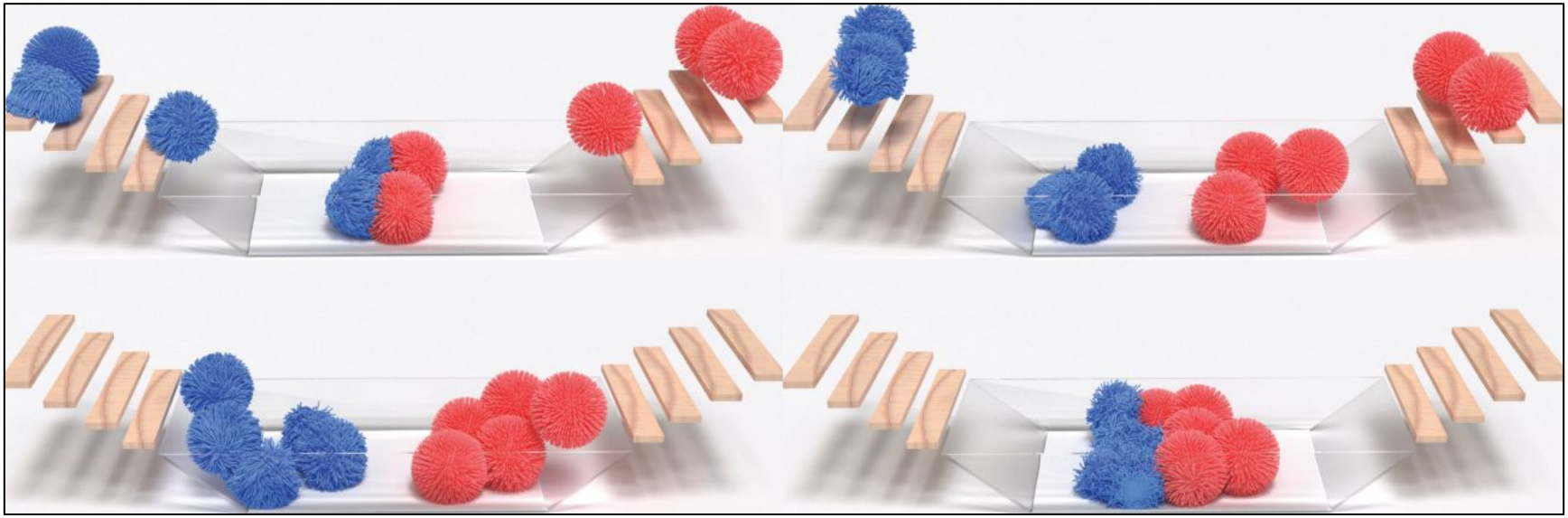
JGS2: Near Second-order Converging Jacobi/Gauss-Seidel for GPU Elastodynamics

Lei Lan, Zixua Lu, Chun Yuan, Weiwei Xu, Hao Su, Huamin Wang, Chenfanfu Jiang, Yin Yang
[2025]



Bastien Saillant
GDL 26/02/2026

Local solvers

Vertex Block Descent

ANKA HE CHEN, University of Utah, USA
ZIHENG LIU, University of Utah, USA
YIN YANG, University of Utah, USA
CEM YUKSEL, University of Utah & Roblox, USA

Fig. 1. *Example simulation results using our solver, both of those methods involve more than 100 million DoFs and 1 million active collisions.*

We introduce vertex block descent, a block coordinate descent solution for the variational form of implicit Euler through vertex-level Gauss-Seidel iterations. It operates with local vertex position updates that achieve reductions in global variational energy with maximized parallelism. This forms a physics solver that can achieve numerical convergence with unconditional stability and exceptional computation performance. It can also fit in a given computation budget by simply limiting the iteration count while maintaining its stability and superior convergence rate.

We present and evaluate our method in the context of elastic body dynamics, providing details of all essential components and showing that it outperforms alternative techniques. In addition, we discuss and show examples of how our method can be used for other simulation systems, including particle-based simulations and rigid bodies.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; *Collision detection.*

Additional Key Words and Phrases: physics-based simulation, elastic body, rigid body, time integration

1 INTRODUCTION

Physics-based simulation is the cornerstone of most graphics applications and the demands from simulation systems to deliver improved stability, accelerated computational performance, and enhanced visual realism are ever-growing. Particularly in real-time graphics applications, the stability and performance requirements are so strict that realism can sometimes be begrudgingly considered of secondary importance.

Notwithstanding the substantial amount of research and groundbreaking discoveries made on physics solvers over the past decades, existing methods still leave some things to be desired. They either deliver high-quality results, but fail to meet the computational demands of many applications or fit in a given computation time by sacrificing quality. Stability, on the other hand, is always a challenge, particularly with strict computation budgets.

In this paper, we introduce *vertex block descent (VBD)*, a physics solver that offers unconditional stability, superior computational performance than prior methods, and the ability to achieve numerical convergence to an implicit Euler integration. Though our method is a general solution that can be used for a variety of simulation problems, we present and evaluate it in the context of elastic body dynamics. Then, we briefly discuss how our method can be applied to some other simulation systems, including particle-based simulations and rigid bodies.

Our VBD method is based on block coordinate descent that performs vertex-based Gauss-Seidel iterations to solve the variational form of implicit Euler. For elastic body dynamics, each iteration runs a loop over the mesh vertices, adjusting the position of a single vertex at a time, temporarily fixing all others. This offers maximized parallelism when coupled with vertex-based mesh coloring, which can achieve an order of magnitude fewer colors (i.e. serialized workloads) as compared to element-based parallelization. Our local position-based updates can ensure that the variational energy is always reduced. Therefore, our method maintains its stability even with a single iteration per time step and large time steps, operating with unconverged solutions containing a large amount of residual. With more iterations, it converges faster than its alternatives. Thus, it can more easily fit in a given computation budget, while maintaining stability with improved convergence.

We present all essential components of using VBD for elastic body dynamics, including formulations for damping, constraints,

Authors' addresses: Anka He Chen, ankachan92@gmail.com, University of Utah, Salt Lake City, UT, USA; Ziheng Liu, NA, University of Utah, Salt Lake City, UT, USA; Yin Yang, NA, University of Utah, Salt Lake City, UT, USA; Cem Yuksel, cem@cemyuksel.com, University of Utah & Roblox, Salt Lake City, UT, USA.

Local energy

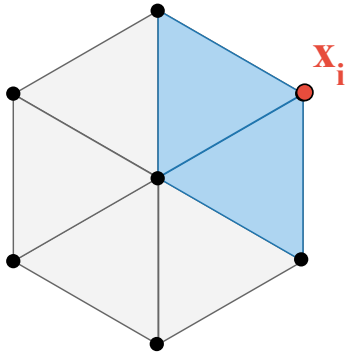
$$G_i(\mathbf{x}) = \frac{m_i}{2h^2}\|\mathbf{x}_i - \mathbf{y}_i\|^2 + \sum_{j\in\mathcal{F}_i} E_j(\mathbf{x})$$

Local Minimization

$$\mathbf{x}_i \leftarrow \underset{\mathbf{x}_i}{\operatorname{argmin}}\, G_i(\mathbf{x})$$

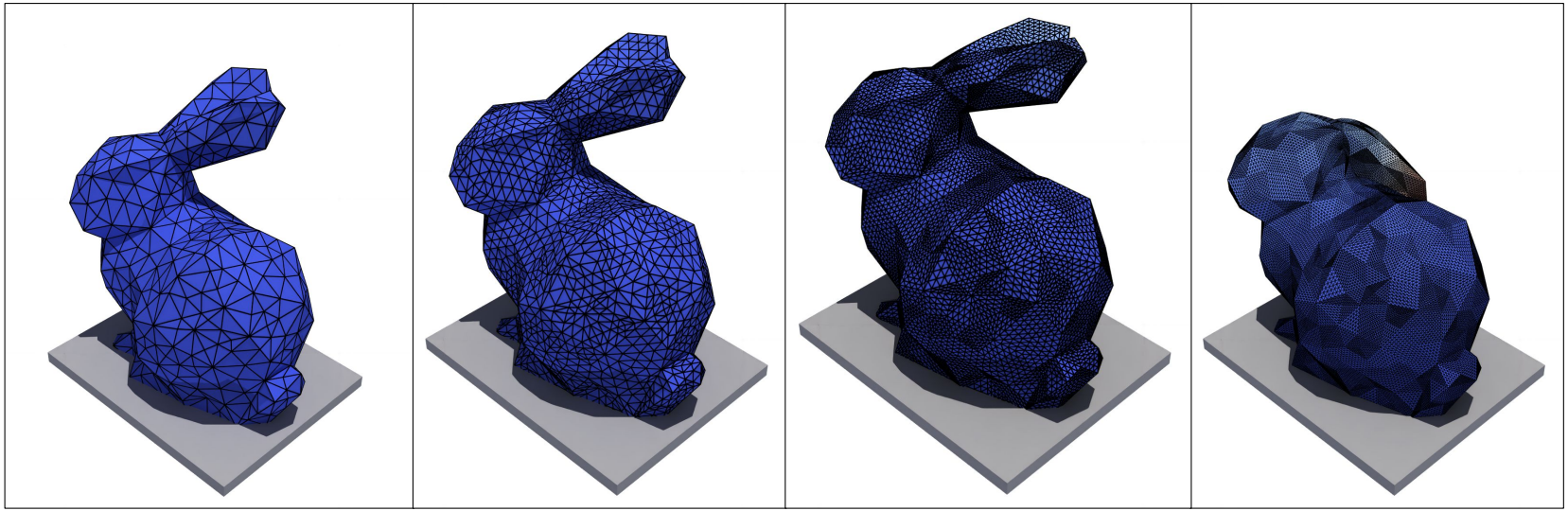
Local system solve

$$\Delta x_i = \underbrace{\left[\frac{\partial^2 G_i(x)}{\partial x_i^2}\right]}_{3\text{x}3}^{-1} \underbrace{\frac{\partial G_i(x)}{\partial x_i}}_{3\text{x}1}$$



- No system assembly
- GPU Friendly
- Simple
- Slow convergence on dense meshes and stiff materials

Why is it slow to converge ?



Local propagation of information

↕

Undershooting / Overshooting

Local resolution only reduce vertex energy
BUT
Can (and will) increase other vertices' energy

Unaware of global solution

How to avoid that ?

Be aware of the global solution duh

Global aware subspace

$$\Delta x = \phi_i(\Delta x_i)$$

subspace

New local minimization

$$\arg\max_{x_i} E_i(x_i) + E_{C_i}(\phi_i(x_i)), \quad E = E_{C_i} + E_i$$

local “global”

Global aware solving

$$\Delta x_i = -[H_i + g_{C_i}^T \frac{\partial^2 \phi_i}{\partial x_i^2} + H_{C_i}^T \frac{\partial \phi_i}{\partial x_i}]^{-1} [g_i + \frac{\partial \phi_i}{\partial x_i}^T g_{C_i}]$$

$$H_i = \frac{\partial^2 E_i}{\partial x_i^2}, \quad g_i = \frac{\partial E_i}{\partial x_i} \quad H_{C_i} = \frac{\partial^2 E_{C_i}}{\partial x \partial x_i}, \quad g_{C_i} = \frac{\partial E_{C_i}}{\partial x}$$

We need to define what is the subspace function

How to avoid that ?

$$\begin{bmatrix} H_{i,i} & H_{i,C_i} \\ H_{i,C_i}^T & H_{C_i,C_i} \end{bmatrix} \begin{bmatrix} \Delta x_i \\ \Delta x_{C_i} \end{bmatrix} = - \begin{bmatrix} g_i \\ g_{C_i} \end{bmatrix}$$

$$\begin{bmatrix} H_{i,i} & H_{i,C_i} \\ H_{i,C_i}^T & H_{C_i,C_i} \end{bmatrix} \begin{bmatrix} \Delta x_i \\ \Delta x_{C_i} \end{bmatrix} = - \begin{bmatrix} g_i \\ 0 \end{bmatrix}$$

$$\Delta x_{C_i} = -[H_{C_i,C_i} H_{i,C_i}^T]^{-1} \Delta x_i$$

$$\Delta x = \begin{bmatrix} \Delta x_i \\ -H_{C_i,C_i}^{-1} H_{i,C_i}^T \Delta x_i \end{bmatrix}$$

$$\Delta x = \begin{bmatrix} I \\ -H_{C_i,C_i}^{-1} H_{i,C_i}^T \end{bmatrix} \Delta x_i$$

$$\Delta x = \phi_i(\Delta x_i)$$

Global impact of local solution

$$U_i = -H_{C_i,C_i}^{-1} H_{i,C_i}^T$$

Like an interpolation function that describe how others particles are influenced by vertex

but

Slower than Newton

Need a big matrix inversion

Let's do some reduction

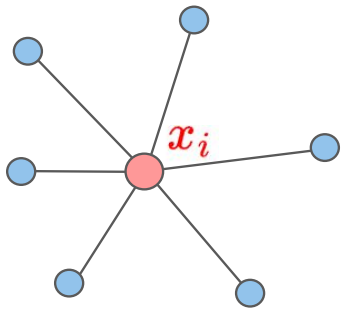
Co-Rotated Subspace

$$\tilde{\phi}_i^k = R^k \begin{bmatrix} I \\ -\bar{H}_{C_i,C_i}^{-1} \bar{H}_{i,C_i}^T \end{bmatrix} [R_i^k]^T \Delta x_i$$

Rest referential

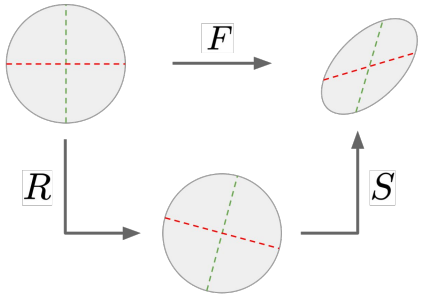
$$\tilde{\phi}_i^k = R^k \begin{bmatrix} I \\ -\bar{U}_i \end{bmatrix} [R_i^k]^T \Delta x_i$$

Local Deformation

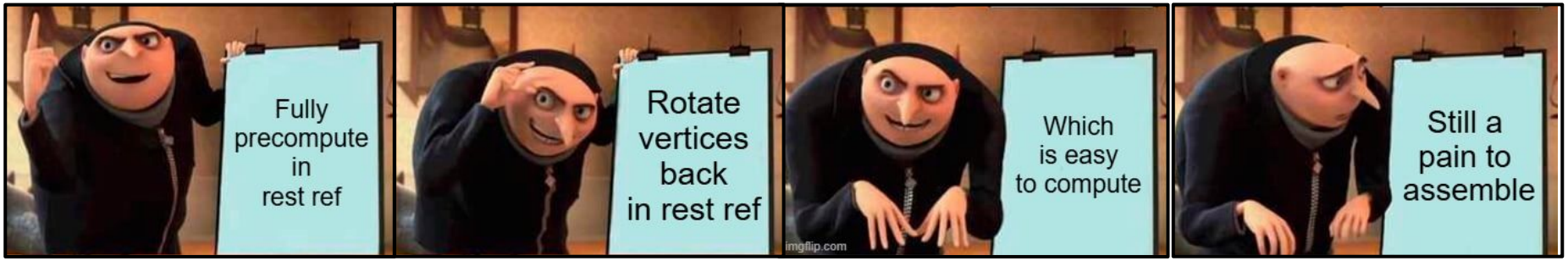


F_i

Polar Decomposition



$$F_i = R_i S_i$$



Very aggressive reduction

Cubature sampling

Optimizing Cubature for Efficient Integration of Subspace Deformations

Steven S. An Theodore Kim Doug L. James
Cornell University

Abstract

We propose an efficient scheme for evaluating nonlinear subspace forces (and Jacobians) associated with subspace deformations. The core problem we address is efficient integration of the subspace force density over the 3D spatial domain. Similar to Gaussian quadrature schemes that efficiently integrate functions that lie in particular polynomial subspaces, we propose cubature schemes (multi-dimensional quadrature) optimized for efficient integration of force densities associated with particular subspace deformations, particular materials, and particular geometric domains. We support generic subspace deformation kinematics, and nonlinear hyperelastic materials. For an r-dimensional deformation subspace with $O(r)$ cubature points, our method is able to evaluate subspace forces at $O(r^2)$ cost. We also describe composite cubature rules for runtime error estimation. Results are provided for various subspace deformation models, several hyperelastic materials (St.Venant-Kirchhoff, Mooney-Rivlin, Arruda-Boyce), and multi-modal (graphics, haptics, sound) applications. We show dramatically better efficiency than traditional Monte Carlo integration.

CR Categories: I.6.8 [Simulation and Modeling]: Types of Simulation—Animation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling G.1.4 [Mathematics of Computing]: Numerical Analysis—Quadrature and Numerical Differentiation

Keywords: Dimensional model reduction; reduced-order modeling; subspace integration; quadrature; subspace dynamics; dynamic deformations; nonlinear solid mechanics; real-time simulation

1 Introduction

Recently, dimensional model reduction has gained attention due to the difficulty of simulating detailed physical models at high rates for multimodal (graphics, haptics, sound) applications. Such reduced-order methods construct a small, r-dimensional subspace that captures the salient features of a much larger, N-dimensional model. If $r \ll N$, simulating these reduced-order models entirely in the r-dimensional subspace holds the promise of superior runtime performance provided costs independent of N are achieved.

Unfortunately, efficient evaluation of internal forces for subspace deformation models has proven difficult for arbitrary geometry and nonlinear materials. In particular, the inability of many models to support more complex materials, such as biological materials for surgical simulation, is unfortunate. Current force evaluation methods either have costs dependent on N [Krysl et al. 2001] or that scale poorly ($O(r^4)$) or are essentially restricted to particular materials (St. Venant-Kirchhoff) in practice [Barbič and James 2005], or are inaccurate.

In this paper, we present a general-purpose force evaluation method that applies to more general materials, subspace kinematics, and geometry, while delivering fast N-independent force evaluations at $O(r^2)$ cost (without exploiting sparsity). We achieve this scalability by performing a cubature optimization preprocess that enables fast runtime evaluation. Additionally, we provide evidence that our cubature schemes are computationally accurate and efficient, are resistant to over-fitting, and provide clear improvements over traditional Monte Carlo integration [Baraff and Witkin 1992].

Subspace Internal Forces: Our method can be applied to general subspace deformation models, but for concreteness of exposition we will focus on the case of dimensional model reduction for detailed finite element meshes [Krysl et al. 2001; Barbič and James 2005]. In a full FEM simulation with N degrees of freedom, the displacement vector would be of length N. However, in dimensional model reduction, the equations of motion have been projected into a linear, r-dimensional subspace of deformations. The reduced-order equations of motion describing a mesh deforming in subspace coordinates can be written as

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{f}(\mathbf{q}) = \mathbf{f}_{\text{ext}}, \tag{1}$$

where, $\mathbf{M} \in \mathbb{R}^{r \times r}$ is the (often constant) mass matrix, $\mathbf{q} \in \mathbb{R}^r$ is the generalized displacement vector of reduced coordinates, $\mathbf{f}(\mathbf{q}) \in \mathbb{R}^r$ is the subspace internal restoring force, $\mathbf{f}_{\text{ext}} \in \mathbb{R}^r$ are external forces, and the overdot denotes differentiation.

Unfortunately, the subspace internal force term $\mathbf{f}(\mathbf{q})$ in (1) is responsible for the poor $O(rN)$ and $O(r^4)$ scalings of previous methods [Krysl et al. 2001; Barbič and James 2005], so its efficient evaluation is the focus of this paper. The subspace force can be formulated in terms of a potential energy function, $E(\mathbf{q}) : \mathbb{R}^r \to \mathbb{R}$, given by the domain integral,

$$E(\mathbf{q}) = \int_\Omega \Psi(X;\mathbf{q})\, d\Omega_X, \tag{2}$$

where $\Psi(X;\mathbf{q})$ is the nonnegative strain energy density at material point X of the undeformed material domain Ω [Bonet and Wood 2008]. The subspace internal force is then the gradient of this energy, and is given by the vector integral

$$\mathbf{f}(\mathbf{q}) = -\nabla_{\mathbf{q}} E(\mathbf{q}) = -\int_\Omega \nabla_{\mathbf{q}} \Psi(X;\mathbf{q})\, d\Omega_X = \int_\Omega \mathbf{g}(X;\mathbf{q})\, d\Omega_X. \tag{3}$$

where we denote the "reduced-force density" integrand by

$$\mathbf{g} = \mathbf{g}(X;\mathbf{q}) = -\nabla_{\mathbf{q}} \Psi(X;\mathbf{q}) \in \mathbb{R}^r. \tag{4}$$

Our approach is to approximate $\mathbf{f}(\mathbf{q})$ using an n-point cubature (multi-dimensional quadrature) scheme,

$$\mathbf{f}(\mathbf{q}) = \int_\Omega \mathbf{g}(X;\mathbf{q})\, d\Omega_X \approx \sum_{i=1}^{n} w_i\, \mathbf{g}(X_i;\mathbf{q}). \tag{5}$$

We precompute estimates of the n positive cubature weights (w_i), and n cubature points (X_i) by minimizing $\mathbf{f}(\mathbf{q})$ integration error over

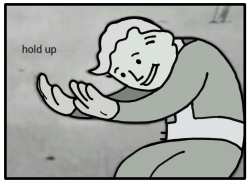
$$\Delta x_i^k = -[H_i^k + g_{C_i}^{k\,T} \frac{\partial^2 \tilde{\phi}_i^k}{\partial x_i^2} + H_{C_i}^T \frac{\partial \tilde{\phi}_i^k}{\partial x_i}]^{-1} [g_i^k + \frac{\partial \tilde{\phi}_i^k}{\partial x_i}^T g_{C_i}^k]$$

$$\frac{\partial \tilde{\phi}_i}{\partial x_i} = \begin{bmatrix} I \\ -\bar{U}_i \end{bmatrix}, \quad \frac{\partial^2 \tilde{\phi}_i}{\partial x_i^2} = 0$$

$$\Delta x_i^k = -[H_i^k + \tilde{H}_{C_i}^k]^{-1} [g_i^k + \tilde{g}_{C_i}^k] \quad \text{use all DoF}$$

$$\tilde{H}_{C_i}^k \approx \sum_{e \in S_i} w_e \tilde{H}_{C_i,e} \qquad \tilde{g}_{C_i}^k \approx \sum_{e \in S_i} w_e \tilde{g}_{C_i,e}^k \quad \text{sample DoF (Tetra ?)}$$

$$|S_i| \in [4, 8] \quad \text{few elements needed}$$



Computation time explodes with size

Authors provide opti (days ⇒ minutes)

Final formula

Initialisation

Cubature

$$|S_i| \in [4, 8] \qquad w_e$$

Precompute for each “cubature elements”

$$-\bar{H}^{-1}_{C_i,C_i} \qquad \bar{H}^{T}_{i,C_i}$$

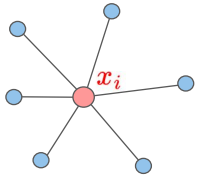
Update

Subspace

$$\tilde{\phi}_i^k = R^k \begin{bmatrix} I \\ \underbrace{-\bar{H}^{-1}_{C_i,C_i} \bar{H}^{T}_{i,C_i}}_{\text{Rest referential}} \end{bmatrix} [R_i^k]^T \Delta x_i$$

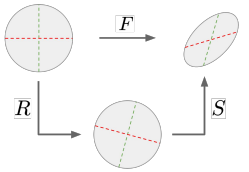
$$\tilde{\phi}_i^k = R^k \begin{bmatrix} I \\ -\bar{U}_i \end{bmatrix} [R_i^k]^T \Delta x_i$$

Local Deformation



F_i

Polar Decomposition



$F_i = R_i S_i$

Forces and Hessians

$$g_i^k + \tilde{g}_{C_i}^k \qquad H_i^k + \tilde{H}_{C_i}^k$$

Local Solve

$$\Delta x_i^k = -[H_i^k + \tilde{H}_{C_i}^k]^{-1} [g_i^k + \tilde{g}_{C_i}^k]$$

Results

Some numbers

Scene	# Element	# DOF	$\lvert\mathcal{S}_i\rvert$	h	$\lVert\Delta x\rVert$	Parallelism	Collision	# Iteration	Pre. time	Sim. time
Teaser (Fig. 1)	3.5M	4.4M	4	1/120	$1E-3$	GS	Penalty	55	37 min.	855 ms ($\infty\times$)
Falling Armadillos (Fig. 4)	6M	3.6M	4	1/150	$5E-4$	Jacobi	Penalty	58	52 min.	883 ms (78×)
House of cards (Fig. 5)	394K	372K	4	1/50	$1E-3$	Jacobi	IPC	23	1 min.	31 ms (120×)
Dragon (Fig. 6)	100K	80K	6	1/100	$1E-3$	Jacobi	Penalty	9	7 min.	7.3 ms (32×)
Letters soft (Fig. 7)	2.1M	1.7M	6	1/120	$5E-4$	GS	Penalty	27	37 min.	176 ms (43×)
Barbarian ships (Fig. 8)	2.5M	2.1M	4	1/120	$3E-4$	Jacobi	Penalty	34	45 min.	333 ms (153×)
Jack-o′-lanterns (Fig. 9)	6.7M	5.7M	4	1/120	$5E-4$	GS	Penalty	32	4 min.	753 ms (40×)
Squeezed puffer ball (Fig. 10)	1.3M	0.9M	6	1/150	$3E-4$	Jacobi	Penalty	69	67 min.	290 ms (173×)
Cactus (Fig. 11)	1.2M	1M	4	1/150	$1E-3$	Jacobi	IPC	40	18 min.	171 ms (82×)
Animal corossing (Fig. 12)	4.8M	4.5M	4	1/150	$1E-3$	GS	IPC	43	10 min.	684 ms (136×)
Cloth (Fig. 13)	2M	3M	4	1/120	$1E-3$	Jacobi	IPC	42	48 min.	469 ms (103×)

"Sim. time is the average simulation time for simulating one time step."

"Jacobi or GS. The difference between those two parallelization methods is negligible."

"[...] show acceleration rate our method offers compared with VBD."

Some numbers

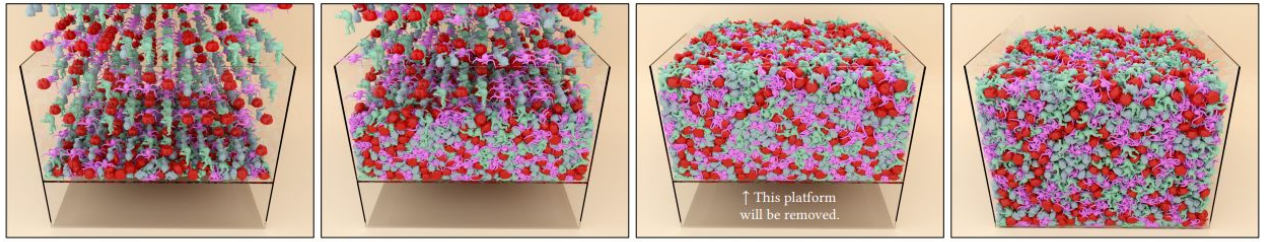
Animal corossing (Fig. 12)	4.8M	4.5M	4	1/150	$1E-3$	GS	IPC	43	10 min.	684 ms (136×)

JGS2

4.8M elements

684ms vs 93s (VBD)

Nvidia 3090 RTX GPU



VBD

124M elements

4.5 s

Nvidia 4090 RTX GPU

Hardware ?	Material ?	Precision ?
Code optimization ?	Experiment ?	VBD version ?

Video time

Conclusion

- Good perf + GPU
- Robust

- Much more complexe
- Pre-computation time

- Scenes with multiple objets
- No idea how it’s implemented

(je chipotte)