

Vertex Block Descent

Ankha He Chen, Ziheng Liu, Yin Yang, Cem Yuskel
[2024]



Optimization Integrator for Large Time Steps

Optimization Integrator for Large Time Steps

Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang and Joseph M. Teran

Abstract—Practical time steps in today's state-of-the-art simulators typically rely on Newton's method to solve large systems of nonlinear equations. In practice, this works well for small time steps but is unstable at large time steps, particularly for difficult or stiff simulations. We show that recasting backward Euler as a minimization problem allows Newton's method to be stabilized by standard optimization techniques with some novel improvements of our own. The resulting solver is capable of solving even the toughest simulations at the 24 Hz frame rate and beyond. We show how simple collisions can be incorporated directly into the solver through constrained minimization without sacrificing efficiency. We also present novel penalty collision formulations for self collisions and collisions against scripted bodies designed for the unique demands of this solver. Finally, we show that these techniques improve the behavior of Material Point Method (MPM) simulations by recasting it as an optimization problem.

Index Terms—Computer Graphics, Three-Dimensional Graphics and Realism, Animation

1 INTRODUCTION

THE most commonly used time integration schemes in use today for graphics applications are implicit methods. Among these, backward Euler [1], [2], [3], [4], [5] or variants on Newmark methods [6], [7], [8] are the most common, though even more sophisticated schemes like BDF-2 [9], [10], implicit-explicit schemes [11], [12], or even the more exotic exponential integrators [13] have received consideration. Integrators have been the subject of comparison before (see for example [3], [9], [14]), seeking good compromises between speed, accuracy, robustness, and dynamic behavior.

These integrators require the solution to one or more nonlinear systems of equations each time step. These systems are typically solved by some variation on Newton's method. Even the most stable simulators are typically run several time steps per 24 Hz frame of simulation. There is growing interest in running simulations at larger time steps [15], so that the selection of Δt can be made based on other factors, such as damping or runtime, and not only on whether the simulator works at all. One of the major factors that limits time step sizes is the inability of Newton's method to converge reliably at large time steps (See Figures 3, 2, and 4), or if a fixed number of Newton iterations are taken, the stability of the resulting simulation. We address this by formulating our nonlinear system of equations as a minimization problem, which we demonstrate can be solved more robustly. The idea that dynamics, energy, and minimization are related has been known since antiquity and is commonly leveraged in variational integrators [6], [12], [16], [17], [18], [19], [20]. The idea that the nonlinear system that occurs from methods like backward Euler can be formulated as a minimization problem has appeared many times in graphics in various forms [2], [4], [5], [13], [19], [19] point out that minimization leads to a method that is both simpler and faster than the equivalent nonlinear root-finding problem, and [5] show that a minimization formulation can be used to solve mass-spring systems more efficiently. [17] use a minimization formulation as a means of ensuring that a solution to their nonlinear system can be found assuming one exists. [21] shows that a minimization formulation can be used to enforce

constraints robustly and efficiently. [2] shows that supplementing Newton's method with a line search greatly improves robustness. [4] also shows that supplementing Newton's method with a line search and a definiteness correction leads to a robust solution procedure. Following their example, we show that recasting the solution of the nonlinear systems that result from implicit time integration schemes as a nonlinear optimization problem results in substantial robustness improvements. We also show that additional improvements can be realized by incorporating additional techniques like Wolfe condition line searches which curve around collision bodies, conjugate gradient with early termination on indefiniteness, and choosing conjugate gradient tolerances based on the current degree of convergence.

This publication is an extended version of [22] in which we have applied the optimization integrator approach to the MPM snow simulator of [23]. This allows us to take much larger time steps than the original method and results in a significant speedup.

2 TIME INTEGRATION

The equations of motion for simulating solids are

$$\dot{\mathbf{x}} = \mathbf{v} \quad \mathbf{M}\dot{\mathbf{v}} = \mathbf{f} \quad \mathbf{f} = \mathbf{f}(\mathbf{x}, \mathbf{v}),$$

where $\mathbf{f}$ are forces. As is common in graphics we assume $\mathbf{M}$ is a diagonal lumped-mass matrix. Since we are interested in robustness and large time steps, we follow a backward Euler discretization. This leads to

$$\frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t} = \mathbf{v}^{n+1} \quad \mathbf{M} \frac{\mathbf{v}^{n+1} - \mathbf{v}^n}{\Delta t} = \mathbf{f}^{n+1} = \mathbf{f}(\mathbf{x}^{n+1}, \mathbf{v}^{n+1}).$$

Eliminating $\mathbf{v}^{n+1}$ yields

$$\mathbf{M} \frac{\mathbf{x}^{n+1} - \mathbf{x}^n - \Delta t \mathbf{v}^n}{\Delta t^2} = \mathbf{f}(\mathbf{x}^{n+1}, \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t}),$$

which is a nonlinear system of equations in the unknown positions $\mathbf{x}^{n+1}$. This system of nonlinear equations is normally solved with Newton's method. If we define

$$\mathbf{h}(\mathbf{x}^{n+1}) = \mathbf{M} \frac{\mathbf{x}^{n+1} - \mathbf{x}^n - \Delta t \mathbf{v}^n}{\Delta t^2} - \mathbf{f}(\mathbf{x}^{n+1}, \frac{\mathbf{x}^{n+1} - \mathbf{x}^n}{\Delta t}), \quad (1)$$

then our nonlinear problem is one of finding a solution to $\mathbf{h}(\mathbf{x}) = \mathbf{0}$. To do this, one would start with an initial guess $\mathbf{x}^{(0)}$, such as the

$$M\ddot{x} = f(x) \quad \ddot{x} = \frac{\dot{x}^{n+1} - \dot{x}^n}{\Delta t} \quad \dot{x}^{n+1} = \frac{x^{n+1} - x^n}{\Delta t}$$

$$M \frac{x^{n+1} - x^n - \Delta t \dot{x}^n}{\Delta t^2} = f_{ext}(x) + f_{int}(x)$$

$$\frac{1}{\Delta t^2} M [x^{n+1} - x^n - \Delta t \dot{x}^n - \Delta t^2 M^{-1} f_{ext}(x)] - f_{int}(x) = 0$$

$$h(x) = \frac{\partial G(x)}{\partial x} = \frac{1}{\Delta t^2} M [x^{n+1} - \tilde{x}] - f_{int}(x) \quad f_{int}(x) = -\frac{\partial E(x)}{\partial x}$$

$$G(x) = \frac{1}{2\Delta t^2} [x^{n+1} - \tilde{x}]^T M [x^{n+1} - \tilde{x}] + \sum E(x)$$

$$x^{n+1} \leftarrow \arg \min_x G(x) \quad x^{n+1} = x^n - \frac{\partial^2 G(x)}{\partial x^2}^{-1} \frac{\partial G(x)}{\partial x}$$

• T. Gast, C. Schroeder, C. Jiang and J. Teran are with the University of California Los Angeles

• A. Stomakhin and J. Teran are with Walt Disney Animation Studios

Manuscript received November 12, 2014.

Simulation Step

$$G_i(\mathbf{x}) = \frac{m_i}{2h^2} \|\mathbf{x}_i - \mathbf{y}_i\|^2 + \sum_{j \in \mathcal{F}_i} E_j(\mathbf{x}) \quad \left| \text{Local definition} \right.$$

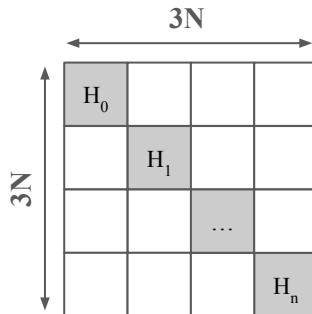
$$\mathbf{x}_i \leftarrow \underset{\mathbf{x}_i}{\operatorname{argmin}} G_i(\mathbf{x}) \quad \mathbf{H}_i \Delta \mathbf{x}_i = \mathbf{f}_i \quad \left| \text{Minimization + System} \right.$$

$$\mathbf{H}_i = \underbrace{\frac{\partial^2 G_i(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i}}_{\substack{\text{3x3} \\ \text{Matrix}}} = \underbrace{\frac{m_i}{h^2} \mathbf{I}}_{\text{const}} + \underbrace{\sum_{j \in \mathcal{F}_i} \frac{\partial^2 E_j(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i}}_{\substack{\text{All Elements} \\ \text{Sum}}} \quad \left| \text{Hessian (all elements of } \mathbf{x}_i) \right.$$

Can be solved analytically : $\Delta \mathbf{x}_i = \mathbf{H}_i^{-1} \mathbf{f}_i$

Still an approximation of the solution

We only compute the hessian “diagonal”



First Guess

Input: $\mathbf{x}^t$: the positions of the previous step; $\mathbf{v}^t$: the velocities of the previous step; $\mathbf{a}_{\text{ext}}$: the external acceleration

Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.

$\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$

Initial DCD using $\mathbf{x}^t$

$\mathbf{x} \leftarrow$ initial guess with adaptive initialization

for each iteration $n \leq n_{\text{max}}$ **do**

if $n \bmod n_{\text{col}} = 1$ **then** CCD using $\mathbf{x}$

for each color c **do**

 // Block-level parallelization

parallel for each vertex i in color c **do**

 // Thread-level parallelization

parallel for each $j \in \mathcal{F}_i$ **do**

 // Variables in shared memory

$$\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$$

$$\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$$

end

 // Local reduction sums

$$\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$$

$$\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$$

$$\Delta \mathbf{x}_i \leftarrow \mathbf{H}_i^{-1} \mathbf{f}_i$$

$\Delta \mathbf{x}_i \leftarrow$ optional line search from $\mathbf{x}_i + \Delta \mathbf{x}_i$ to $\mathbf{x}_i$

$$\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$$

end

 // Copy updated positions back to the vertex buffer

parallel for each vertex i in color c **do**

$$\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$$

end

end

 // Optional: accelerated iteration

parallel for each vertex i **do**

 Update $\mathbf{x}_i$ using Equation 18.

end

end

$$\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$$

return $\mathbf{x}$, $\mathbf{v}$

Previous position:

$$\mathbf{x} = \mathbf{x}^t$$

Inertia:

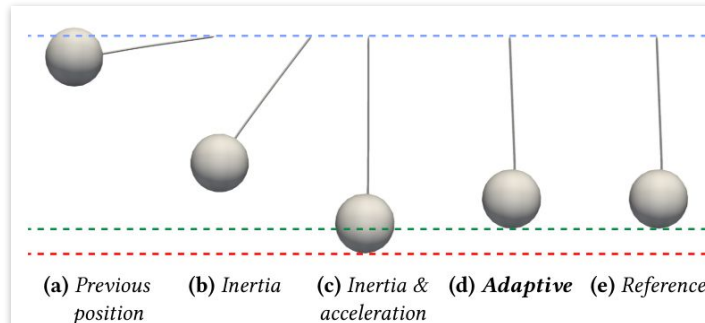
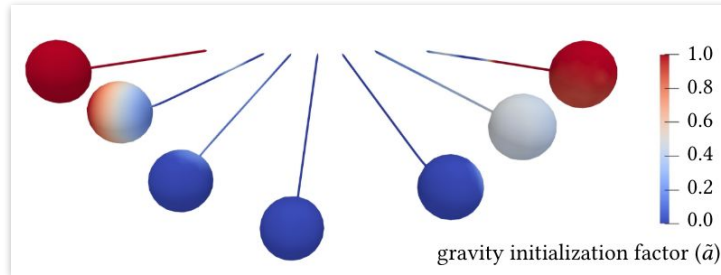
$$\mathbf{x} = \mathbf{x}^t + h\mathbf{v}^t$$

Inertia and acceleration:

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

Adaptive:

$$\mathbf{x} = \mathbf{x}^t + h\mathbf{v}^t + h^2\tilde{\mathbf{a}}$$



Adaptative first guess scale the acceleration by a factor $[0,1]$

Includes external acceleration when the motion resembles free fall

Avoid stretching artifacts
+
Avoid interpenetration at rest

Contact and Friction

Input: $\mathbf{x}^t$: the positions of the previous step; $\mathbf{v}^t$: the velocities of the previous step; $\mathbf{a}_{\text{ext}}$: the external acceleration

Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.

$\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$

Initial DCD using $\mathbf{x}^t$

$\mathbf{x} \leftarrow$ initial guess with adaptive initialization

for each iteration $n \leq n_{\text{max}}$ **do**

if $n \bmod n_{\text{col}} = 1$ **then** CCD using $\mathbf{x}$

for each color c **do**

 // Block-level parallelization

parallel for each vertex i in color c **do**

 // Thread-level parallelization

parallel for each $j \in \mathcal{F}_i$ **do**

 // Variables in shared memory

$$\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$$

$$\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$$

end

 // Local reduction sums

$$\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$$

$$\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$$

$$\Delta \mathbf{x}_i \leftarrow \mathbf{H}_i^{-1} \mathbf{f}_i$$

$$\Delta \mathbf{x}_i \leftarrow \text{optional line search from } \mathbf{x}_i + \Delta \mathbf{x}_i \text{ to } \mathbf{x}_i$$

$$\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$$

end

 // Copy updated positions back to the vertex buffer

parallel for each vertex i in color c **do**

$$\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$$

end

end

 // Optional: accelerated iteration

parallel for each vertex i **do**

 Update $\mathbf{x}_i$ using Equation 18.

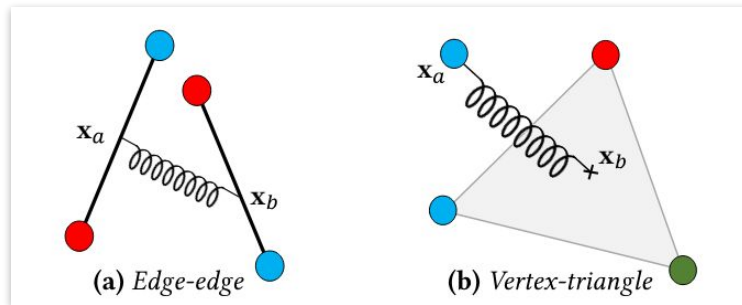
end

end

$$\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$$

return $\mathbf{x}$, $\mathbf{v}$

$$E_c(\mathbf{x}) = \frac{1}{2} k_c d^2 \quad \text{with} \quad d = \max \left(0, (\mathbf{x}_b - \mathbf{x}_a) \cdot \hat{\mathbf{n}} \right)$$



Continuous collision Detection

[2023] Shortest Path to Boundary for Self-Intersecting Meshes

Friction

[2020] IPC : Intersection- and Inversion-free, Large-Deformation Dynamics

Friction Hessian approximation

[2020] Primal/Dual Descent Methods for Dynamics



Graph coloring for multithreading

Input: $\mathbf{x}^t$: the positions of the previous step; $\mathbf{v}^t$: the velocities of the previous step; $\mathbf{a}_{\text{ext}}$: the external acceleration

Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.

$\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$

Initial DCD using $\mathbf{x}^t$

$\mathbf{x} \leftarrow$ initial guess with adaptive initialization

for each iteration $n \leq n_{\text{max}}$ **do**

if $n \bmod n_{\text{col}} = 1$ **then** CCD using $\mathbf{x}$

for each color c **do**

 // Block-level parallelization

parallel for each vertex i in color c **do**

 // Thread-level parallelization

parallel for each $j \in \mathcal{F}_i$ **do**

 // Variables in shared memory

$$\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$$

$$\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$$

end

 // Local reduction sums

$$\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$$

$$\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$$

$$\Delta \mathbf{x}_i \leftarrow \mathbf{H}_i^{-1} \mathbf{f}_i$$

$$\Delta \mathbf{x}_i \leftarrow \text{optional line search from } \mathbf{x}_i + \Delta \mathbf{x}_i \text{ to } \mathbf{x}_i$$

$$\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$$

end

 // Copy updated positions back to the vertex buffer

parallel for each vertex i in color c **do**

$$\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$$

end

end

 // Optional: accelerated iteration

parallel for each vertex i **do**

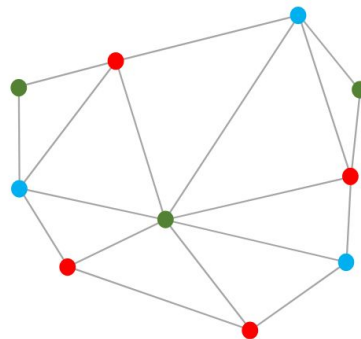
 Update $\mathbf{x}_i$ using Equation 18.

end

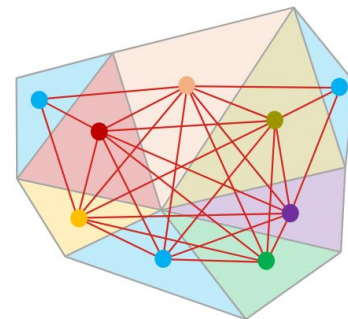
end

$$\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$$

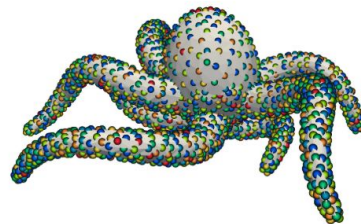
return $\mathbf{x}, \mathbf{v}$



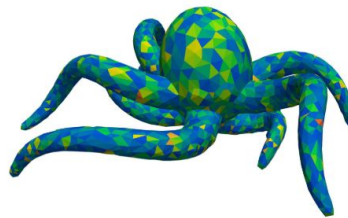
(a) Vertex colors: 4
4 :)



(b) Dual graph & element colors: 7



(c) Vertex colors: 8



(d) Element colors: 76

Mesh graph coloration is better

Less step for GPU => faster

Correction Step

Input: $\mathbf{x}^t$: the positions of the previous step; $\mathbf{v}^t$: the velocities of the previous step; $\mathbf{a}_{\text{ext}}$: the external acceleration

Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.

$\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$

Initial DCD using $\mathbf{x}^t$

$\mathbf{x} \leftarrow$ initial guess with adaptive initialization

for each iteration $n \leq n_{\text{max}}$ **do**

if $n \bmod n_{\text{col}} = 1$ **then** CCD using $\mathbf{x}$

for each color c **do**

 // Block-level parallelization

parallel for each vertex i in color c **do**

 // Thread-level parallelization

parallel for each $j \in \mathcal{F}_i$ **do**

 // Variables in shared memory

$\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$

$\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$

end

 // Local reduction sums

$\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$

$\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$

$\Delta \mathbf{x}_i \leftarrow \mathbf{H}_i^{-1} \mathbf{f}_i$

$\Delta \mathbf{x}_i \leftarrow$ optional line search from $\mathbf{x}_i + \Delta \mathbf{x}_i$ to $\mathbf{x}_i$

$\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$

end

 // Copy updated positions back to the vertex buffer

parallel for each vertex i in color c **do**

$\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$

end

end

 // Optional: accelerated iteration

parallel for each vertex i **do**

 Update $\mathbf{x}_i$ using Equation 18.

end

end

$\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$

return $\mathbf{x}, \mathbf{v}$

$$G_i(\mathbf{x}) = \frac{m_i}{2h^2} \|\mathbf{x}_i - \mathbf{y}_i\|^2 + \sum_{j \in \mathcal{F}_i} E_j(\mathbf{x}) \quad \left| \text{Local definition} \right.$$

$$\mathbf{x}_i \leftarrow \underset{\mathbf{x}_i}{\text{argmin}} G_i(\mathbf{x}) \quad \mathbf{H}_i \Delta \mathbf{x}_i = \mathbf{f}_i \quad \left| \text{Minimization + System} \right.$$

$$\mathbf{H}_i = \underbrace{\frac{\partial^2 G_i(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i}}_{\substack{\text{3x3} \\ \text{Matrix}}} = \underbrace{\frac{m_i}{h^2} \mathbf{I}}_{\text{const}} + \underbrace{\sum_{j \in \mathcal{F}_i} \frac{\partial^2 E_j(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_i}}_{\substack{\text{All Elements} \\ \text{Sum}}} \quad \left| \text{Hessian (all elements of } \mathbf{x}_i) \right.$$

Every forces and hessian can be computed in parallel (for a same color)

Sum at each vertice is done with a reduction => classic GPU algorithm

Then compute the analytical solution for the correction

“Optional” Acceleration

Input: $\mathbf{x}^t$: the positions of the previous step; $\mathbf{v}^t$: the velocities of the previous step; $\mathbf{a}_{\text{ext}}$: the external acceleration

Output: This step's position $\mathbf{x}^{t+1}$ and velocity $\mathbf{v}^{t+1}$.

$\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$

Initial DCD using $\mathbf{x}^t$

$\mathbf{x} \leftarrow$ initial guess with adaptive initialization

for each iteration $n \leq n_{\text{max}}$ **do**

if $n \bmod n_{\text{col}} = 1$ **then** CCD using $\mathbf{x}$

for each color c **do**

 // Block-level parallelization

parallel for each vertex i in color c **do**

 // Thread-level parallelization

parallel for each $j \in \mathcal{F}_i$ **do**

 // Variables in shared memory

$$\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$$

$$\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$$

end

 // Local reduction sums

$$\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$$

$$\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$$

$$\Delta \mathbf{x}_i \leftarrow \mathbf{H}_i^{-1} \mathbf{f}_i$$

$\Delta \mathbf{x}_i \leftarrow$ optional line search from $\mathbf{x}_i + \Delta \mathbf{x}_i$ to $\mathbf{x}_i$

$$\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$$

end

 // Copy updated positions back to the vertex buffer

parallel for each vertex i in color c **do**

$$\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$$

end

end

 // Optional: accelerated iteration

parallel for each vertex i **do**

 Update $\mathbf{x}_i$ using Equation 18.

end

end

$$\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$$

return $\mathbf{x}$, $\mathbf{v}$

Chebyshev semi-iterative approach

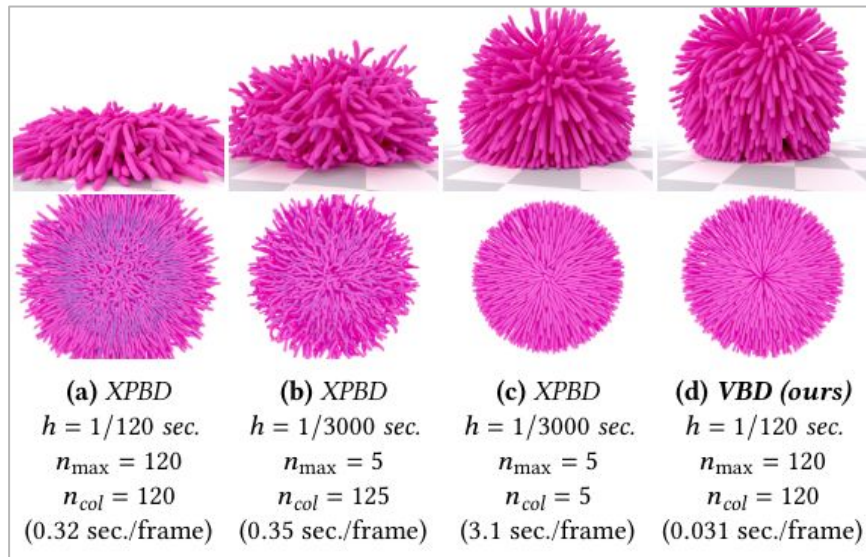
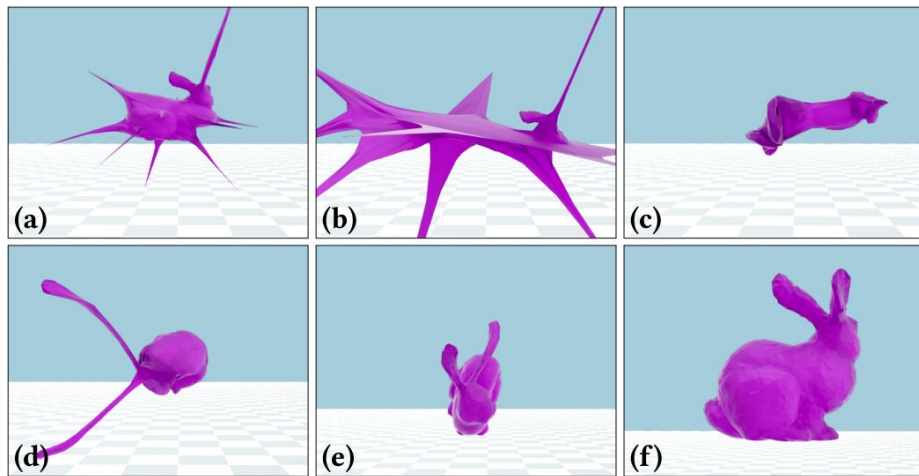
[2015] A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-based Dynamics



Cheap and Simple

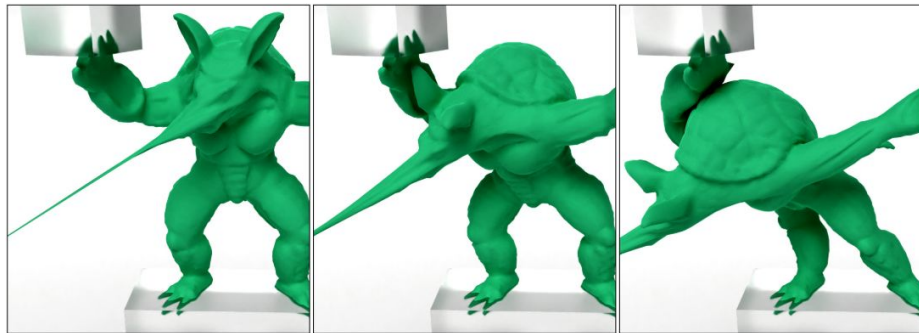
Significantly better result

Robustness



Very robust method and quite fast !

The comparison with XPBD is not fair though



Video

Nvidia RTX 4090

Bibliography

[2024] Vertex Block Descent

[2023] Shortest Path to Boundary for Self-Intersecting Meshes

[2020] IPC : Intersection- and Inversion-free, Large-Deformation Dynamics

[2020] Primal/Dual Descent Methods for Dynamics

[2015] A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-based Dynamics

[2015] Scalable Partitioning for Parallel Position Based Dynamics