Macklin et al. 2020], it can also easily handle high mass ratios, which is another important advantage.

However, VBD is not without shortcomings. First of all, it struggles to converge for problems involving high stiffness ratios. Furthermore, there is no mechanism for enforcing explicit hard constraints (unless the corresponding degree of freedom can be entirely eliminated). Unfortunately, such cases are not uncommon. Collisions, for example, often require highly stiff penalty forces or hard constraints. It is also common to connect objects with hard constraints to form articulated rigid bodies or other mechanical systems. Thus, these issues prevent widespread use of VBD.

In this paper, we directly address these limitations of VBD and extend it with an *augmented Lagrangian* formulation that allows efficiently handling hard constraints via progressively increasing stiffness, without introducing numerical instability into the optimization through extreme stiffness values. This adds a *dual* step after each *primal* iteration of VBD, making our approach a hybrid, *primal-dual* method. To substantially improve convergence rate in the presence of high stiffness ratios, we apply the same progressive stiffness increment used in augmented Lagrangian to arbitrary, finite stiffness forces. Due to the use of augmented Lagrangian techniques, we call our method *Augmented-VBD* or *AVBD* for short.

Our evaluation involves challenging experiments with hard constraints, such as articulated rigid bodies and attachment constraints for soft bodies, extreme stiffness ratios, and complex collisions and contacts, such as rigid body stacking, piling, and friction. Our results show that AVBD can successfully handle these complex interactions, requiring a small number of iterations to maintain the constraints

Authors' Contact Information: Chris Giles, cgiles@roblox.com, Roblox, San Mateo, CA, USA; Elie Diaz, elie.diaz@utah.edu, University of Utah, Salt Lake City, UT, USA; Cem Yuksel, cem@cemyuksel.com, University of Utah, Salt Lake City, UT, USA.

ACM 1557-7368/2025/8-ART90
https://doi.org/10.1145/3731195

and produce stable simulations (Figure 1), providing a substantial improvement over VBD in these scenarios in terms of both visual quality and convergence speed/performance.

2 Background

There is a large body of work on physics-based simulations in computer graphics. In this section, we briefly summarize them and then describe the vertex block descent method in detail.

2.1 Prior Work

Baraff [1997] introduced impulse based simulation methods to the computer graphics community, which was later refined in Catto [2006]. This variant, referred to as a *sequential impulse* solver, solves for the *dual variables* (impulses) sequentially using projected Gauss-Seidel (PGS), followed by an application of the impulses to get an updated velocity and position using semi-implicit Euler integration. This method forms the cornerstone of most real time rigid body simulators used in video games, VR applications, CGI, and many others. This is primarily due to its simplicity and robustness, particularly at low iteration counts. It is very effective for the hard equality and inequality constraints present in contact, friction, and articulated joints. There are some limitations, however. For one, it struggles to deal with high mass ratios due to poor conditioning of the linear system [Macklin et al. 2020]. Andrews et al. [2017a] proposed a robust way of including a *geometric stiffness* term which improves convergence of the dual formulation in the presence of high mass ratios, but only when using a computationally intensive direct solver. Being a purely constraint centric approach, it is not always possible to include arbitrary non-constraint forces in dual formulations. Finally, due to linearizations inherent to the methods, they struggle with highly non-linear constraints.

More recently, *position-based dynamics* (PBD) [Müller et al. 2007] and related extensions (XPBD) [Macklin et al. 2016, 2019; Müller et al. 2020] were introduced. Unlike sequential impulse methods, which solve for constraint impulses then integrate velocity to get position, these position-based methods solve for constraint forces then directly update position with a fully implicit Euler integrator. They solve constraints in an iterative fashion using a non-linear Gauss-Seidel solver. This allows these methods to handle non-linear constraints and forces much more robustly. They are still solving the dual problem, which is to say, they still suffer from high mass ratio issues and implementation challenges when including non-constraint forces.

Macklin et al. [2019] showed that trading Gauss-Seidel iterations for sub-steps results in better convergence for a similar computational cost, especially in the presence of high mass ratios, with the added benefit of better energy conservation. This approach can be applied to impulse or position based methods, having been coined recently as *temporal Gauss-Seidel*. However, the number of sub-steps required depends on the specific mass ratios present, and it is not easy to predict how many are required in the worst case. Giles and Andrews [2024] showed a method for adaptively determining the largest safe time-step for maintaining stability in dual methods, but this approach does not reduce the upper bound on the number of sub-steps required.

Position-based methods have seen recent adoption for soft body and cloth simulation, but less so for constrained rigid body simulation. This is because position-based methods by design will correct constraint error (potential energy) by converting it into momentum (kinetic energy). For soft bodies and other non-constraint forces, this is desirable. However, for hard constraints, which should never be violated, this can result in undesirably energetic simulations. This violation can occur when under-solving due to a fixed iteration count, or if constraints are violated at the start of the simulation for any other reason. This issue is less apparent or non-existent in sequential impulse techniques through the use of post stabilization [Cline and Pai 2003]. Macklin et al. [2019] proposed a post stabilization method at the velocity level to help alleviate this issue, though it can take many iterations to remove all added energy.

Recently, *primal* type methods have received much attention in the computer graphics space. These methods, rather than solve for constraint impulse or forces, solve directly for position or velocity updates. Macklin et al. [2020] describes how primal and dual methods are related, and shows that primal methods are not sensitive to high mass ratios, and instead are sensitive to high stiffness ratios. They showed promising results, but are somewhat limited by the convergence rate of Jacobi iterations, which is particularly sensitive to high stiffness ratios. Lan et al. [2022] use the IPC framework to efficiently simulate rigid bodies, while preventing penetration by treating them as stiff affine bodies. Guo et al. [2024] use a Barrier-augmented Lagrangian formulation to improve the conditioning and convergence for elastodynamic systems, using an adaptive primal-dual optimization scheme. Chen et al. [2024b] introduce an interior point primal-dual method solved using Newton's method, specifically targeted at contact and friction. Similar to our work, they show that hybrid primal-dual methods inherit the benefits of both primal and dual formulations. They do not, however, investigate constraint types or forces other than contacts and friction.

These methods use *global solvers*, which are very accurate, but struggle to provide real-time performance for even moderately sized scenes, since they do not scale linearly with the number of bodies. Our method can achieve comparable accuracy to global methods with enough iterations.

2.2 Vertex Block Descent

Another recent primal type method is *vertex block descent* (VBD) [Chen et al. 2024a] for computing implicit Euler steps [Baraff and Witkin 1998; Hirota et al. 2001; Martin et al. 2011; Volino and Magnenat-Thalmann 2001], which was traditionally approximated using a single Newton step by solving a global linear system [Baraff and Witkin 1998]. This global solution can result in stability issues with stiff problems and large time steps. VBD avoids this by using local Gauss-Seidel iterations to minimize the variational energy [Kane et al. 1999, 2000; Kharevych et al. 2006; Lew et al. 2004; Simo et al. 1992]

$$\mathbf{x}^{t+\Delta t} = \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2\,\Delta t^2}\|\mathbf{x}-\mathbf{y}\|_M^2 + E(\mathbf{x})\,, \tag{1}$$

where Δt is the timestep size, $\mathbf{x}^{t+\Delta t}$ are the positions (i.e. degrees of freedom) at the end of the timestep, $\mathbf{y}$ is the inertial positions, computed using the positions $\mathbf{x}^t$ and velocities $\mathbf{v}^t$ at the begining of

the timestep, along with external acceleration, such that

$$\mathbf{y} = \mathbf{x}^t + \Delta t \mathbf{v}^t + \Delta t^2 \mathbf{a}_{\text{ext}} \tag{2}$$

combined with the mass-weighted norm $\|\cdot\|_M$, and $E(\mathbf{x})$ is the *total potential energy* evaluated at $\mathbf{x}$.

Instead of solving Equation 1 directly, VBD moves a single mass (i.e. vertex) i at a time, assuming all others are fixed, to iteratively minimize the local variational energy

$$\mathbf{x}_i \leftarrow \underset{\mathbf{x}_i}{\text{argmin}} \frac{1}{2\,\Delta t^2}\|\mathbf{x}_i - \mathbf{y}_i\|^2_{\mathbf{M}_i} + \sum_{j\in\mathcal{F}_i} E_j(\mathbf{x}) \,, \tag{3}$$

where $\mathbf{M}_i$ is the mass matrix and E_j is the energy of a force element j in $\mathcal{F}_i$. Equation 3 is solved using a single Newton iteration that corresponds to the linear system

$$\mathbf{H}_i \,\Delta\mathbf{x}_i = \mathbf{f}_i \,, \tag{4}$$

where Δx_i is the position change between iterations, $\mathbf{f}_i$ is the force

$$\mathbf{f}_i = -\frac{1}{\Delta t^2}\mathbf{M}_i(\mathbf{x}_i - \mathbf{y}_i) + \sum_{J\in\mathcal{F}_i} \mathbf{f}_{ij} \,, \tag{5}$$

using $\mathbf{f}_{ij} = -\partial E_j(\mathbf{x})/\partial \mathbf{x}_i$, and $\mathbf{H}_i$ is the Hessian

$$\mathbf{H}_i = \frac{\mathbf{M}_i}{\Delta t^2} + \sum_{J\in\mathcal{F}_i} \mathbf{H}_{ij} \,, \tag{6}$$

such that $\mathbf{H}_{ij} = \partial^2 E_j(x)/\partial \mathbf{x}_i^2$ is the Hessian of each force element j acting on mass i. When using particles/vertices of 3 degrees of freedom (DOF) each and mass m_i, $\mathbf{M}_i = m_i\mathbf{I}$ and $\mathbf{H}_i$ are 3×3 matrices, where $\mathbf{I}$ is the identity matrix. For rigid bodies, $\mathbf{M}_i$ is a 6×6 matrix containing the mass in the upper diagonal block, and rotated moment of inertia in the lower diagonal block.

The linear systems in Equation 4 are small enough that we can solve for $\Delta\mathbf{x}_i$ by directly inverting $\mathbf{H}_i$, such that $\Delta\mathbf{x}_i = \mathbf{H}_i^{-1}\mathbf{f}_i$.

$\mathbf{H}_i$ is not guaranteed to be symmetric positive definite (SPD). If we can ensure that the system is SPD, we can use more efficient and robust methods for solving the system such as using an LDL^T decomposition. We will show how to construct an approximation of $\mathbf{H}_i$ in Section 3.5 which guarantees an SPD system.

The equations for rotational degrees of freedom involve minor modifications to the ones above, as we detail in Section 4.

When combined with line search [Hirota et al. 2001], VBD can guarantee descent in variational energy with each step, though line search is often unnecessary in practice.

Algorithmically, VBD resembles XPBD, which is also iterative, but in the dual space that operates on one constraint at a time. Thus, parallelization with XPBD is achieved by graph coloring the constraints [Fratarcangeli and Pellacini 2015; Fratarcangeli et al. 2016; Ton-That et al. 2023], while VBD colors the vertices, resulting in significantly fewer colors than XPBD, which leads to improved parallelization. More importantly, while VBD directly solves the implicit Euler step, XBPD's formulation suffers from approximation errors that negatively impact the visual behavior. Also, being a dual method, XPBD can result in significant error with high mass ratios, which is entirely avoided by VBD's primal formulation.

On the other hand, VBD struggles with high stiffness ratios and cannot model hard constraints, which are easy to handle with XPBD's dual formulation. Our method directly addresses these limitations of VBD, which are particularly important for robustly and efficiently handling attachment, collision, and contact constraints, and critical for simulating articulated rigid bodies.

3 VBD with Augmented Lagrangian

VBD can handle constraints using a quadratic energy potential

$$E_j(\mathbf{x}) = \frac{1}{2}k_j\Big(C_j(\mathbf{x})\Big)^2 \tag{7}$$

where k_j and C_j are the stiffness and the constraint error for constraint j. This works well for soft constraints, but stiffer constraints can lead to convergence issues. Hard constraints, on the other hand, cannot be handled this way, as their stiffness would be infinite. Using finite but large stiffness values, on the other hand, negatively impacts the convergence of VBD.

We extend the VBD formulation to support hard constraints using the augmented Lagrangian method (Section 3.1). Then, we explain how it can be used for handling contacts and inequality constraints (Section 3.2). Motivated by this approach, we present a method that significantly improves the convergence of VBD in the presence of forces with high stiffness ratios (Section 3.4). Since stiff systems are more likely to contain indefinite Hessians, we describe a simple method for handling them (Section 3.5). Then, we present a simple method to resolve the explosive error correction introduced by hard constraints when the constraint error exists at the beginning of the step (Section 3.6). Finally, we present a simple warm-starting approach that can significantly improve convergence, (Section 3.7).

3.1 Hard Constraints

To support hard constraints with $k_j = \infty$, we use an augmented Lagrangian formulation for the constraint energy, such that for each iteration n we use

$$E_j^{(n)}(\mathbf{x}) = \frac{1}{2}k_j^{(n)}\Big(C_j(\mathbf{x})\Big)^2 + \lambda_j^{(n)}C_j(\mathbf{x}) \,, \tag{8}$$

where $k_j^{(n)}$ is the finite stiffness and $\lambda_j^{(n)}$ the Lagrange multiplier (also called *dual variable*) used for AVBD iteration n. The resulting constraint force is then

$$\mathbf{f}_{ij}^{(n)} = -\left(k_j^{(n)}C_j(\mathbf{x}) + \lambda_j^{(n)}\right)\frac{\partial C_j(\mathbf{x})}{\partial \mathbf{x}_i} \,. \tag{9}$$

For the first iteration $n = 0$, we can initialize the variables using

$$k_j^{(0)} = k_{\text{start}} \qquad \text{and} \qquad \lambda_j^{(0)} = 0 \,, \tag{10}$$

where $k_{\text{start}} > 0$ is an initial stiffness parameter. Before we begin the next iteration, the dual variable is updated using

$$\lambda_j^{(n+1)} = k_j^{(n)}C_j(\mathbf{x}) + \lambda_j^{(n)} \,. \tag{11}$$

Notice that with this updated dual variable, the Lagrangian term in Equation 9 is sufficient to apply the same amount of force as before without violating the constraint at all.

Here, the stiffness variable merely determines how fast the dual variable grows over multiple iterations to provide the necessary force. Thus, it is helpful to adjust the stiffness variable between iterations to improve convergence. We use

$$k_j^{(n+1)} = k_j^{(n)} + \beta\,|C_j(\mathbf{x})| \,, \tag{12}$$

where β is a scaling parameter that controls how quickly the stiffness is increased. We use $\beta = 10$ for all examples in this paper, though the results are not sensitive to the value of β used and we observed similar results for any $\beta \in [1, 1000]$ we tested. Even using $\beta = 0$, which keeps the stiffness at k_{start}, works, but typically requires more iterations to reduce the constraint error.

Using this stiffness update with $\beta > 0$, the value of k_{start} is not crucial either, but picking a good initial stiffness improves convergence, as it controls how fast the dual variable is updated after the first iteration. In Section 3.7, we discuss a method for warm-starting the stiffness and dual variable to improve convergence.

With this formulation, the stiffness term often remains relatively small, while the constraint can be perfectly satisfied. This approach not only avoids numerical instabilities of high stiffness, but also offers strong theoretical convergence [Birgin and Martínez 2014].

3.2 Inequality Constraints

There are many types of hard constraints which must be bounded by some lower or upper force limits. Some examples include contacts, friction, and joint limits. Previous works using a primal formulation circumvent this by approximating inequalities using a function with a smooth boundary [Chen et al. 2024a; Lan et al. 2022; Macklin et al. 2020]. However, these smooth functions do not exactly model the bounds, which are particularly important for accurate modeling of Coulomb friction needed to achieve stable stacking [Chen et al. 2024b].

Dual methods based on projected Gauss-Seidel model bounds by clamping the Lagrange multipliers at each iteration [Catto 2006]. We take a similar approach and model bounds by clamping the Lagrange multiplier, as it controls the magnitude of the force. Note that we can rewrite Equation 9 as

$$\mathbf{f}_{ij}^{(n)} = -\lambda_j^+ \frac{\partial C_j(\mathbf{x})}{\partial \mathbf{x}_i} \qquad \text{where} \qquad \lambda_j^+ = k_j^{(n)} C_j(\mathbf{x}) + \lambda_j^{(n)} \,. \tag{13}$$

Notice that this definition for λ_j^+ above is the same as $\lambda_j^{(n+1)}$ using Equation 11, except that $C_j(\mathbf{x})$ above is computed before updating $\mathbf{x}_i$. Since $\|\partial C_j(\mathbf{x})/\partial \mathbf{x}_i\| = 1$ for typical constraints, bounding the Lagrange multiplier in turn bounds the magnitude of the force. Specifically, we define $\lambda_j^{\min}$ and $\lambda_j^{\max}$ as the minimum and maximum force bounds, respectively. We use them to clamp λ_j^+, when needed.

This results in a discontinuous force with zero Hessian when the force exceeds the bounds, preventing the optimizer from making progress.

One simple solution is using the exact clamped Lagrange multipliers, as described above, but approximating the Hessian without clamping. This is a more conservative choice as it can cause smaller updates at boundaries, but lacks any discontinuities and in practice converges to the exact bounds with enough iterations.

A better Hessian approximation for the clamped force can be achieved by *stiffness rescaling*, inspired by the friction model of [Macklin et al. 2020], such that

$$\tilde{k}_j^{(n)} = \begin{cases} \left| \dfrac{\lambda_j^{\min} - \lambda_j^{(n)}}{C_j(\mathbf{x})} \right|, & \text{if } \lambda_j^+ < \lambda_j^{\min} \text{ and } C_j(\mathbf{x}) \neq 0, \\ \left| \dfrac{\lambda_j^{\max} - \lambda_j^{(n)}}{C_j(\mathbf{x})} \right|, & \text{if } \lambda_j^+ > \lambda_j^{\max} \text{ and } C_j(\mathbf{x}) \neq 0, \\ k_j^{(n)}, & \text{otherwise} \,, \end{cases} \tag{14}$$

where λ_j^+ is the force before clamping is applied, This rescaled stiffness is used in place of the original one only for calculating the Hessian, when building the left-hand side of Equation 4.

We can further optimize with a simple modification to the stiffness increment of Equation 12. If the force for the current iteration is outside of the desired force bounds, then there is no need to increase the stiffness value as this could potentially push the solution further away from the force bounds. Therefore, we only apply the stiffness update of Equation 12 when $\lambda_j^{\min} < \lambda_j^+ < \lambda_j^{\max}$.

3.3 Frictional Contacts

We model contacts along with friction using a 3D constraint

$$\mathbf{C}_{contact}(\mathbf{x}) = \begin{bmatrix} \hat{\mathbf{t}} & \hat{\mathbf{b}} & \hat{\mathbf{n}} \end{bmatrix}^T (\mathbf{r}_a - \mathbf{r}_b) \tag{15}$$

where $\hat{\mathbf{n}}$, $\hat{\mathbf{t}}$, and $\hat{\mathbf{b}}$ form a 3×3 orthonormal basis about the contact normal $\hat{\mathbf{n}}$, and $\mathbf{r}_a$ and $\mathbf{r}_b$ are the world-space contact points for the colliding bodies/particles a and b. The first two components of this constraint model the friction force and the last one is the normal force along the collision direction.

To ensure that the normal force cannot pull the colliders together, we use bounds $\lambda_{\mathbf{n}}^{\min} = 0$ and $\lambda_{\mathbf{n}}^{\max} = \infty$. To model a friction cone, we bound the other two Lagrange multipliers $\lambda_{\mathbf{t}}^+$ and $\lambda_{\mathbf{b}}^+$ that determine the friction force along $\hat{\mathbf{t}}$ and $\hat{\mathbf{b}}$, respectively. We achieve this by bounding the magnitude of the vector $\boldsymbol{\lambda}_{\mathbf{tb}}^+ = [\, \lambda_{\mathbf{t}}^+ \;\; \lambda_{\mathbf{b}}^+ \,]^T$, such that $\|\boldsymbol{\lambda}_{\mathbf{tb}}^+\| \leq \mu \lambda_{\mathbf{n}}^+$, where μ is the coefficient of friction. So, we use $\lambda_{\mathbf{tb}}^{\min} = -\mu \lambda_{\mathbf{n}}^+$ and $\lambda_{\mathbf{tb}}^{\max} = \mu \lambda_{\mathbf{n}}^+$. To improve static friction accuracy further and to prevent numerical drift, we modify our collision detection routine such that, if on the previous time step, $\|\boldsymbol{\lambda}_{\mathbf{tb}}^+\| \leq \mu \lambda_{\mathbf{n}}^+$, we do not allow the contact points to move along the tangent directions. This is not required, but can give better static friction behavior when using a lower maximum iteration count.

We can also switch between using static and dynamic friction coefficients, μ_s and μ_d, respectively, using the same test. If $\|\boldsymbol{\lambda}_{\mathbf{tb}}^+\| \leq \mu \lambda_{\mathbf{n}}^+$ in the previous frame, we switch to static friction with $\mu = \mu_s$. If we must clamp $\boldsymbol{\lambda}_{\mathbf{tb}}^+$ because otherwise it would be $\|\boldsymbol{\lambda}_{\mathbf{tb}}^+\| > \mu_s \lambda_{\mathbf{n}}^+$, we immediately switch to dynamic friction using $\mu = \mu_d$.

3.4 High Stiffness Ratios

The convergence of VBD struggles with high stiffness ratios, because each step updates the position of a single mass (i.e. vertex) only, assuming all other masses are fixed. Consider a mass that is connected to two force elements, one of which is significantly stiffer than the other. The stiff force corresponds to a steeper gradient, so any incremental movement of the mass corresponds to a larger change in energy for the stiffer force. This is exactly why VBD itera-

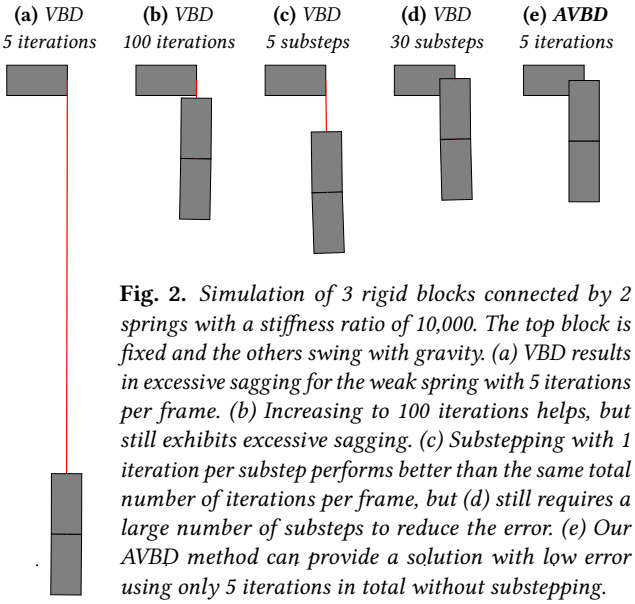


Fig. 2. *Simulation of 3 rigid blocks connected by 2 springs with a stiffness ratio of 10,000. The top block is fixed and the others swing with gravity. (a) VBD results in excessive sagging for the weak spring with 5 iterations per frame. (b) Increasing to 100 iterations helps, but still exhibits excessive sagging. (c) Substepping with 1 iteration per substep performs better than the same total number of iterations per frame, but (d) still requires a large number of substeps to reduce the error. (e) Our AVBD method can provide a solution with low error using only 5 iterations in total without substepping.*

tions bias the position updates towards the local solutions of the stiff forces. This is indeed the correct strategy for locally minimizing the energy for a mass, but it prevents forces with lower stiffness from propagating their information globally across multiple masses. Thus, the locally optimal solutions lead to a globally slow convergence.

This is demonstrated with a simple example in Figure 2a, using two springs with a high stiffness ratio connecting 3 blocks. Though both springs are sufficiently stiff to withstand the weight of the blocks, using 5 VBD iterations leads to excessive stretching for the weaker spring. Using 100 iterations (Figure 2b) mitigates the issue, but the result still remains visually far from the converged solution.

One simple solution is reducing the timestep size. This can be accomplished by replacing iterations with substeps, using the same total computation cost. Yet, 5 substeps (Figure 2c) is insufficient in this case and we start getting results visually close to convergence with using 30 or more substeps (Figure 2d) in this example.

To alleviate this issue, for forces with finite stiffness, we use a similar formulation as Equation 12 that ramps up the stiffness, but up to the actual stiffness of the force k_j^*, such that

$$k_j^{(n+1)} = \min\left(k_j^*,\ k_j^{(n)} + \beta\,|C_j(\mathbf{x})| \right). \tag{16}$$

Since the stiffness is always finite, we do not add the Lagrangian term to the constraint energy and we use Equation 7 with the stiffness variable, instead of the actual stiffness.

Similar to the augmented Lagrangian solution above, the stiffness is increased with each iteration, based on how much the constraint is violated, but up to the actual stiffness. This limits the stiffness ratio acting on a mass in the earlier iterations, so the local solution for the first iteration is no longer heavily biased toward the stiff forces. Thus, the weaker forces get a chance to globally propagate their information. The stiffness bound is gradually lifted, as needed, and the local solutions correctly bias the position updates toward stiffer forces. With this formulation, in the same example, 5 iterations (without substeps) provides a good solution (Figure 2e).

3.5 Approximate Hessians

VBD cannot guarantee that the Hessian $\mathbf{H}_{ij}$ of the force element j acting on mass i is always positive definite and, thereby, invertible. In the rare cases when it is not, VBD skips updating mass i for that iteration, expecting that it will be positive definite in the next iteration.

However, the Hessian can easily become indefinite for hard constraints modeled with augmented Lagrangian, using

$$\mathbf{H}_{ij} = k_j^{(n)} \left(\frac{\partial C_j(\mathbf{x})}{\partial \mathbf{x}_i} \right)^T \frac{\partial C_j(\mathbf{x})}{\partial \mathbf{x}_i} + \mathbf{G}_{ij} \tag{17}$$

where $\mathbf{G}_{ij}(\mathbf{x}) = \lambda_j^+ \, \partial^2 C_j(\mathbf{x}) / \partial \mathbf{x}_i^2$ is the second derivative of the constraint scaled by λ_j^+. Here, the first term is always symmetric positive definite, but the second term can be indefinite and non-symmetric.

We avoid this by using a diagonal approximation of the second term based on the norm of its columns (as in Andrews et al. [2017b]). Thus, we replace $\mathbf{G}_{ij}$ in Equation 17 with the diagonal matrix $\tilde{\mathbf{G}}_{ij} = \mathrm{diag}(\mathbf{g}_{ij})$, where each element $g_{ij,c}$ of vector $\mathbf{g}_{ij}$ is the norm of vector $\mathbf{G}_{ij,c}$ forming column c of $\mathbf{G}_{ij}$, such that $g_{ij,c} = \|\mathbf{G}_{ij,c}\|$. This guarantees that the resulting approximate Hessian is symmetric positive definite, and that stable progress can be achieved with each AVBD iteration. Additionally, this type of system can be solved more accurately and efficiently using an LDL^T decomposition, which we use in our implementation instead of a direct inverse. This Hessian approximation corresponds to a quasi-Newton step.

Compared to skipping the update for a vertex when the determinant is small, as in the original VBD method, this approach results in improved convergence and a reduction in artifacts for stiff forces at low iteration bounds.

3.6 Preventing Explosive Error Correction

One important advantage of VBD is that it remains stable even when the number of iterations per frame is limited to a small number. This is particularly helpful for real-time graphics applications with strict computation budgets. If the number of iterations is insufficient for convergence, the resulting simulation can contain a large amount of constraint error. This can manifest itself as excessive momentum, but does not break numerical stability.

AVBD maintains this property. However, large constraint error for hard constraints exacerbates this issue and can inject a significant amount of momentum. This is because, if a hard constraint is violated in the previous frame (due to limited iterations), in the very next frame the hard constraint can apply an arbitrarily large force to enforce the constraint, resulting in a sudden motion and a large spike in momentum. This behavior is expected, since the hard constraints should not have been violated in the first place, but this cannot be guaranteed when the number of iterations is bounded.

We address this by limiting the amount of position correction that takes place in the next frame for hard constraints. This is accomplished using

$$C_j(\mathbf{x}) = C_j^*(\mathbf{x}) - \alpha\, C_j^*(\mathbf{x}^t). \tag{18}$$

where C_j^* is the original constraint function and $\alpha \in [0, 1]$ is the regularization parameter, determining what portion of the existing constraint error from the previous time step $C_j^*(\mathbf{x}^t)$ is ignored. We use $\alpha = 0.95$ for all examples in this paper. Note that if $\alpha = 1$, all pre-existing constraint error is removed, so constraint error must be explicitly corrected using something like post stabilization.

Notice that the regularized constraint function has the same derivatives as the original function, i.e. $\partial C_j(\mathbf{x})/\partial \mathbf{x} = \partial C_j^*(\mathbf{x})/\partial \mathbf{x}$. Since it partially ignores the remaining error from the previous time step, it leads to a gradual error correction over multiple steps, thereby mitigating the momentum injection of error correction.

3.7 Warm Starting

We can use Equation 10 to reinitialize the stiffness and dual variables every frame. To improve convergence, however, it is often helpful to warm-start these, using the values computed at the end of the previous time step. This way, the very first iteration uses the stiffness and dual values needed to maintain the constraint.

Yet, notice that the update rules in Equation 12 and Equation 16 always increase the stiffness. Therefore, using the stiffness value of the previous frame prevents the simulation from using a smaller stiffness later on, and a high stiffness needed for a previous frame might be unnecessary to maintain the constraint in the later frames.

That is why, we simply scale down the values computed in the previous frame before we use them to initialize the stiffness and dual variables for the current frame, such that

$$k_j^{(0)} = \max\left(\gamma\, k_j^t,\ k_{\text{start}}\right) \qquad \text{and} \qquad \lambda_j^{(0)} = \alpha\gamma\, \lambda_j^t \tag{19}$$

where k_j^t and λ_j^t are the dual parameters computed after the last iteration of the previous frame and $\gamma \in [0, 1)$ is the scaling parameter. We use $\gamma = 0.99$ for all examples in this paper. Note that λ is scaled by α, because we do not want energy introduced to correct pre-existing constraint error to be included in warm starting, as this would add energy to the system over time, causing instability.

For the simple example in Figure 2e, we can achieve the same result with only a single iteration if we use warm starting.

4 Implementation Details

We have implemented AVBD entirely on the GPU using DirectX 11 compute shaders. The pseudocode of our implementation is shown in Algorithm 1, with portions that are new to AVBD over VBD marked in red. For broad-phase collision detection, we build a bounding volume hierarchy using the LBVH approach of [Lauterbach et al. 2009]. From there, we can build a list of pairs by traversing and intersecting the bounding box of each object with the BVH. Once a list of pairs is available, we perform discrete narrow-phase collision detection, making sure to persist the constraint force and stiffness variables for warm-starting purposes. One sweep of greedy vertex coloring is then performed, followed by an initialization and warm-starting step for all variables. Then, the main iteration begins. For n iterations, we solve for the primal variables (Equation 4) for each color in parallel, followed by updating the dual variables and stiffnesses in parallel using Equation 11 and Equation 12. Finally, we compute the velocities for all degrees of freedom. Since we already have a BVH from the broad-phase step, we use it to perform compute shader based ray-tracing of primary, shadow, and ambient occlusion rays for the final rendering.

Algorithm 1 *AVBD simulation for one time-step*

collision detection using $\mathbf{x}$
update colorization
$\mathbf{y} \leftarrow \mathbf{x} + h\mathbf{v} + \Delta t^2 \mathbf{a}_{ext}$
$\mathbf{x} \leftarrow$ initial guess with adaptive initialization
$\lambda \leftarrow \alpha\gamma\lambda$
$\mathbf{k} \leftarrow \max(k_{\text{start}}, \gamma\mathbf{k})$
for $n_{\max}$ iterations **do**
  for each color c **do**
    for each vertex i in color c (in parallel) **do**
      $\mathbf{f}_i \leftarrow -\frac{\mathbf{M}_i}{\Delta t^2}(\mathbf{x}_i - \mathbf{y}_i)$
      $\mathbf{H}_i \leftarrow \frac{\mathbf{M}_i}{\Delta t^2}$
      for each force j affecting vertex i **do**
        if j is hard constraint **then**
          $\mathbf{f}_i \leftarrow \mathbf{f}_i - \text{clamp}\left(k_j C_j(\mathbf{x}) + \lambda_j, \lambda_j^{min}, \lambda_j^{max}\right)\frac{\partial C_j}{\partial \mathbf{x}_i}$
        else
          $\mathbf{f}_i \leftarrow \mathbf{f}_i - k_j C_j(\mathbf{x})\frac{\partial C_j}{\partial \mathbf{x}_i}$
        end if
        $\mathbf{H}_i \leftarrow \mathbf{H}_i + k_j(\frac{\partial C_j}{\partial \mathbf{x}_i})^T \frac{\partial C_j}{\partial \mathbf{x}_i} + \mathbf{G}'_{ij}$
      end for
      $\mathbf{x}_i^{new} \leftarrow \mathbf{x}_i + \mathbf{H}_i^{-1}\mathbf{f}_i$
    end for
    for each vertex i in color c (in parallel) **do**
      $\mathbf{x}_i \leftarrow \mathbf{x}_i^{new}$
    end for
  end for
  for each force j (in parallel) **do**
    if j is hard constraint **then**
      $\lambda_j \leftarrow \text{clamp}\left(k_j C_j(\mathbf{x}) + \lambda_j, \lambda_j^{\min}, \lambda_j^{\max}\right)$
      if $\lambda_j^{\min} < \lambda_j < \lambda_j^{\max}$ **then**
        $k_j \leftarrow k_j + \beta|C_j(\mathbf{x})|$
      end if
    else
      $k_j \leftarrow \min\left(k_j^*, k_j + \beta|C_j(\mathbf{x})|\right)$
    end if
  end for
end for
$\mathbf{v} \leftarrow (\mathbf{x} - \mathbf{x}^t)/\Delta t$

Bounding the Dual Variables. In general, it is also a good idea to bound $k_j^{(n)}$ and $\lambda_j^{(n)}$ to some large maximum value for numerical stability. This is typically unnecessary, but it can be important if the simulation contains conflicting hard constraints that cannot be satisfied together. Thus, we prevent the stiffness values from increasing arbitrarily by bounding the dual variables to large (but finite) values. Note that $\lambda_j^{(n)}$ can be negative, so we bound its magnitude.

Approximating Constraints. For hard constraints which are expensive to evaluate, such as collisions, we optimize the computation of $C_j(\mathbf{x})$ using the first few terms of a Taylor series expansion about

the positions at the beginning of the time step $\mathbf{x}^t$, resulting

$$C_j(\mathbf{x}) \approx (1-\alpha)\, C_j^*(\mathbf{x}^t) + \frac{\partial C_j^*(\mathbf{x}^t)}{\partial \mathbf{x}}(\mathbf{x}-\mathbf{x}^t) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^t)^T \frac{\partial^2 C_j^*(\mathbf{x}^t)}{\partial \mathbf{x}^2}(\mathbf{x}-\mathbf{x}^t).$$

This avoids recomputing the constraint error and its derivatives at each iteration. We can instead compute these terms once at the beginning of time step and cache them. For some constraints which are linear or nearly linear, we can even drop the second order term of this equation to no ill effect. In our implementation, we drop this second order term for contacts, as it tends to be close to zero. When using the Taylor series approximation, we must also use it for approximating the derivatives of C_j when computing $\mathbf{f}_{ij}$ and $\mathbf{H}_{ij}$.

Parallelization. AVBD maintains the excellent parallelization of the original VBD method. Our approach adds only one additional pass between iterations: the dual variable and stiffness update, which is run for all constraints in parallel. For the primal step, like VBD, we perform coloring such that we are able to process all masses with the same color in parallel. In our implementation, we perform an incremental greedy coloring once per time step, considering all constraints, forces, and collisions. During this process, for each mass we assign a color that is different from the colors of the connected masses with smaller indices in a parallel Jacobi fashion. This coloring scheme typically converges after a few iterations, where most, if not all, masses have different colors than their connected neighbors. We double buffer the position updates like VBD, such that in the rare case where two masses have the same color, the solver effectively becomes equivalent to Jacobi for those masses on that time step, while others follow the Gauss-Seidel order. We have observed that in some difficult cases, such as those involving a series of connections with high stiffness ratios, randomizing the order with which the colors are processed can improve convergence. However, we do not randomize the order for any results presented in this paper.

Rigid Bodies. The equations we provided above are for linear degrees of freedom. For rigid bodies, however, we also have rotational degrees of freedom that require special handling. Let $\mathbf{q}_i = (q_{i,\mathrm{s}}, \mathbf{q}_{i,\mathrm{v}})$ be the quaternion representing the rotation of a rigid body, such that $q_{i,\mathrm{s}}$ and $\mathbf{q}_{i,\mathrm{v}} = (\mathbf{q}_i)_\mathrm{v}$ are its scalar and vector components, respectively. We represent $\mathbf{x}_i$ of a rigid body i as a combination of its linear position $\mathbf{p}_i$ and rotation $\mathbf{q}_i$. Since $\mathbf{q}_i$ must be a unit quaternion, $\mathbf{x}_i$ contains 6 degrees of freedom. For using this representation with the equations above, we define its subtraction as an operation that produces a 6D vector using

$$\mathbf{x}_i - \mathbf{x}_j := \begin{bmatrix} \mathbf{p}_i - \mathbf{p}_j \\ (2\mathbf{q}_i \mathbf{q}_j^{-1})_\mathrm{v} \end{bmatrix}. \tag{20}$$

Its update $\Delta\mathbf{x}_i = [\Delta\mathbf{p}_i\ \Delta\mathbf{w}_i]^T$ also contains a linear $\Delta\mathbf{p}_i$ and an angular $\Delta\mathbf{w}_i$ component and we define the update equation as

$$\mathbf{x}_i + \Delta\mathbf{x}_i := \begin{bmatrix} \mathbf{p}_i + \Delta\mathbf{p}_j \\ \mathrm{normalize}\left(\mathbf{q}_i + \frac{1}{2}(0, \Delta\mathbf{w}_i)\mathbf{q}_i\right) \end{bmatrix}. \tag{21}$$

Approximate Hessians. In our implementation, we always use the approximate Hessians described in Section 3.5 with quasi-Newton steps. Alternatively, it is possible to use $\tilde{\mathbf{G}}_{ij}$ only when the determinant $|\det(\mathbf{G}_{ij})| < \epsilon$ for a small ϵ parameter and otherwise continue with $\mathbf{G}_{ij}$. However, in our tests we observed minor vibrations when alternating between Newton and quasi-Newton steps without any improvement in convergence. Also, computing the determinant for rigid bodies (i.e. 6×6 matrix) incurs some performance penalty. That is why our implementation always uses quasi-Newton steps.

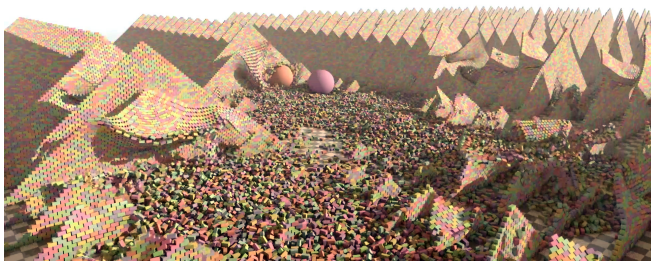


Fig. 3. *Piles of 510,000 blocks smashed by two spheres, simulated using our AVBD method with 4 iterations, taking 10.3 ms (17.6 ms including collision detection) per frame on an NVIDIA RTX 4090 GPU.*

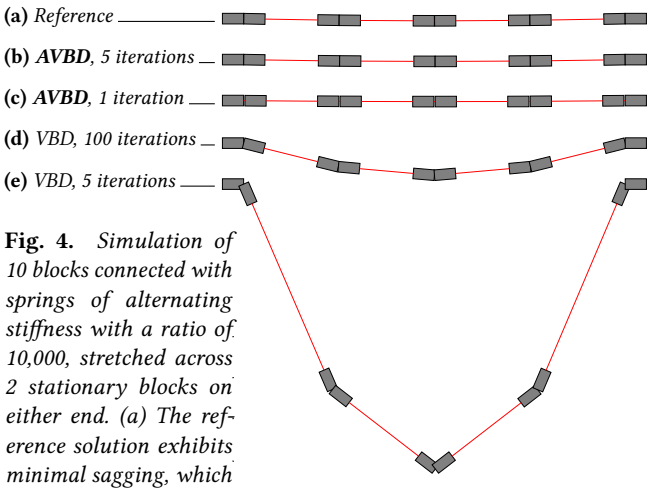
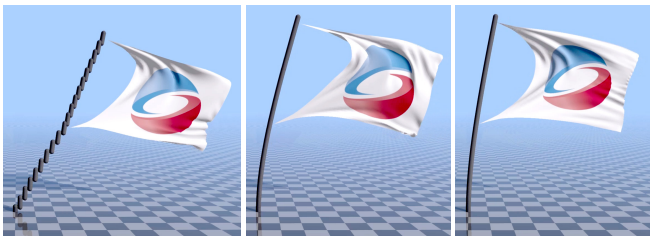


Fig. 4. *Simulation of 10 blocks connected with springs of alternating stiffness with a ratio of 10,000, stretched across 2 stationary blocks on either end. (a) The reference solution exhibits minimal sagging, which is matched by (b) our AVBD method using 5 iterations. (c) AVDB with only a single iteration leads to a small amount of error for the stiff springs. (d) VBD, in comparison, leads to overstretching even with 100 iterations and (e) using 5 iterations with VBD produces extreme stretching for the weaker springs.*

5 Results

We begin evaluating our AVBD method by comparing it to VBD [Chen et al. 2024a]. Then, we add comparisons to popular dual methods: XPBD using iterations or substeps [Macklin et al. 2016; Müller et al. 2020] and the sequential impulse method [Catto 2006]. We first compare them in terms of animation quality and convergence behavior. Finally, we provide performance results using large scenes (Figure 1 and Figure 3). We use a timestep of 1/60 seconds for all examples, with all AVBD parameters fixed. The iteration/substep counts are noted for each example. For friction constraints, we use the simple solution of calculating the Hessian without clamping, instead of stiffness rescaling.



(a) *VBD, 20 iterations* **(b)** ***AVBD****, 5 iter.* **(c)** ***AVBD****, 20 iter.*

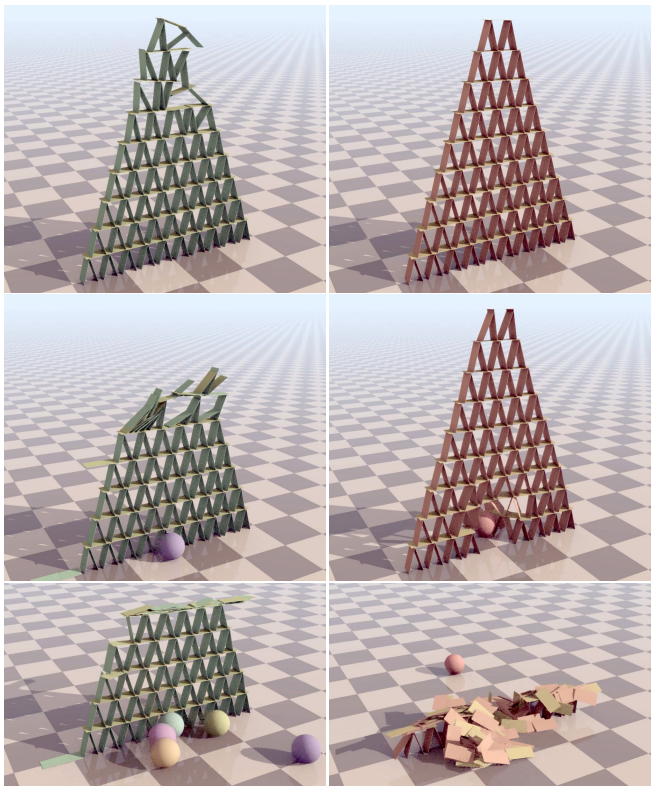
Fig. 5. *Simulation of a flag represented as a mass-spring system with 1,500 vertices and 6,000 springs. The flag is attached to a pole at two points with hard constraints. The pole is formed by 20 rigid body segments connected with ball joints and angular springs that resist bending. The high stiffness ratio between the cloth, the pole, and the attachment constraints causes excessive stretching of the pole in VBD. AVBD corrects these issues even with relatively few iterations, using augmented Lagrangian for the hard constraints forming the attachment of the flag and the ball joints of the pole. Using more iterations with AVBD improves the animation of the flag and the deformation of the pole. All hard constraints are always satisfied with low error even with low iteration limits.*

5.1 High Stiffness Ratios and High Stiffness

In Figure 4a we show a more complex example of high stiffness ratios than in Figure 2, comparing our method to VBD. Notice that, VBD with 5 iterations produces excessive stretching for weaker springs, which is improved but not resolved even after 100 iterations. In contrast, AVBD can produce a close solution to the reference after only a single iteration. This is mainly due to our warm-start of stiffness and dual variables (Section 3.7). The results after 5 iterations with AVBD are visually indistinguishable from the reference.

Another example of high stiffness ratios is shown in Figure 5, where a soft flag is attached with hard constraints to a stiffer but deformable pole. VBD struggles to balance the stiff force of the hard constraints with the weaker bending stiffness of the pole and the pull of the flag. Our AVBD method, on the other hand, can produce a reasonable animation, satisfying the hard attachment constraints and properly bending the pole, even using 4× fewer iterations. Obviously, the results improve with more iterations. Notice that the flag pole in this example also shows that AVBD can handle long articulated chains (of 20 bodies in this case) with few iterations.

Aside from high stiffness ratios, using high absolute stiffness values can be detrimental to convergence of VBD. Figure 6 shows a delicate card tower with very lightweight bodies, requiring accurate and stiff static friction forces to keep it standing. With the stiffness required to keep the tower (partially) standing, the potential energy term dominates the kinetic energy term of VBD, causing the cards to appear heavier than they really are. In this example, relatively heavier balls are thrown to hit the bottom of the tower, but they fail to knock the cards down and bounce back instead. In comparison, our AVBD method does not require high stiffness to satisfy the constraints, thus eliminating this issue. So, a single ball hitting the bottom easily passes through and the tower collapses.



(a) *VBD* **(b)** ***AVBD***

Fig. 6. *A card tower held by friction: (a) VBD requires carefully tuning the contact stiffness and can still fail to preserve the structure and generate excessive friction, (b) hard constraints of AVBD can easily handle this case without any parameter tuning. Both methods use 5 substeps for discrete collision detection and 5 iterations per substep.*

5.2 High Mass Ratios

Dual methods, on the other hand, are known to struggle with high mass ratios. This can be seen in the comparison using a swinging pendulum simulation, shown in Figure 7. Here, the pendulum is formed by a long chain of 50 rigid bodies connected with ball sockets. Our AVBD method is capable of maintaining attachment constraints with minimal stretching even using low iteration counts. Despite the high stiffness, VBD cannot prevent stretching, which is more obvious with lower iteration counts. XPBD using substeps produces a similar stretching, though with some instabilities in animation due to high mass ratio, as can be seen in our supplemental video. Using even a larger number of iterations, instead of substeps, XPBD entirely fails to limit stretching and the pendulum hits the ground. This is also the case with sequential impulse in this example.

We present the constraint error across frames using 20 iterations/substeps with XPBD, VBD, and our AVBD method in Figure 8 for the same swinging pendulum example. Notice that AVBD consistently maintains a lower constraint error and does not have the error oscillations of XPBD. Sequential impulse is not included in this graph, as it fails to generate a swinging motion.

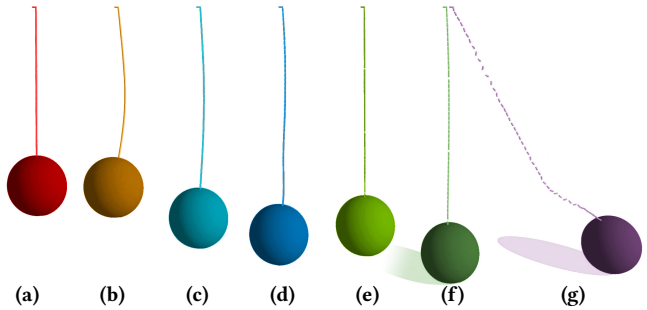


Fig. 7. *Simulation of a pendulum formed by a chain of 50 bodies connected with ball-socket constraints and a heavy object at the end with 50,000:1 mass ratio, simulated using different methods with different numbers of iterations/sub-steps. Notice that our AVBD formulation is the only one that can prevent excessive stretching of the chain.*

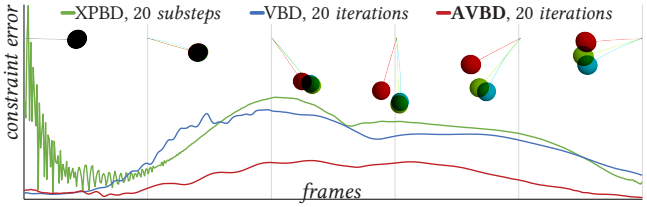


Fig. 8. *The constraint error across frames for a swinging pendulum, using the same construction as in Figure 7.*

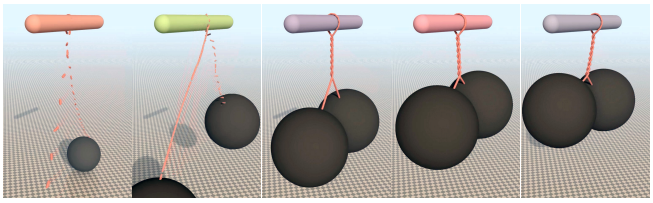
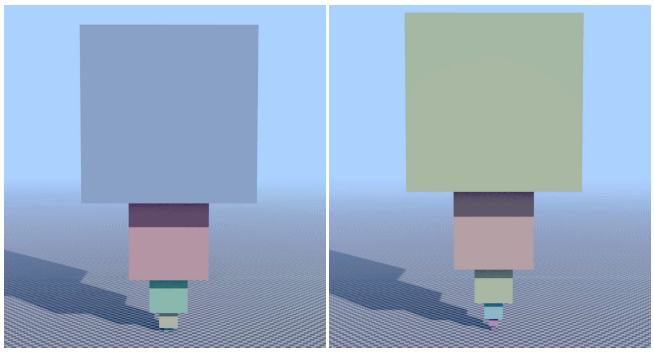


Fig. 9. *Two heavy balls attached by a chain of 50 bodies. Primal methods can handle the mass ratio, while dual methods quickly fail. VBD leads to some excessive stretching, as compared to AVBD.*

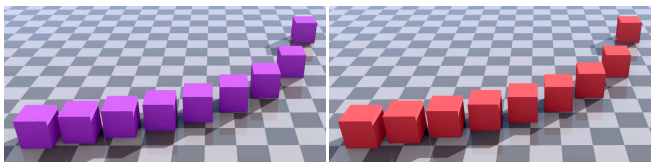
A more challenging example is shown in Figure 9, where two heavy balls are joined with a chain. The dual methods (XPBD and sequential impulse) quickly fail to maintain the constraints that hold the chain together due to mass ratio. Primal methods handle this test well, though VBD leads to excessive stretching, but our AVBD method can use hard constraints to preserve the chain length.

Problems regarding mass ratios with dual methods are also apparent in Figure 10. Notice that the block at the bottom is completely crushed with sequential impulse, while our AVBD method can properly maintain the collision constraints in this example.



(a) *Sequential Impulse, 20 iterations* **(b)** ***AVBD**, 20 iterations*

Fig. 10. *Simulation of a stack of boxes. (a) The bottom of the stack collapses with sequential impulse. (b) AVBD maintains the stack.*



(a) *Sequential Impulse, 4 iterations* **(b)** ***AVBD**, 4 iterations*

Fig. 11. *Blocks of different friction coefficient with the same initial velocity, sliding on the ground until they stop, showing that (a) sequential impulse and (b) AVBD produce visually identical friction behavior.*

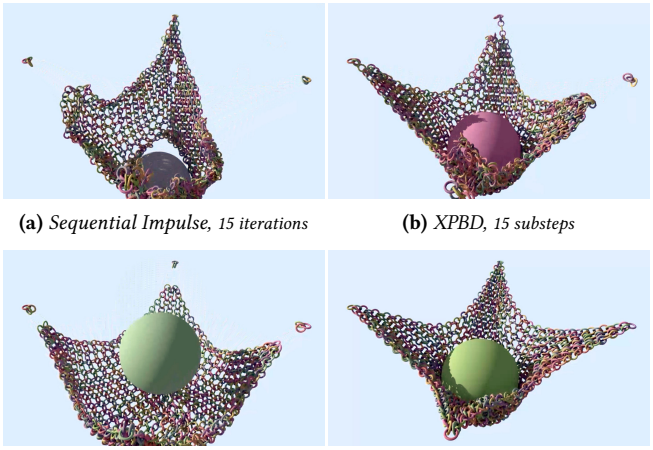
5.3 Friction

On the other hand, sequential impulse can accurately model the behavior of different friction coefficients. Figure 11 shows that AVBD can match the same friction behavior as sequential impulse. Unlike our other examples, where using stiffness rescaling (Equation 14) instead of our simple solution of calculating the Hessian without clamping makes no noticeable improvement, stiffness rescaling improves convergence in this test, allowing us to match the behavior of Sequential Impulse using only a single AVBD iteration (instead of the 4 iterations used in Figure 11).

5.4 Complex Collisions and Constraints

Figure 12 shows a challenging collision case, where a heavy ball is dropped on a chain mail. With sequential impulse, the ball goes through the chain mail, as the collision constraints cannot overcome the momentum of the ball. XPBD with substeps can maintain the integrity of the chain mail near the collision with the ball, but the contacts at the corners fail. These same contacts at the corners also fail with VBD, but even before the ball touches the chain mail, this is because the quadratic energy potential cannot produce large enough forces for those contacts. Our AVBD formulation successfully handles this case without any visual artifacts, maintaining all contacts using hard constraints.

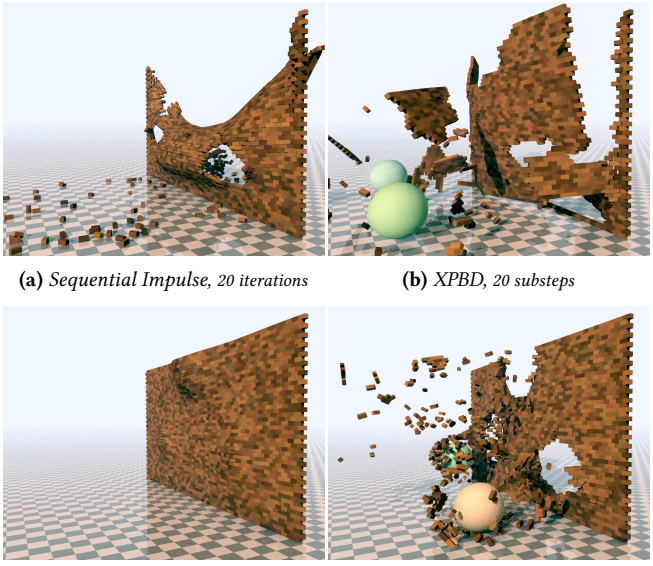
Figure 13 shows a simplified wall break simulation, where three balls with high momentum smash into a brick wall. The bricks are attached via breakable hard constraints with a maximum force threshold. In this example, sequential impulse generates a clean



(a) *Sequential Impulse, 15 iterations* **(b)** *XPBD, 15 substeps*

(c) *VBD, 15 iterations* **(d)** ***AVBD**, 15 iterations*

Fig. 12. *A heavy ball dropping on a chain mail. Aside form AVBD, all other methods fail to properly resolve the collisions. All methods use 15 iterations/substeps in this test.*



(a) *Sequential Impulse, 20 iterations* **(b)** *XPBD, 20 substeps*

(c) *VBD, 20 iterations* **(d)** ***AVBD**, 20 iterations*

Fig. 13. *A wall constructed using bricks that are joined via breakable attachment constraints, smashed by three balls with high momentum. The dual methods fail to maintain the constraints, resulting in bending artifacts and unnatural fracture. VBD bends, but does not break the wall. Our AVBD method produces plausible breaks.*

fracture at first, but it cannot maintain the constraints and the disturbance of the initial fractures makes the wall slowly bend and finally collapse. XPBD also struggles to maintain the constraints after the collision and its inaccuracies makes the wall tear at unnatural places, producing a rubbery behavior. VBD, on the other hand, entirely prevents fracture, but bends the wall instead, smoothly distributing the constraint error. Our AVBD method forms clean wall breaks with plausible fractures and successfully maintains the attachment constrains that are retained after the initial impacts.

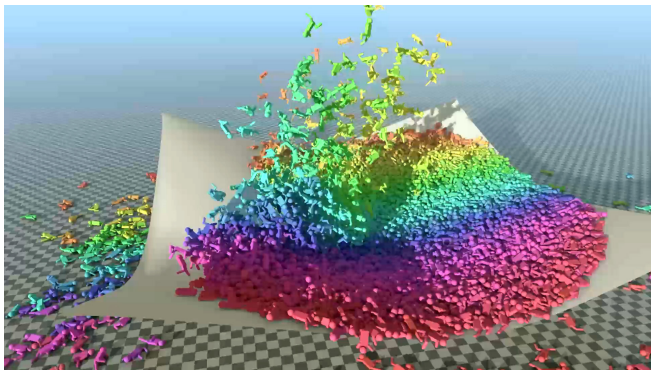


Fig. 14. *35,000 rigid bodies connected with 72,000 joints falling onto a cloth with 10,000 vertices (20,000 triangles), simulated using AVBD with 10 iterations in 16 ms per frame.*

Figure 14 shows a complex example involving a large deformable cloth and articulated rigid bodies falling onto it, where all joints, collision constraints, and cloth forces are solved in a unified fashion using our augmented Lagrangian formulation. Our AVBD method can efficiently simulate this scene, simulated here with 10 iterations taking 16 ms per frame including collision detection.

5.5 Performance

The computational cost of all of these methods scale linearly with the number of iterations/substeps used. The cost of each iteration scales linearly with the number of bodies for primal methods (VBD and AVBD) and with the number of forces/constraints for dual methods (XPBD and sequential impulse). Most of the computation time is used by iterating over bodies/constraints with a minor overhead for coloring and synchronization.

Table 1. *Solver performance of different methods in our large scenes.*

	Seq. Imp.	XPBD	VBD	**AVBD**
Figure 1	15 iter.	6 sub.	15 iter.	4 iter.
110,000 bodies	14.1 ms	19.5 ms	10.8 ms	3.5 ms
Figure 3	27 iter.	26 sub.	8 iter.	3 iter.
510,000 bodies	78.4 ms	524.7 ms	26.5 ms	10.3 ms

We provide a performance comparison using our large scenes in Table 1. Here, the performance differences between methods is mainly due to the number of iterations used. In Figure 1 and Figure 3, our AVBD method can produce stable simulations with only 4 and 3 iterations, respectively. Other methods, however, require more iterations/substeps just to prevent the piles of blocks from collapsing with gravity. As a result, they require significantly longer simulation times. In particular, the scene in Figure 3 turned out to be a challenging case for XPBD and even with 26 substeps the piles of blocks were collapsing after several seconds in our tests. Thus, the stability of the simulation is a primary factor in determining its relative performance.

Table 2. *Parameters used in our formulation of AVBD.*

	Range	Used	Reference	Description
β	$(0, \infty)$	10	Equation 12	Stiffness ramping controlling convergence rate
α	$[0, 1]$	0.95	Equation 18	Regularization to prevent explosive error correction
γ	$[0, 1)$	0.99	Equation 19	Warm starting of stiffness and dual variables

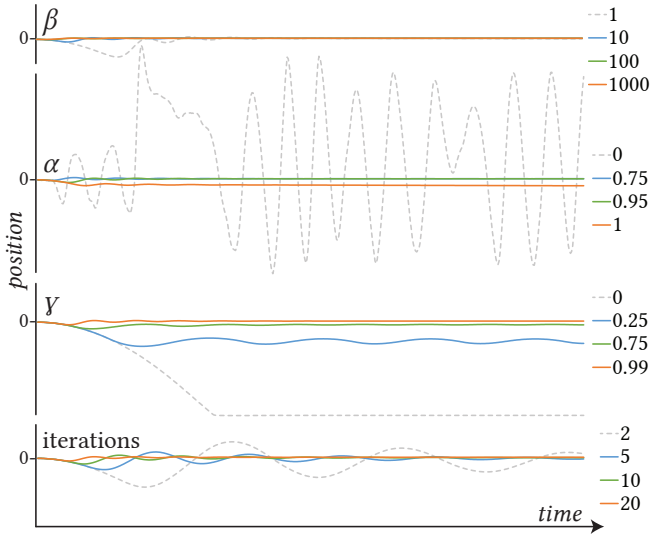


Fig. 15. *The effect of different parameter values. The pendulum in Figure 7a is initialized in the vertical pose shown and the vertical position of the heavy ball is recorded over time with different parameter values, using the default values in Table 2 with 20 iterations by default.*

5.6 Parameter Tests

Our formulation of AVBD relies on a number of user-defined parameters, shown in Table 2. Though none of these parameters require careful tuning, they can impact the behavior of the solver. Yet, with simple examples, it is difficult to see any behavioral change with different parameters, and complex animations obfuscate their impact. Therefore, below we discuss these parameters using a relatively challenging quasi-static pendulum example shown in Figure 7a, involving 50 rigid links, forming a chain that holds a heavy ball. The pendulum is initialized in this vertical pose, and the vertical position of the ball is tracked over time, using different parameters. The results are shown in Figure 15. The corresponding animation sequences are included in our supplemental video.

The β parameter controls how quickly the stiffness is ramped up using Equation 12. It makes little difference in a quasi-static case using warm starting. Larger values can improve convergence, but can also lead to excessive stiffness. When β is too small, it takes a number of frames to build up sufficient stiffness and dual variable.

The α parameter controls the portion of accumulated constraint error from prior frames to be corrected in the current frame. $\alpha = 0$ results in explosive error correction, effectively turning off our prevention mechanism (see Section 3.6). Using larger values for α avoids this instability by reducing the energy injection due to error correction. $\alpha = 1$ removes all existing constraint error and only attempts to prevent new error, and so constraint error gradually accumulates as the simulation proceeds. Post stabilization could be applied to correct the error with this configuration.

The γ parameter controls how much of the previous frame's stiffness and dual variable to use for warm starting. $\gamma = 0$ results in poor convergence, since it effectively disables warm starting, and every frame must recompute the stiffness and dual variables from scratch. In this example, the chain fails to accumulate sufficient tension (using the default values for the other parameters), and the ball hits the ground. Larger values for γ improve convergence. Yet, $\gamma = 1$ is not a valid configuration, since this would never allow the stiffness value to decrease.

It is important to note that regardless of the choices of γ and β, with enough iterations, the simulation converges towards zero constraint error. This is unlike VBD, where the chosen stiffness value determines how stiff the constraints will be. In this test, using the default values, even with a small number of iterations per frame, convergence can be achieved after a number frames.

We do not discuss the k_{start} parameter here, because even in this relatively complex test case, it makes no visible difference. In fact, in this case, it virtually has no impact, except for the very first iteration of the very first frame.

6 Discussion

The main tunable parameter of our approach is β used to control how quickly stiffness is ramped up. It is worth pointing out that this parameter does not affect the converged result, only how quickly convergence is achieved. This is due to the augmented Lagrangian term, which converges to the same constraint forces as the Lagrange multipliers of the dual formulations like XPBD, regardless of the stiffness value. Indeed, one could use a fixed stiffness value and still converge to the correct result given enough iterations.

Position based methods, including our work, inject energy into the system to correct constraint error. The amount of energy added increases as the time-step decreases. This motivates our approach of subtracting some of the initial constraint error which exists at the start of the step, in order to spread out the error correction over several frames. This can be viewed as a *Baumgarte stabilization* like approach, as it adds a small amount of energy to correct positional error, depending on the value of α.

If it is desired to add zero additional energy, we could use post stabilization [Cline and Pai 2003]. In our framework, this involves first solving using $\alpha = 1$, such that no existing position error is corrected for, then performing a second *post stabilization* solve using $\alpha = 0$, and leaving out the velocity update at the end of the solve. This can improve stability, however it adds a small performance cost due to the extra solve. Therefore, we opt for the more efficient Baumgarte stabilization like approach for all our examples.

Depending on the scene complexity, collision detection can take close to or even longer than the simulation step. For example, our collision detection implementation takes about 6.3 ms in Figure 1 and 7.2 ms in Figure 3. Modern GPUs contain hardware specifically for

ray tracing and construction of spatial acceleration data structures. Unfortunately, the APIs currently only allow intersecting rays with these structures. If they were to provide the ability to intersect these structures with a bounding box, or get the closest triangle to a query point, we could massively accelerate the broad-phase and narrow-phase collision detection steps.

One fundamental limitation of local iterative techniques like AVBD is information propagation (ie. forces) across constraints is limited by the number of iterations. This can be noticeable with very large mechanisms and chains of bodies with low iteration counts. Though we demonstrated long chains in our examples, propagation can take multiple frames. It would be interesting to explore combining AVBD with a global solver. Also, as AVBD uses backwards Euler integration (BDF1), there is naturally some energy dissipation with higher time steps. Investigation into higher order integrators like BDF2 would also be interesting future work.

7 Conclusion

We have introduced Augmented-VBD that extends the VBD method with an augmented Lagrangian formulation to provide a mechanism for handling hard constraints. Using these hard constraints, we have also described methods for efficiently modeling inequality constraints and frictional contacts and we have shown that we can produce the expected behavior for a range of friction coefficients. Motivated by our formulation of hard constraints, we have presented a simple technique that can overcome arguably one of the biggest limitations of VBD: simulating high stiffness ratios. Finally, we have presented methods for improving stability by approximating Hessians and avoiding explosive error correction. Our experiments show that AVBD delivers improved animation quality, superior performance, and better stability as compared to alternative methods, and that it is able to handle complex scenes involving stacking, frictional contact, and other hard constraints using a limited number of iterations per frame.

References

Sheldon Andrews, Marek Teichmann, and Paul G. Kry. 2017a. Geometric Stiffness for Real-time Constrained Multibody Dynamics. *Computer Graphics Forum* 36, 2 (2017), 235–246. doi:10.1111/cgf.13122

Sheldon Andrews, Marek Teichmann, and Paul G. Kry. 2017b. Geometric Stiffness for Real-time Constrained Multibody Dynamics. *Computer Graphics Forum* 36, 2 (2017). doi:10.1111/cgf.13122

David Baraff. 1997. *An Introduction to Physically Based Modeling: Rigid Body Simulation I—Unconstrained Rigid Body Dynamics.* Technical Report CMU-RI-TR-97-13. Robotics Institute, Carnegie Mellon University. Available at https://www.cs.cmu.edu/~baraff/sigcourse/notesd1.pdf.

David Baraff and Andrew Witkin. 1998. Large Steps in Cloth Simulation. (1998).

E G Birgin and J M Martínez. 2014. *Practical augmented Lagrangian methods for constrained optimization.* Society for Industrial and Applied Mathematics, Philadelphia, PA.

Erin Catto. 2006. Fast and Simple Physics using Sequential Impulses. In *Proceedings of the Game Developers Conference.* Available at https://box2d.org/files/ErinCatto_SequentialImpulses_GDC2006.pdf.

Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. 2024a. Vertex Block Descent. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2024)* 43, 4, Article 116 (07 2024), 16 pages. doi:10.1145/3658179

Yi-Lu Chen, Mickaël Ly, and Chris Wojtan. 2024b. Primal–Dual Non-Smooth Friction for Rigid Body Animation. In *SIGGRAPH 2024 Conference Papers.* Association for Computing Machinery, New York, NY, USA. doi:10.1145/3641519.3657485

M.B. Cline and D.K. Pai. 2003. Post-stabilization for rigid body simulation with contact and constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation.* 3744–3751. doi:10.1109/ROBOT.2003.1242171

M. Fratarcangeli and F. Pellacini. 2015. Scalable Partitioning for Parallel Position Based Dynamics. *Computer Graphics Forum* 34, 2 (May 2015), 405–413. doi:10.1111/cgf.12570

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: a practical gauss-seidel method for stable soft body dynamics. *ACM Transactions on Graphics* 35, 6 (Nov. 2016), 1–9. doi:10.1145/2980179.2982437

Chris Giles and Sheldon Andrews. 2024. Adaptive Sub-stepping for Constrained Rigid Body Simulations. In *Proceedings of the 17th ACM SIGGRAPH Conference on Motion, Interaction, and Games* (Arlington, VA, USA) *(MIG '24).* doi:10.1145/3677388.3696331

Dewen Guo, Minchen Li, Yin Yang, Sheng Li, and Guoping Wang. 2024. Barrier-Augmented Lagrangian for GPU-based Elastodynamic Contact. *ACM Trans. Graph.* 43, 6, Article 225 (Nov. 2024), 17 pages. doi:10.1145/3687988

G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs. 2001. An implicit finite element method for elastic solids in contact. In *Proceedings Computer Animation 2001. Fourteenth Conference on Computer Animation (Cat. No.01TH8596).* IEEE Comput. Soc, Seoul, South Korea, 136–254. doi:10.1109/CA.2001.982387

C. Kane, J. E. Marsden, and M. Ortiz. 1999. Symplectic-energy-momentum preserving variational integrators. *J. Math. Phys.* 40, 7 (July 1999), 3353–3371. doi:10.1063/1.532892

Couro Kane, Jerrold E Marsden, Michael Ortiz, and Matthew West. 2000. Variational integrators and the Newmark algorithm for conservative and dissipative mechanical systems. *International Journal for numerical methods in engineering* 49, 10 (2000), 1295–1325.

Liliya Kharevych, Weiwei Yang, Yiying Tong, Eva Kanso, Jerrold E. Marsden, Peter Schröder, and Matthieu Desbrun. 2006. Geometric, Variational Integrators for Computer Animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Marie-Paule Cani and James O'Brien (Eds.). The Eurographics Association. doi:10.2312/SCA/SCA06/043-051

Lei Lan, Danny M. Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. 2022. Affine Body Dynamics: Fast, Stable & Intersection-free Simulation of Stiff Materials. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 67:1–67:14. doi:10.1145/3528223.3530064

Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. 2009. Fast BVH Construction on GPUs. *Computer Graphics Forum* 28, 2 (2009), 375–384. doi:10.1111/j.1467-8659.2009.01377.x

A. Lew, J. E. Marsden, M. Ortiz, and M. West. 2004. Variational time integrators. *Internat. J. Numer. Methods Engrg.* 60, 1 (May 2004), 153–212. doi:10.1002/nme.958

M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T.Y. Kim. 2020. Primal/Dual Descent Methods for Dynamics. *Computer Graphics Forum* 39, 8 (2020), 89–100. doi:10.1111/cgf.14104 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14104

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games.* ACM, Burlingame California, 49–54. doi:10.1145/2994258.2994272

Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. 2019. Small Steps in Physics Simulation. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 1–7. doi:10.1145/3309486.3340247

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-based elastic materials. In *ACM SIGGRAPH 2011 papers.* 1–8.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. 2020. Detailed Rigid Body Simulation with Extended Position Based Dynamics. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA).* 101–112. doi:10.1111/cgf.14105

J.C. Simo, N. Tarnow, and K.K. Wong. 1992. Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics. *Computer Methods in Applied Mechanics and Engineering* 100, 1 (Oct. 1992), 63–116. doi:10.1016/0045-7825(92)90115-Z

Quoc-Minh Ton-That, Paul G. Kry, and Sheldon Andrews. 2023. Parallel block Neo-Hookean XPBD using graph clustering. *Computers & Graphics* 110 (Feb. 2023), 1–10. doi:10.1016/j.cag.2022.10.009

P. Volino and N. Magnenat-Thalmann. 2001. Comparing efficiency of integration methods for cloth simulation. In *Proceedings. Computer Graphics International 2001.* IEEE Comput. Soc, Hong Kong, China, 265–272. doi:10.1109/CGI.2001.934683