

Vertex Block Descent

Par Anka He Chen, Ziheng Liu, Yin Yang, Cem Yuksel

Présenté par Pierre-Antoine Heredero

Diapositives de la présentation et notes accompagnatrices disponibles en format Markdown (**.md**) et PDF (**.pdf**).



papshed/MTI855-Presentation



1 Contributor 0 Issues 0 Stars 0 Forks

papshed/MTI855-Presentation
Contribute to papshed/MTI855-Presentation development by creating an account on GitHub.

 GitHub

Les illustrations utilisées dans les diapositives sont tirées du domaine public ou de la publication faisant l'objet de la présentation.

Méthodes existantes

- Résultat de haute qualité, système glouton.
- Système efficace, sacrifice en réalisme et qualité.

Motivations

- Convergence numérique avec stabilité inconditionnelle.
- Performance de calcul supérieure.

The Pitch

1. Stabilité améliorée.
2. Performance de calcul accélérée.
3. Réalisme visuel accru.



Défi principal

Développer une méthode capable d'opérer efficacement au niveau des sommets tout en:

1. garantissant une réduction globale de l'énergie;
2. maximisant le parallélisme.

Vertex Block Coordinate

Méthode basée sur la technique de **Block Coordinate Descent** qui performe des itérations de **Gauss-Seidel** basées sur les **sommets** pour résoudre la **forme variationnelle** d'**Euler implicite**.



Tangente #1

Block Coordinate Descent

QUOI? Algorithme d'optimisation.

COMMENT? Minimise une fonction en mettant à jour itérativement un bloc de variables à la fois, tout en gardant les autres fixes.

1. Fonction divisée en blocs de variables.
2. Parcours de chaque bloc de variable.
3. Minimiser la fonction par rapport au bloc en gardant tous les autres blocs fixes.
4. Converger, ou répéter étapes 1 à 3.



Tangente #2

Forme variationnelle

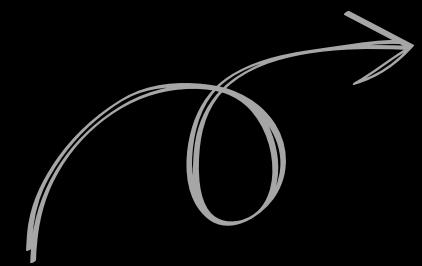
QUOI? Méthode alternative de formuler la méthode d'Euler implicite pour une équation différentielle.

COMMENT? Reformule le problème de pas de temps comme un problème d'optimisation.

Au lieu de résoudre directement les équations de mise à jour implicites, elle cherche à minimiser une certaine fonctionnelle (une fonction de fonctions) qui équilibre deux termes concurrents.



Optimization globale



Tangente #2

Forme variationnelle

TL;DR

La technique d'optimization tombe dans la catégorie des méthodes de descente de coordonnées, pour minimiser efficacement l'énergie variationnelle, en fonction de la position.

$$x_{t+1} = \operatorname{argmin}_x G(x)$$

où $G(x)$

énergie variationnelle

$$x_{t+1}$$

position au pas de temps suivant

$$G(x) = \frac{1}{2h^2} \|x - y\|_M^2 + E(x)$$

où $E(x)$

énergie potentielle totale

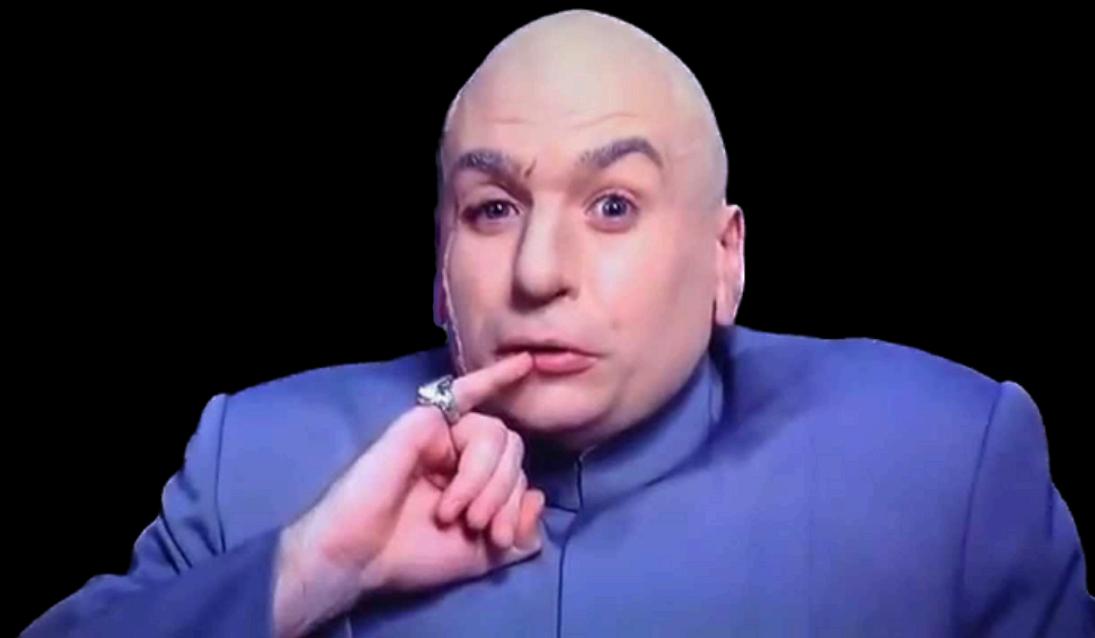
$$\frac{1}{2h^2} \|x - y\|_M^2$$

inertie potentielle

$$\text{avec } y = x^t + hv^t + h^2 a_{ext}$$

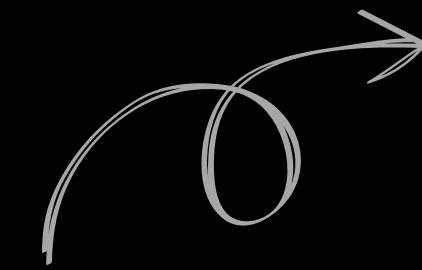
où h pas de temps

a_{ext} accélération externe (fixe)



Optimization locale

Tangente #1
Block Coordinate Descent



COMMENT? Modifier un seul sommet à la fois en fixant tous les autres sommets.

POURQUOI? La partie du terme énergétique $E(x)$ qui est affectée ne comprend que l'ensemble des éléments de force F_i qui agissent sur (ou utilisent la position de) le sommet i .

$$G_i(x) = \frac{m_i}{2h^2} \|x_i - y_i\|^2 + \sum_{j \in F_i} E_j(x)$$

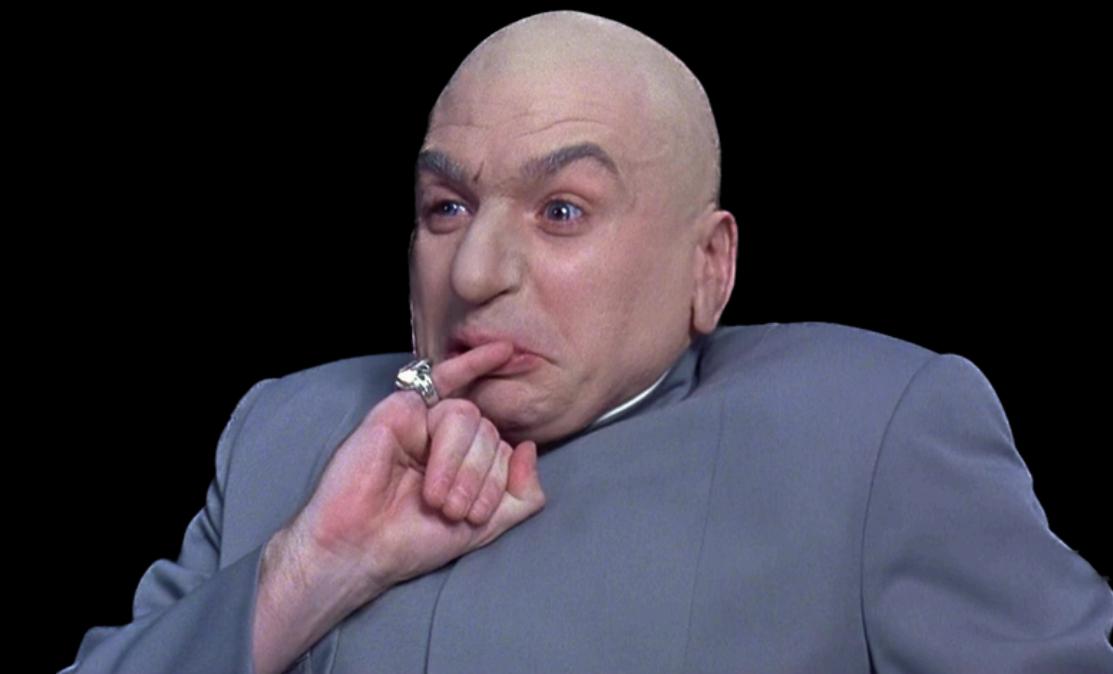
où

$G_i(x)$	énergie variationnelle locale au sommet i
m_i	masse du sommet i
E_j	énergie de l'élément de force j du sommet i

Le problème d'optimization globale peut être reformulé comme une suite de problèmes d'optimization locale.

$$x_i \leftarrow \operatorname{argmin}_{x_i} G_i(x)$$

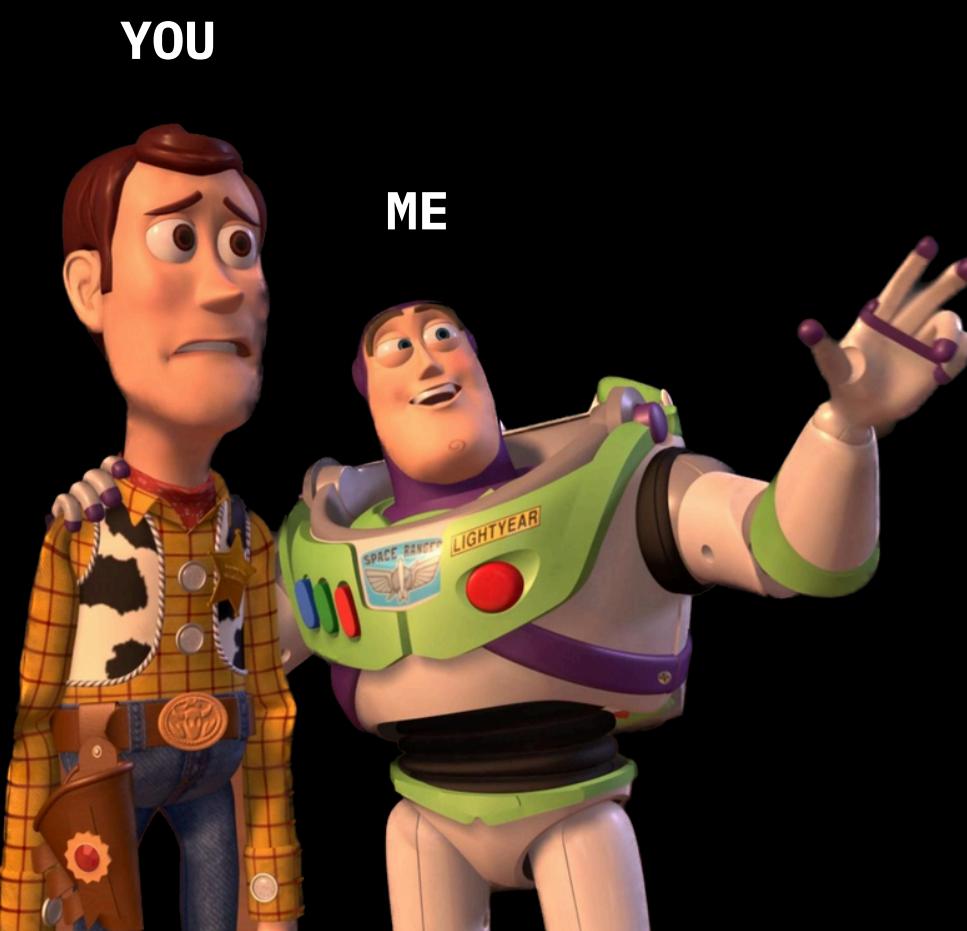
On résoudra le système à l'aide d'itérations de Gauss-Seidel.



TL;DR

VBD décompose le problème d'optimization globale en problème d'optimization locale pour chaque sommet.

La vitesse résultante est calculée selon la formulation d'Euler implicite, après n itérations de Gauss-Seidel.



$$v^{t+1} = \frac{1}{h}(x^{t+1} - x^t)$$

Solveur de système local

Système à résoudre:

$$H_i \Delta x_i = f_i \quad \text{où} \quad f_i = -\frac{\partial G_i(x)}{\partial x_i} = -\frac{m_i}{h^2}(x_i - y_i) - \sum_{j \in F_i} \frac{\partial E_j(x)}{\partial x_i} \quad \begin{matrix} \text{force agissant} \\ \text{sur le sommet } i \end{matrix}$$
$$H_i = \frac{m_i}{h^2} I + \sum_{j \in F_i} \frac{\partial^2 E_j}{\partial x_i \partial x_i} \quad \text{hessienne de } G_i$$

Le système est résolu analytiquement!

$$\Delta x_i = H_i^{-1} f_i$$



Amortissement

Ajout d'un coefficient d'amortissement k_d pour donner le contrôle à l'utilisateur.

$$f_i = -\frac{m_i}{h^2}(x_i - y_i) - \sum_{j \in F_i} \frac{\partial E_j(x)}{\partial x_i} - \left(\sum_{j \in F_i} \frac{k_d}{h} \frac{\partial^2 E_j}{\partial x_i \partial x_i} \right) (x_i - x_i^t)$$

$$H_i = \frac{m_i}{h^2} I + \sum_{j \in F_i} \frac{\partial^2 E_j}{\partial x_i \partial x_i} + \left(\sum_{j \in F_i} \frac{k_d}{h} \frac{\partial^2 E_j}{\partial x_i \partial x_i} \right)$$

Collisions

2 types de collisions dans le modèle:

- collision arrête-arrête (CCD)
- collision arrête-face (CCD ou DCD)



Soient

x_a, x_b

points de contact

$$d = \max(0, (x_b - x_a) \cdot \hat{n}) \quad \text{profondeur de pénétration}$$

k_c

coefficient de rigidité

\hat{n}

normale de contact

Les collisions sont gérées en introduisant une énergie de collision quadratique par sommet, basée sur la profondeur de pénétration.

$$E_c(x) = -\frac{1}{2}k_c d^2$$

Friction

Soient $\delta x_c = (x_a - x_a^t) - (x_b - x_b^t)$ mouvement relatif au point de contact

où x_a^t x_b^t positions des corps au début du pas de temps

$\lambda_{c,i} = \frac{\partial E_c}{\partial x_i} \cdot \hat{n}$ magnitude signée de la force engendrée par la collision c appliquée sur le sommet i

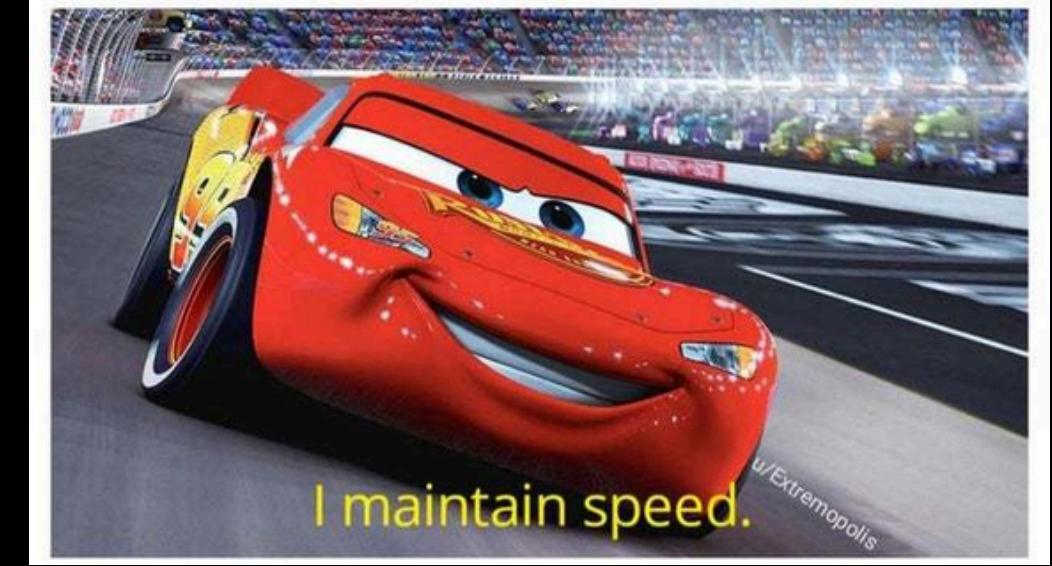
Soit u_c coefficient de friction

La force de friction au sommet i engendrée par la collision c

et le terme hessien de friction

sont ajoutés à f_i et H_i respectivement.

Friction was discovered in 1493
Moving objects before 1493:



Initialization

Tout solveur itératif demande une approximation initiale.

VBD n'oblige pas l'utilisation d'une initialization particulière.

- a) position précédente
- b) inertie
- c) inertie et accélération
- d) adaptive

$$x = x^t$$

$$x = x^t + hv^t$$

$$x = x^t + hv^t + h^2 a_{ext}$$

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

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

où $\tilde{a} = \begin{cases} 1 & \text{si } a_{ext}^t > \|a_{ext}\| \\ 0 & \text{si } a_{ext}^t < 0 \\ \frac{a_{ext}^t}{\|a_{ext}\|} & \text{sinon} \end{cases}$

avec $a_{ext}^t = a^t \cdot \hat{a}_{ext}$
 $a^t = (v^t - v^{t-1})$

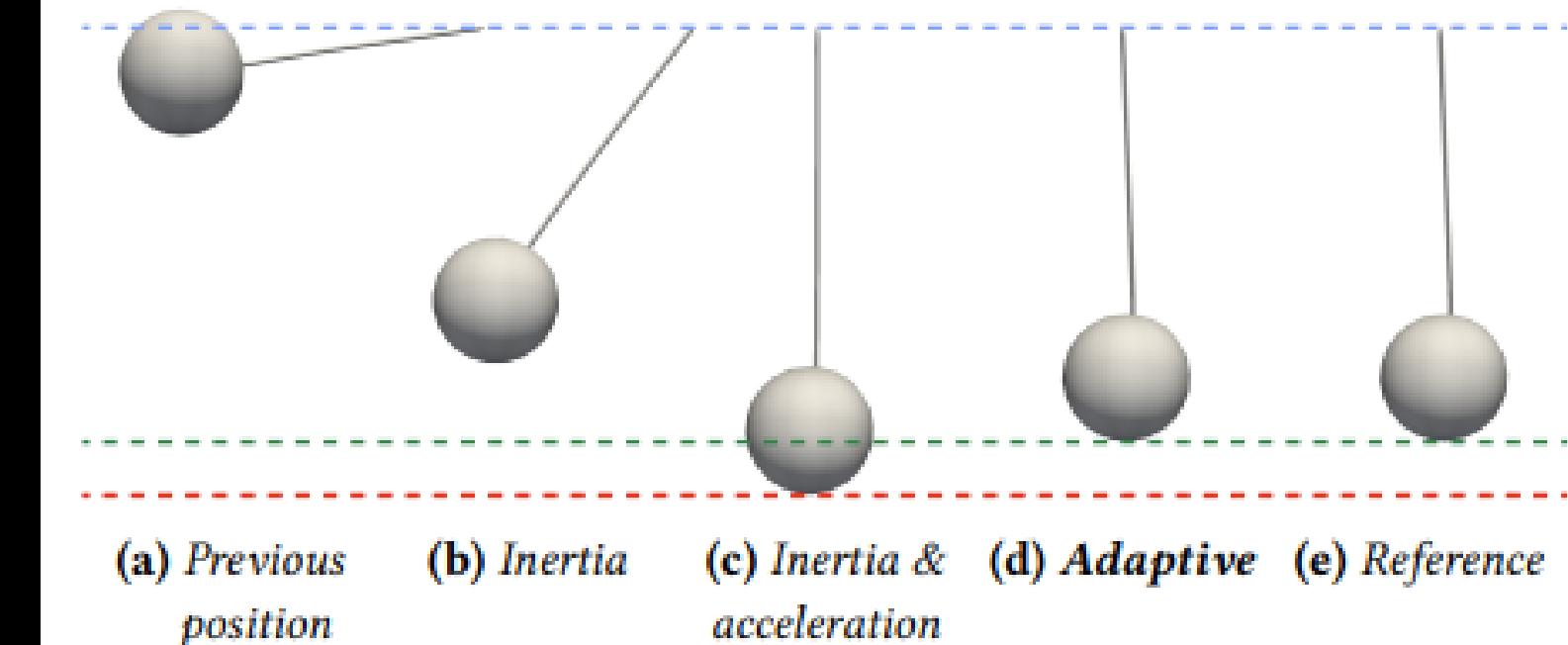


Fig. 5. Different initialization options for a swinging elastic pendulum dropped from the same height (blue line) simulated with our method using only 20 iterations per frame, showing the same frame of the simulation. Notice that initializing using (a) previous position and (b) inertia fail to properly move under gravity, while (c) inertia and acceleration leads to excessive stretching (red line) when VBD does not run to convergence. (d) Our adaptive solution closely matches (e) the reference generated by fully converged Newton's method (green line).

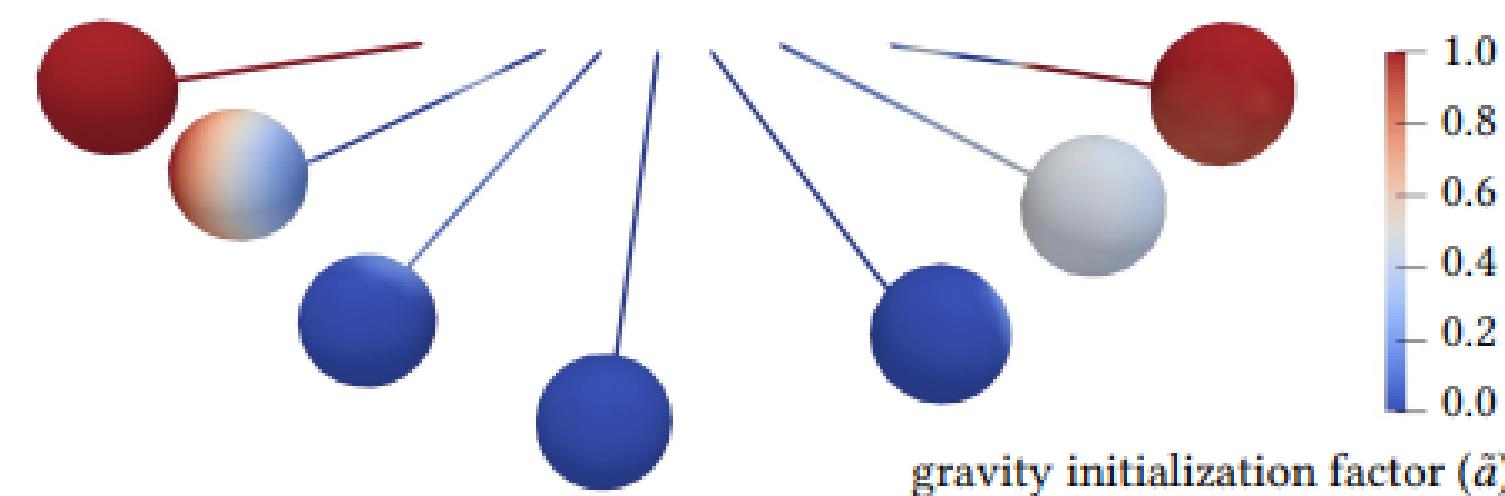


Fig. 6. The ratio of gravity \tilde{a} used with adaptive initialization during the swinging of an elastic pendulum. The model is a single-piece tetrahedral mesh. It automatically distinguishes vertices in approximate free-fall (red) and those where elasticity counteracts gravity (blue).

Approche semi-itérative de Chebyshev

Utilisée pour améliorer et accélérer la convergence de la méthode de Vertex Block Descent.

Calcule de manière itérative un rapport d'accélération basé sur l'approximation du rayon spectral du système.

Recalcule les positions à la fin de l'itération de Gauss-Seidel:

$$x^{(n)} = \omega_n (\bar{x}^{(n)} - x^{(n-2)}) + x^{(n-2)}$$

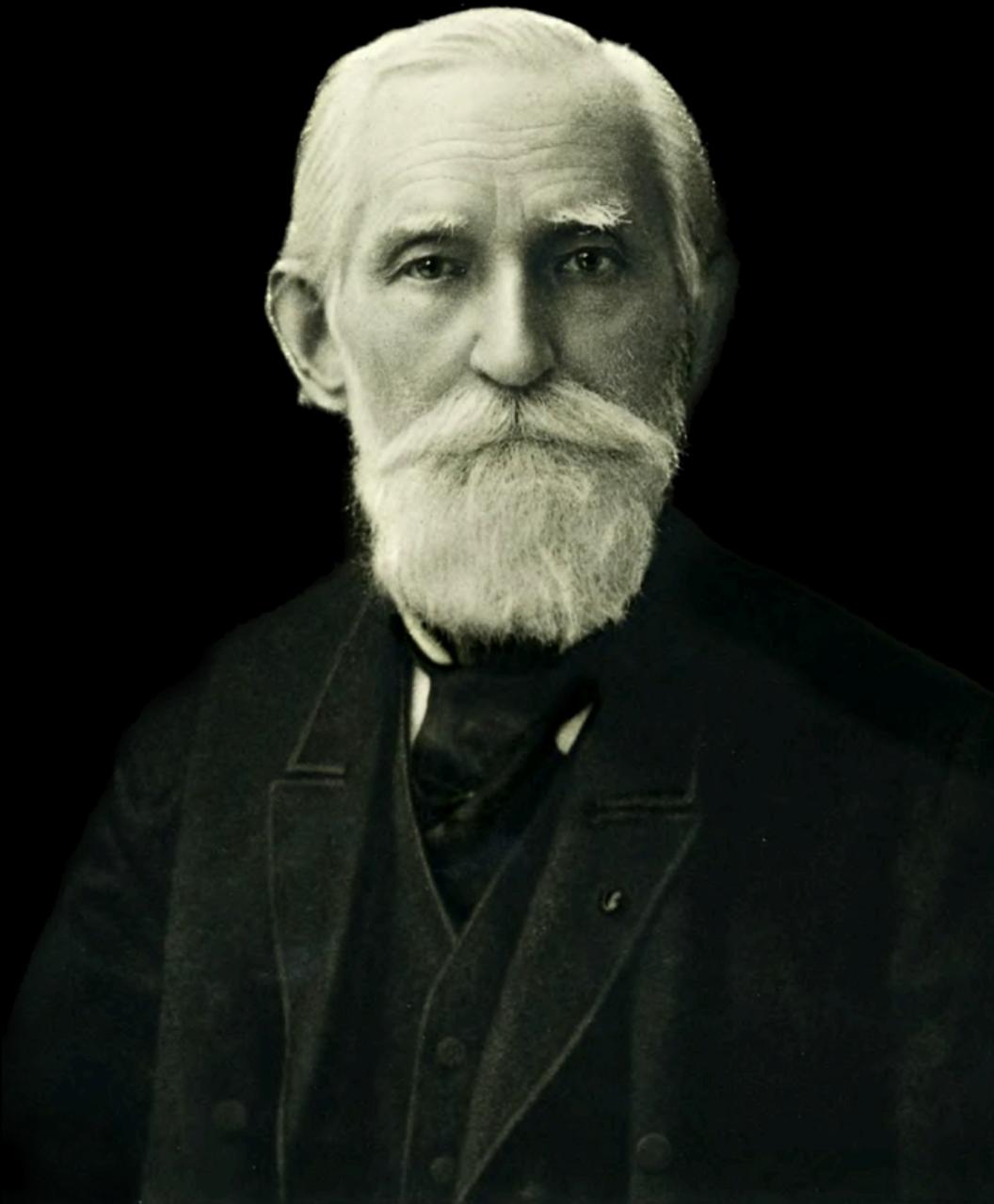
où ω_n rapport d'accélération (momentum) qui a changé à l'itération n

avec $\omega_1 = 1$

$$\omega_2 = \frac{2}{2-\rho^2}$$

$$\omega_n = \frac{4}{4-\rho^2 \omega_{n-1}} \quad \text{où } \rho \in (0, 1) \quad \text{rayon spectral du système}$$

avec $\rho(A) = \max\{|\lambda_1|, \dots, |\lambda_n|\}$

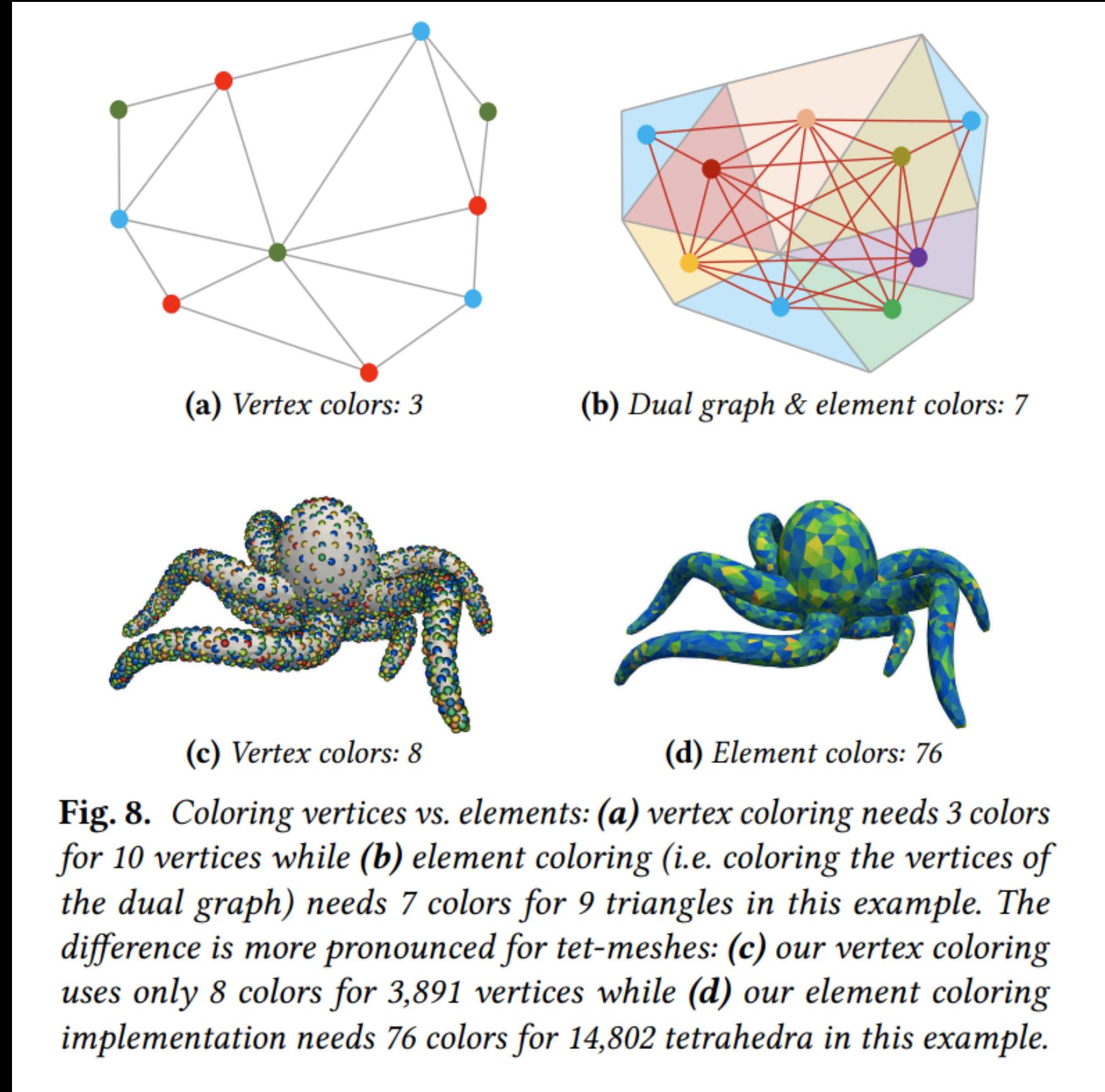


Parallélisation

VBD est parallélisé en colorant le graphe de sommets original, ce qui conduit à un niveau de parallélisme beaucoup plus élevé.

Le nombre de couleurs dans le graphe est ainsi minimisé.

La suppression d'éléments de force ne nécessite aucune modification de la coloration des sommets.



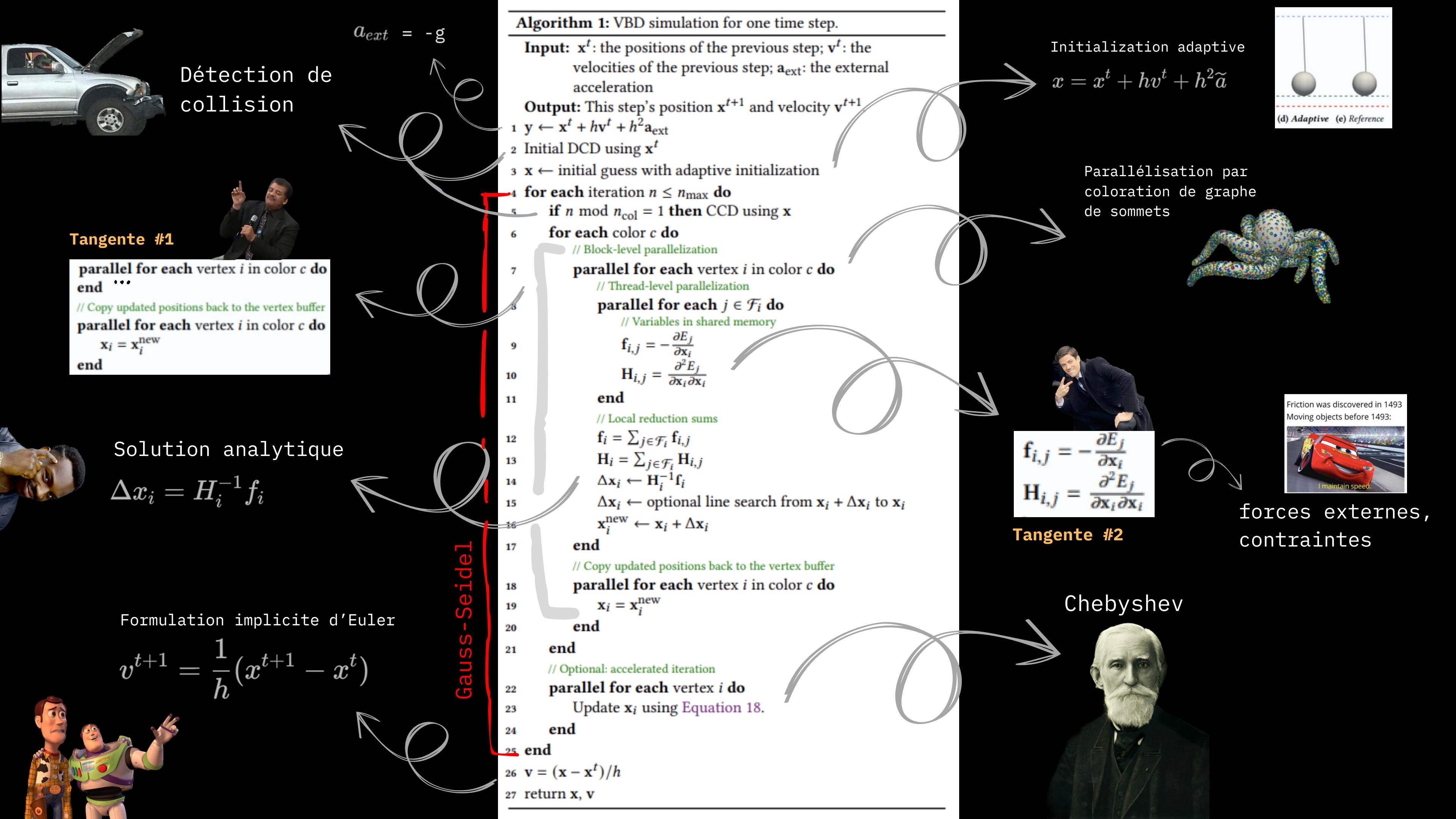
Algorithm 1: VBD simulation for one time step.

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

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

```
1  $\mathbf{y} \leftarrow \mathbf{x}^t + h\mathbf{v}^t + h^2\mathbf{a}_{\text{ext}}$ 
2 Initial DCD using  $\mathbf{x}^t$ 
3  $\mathbf{x} \leftarrow$  initial guess with adaptive initialization
4 for each iteration  $n \leq n_{\max}$  do
5   if  $n \bmod n_{\text{col}} = 1$  then CCD using  $\mathbf{x}$ 
6   for each color  $c$  do
7     // Block-level parallelization
8     parallel for each vertex  $i$  in color  $c$  do
9       // Thread-level parallelization
10      parallel for each  $j \in \mathcal{F}_i$  do
11        // Variables in shared memory
12         $\mathbf{f}_{i,j} = -\frac{\partial E_j}{\partial \mathbf{x}_i}$ 
13         $\mathbf{H}_{i,j} = \frac{\partial^2 E_j}{\partial \mathbf{x}_i \partial \mathbf{x}_i}$ 
14      end
15      // Local reduction sums
16       $\mathbf{f}_i = \sum_{j \in \mathcal{F}_i} \mathbf{f}_{i,j}$ 
17       $\mathbf{H}_i = \sum_{j \in \mathcal{F}_i} \mathbf{H}_{i,j}$ 
18       $\Delta \mathbf{x}_i \leftarrow \mathbf{H}_i^{-1} \mathbf{f}_i$ 
19       $\Delta \mathbf{x}_i \leftarrow$  optional line search from  $\mathbf{x}_i + \Delta \mathbf{x}_i$  to  $\mathbf{x}_i$ 
20       $\mathbf{x}_i^{\text{new}} \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$ 
21    end
22    // Copy updated positions back to the vertex buffer
23    parallel for each vertex  $i$  in color  $c$  do
24       $\mathbf{x}_i = \mathbf{x}_i^{\text{new}}$ 
25    end
26  end
27  // Optional: accelerated iteration
28  parallel for each vertex  $i$  do
29    Update  $\mathbf{x}_i$  using Equation 18.
30  end
31 end
32  $\mathbf{v} = (\mathbf{x} - \mathbf{x}^t)/h$ 
33 return  $\mathbf{x}, \mathbf{v}$ 
```

Assemblons le tout.



Différences avec les méthodes standard

Comme PBD et XPBD, VBD travaille avec les mises-à-jour de positions, mais utilise directement les formulations de force, sans les convertir en contraintes.

Les approximations dans la formulation de XPBD introduisent des erreurs qui font diverger le résultat de la solution d'Euler implicite

Résilient aux grands pas de temps, aux nombres d'itérations limitées et aux rapports de masses élevés.

Contrairement à XPBD, VBD ne dicte pas une initialization du système particulière.

Parallélisé à partir du graphe de sommets original, ce qui conduit à un nombre de couleur plus petit et à un parallélisme supérieur.

Résultats



Fig. 11. Simulation of 216 squishy balls with tentacles, a total of 48 million vertices and 151 million tetrahedra, dropped into a Utah teapot, forming a stable pile with active frictional contacts. The average and maximum computation time per time step is 3.6 and 3.9 seconds, respectively, using $S = 4$ substeps per frame and $n_{\max} = 40$ iterations per step. The final frame of this simulation is shown in Figure 1.

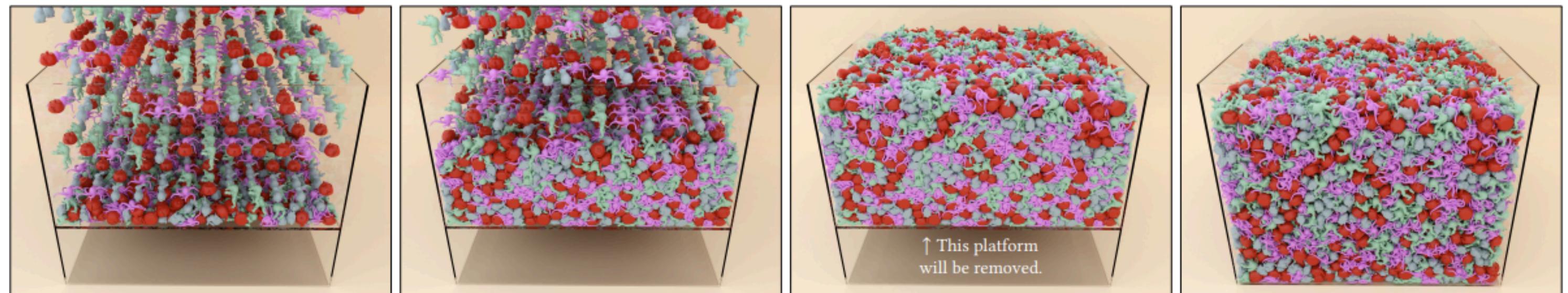


Fig. 12. Simulation of 10,368 deformable objects, totaling over 36 million vertices and 124 million tetrahedra, dropped onto a platform inside a box container. Then, the platform is suddenly removed and the objects collectively fall onto the ground, forming stable piles both before and after the platform is removed. The average and maximum computation times per time step are 4.2 and 4.7 seconds, respectively, using $S = 2$ substeps and $n_{\max} = 60$ iterations per step. The final frame of this simulation is shown in Figure 1.

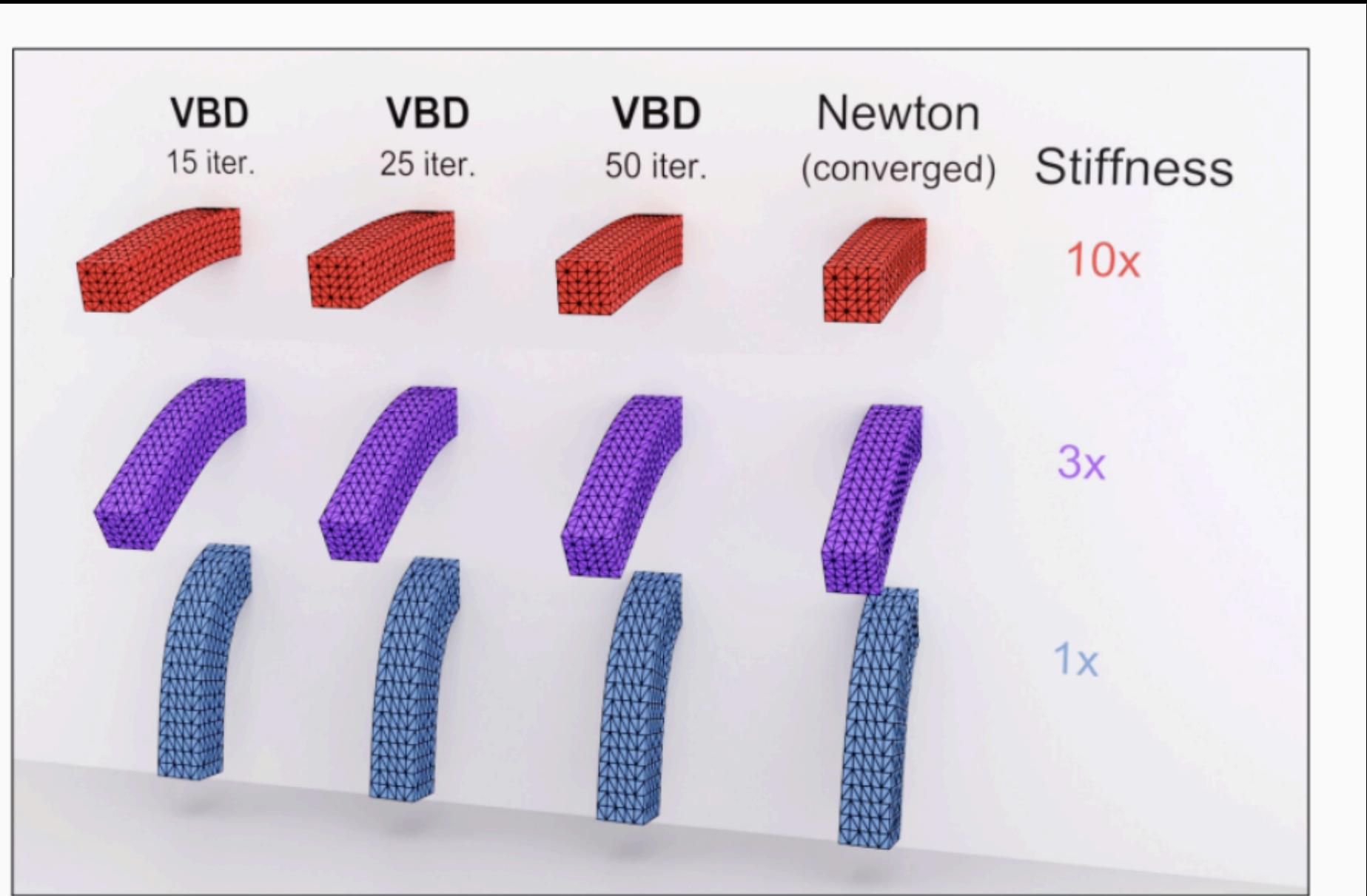


Fig. 13. Visual convergence with different numbers of iterations per frame for different material stiffness (with accelerated iterations using $\rho = 0.75, 0.86, 0.93$ top to bottom), simulating a beam with 463 vertices and 1.5 thousand tetrahedra.

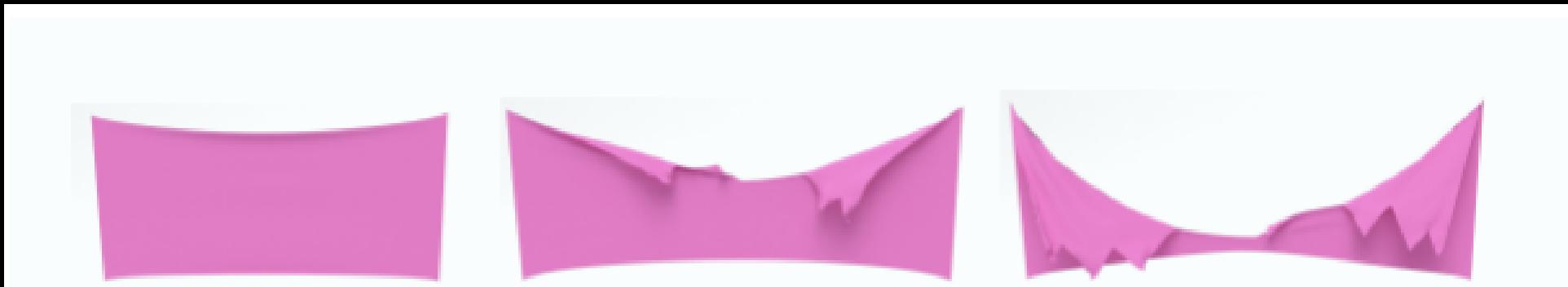


Fig. 10. Tearing a piece of cloth with 2500 vertices and 4800 triangles.

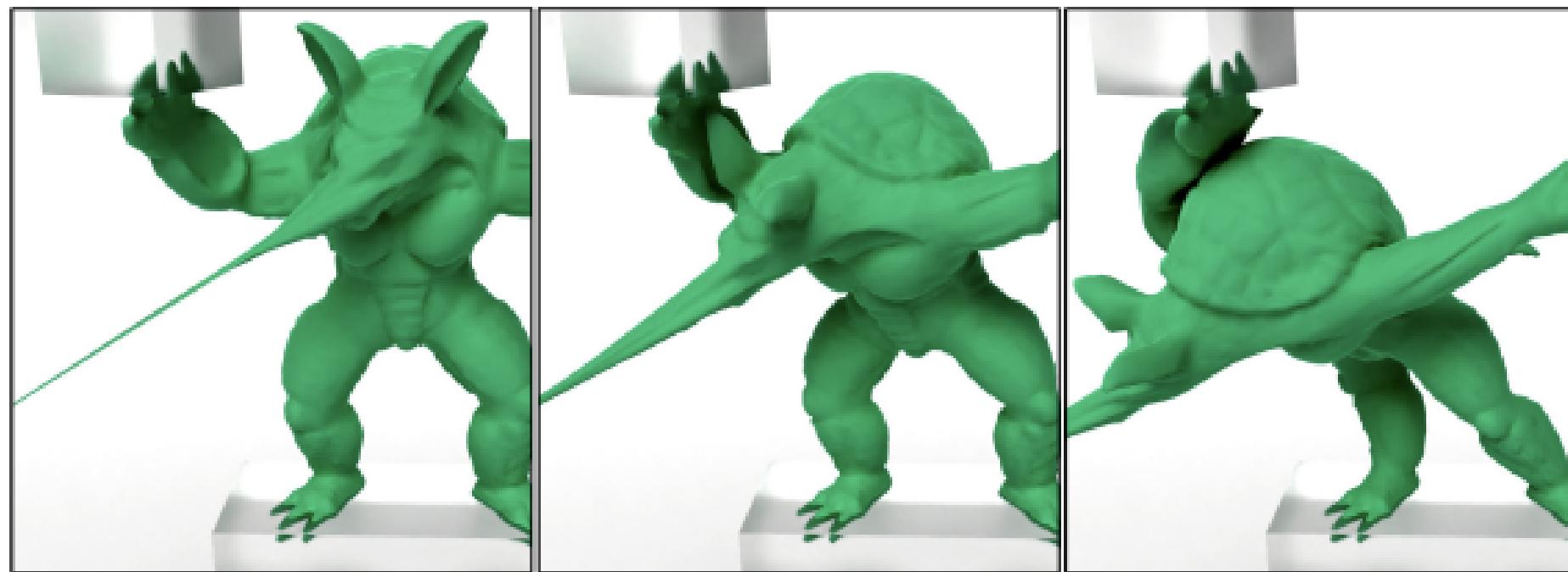


Fig. 16. A stress test using only a single iteration per frame (i.e. a time step of $h = 1/60$ seconds and $n_{\max} = 1$). One vertex on the armadillo model's nose is pulled while the finger and toe vertices are fixed. The model has 15 thousand vertices and 50 thousand tetrahedra.

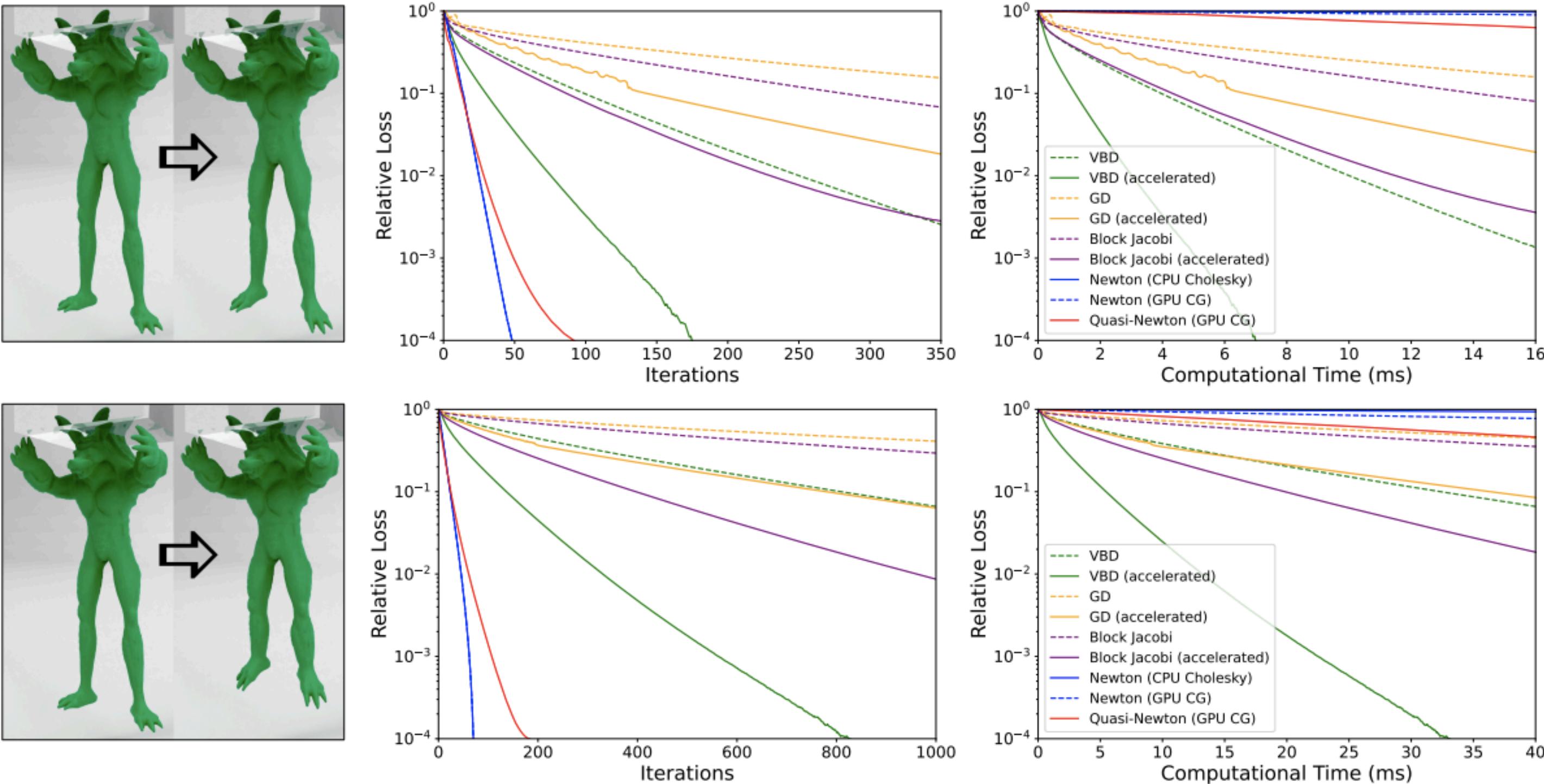


Fig. 17. Convergence of different descent methods for simulating an armadillo model with 15 thousand vertices and 50 thousand tetrahedra with (top) a relatively soft material and (bottom) a 10 \times stiffer material. Vertices near the top inside the glass block are fixed and the models are initially stretched, as shown on the left, by pulling down foot vertices. Then, the position constraints on foot vertices are suddenly removed, allowing the model to deform for 33 ms. The deformation is computed using a single time step of $h = 33$ ms. The graphs show relative loss over iterations and computation time. All methods are implemented on the GPU using the same framework with single precision (32-bit) floating-point numbers, except for Newton's method with Cholesky factorization, which runs on the CPU using double precision (64-bit). Accelerated versions use $\rho = 0.95$.

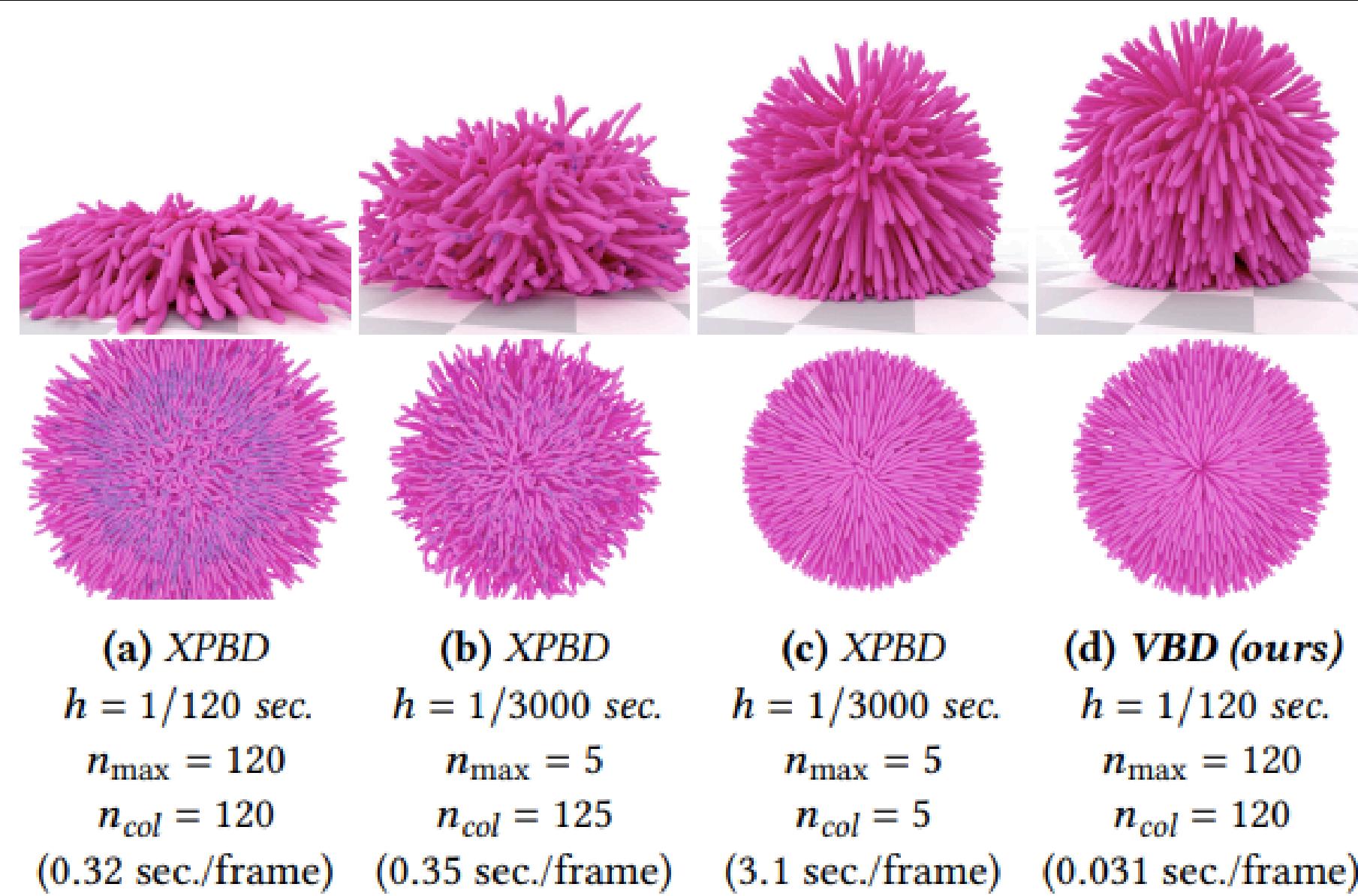


Fig. 19. A squishy ball with tentacles, comprising 230 thousand vertices and 700 thousand tetrahedra, dropped on the ground, simulated using (a) XPBD with a large time step and 240 iterations per frame, (b) XPBD with a 25× smaller time step and 250 total iterations per frame, (c) XPBD with the same small time step and iteration count but with 25× more frequent collision detection, and (d) VBD with a large time step and 240 iterations per frame. Comparing (a) and (d), VBD is faster than XPBD with the same settings. XPBD’s solution approaches VBD as the time step decreases, but it also requires more frequent collision detection to achieve a visually similar result to VBD.

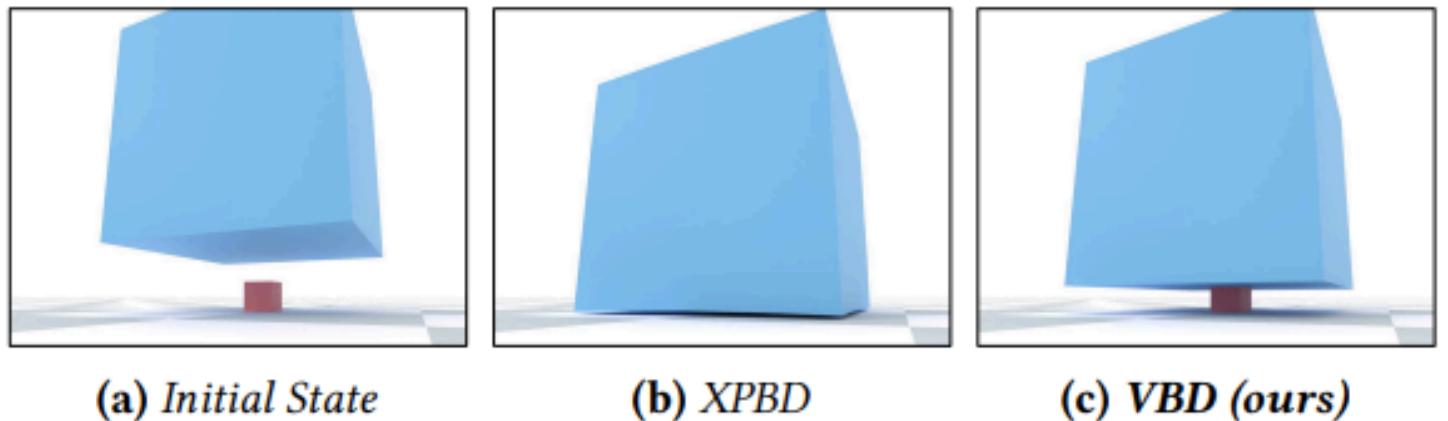


Fig. 20. Dropping a large and heavy elastic cube onto a smaller and much lighter box with a mass ratio of 1:2000. Each cube has 400 vertices and 1.5 thousand tetrahedra.

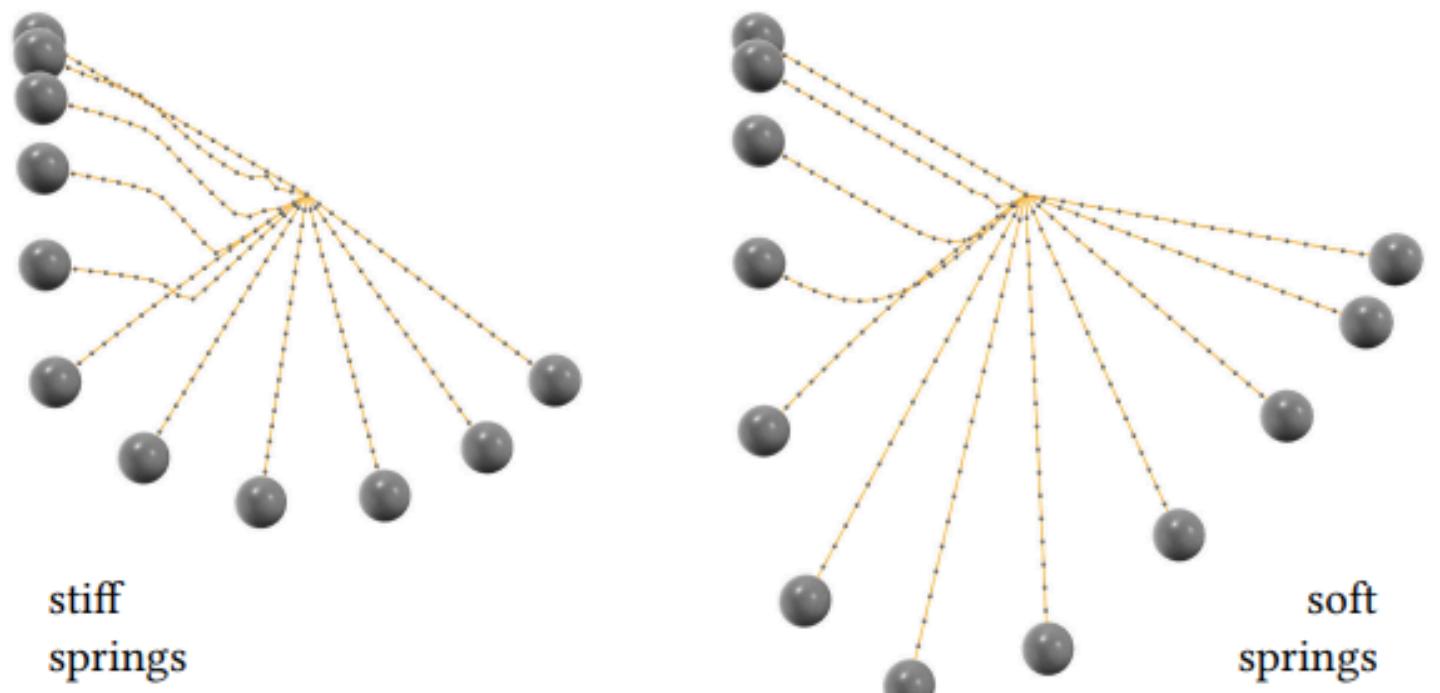


Fig. 21. 20 particles attached with springs, forming a swinging chain, simulated using VBD with a $S = 1$ substep and 100 iterations per step. The particle on one end of the chain is fixed and the particle on the other end has 1000× more mass than the others. (Left) using sufficiently stiff springs, they expand no more than 0.7% of their rest lengths, despite the substantial mass difference. (Right) using 100× less stiff springs, the chain undergoes a visible expansion as it swings.



Fig. 22. 5 rigid bodies, each with 6 DoF, forming a chain through collisions, simulated using our VBD formulation for rigid body dynamics.



Fig. 23. Dropping 60 rigid bodies into a Utah teapot, showcasing collisions and frictional contact. Remarkably, one rigid body stays on the spout due to friction.

Travaux futur

- Explorer les stratégies à pas de temps dynamique.
- Étude d'approches multi-résolution ou hiérarchiques afin d'améliorer encore les performances pour les simulations à très grande échelle.
- Développer des implémentations GPU spécialisées afin d'exploiter pleinement les architectures de calcul parallèle.

Projet Personnel

- Explorer comment VBD peut être utilisé pour simuler du *rubber*, par exemple, les pneus d'une voiture de course et la dégradation de ceux-ci.