POSITION-BASED SIMULATION OF ELASTIC MODELS ON THE GPU WITH ENERGY AWARE GAUSS-SEIDEL ALGORITHM

Ozan Cetinaslan

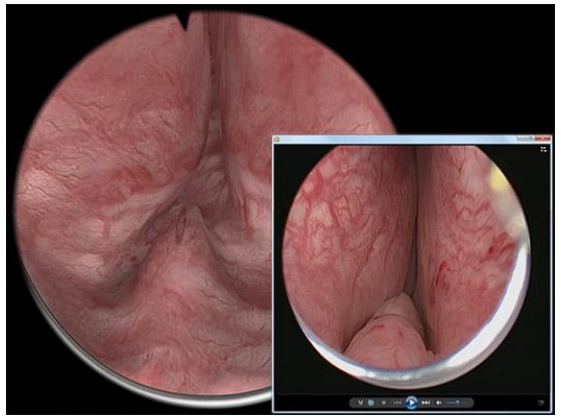
Instituto de Telecomunicações & Faculdade de Ciências, Universidade do Porto, Portugal

Introduction

Physics-based visual simulations have become ubiquitous in movies, cartoons, medical applications and computer games.

Collapsed Buildings, 2012 © DigitalDomain

Sulley from Monsters Inc, © Disney/Pixar



Medical App. © VirtaMed

Assassin's Creed, © Ubisoft

Introduction

Physics-based Simulations

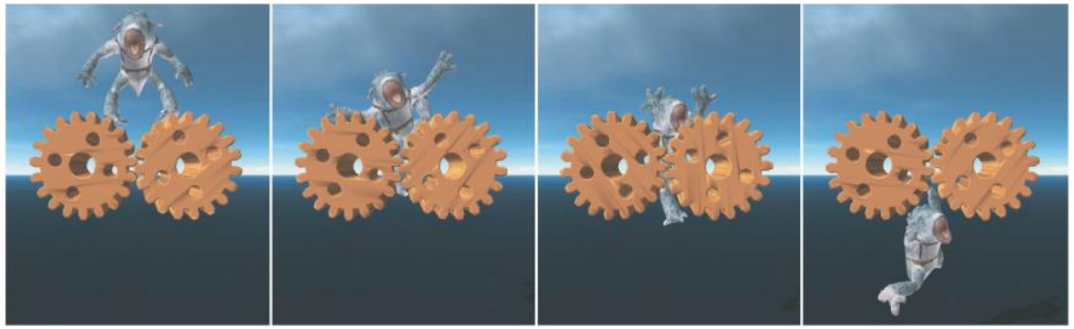
imitate the life-like motions!

Introduction

To achieve physics-based simulations, numerical methods are employed:

- Integration Methods
  - Implicit / Explicit Euler methods
  - Verlet method
  - Second Order Integration method
  - Runge–Kutta method
  - …
- Iteration Approaches
  - Newton’s method
  - Jacobi method
  - Gauss-Seidel method

Introduction



Muller, M., Heidelberger B., Hennix M., Ratcliff J.,

"Position Based Dynamics "

VRIPhys, 2006

Journal of Visual Communication and Image Representation, 2007

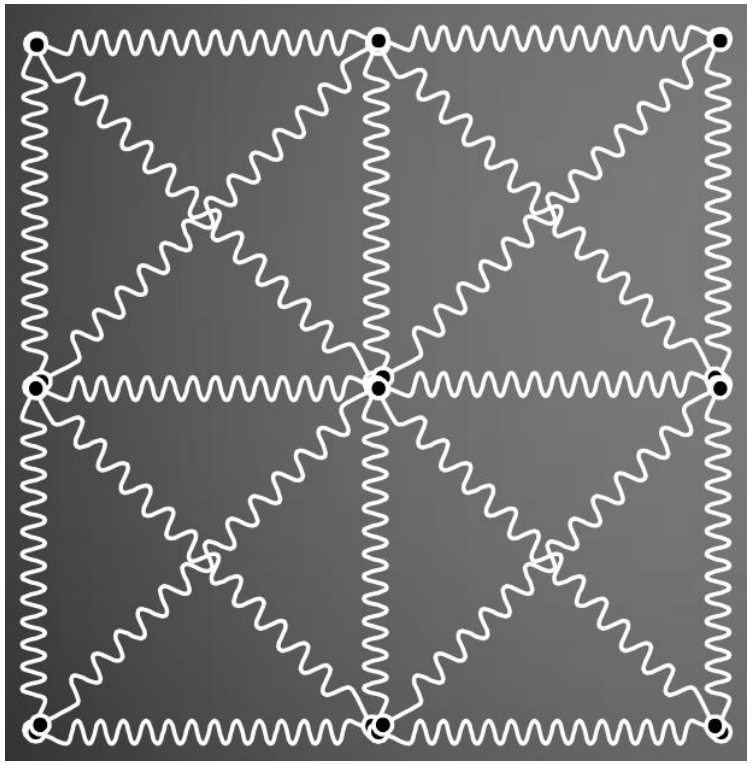
Introduction

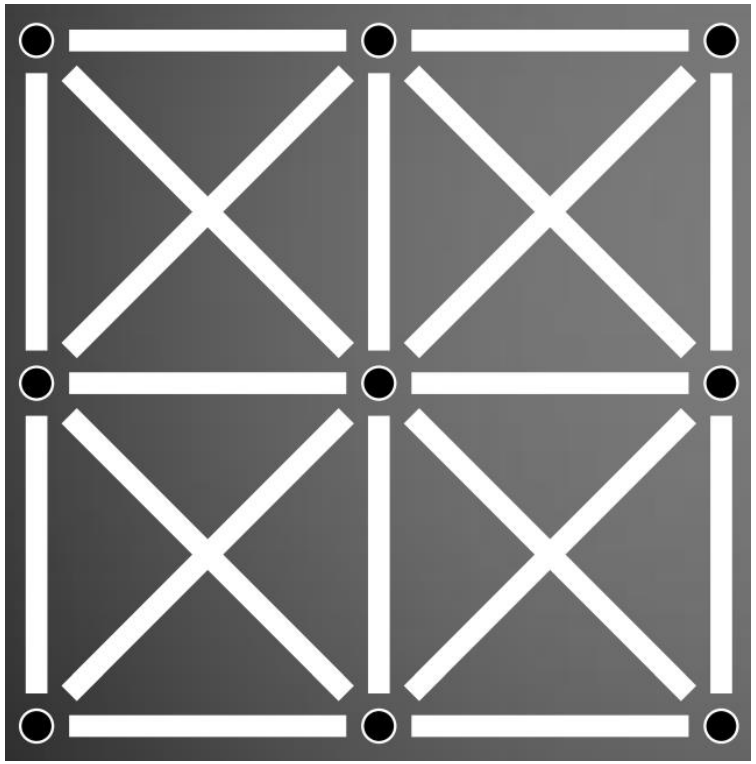
Position Based Dynamics (PBD):

- Integration Methods
  - Implicit / Explicit Euler methods
  - **Verlet method**
  - Second Order Integration method [Bender et al. 17]
  - Runge–Kutta method
  - …
- Iteration Methods
  - Newton’s method
  - Jacobi method [Macklin et al. 14]
  - **Gauss-Seidel method**

Introduction

The main idea is to consider the mass-spring networks as connected particles





[Stam 2009]

Introduction

In a very general perpective, PBD algorithm:

- First, shoots the particles with the integration method.
- After, pulls them back with the non-linear Gauss-Seidel iteration.

Introduction

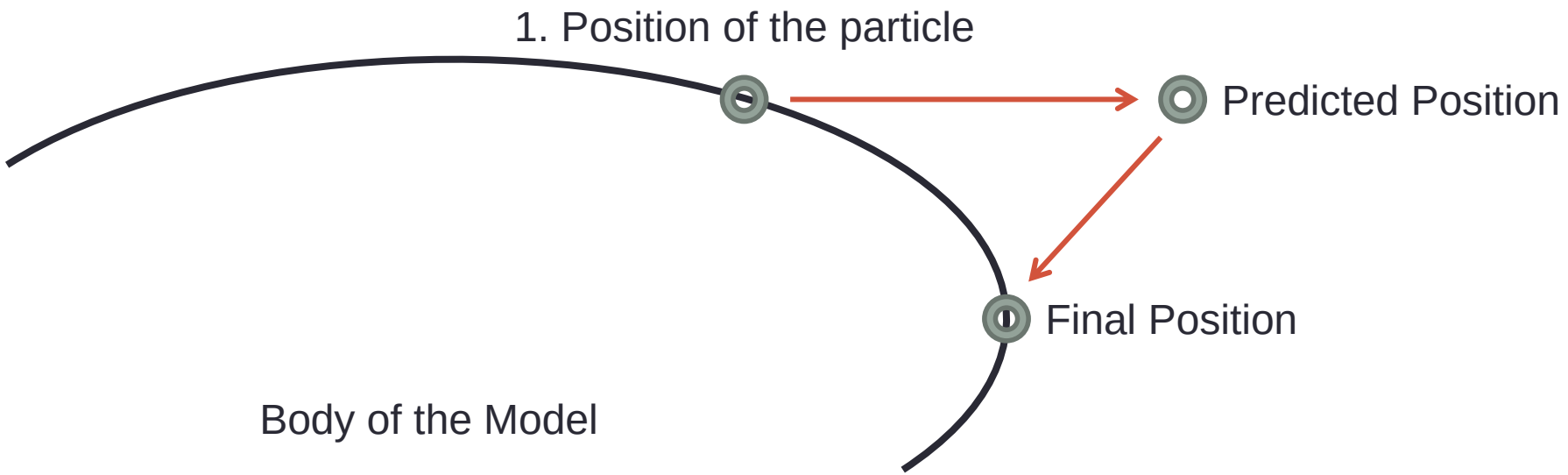
In a very general perpective, PBD algorithm:

- First, shoots the particles with the integration method.
- After, pulls them back with the non-linear Gauss-Seidel iteration.

Introduction

That implies:

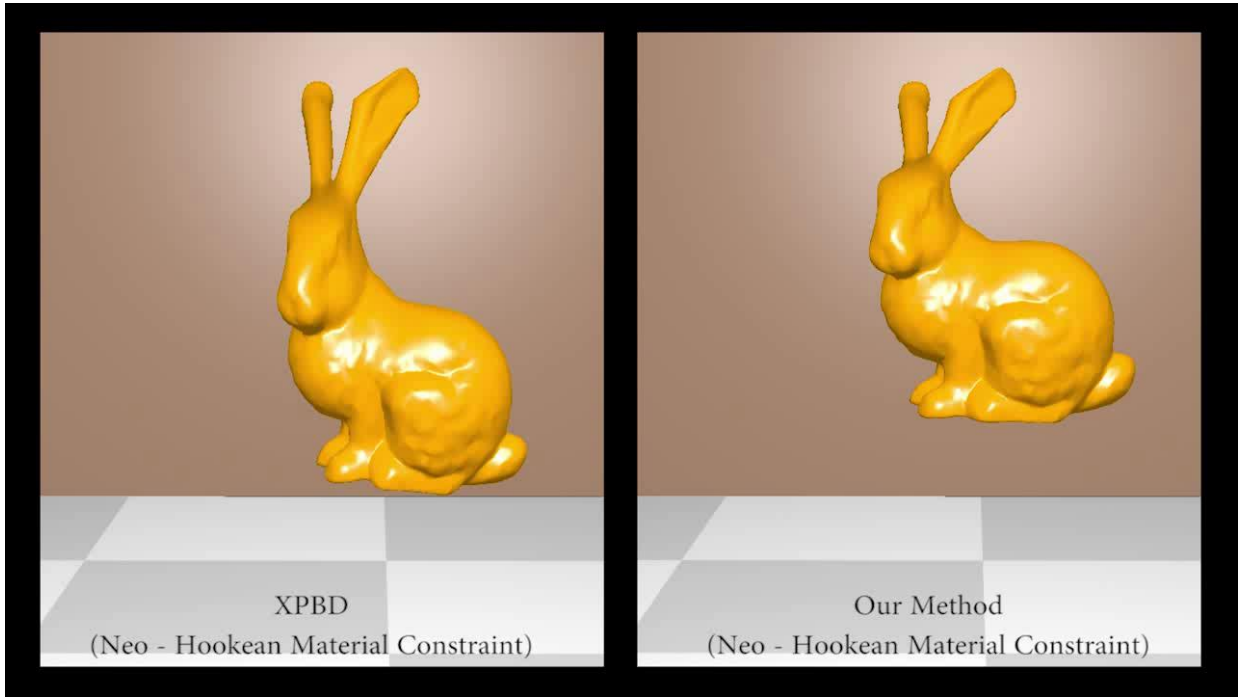
- Verlet integration predicts the position of a particle
- Gauss-Seidel loop iterates the particle till the final position is obtained
- Gauss-Seidel iteration operates based on projected constraints.



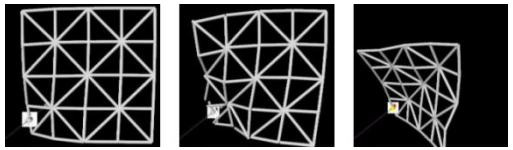
Motivation

Due to extreme dependency to **Gauss-Seidel**:

- Is it possible to improve the visual results within **Gauss-Seidel** on the GPU?



PBD in Literature



Faure, F.,
“Interactive Solid Animation Using Linearized Displacement Constraints”
Computer Animation and Simulation Workshop, 1998

Jakobsen, T.,
“Advanced Character Physics”
Proceedings of the Game Developers Conference, 2001

Stam J.,
"Nucleus: Towards a Unified Dynamics Solver for Computer Graphics"
Proceedings of Computer-Aided Design and Computer Graphics, 2009

PBD in Literature

Wrinkle Meshes

Matthias Müller & Nuttapong Chentanez

NVIDIA

Solid Simulation with Oriented Particles

Matthias Müller Nuttapong Chentanez
NVIDIA PhysX Research

Long Range Attachments - A Method to Simulate Inextensible Clothing in Computer Games

Tae-Yong Kim, Nuttapong Chentanez and Matthias Müller-Fischer†

NVIDIA PhysX Research

Fast Simulation of Inextensible Hair and Fur

M. Müller T.Y. Kim N. Chentanez

Nvidia PhysX Research

Position Based Fluids

Miles Macklin * Matthias Müller †

NVIDIA

Strain Based Dynamics

Matthias Müller Nuttapong Chentanez Tae-Yong Kim Miles Macklin

NVIDIA

Position-Based Simulation of Continuous Materials

Jan Bender[a], Dan Koschier[a], Patrick Charrier[a], Daniel Weber[b]

[a] *Graduate School CE, TU Darmstadt*
[b] *Fraunhofer IGD, Darmstadt*

Direct Position-Based Solver for Stiff Rods

Crispin Deul[1], Tassilo Kugelstadt[2], Marcel Weiler[1] and Jan Bender[2]

[1]Graduate School CE, TU Darmstadt
[2] RWTH Aachen University

General (X)PBD Algorithm

$x^{n+1} = x^n + v^n h + h^2 w_i f_{ext}$

initialize total Lagrange multiplier $\lambda_0 = 0$,

while k < Iterations do

    for all constraints do

        GAUSS-SEIDEL PROCESS

    end for

    k = k +1

end while

update x^{n+1}

update velocity $v^{n+1} = (x^{n+1} - x^n) / h$

Verlet Integration

$$\mathbf{x^{n+1} = x^n + v^n h + h^2 w_i f_{ext}}$$

initialize total Lagrange multiplier $\lambda_0 = 0$,

while k < Iterations do

    for all constraints do

        GAUSS-SEIDEL PROCESS

    end for

    k = k +1

end while

update x^{n+1}

update velocity $v^{n+1} = (x^{n+1} - x^n) / h$

Verlet Integration

$$\mathbf{x^{n+1} = x^n + (x^n - x^{n-1}) + h^2 w_i f_{ext}}$$

initialize total Lagrange multiplier $\lambda_0 = 0$,

while k < Iterations do

    for all constraints do

        GAUSS-SEIDEL PROCESS

    end for

    k = k +1

end while

update x^{n+1}

~~update velocity $v^{n+1} = (x^{n+1} - x^n) / h$~~

Gauss-Seidel Process

After, the displacements of each point is calculated with a constraint function:

$$(C(x + \Delta x) = 0)$$

With the help of 1st order Taylor-expansion:

$$(C(x + \Delta x) \approx C(x) + \nabla_x C(x) \cdot \Delta x = 0)$$

Position updates are obtained as:

$$\boxed{\Delta x_i = w_i \nabla_{x_i} C(x) \lambda_i} \quad \text{where} \quad \lambda_i = -\frac{C(x)}{\sum_i w_i |\nabla_{x_i} C(x)|^2}$$

Gauss-Seidel Process

In the recent extension to PBD (XPBD):

XPBD: Position-Based Simulation of Compliant Constrained Dynamics

Miles Macklin Matthias Müller Nuttapong Chentanez

NVIDIA

Position updates are associated with compliance stiffness matrix in order to prevent iteration count and step-size dependency:

$$\boxed{\Delta x_i = w_i \nabla_{x_i} C(x) \Delta\lambda_i}$$ where $$\Delta\lambda_i = -\frac{C(x) + \tilde{\alpha}\lambda_i}{\left(\sum_i w_i |\nabla_{x_i} C(x)|^2\right) + \tilde{\alpha}}$$

Gauss-Seidel Process

$x^{n+1} = x^n + (x^n - x^{n-1}) + h^2 w_i f_{ext}$

initialize total Lagrange multiplier $\lambda_0 = 0$,

while k < Iterations do

    for all constraints do

        compute $\Delta\lambda$ and Δx

        $\boldsymbol{\lambda_{k+1} = \lambda_k + \Delta\lambda}$

        $\boldsymbol{x_{k+1} = x_k + \Delta x}$

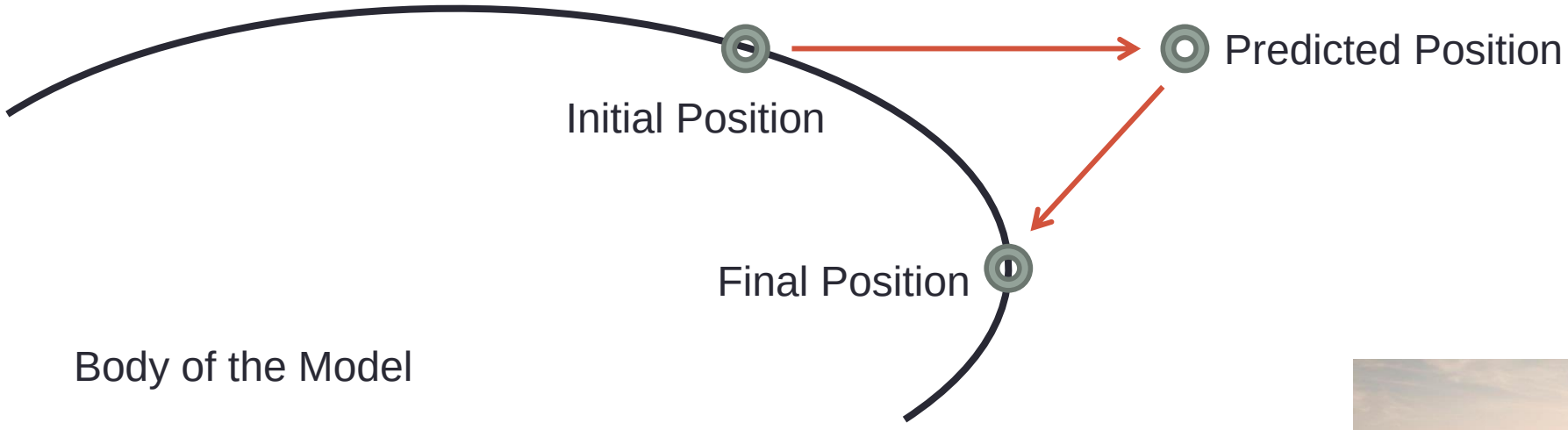
    end for

    k = k +1

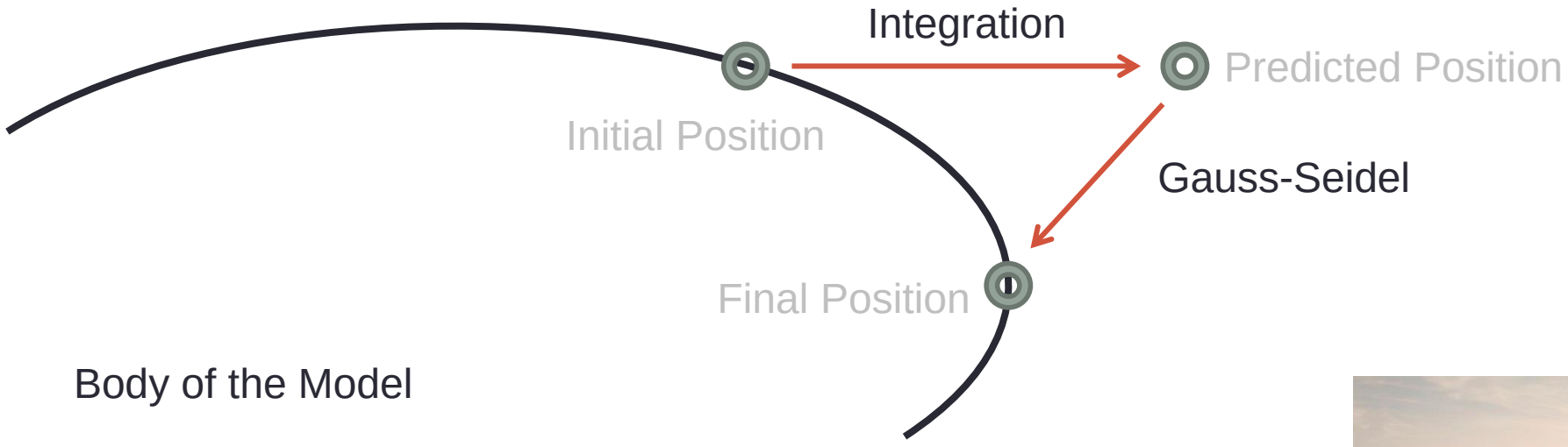
end while

update x^{n+1}

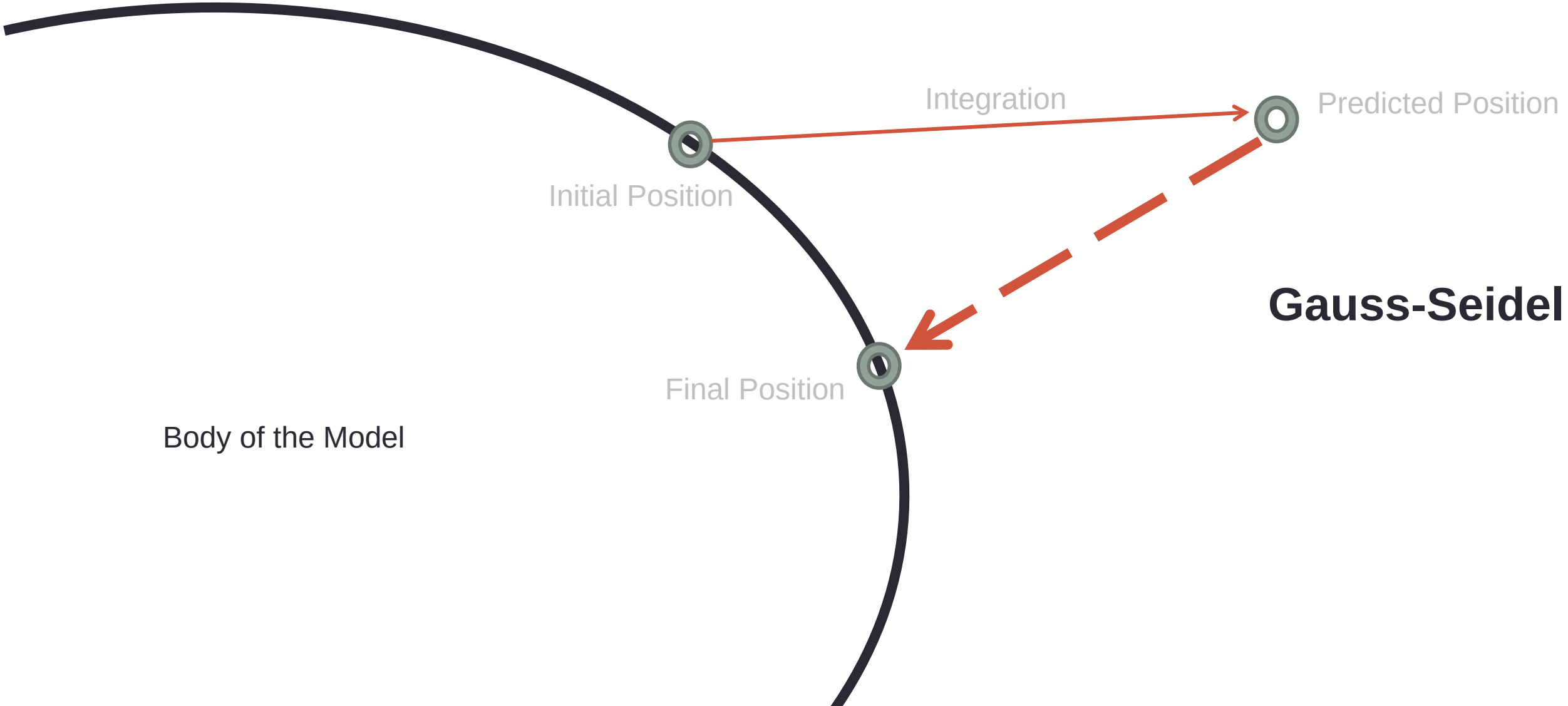
Energy Aware Gauss-Seidel Algorithm



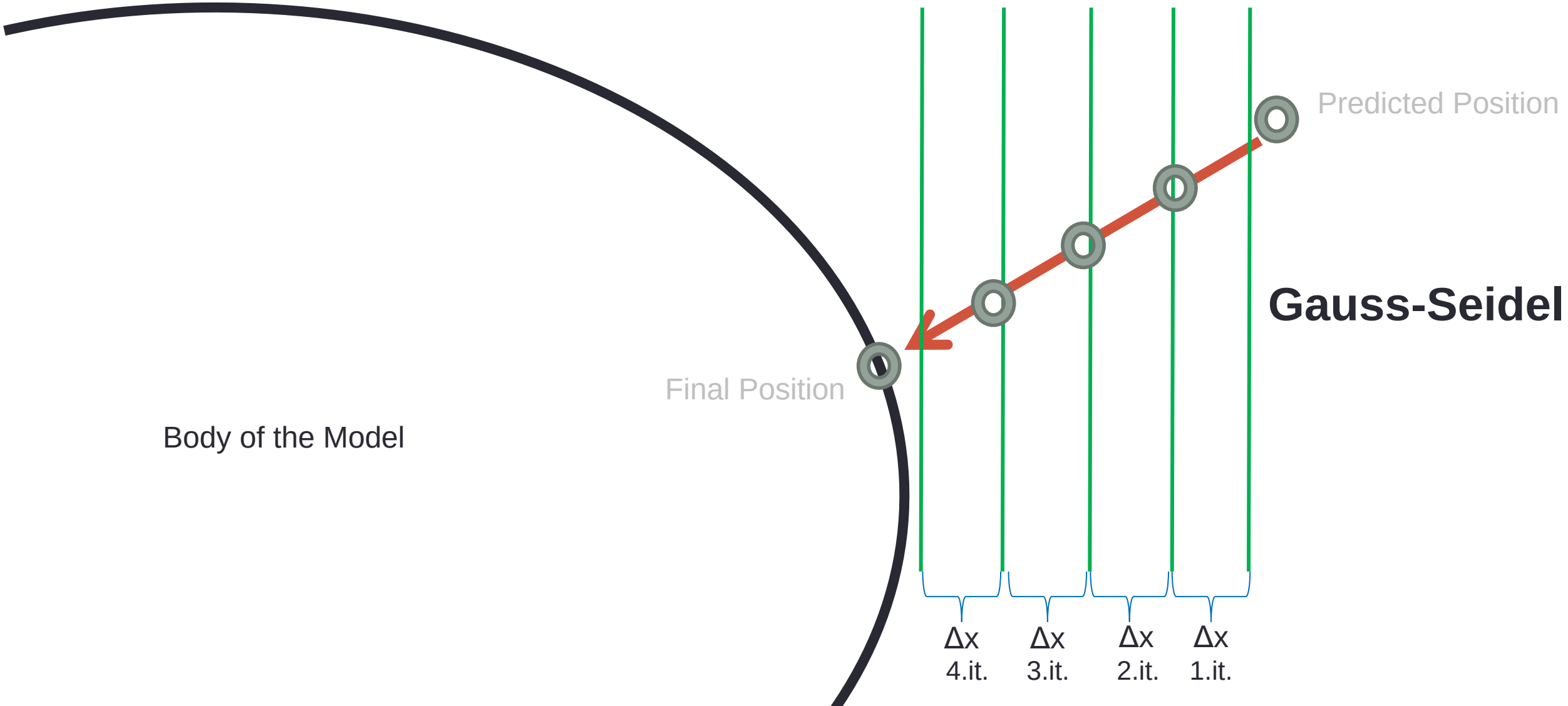
Energy Aware Gauss-Seidel Algorithm



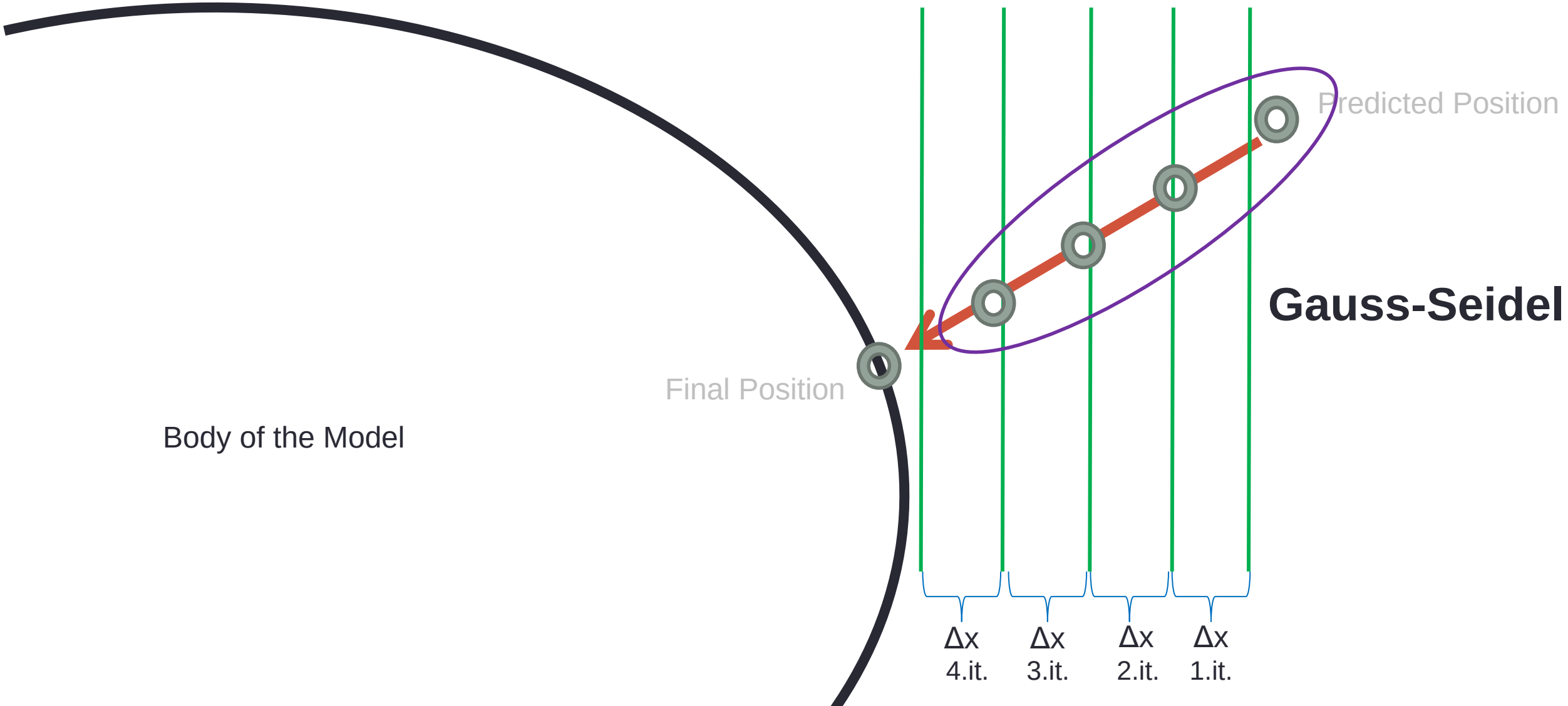
Energy Aware Gauss-Seidel Algorithm



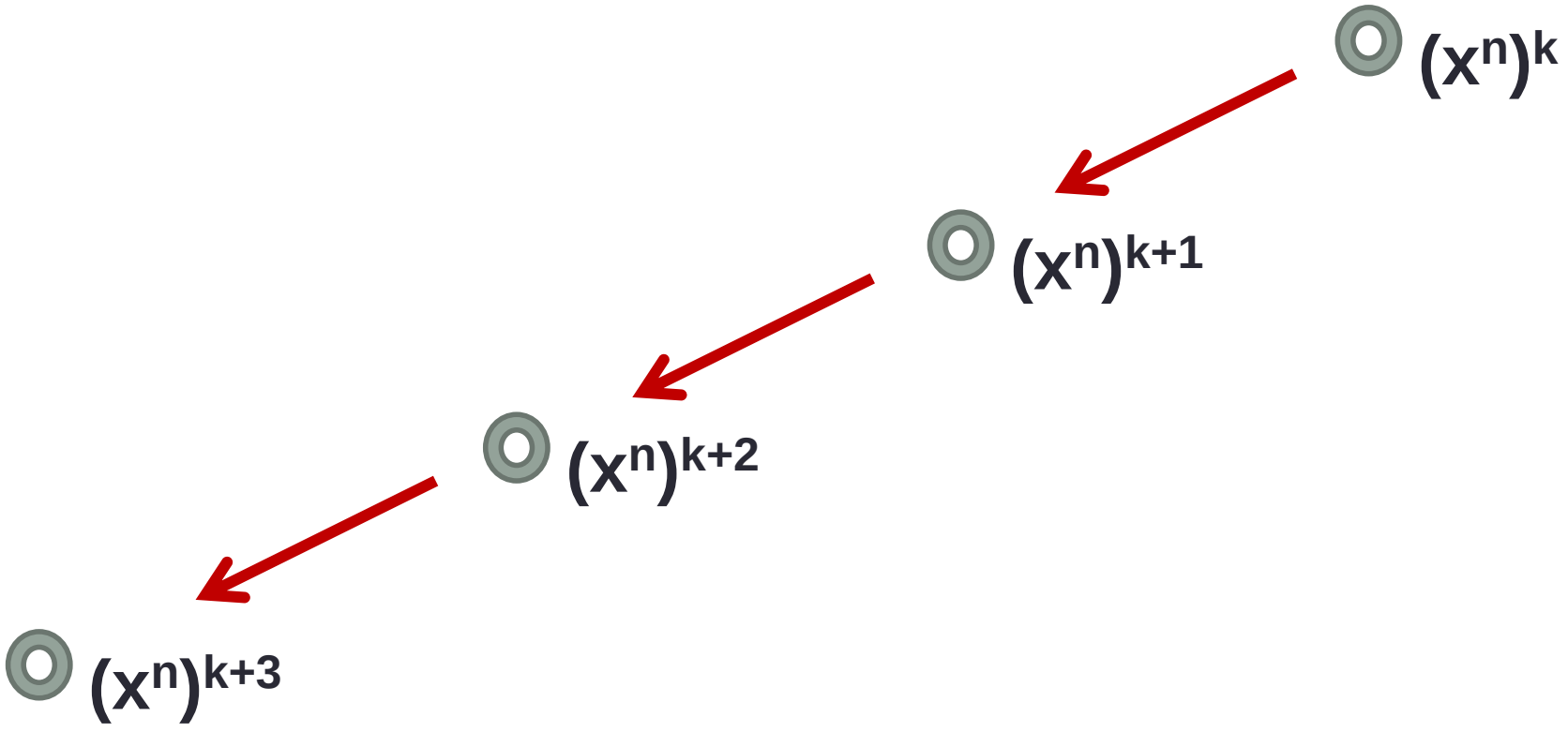
Energy Aware Gauss-Seidel Algorithm



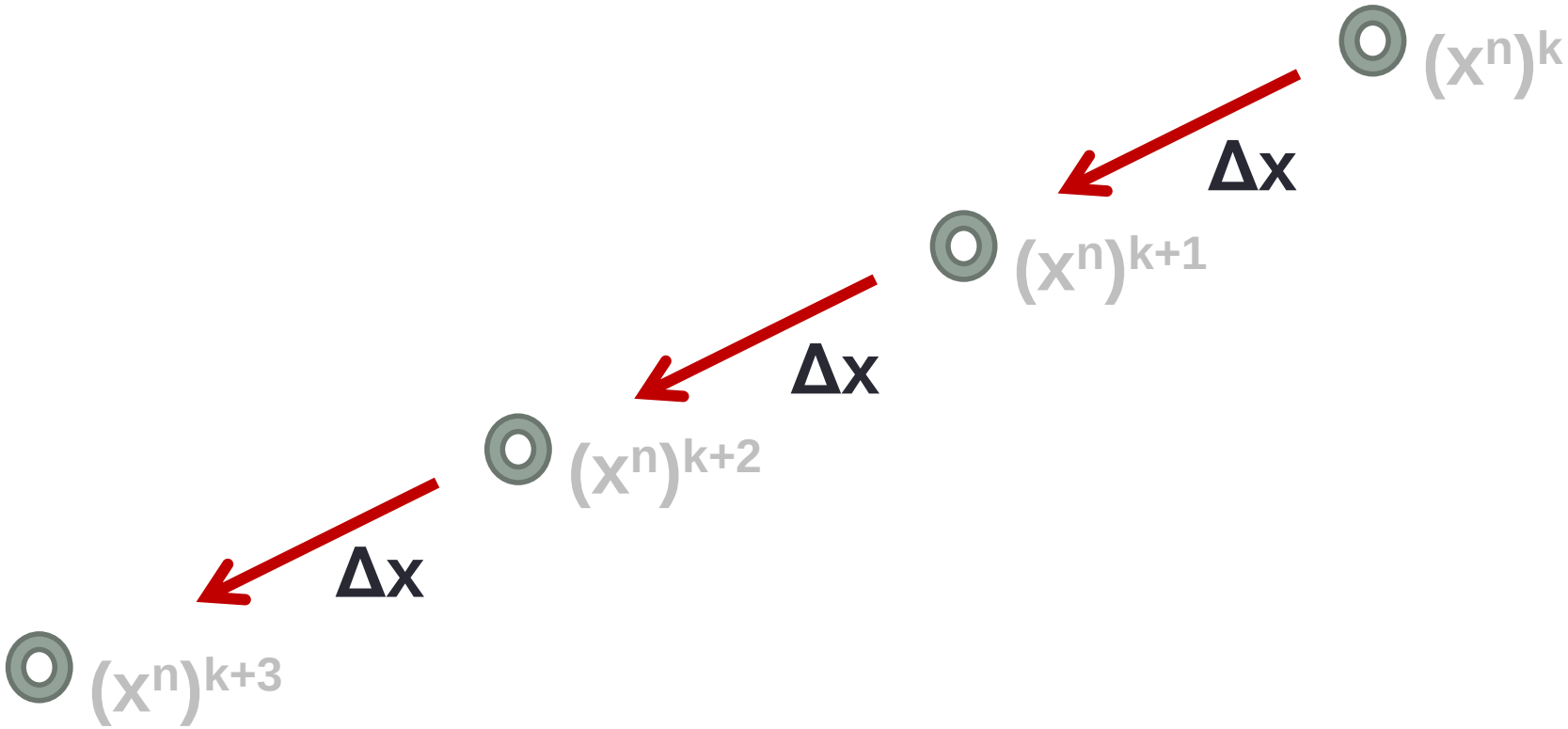
Energy Aware Gauss-Seidel Algorithm



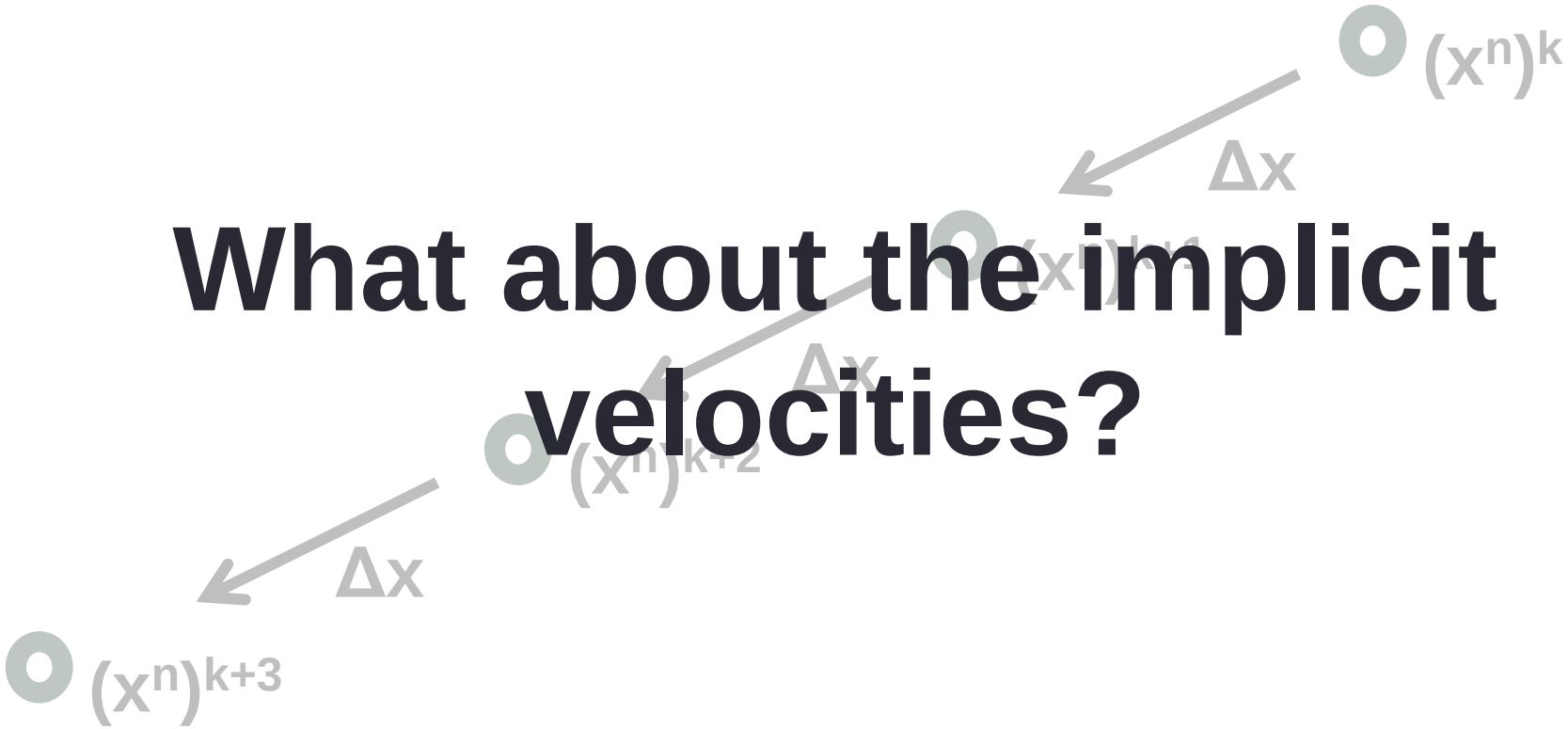
Energy Aware Gauss-Seidel Algorithm



Energy Aware Gauss-Seidel Algorithm

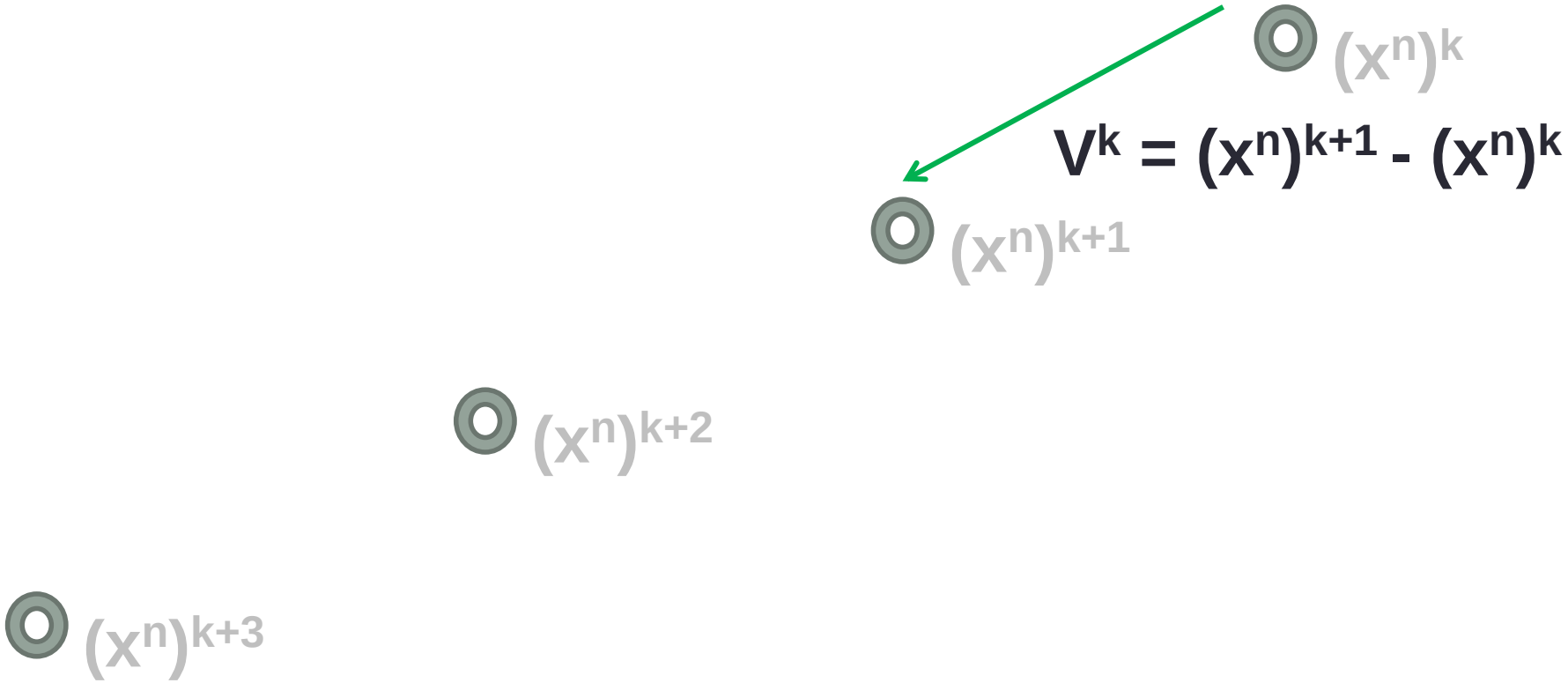


Energy Aware Gauss-Seidel Algorithm

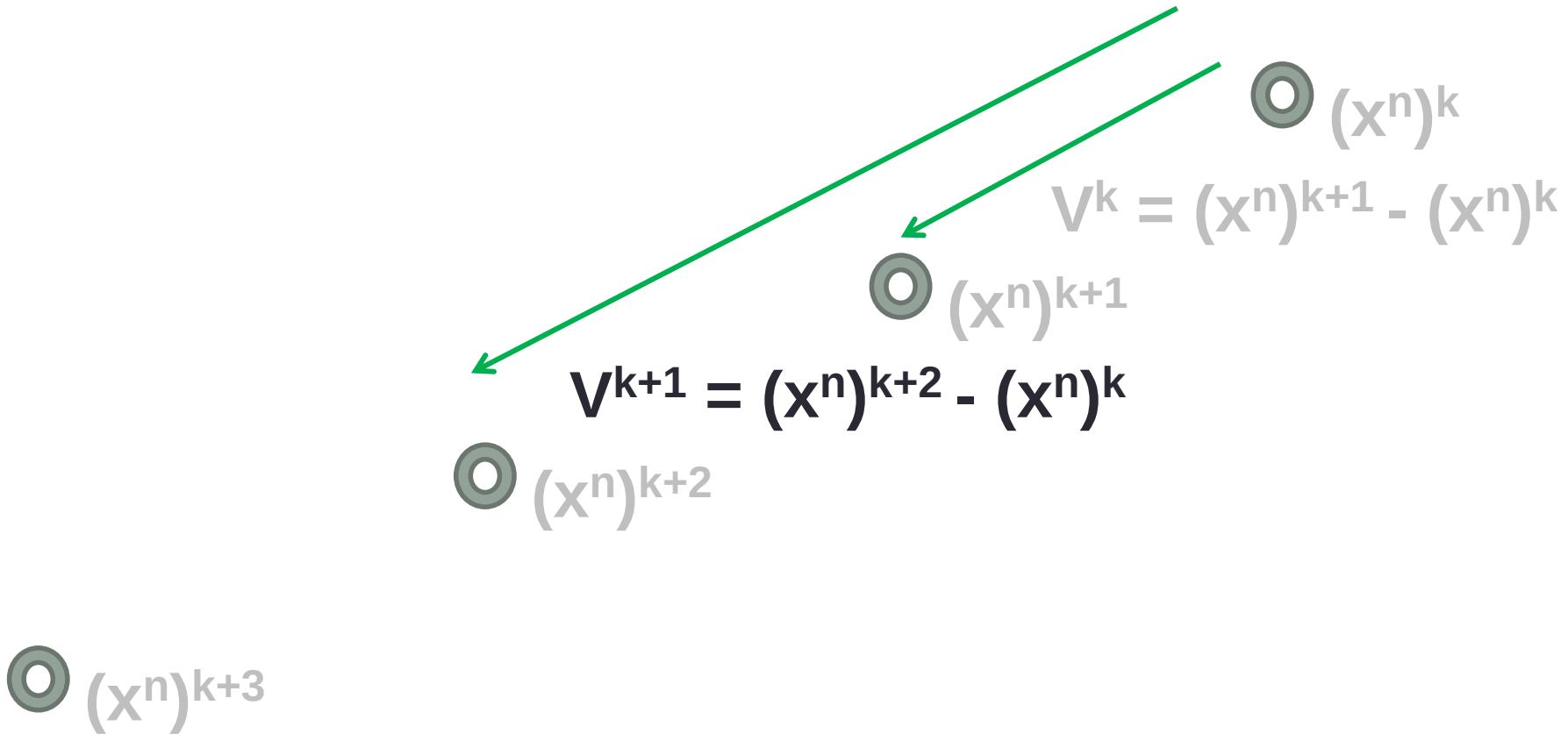


What about the implicit velocities?

Energy Aware Gauss-Seidel Algorithm



Energy Aware Gauss-Seidel Algorithm



Energy Aware Gauss-Seidel Algorithm

$(x^n)^k$

$V^k = (x^n)^{k+1} - (x^n)^k$

$(x^n)^{k+1}$

$V^{k+1} = (x^n)^{k+2} - (x^n)^k$

$(x^n)^{k+2}$

$V^{k+2} = (x^n)^{k+3} - (x^n)^k$

$(x^n)^{k+3}$

Energy Aware Gauss-Seidel Algorithm

Now, what do we know within the Gauss-Seidel Alg.:

- Positions
- Position changes
- Implicit velocities
- Constraint values (Potential energy values)

Energy Aware Gauss-Seidel Algorithm

Now, what do we know within the Gauss-Seidel Alg.:

- Positions
- Position changes
- Implicit velocities
- Constraint values (Potential energy values)

Is it posible to create an energy balance within the Gauss-Seidel iteration?

Energy Aware Gauss-Seidel Algorithm

IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS

Energy Conservation for the Simulation of Deformable Bodies

Jonathan Su, Rahul Sheth, and Ronald Fedkiw

Su et al. (2012) applied the conservation principle to a mass-spring network where:

- Velocities are based on forces.
- Hooke’s law dominates the framework.
- There is an integration dependency.

Energy Aware Gauss-Seidel Algorithm

However, inspired by their work, we alter the Gauss-Seidel Algorithm with the energy balance in its each iteration:

That means each iteration satisfies:

$$\boxed{\Delta KE = -\Delta PE}$$

Energy Aware Gauss-Seidel Algorithm

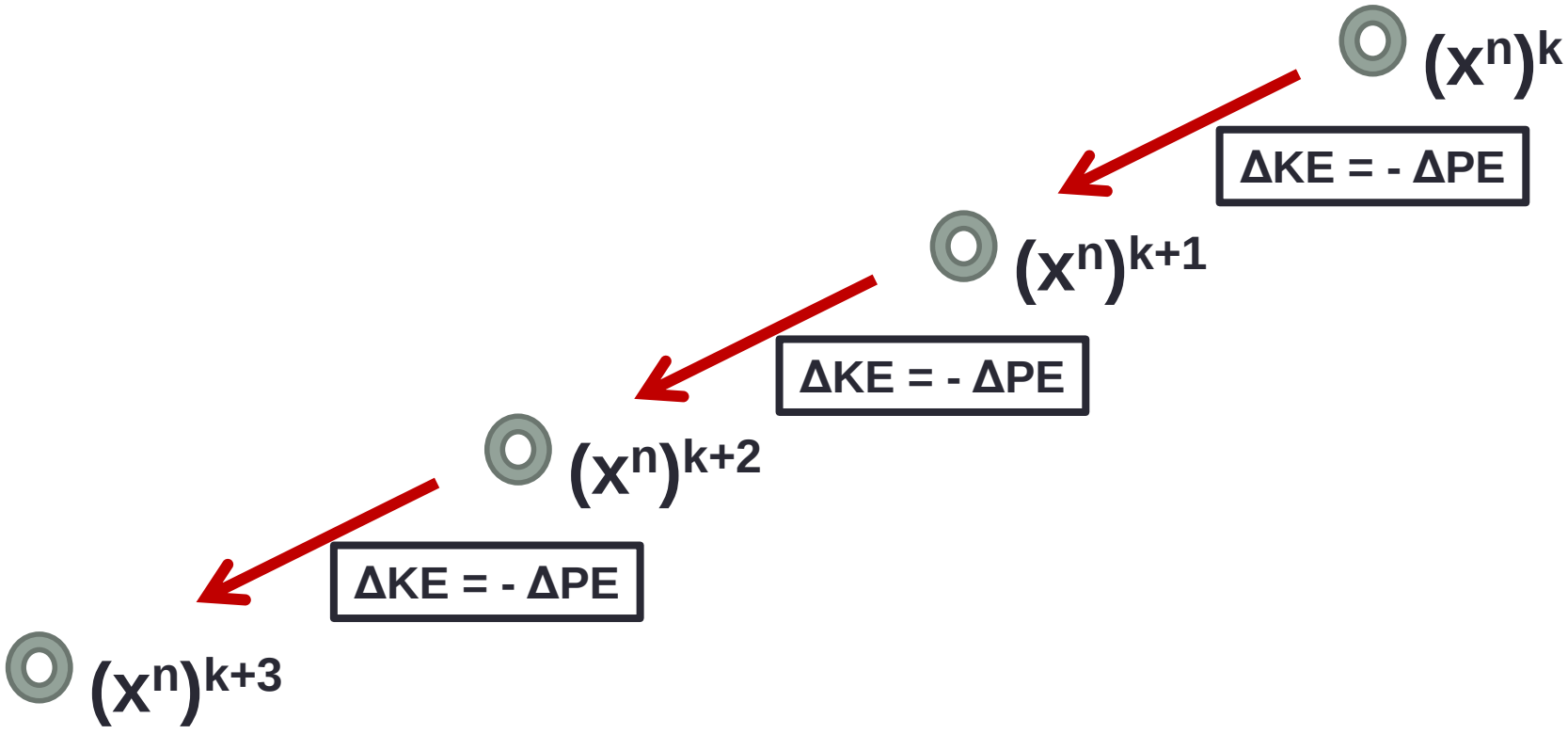
However, inspired by their work, we alter the Gauss-Seidel Algorithm with the energy balance in its each iteration:

That means each iteration satisfies:

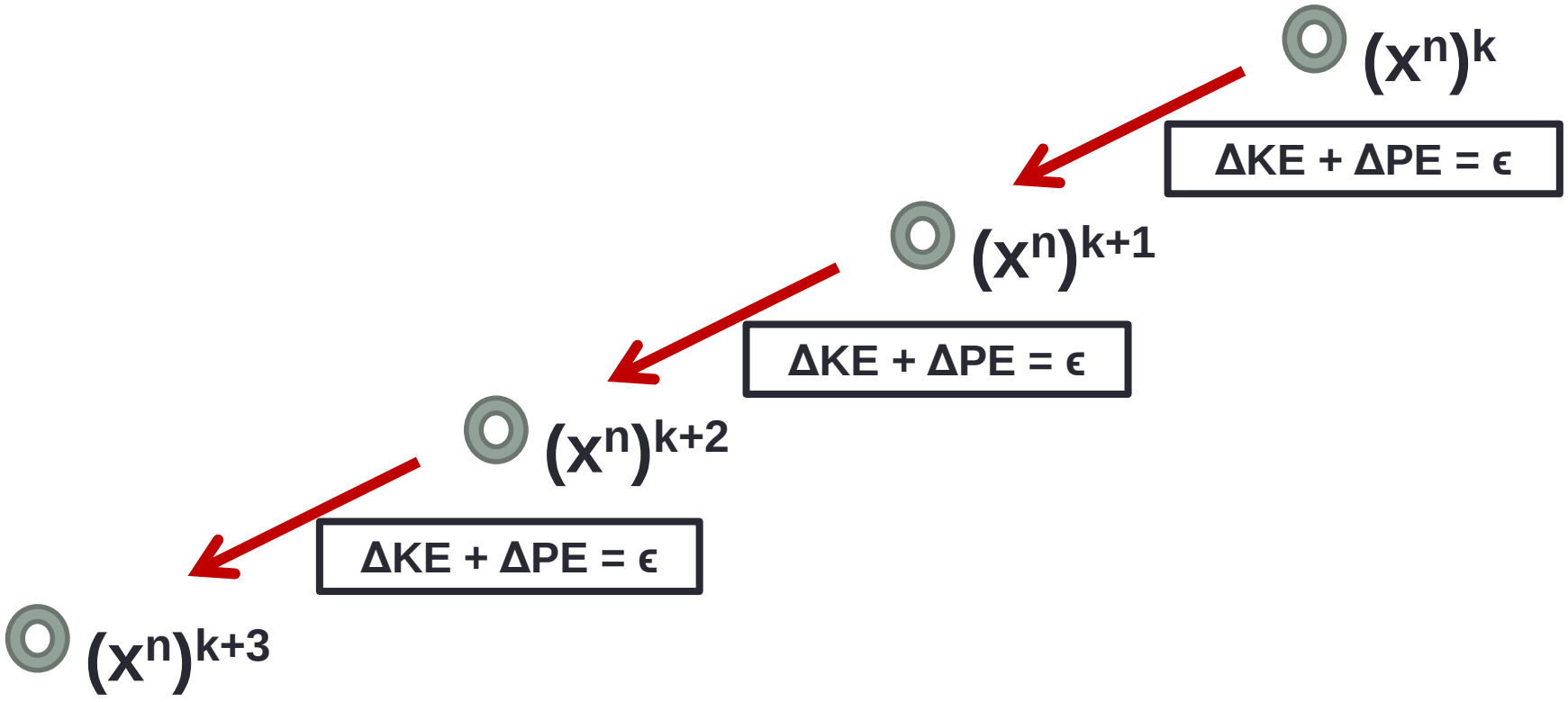
$$\boxed{\Delta KE = -\Delta PE}$$

$$\boxed{\frac{1}{2}m\left[\left(V^{k+1}\right)^{T}V^{k+1} - \left(V^{k}\right)^{T}V^{k}\right] = -[C(x^{n})^{k} - C\left(x^{n-1}\right)]}$$

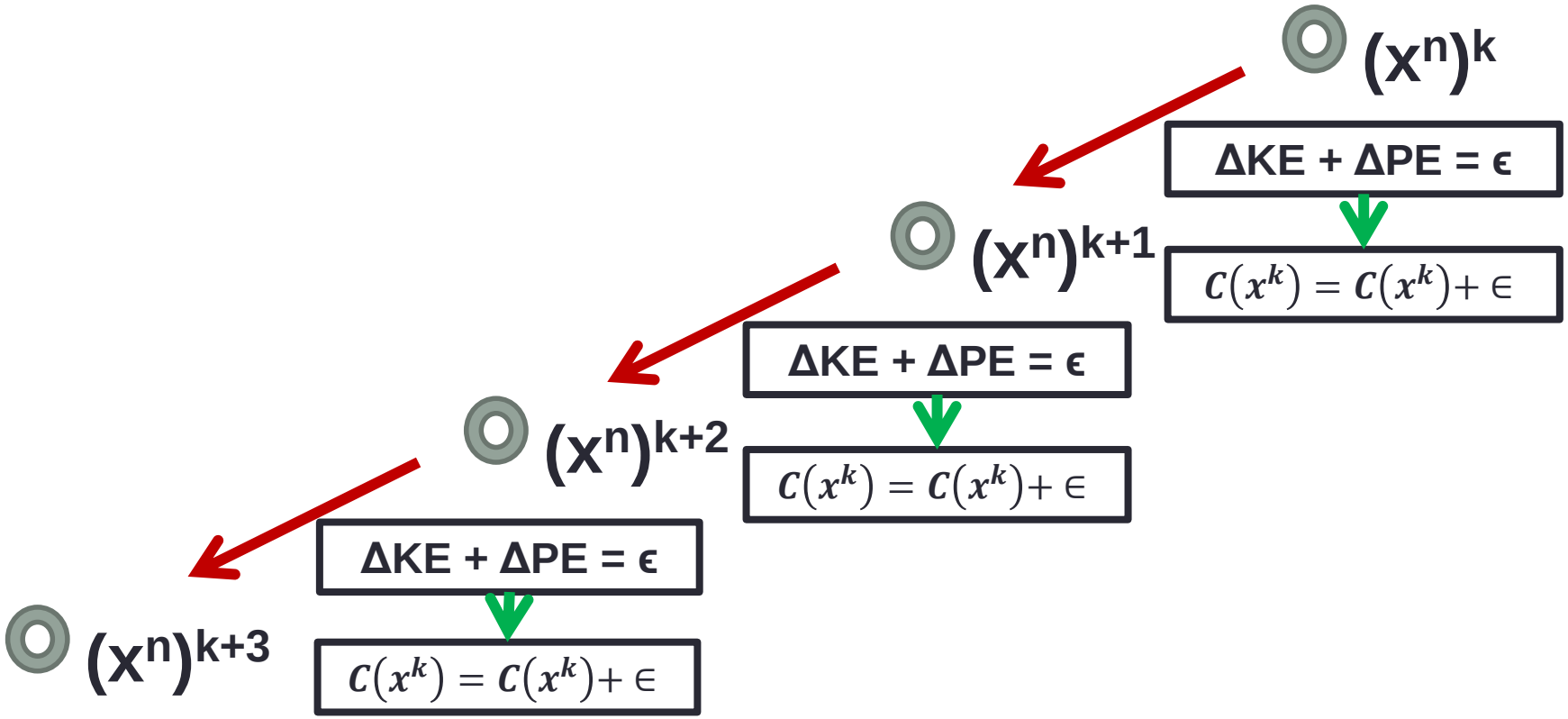
Energy Aware Gauss-Seidel Algorithm



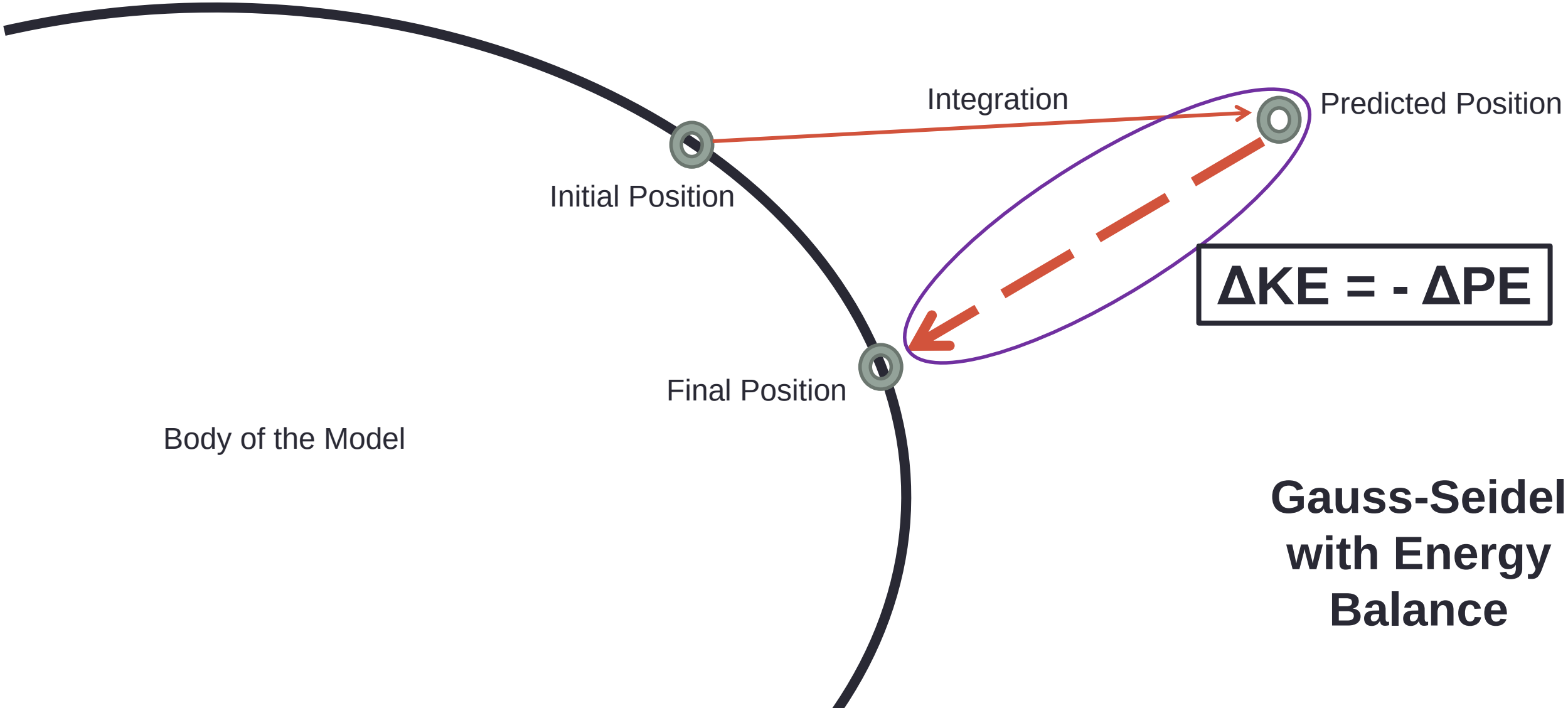
Energy Aware Gauss-Seidel Algorithm



Energy Aware Gauss-Seidel Algorithm



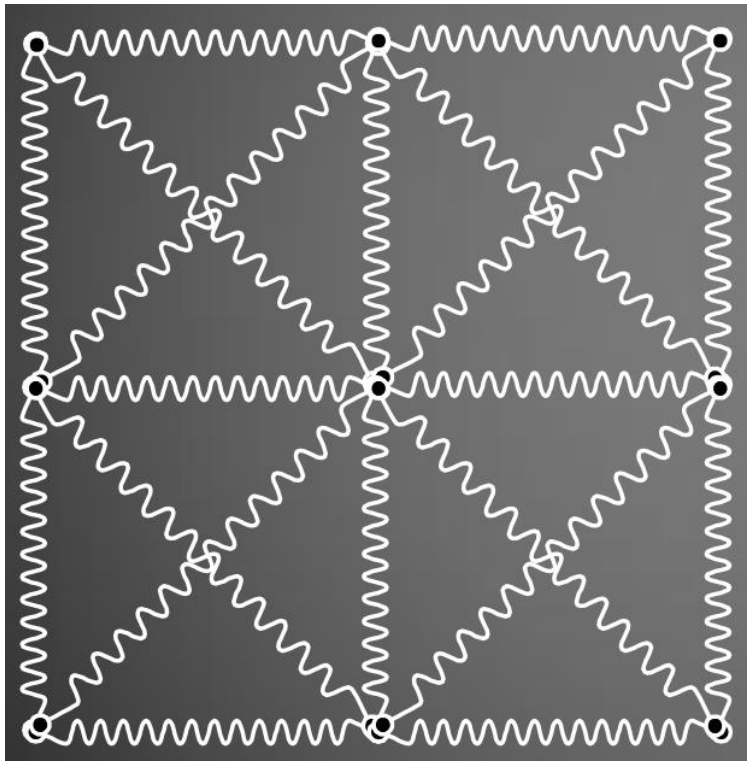
Energy Aware Gauss-Seidel Algorithm

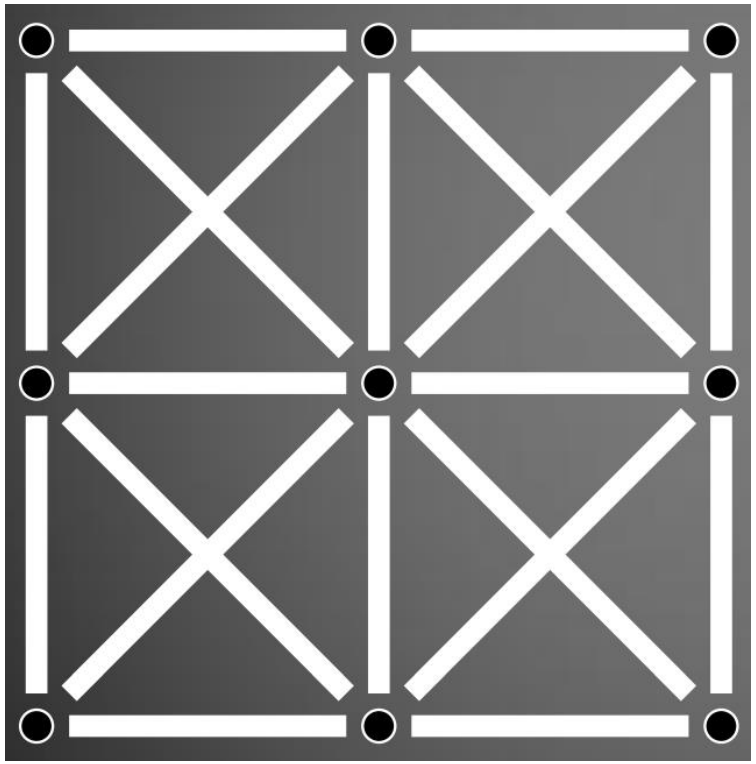


Gauss-Seidel with Energy Balance

Mesh Coloring Alg.

Connectivity of the particles is a limitation for parallelism:

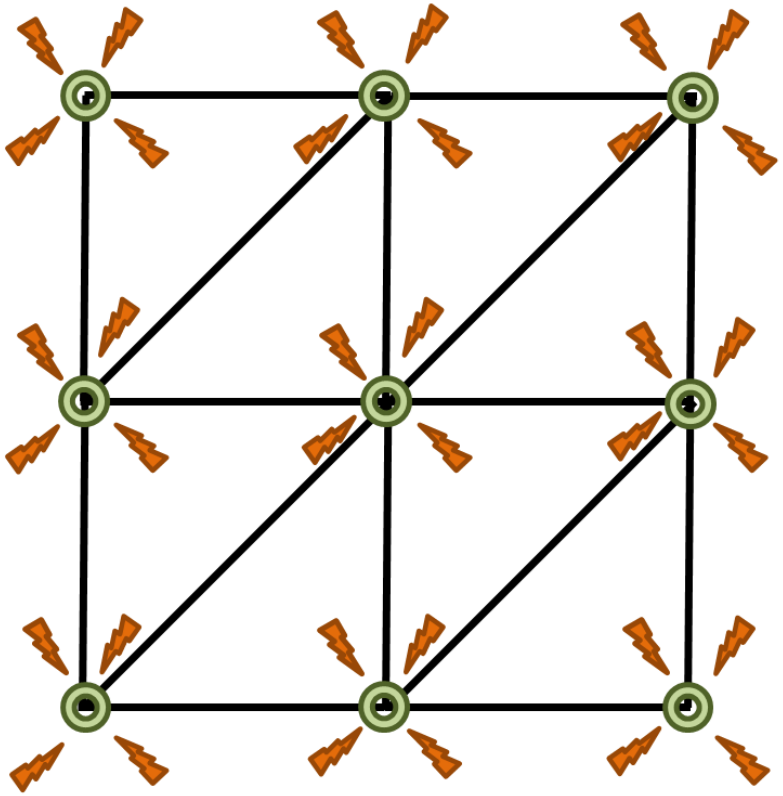




[Stam 2009]

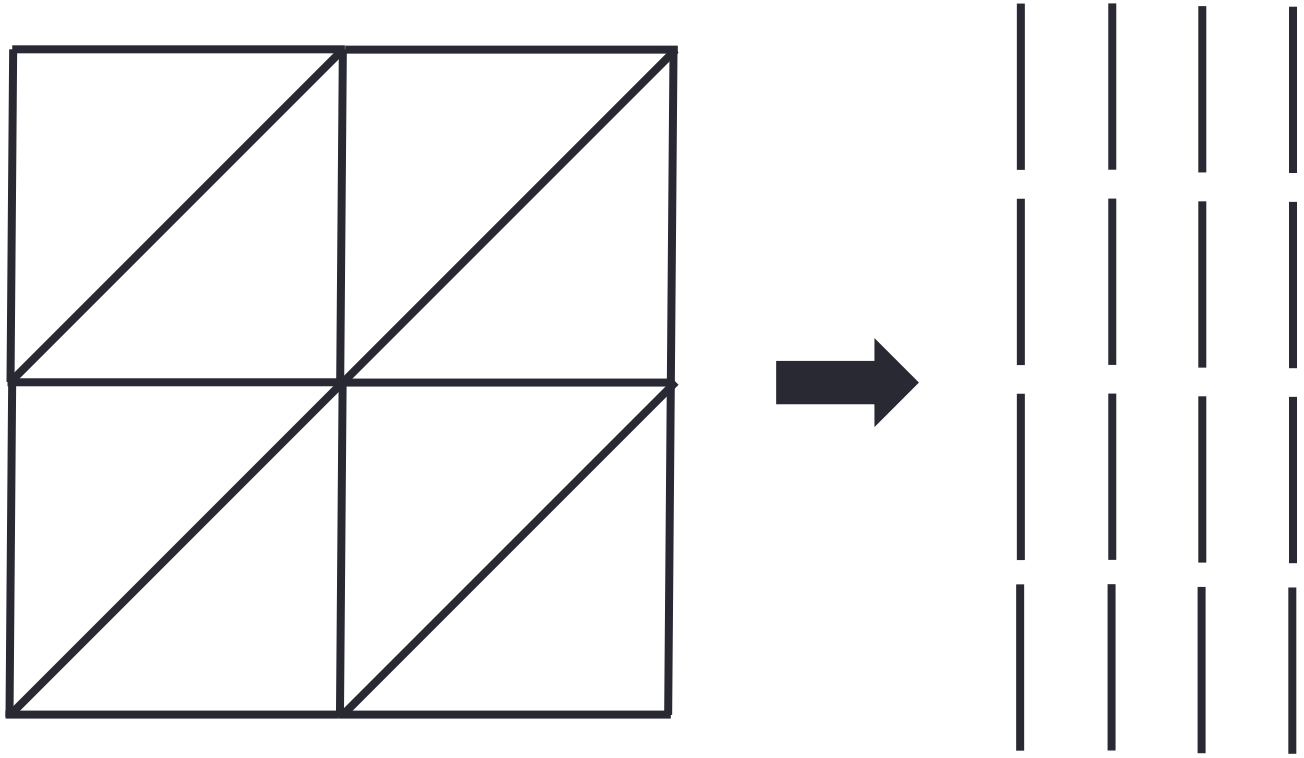
Mesh Coloring Alg.

This limitation is the well-known race condition which is the attack of many threads to one particle simultaneously:



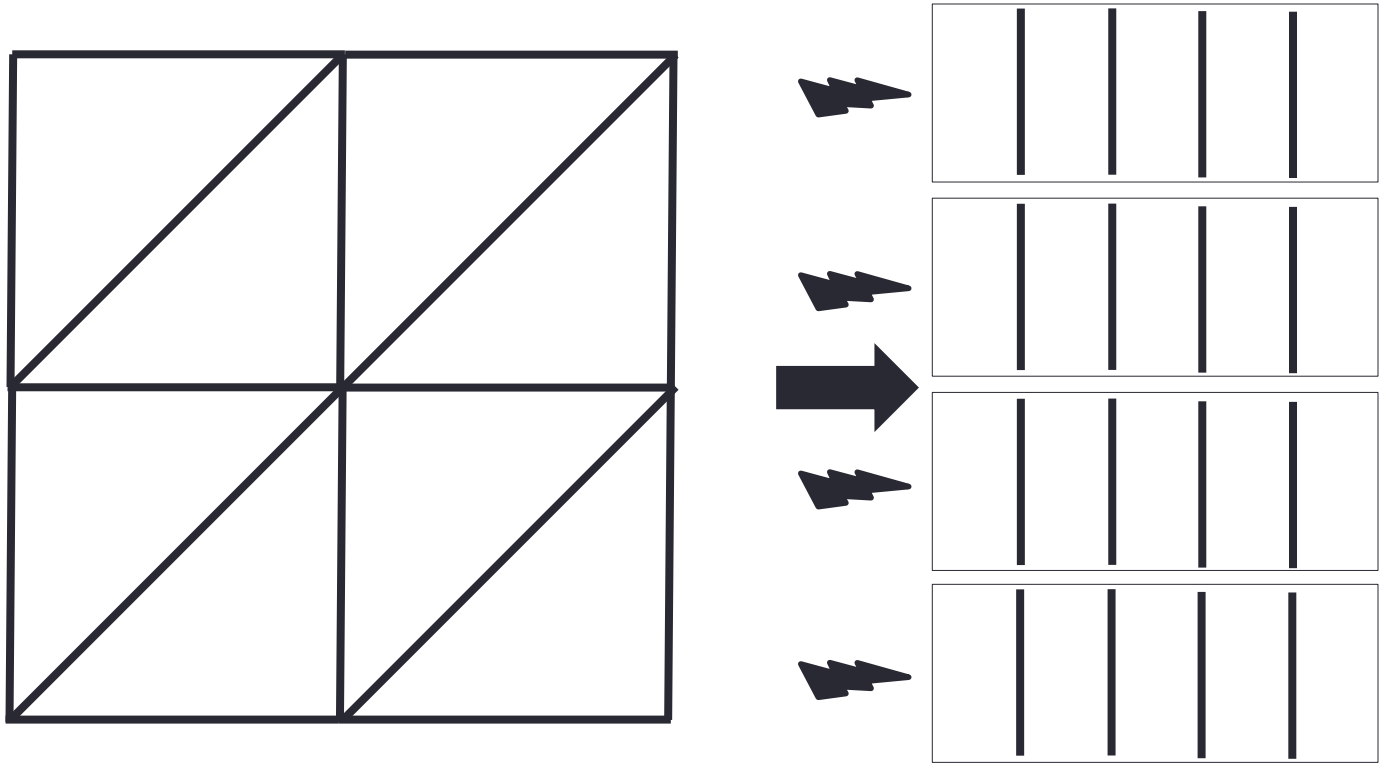
Mesh Coloring Alg.

In order to avoid race condition, we have to partition the mesh structure:



Mesh Coloring Alg.

This partitioning has to be done in such a way that none of the elements share any connectivity:



Mesh Coloring Alg.

In order to achieve that, an intuitive **Mesh Coloring Algorithm** is implemented:

Mesh Coloring Alg.

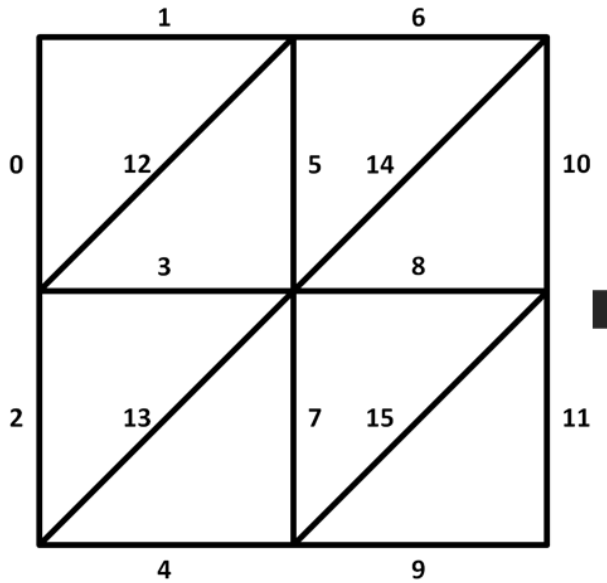
In order to achieve that, an intuitive **Mesh Coloring Algorithm** is implemented:

- It is a two step algorithm:

Mesh Coloring Alg.

In order to achieve that, an intuitive **Mesh Coloring Algorithm** is implemented:

- It is a two step algorithm:
  - First, a non-adjacent array list for each member is created:



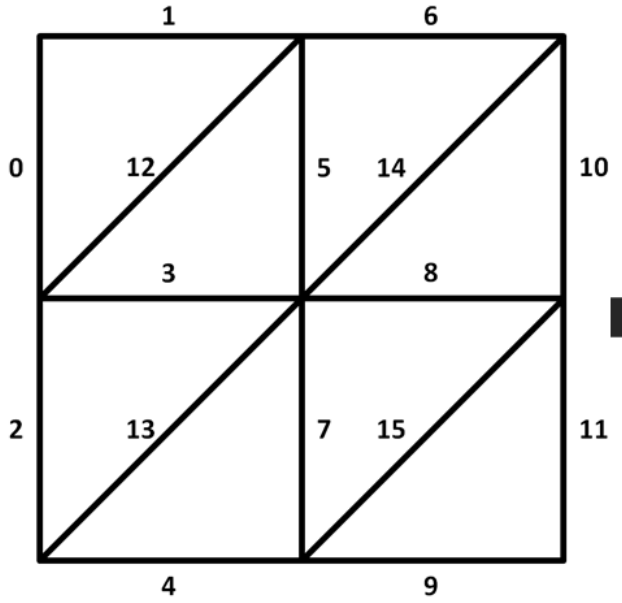
$$
\begin{array}{l}
0 \rightarrow [4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 13 \quad 14] \\
1 \rightarrow [2 \quad 3 \quad 7 \quad 8 \quad 10 \quad 13 \quad 14] \\
2 \rightarrow [1 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 14 \quad 15] \\
3 \rightarrow [1 \quad 4 \quad 6 \quad 9 \quad 10 \quad 11 \quad 15] \\
4 \rightarrow [0 \quad 3 \quad 5 \quad 8 \quad 10 \quad 11 \quad 12 \quad 14] \\
5 \rightarrow [0 \quad 2 \quad 4 \quad 9 \quad 10 \quad 11 \quad 15] \\
6 \rightarrow [0 \quad 2 \quad 3 \quad 7 \quad 8 \quad 11 \quad 13 \quad 15] \\
7 \rightarrow [0 \quad 1 \quad 2 \quad 6 \quad 10 \quad 11 \quad 12] \\
8 \rightarrow [0 \quad 1 \quad 2 \quad 4 \quad 6 \quad 9 \quad 12] \\
9 \rightarrow [2 \quad 3 \quad 5 \quad 8 \quad 10 \quad 13 \quad 14] \\
10 \rightarrow [1 \quad 3 \quad 4 \quad 5 \quad 7 \quad 9 \quad 12 \quad 13] \\
11 \rightarrow [3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 13 \quad 14] \\
12 \rightarrow [4 \quad 7 \quad 8 \quad 10 \quad 13 \quad 14] \\
13 \rightarrow [0 \quad 1 \quad 6 \quad 9 \quad 10 \quad 11 \quad 12 \quad 15] \\
14 \rightarrow [0 \quad 1 \quad 2 \quad 4 \quad 9 \quad 11 \quad 12 \quad 15] \\
15 \rightarrow [2 \quad 3 \quad 5 \quad 6 \quad 13 \quad 14]
\end{array}
$$

Mesh - Before Coloring **Non-Adjacent Array List**

Mesh Coloring Alg.

In order to achieve that, an intuitive **Mesh Coloring Algorithm** is implemented:

- It is a two step algorithm:
  - Second, colored groups are obtained from the non-adjacent array list:



$$
\begin{aligned}
0 &\rightarrow [4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 13\ \ 14]\\
1 &\rightarrow [2\ \ 3\ \ 7\ \ 8\ \ 10\ \ 13\ \ 14]\\
2 &\rightarrow [1\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 14\ \ 15]\\
3 &\rightarrow [1\ \ 4\ \ 6\ \ 9\ \ 10\ \ 11\ \ 15]\\
4 &\rightarrow [0\ \ 3\ \ 5\ \ 8\ \ 10\ \ 11\ \ 12\ \ 14]\\
5 &\rightarrow [0\ \ 2\ \ 4\ \ 9\ \ 10\ \ 11\ \ 15]\\
6 &\rightarrow [0\ \ 2\ \ 3\ \ 7\ \ 8\ \ 11\ \ 13\ \ 15]\\
7 &\rightarrow [0\ \ 1\ \ 2\ \ 6\ \ 10\ \ 11\ \ 12]\\
8 &\rightarrow [0\ \ 1\ \ 2\ \ 4\ \ 6\ \ 9\ \ 12]\\
9 &\rightarrow [2\ \ 3\ \ 5\ \ 8\ \ 10\ \ 13\ \ 14]\\
10 &\rightarrow [1\ \ 3\ \ 4\ \ 5\ \ 7\ \ 9\ \ 12\ \ 13]\\
11 &\rightarrow [3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 13\ \ 14]\\
12 &\rightarrow [4\ \ 7\ \ 8\ \ 10\ \ 13\ \ 14]\\
13 &\rightarrow [0\ \ 1\ \ 6\ \ 9\ \ 10\ \ 11\ \ 12\ \ 15]\\
14 &\rightarrow [0\ \ 1\ \ 2\ \ 4\ \ 9\ \ 11\ \ 12\ \ 15]\\
15 &\rightarrow [2\ \ 3\ \ 5\ \ 6\ \ 13\ \ 14]
\end{aligned}
$$

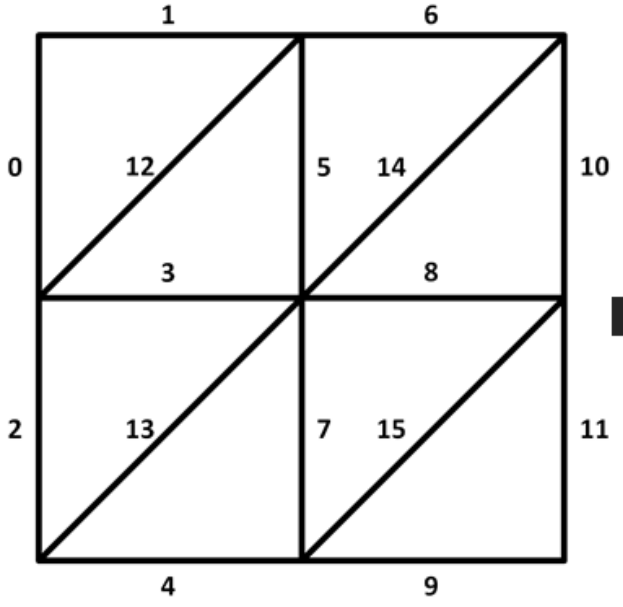
$$
\begin{aligned}
\text{Group 1} &\rightarrow [0\ \ 4\ \ 5\ \ 10]\\
\text{Group 2} &\rightarrow [1\ \ 2\ \ 7\ \ 11]\\
\text{Group 3} &\rightarrow [3\ \ 6\ \ 9]\\
\text{Group 4} &\rightarrow [12\ \ 8]\\
\text{Group 5} &\rightarrow [13]\\
\text{Group 6} &\rightarrow [14\ \ 15]
\end{aligned}
$$

Mesh - Before Coloring **Non-Adjacent Array List** **Colored Groups**

Mesh Coloring Alg.

In order to achieve that, an intuitive **Mesh Coloring Algorithm** is implemented:

- It is a two step algorithm:
  - Second, independent edges list is obtained from the non-adjacent array list:



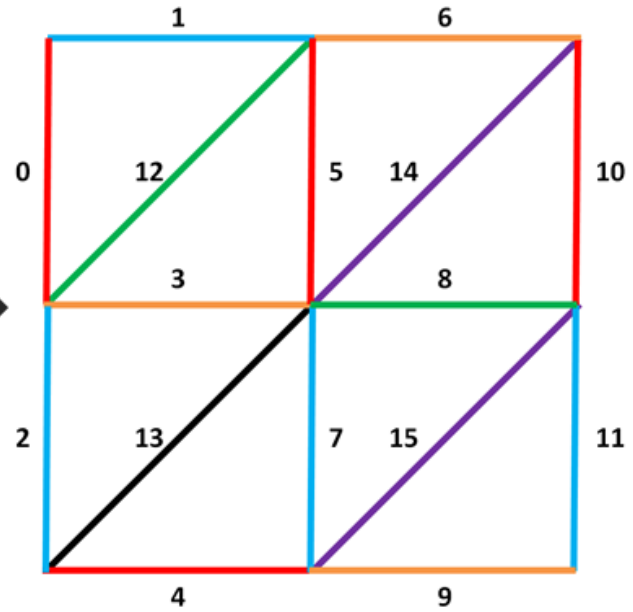
Mesh - Before Coloring

$$
\begin{aligned}
0 &\rightarrow [4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 13 \quad 14] \\
1 &\rightarrow [2 \quad 3 \quad 7 \quad 8 \quad 10 \quad 13 \quad 14] \\
2 &\rightarrow [1 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 14 \quad 15] \\
3 &\rightarrow [1 \quad 4 \quad 6 \quad 9 \quad 10 \quad 11 \quad 15] \\
4 &\rightarrow [0 \quad 3 \quad 5 \quad 8 \quad 10 \quad 11 \quad 12 \quad 14] \\
5 &\rightarrow [0 \quad 2 \quad 4 \quad 9 \quad 10 \quad 11 \quad 15] \\
6 &\rightarrow [0 \quad 2 \quad 3 \quad 7 \quad 8 \quad 11 \quad 13 \quad 15] \\
7 &\rightarrow [0 \quad 1 \quad 2 \quad 6 \quad 10 \quad 11 \quad 12] \\
8 &\rightarrow [0 \quad 1 \quad 2 \quad 4 \quad 6 \quad 9 \quad 12] \\
9 &\rightarrow [2 \quad 3 \quad 5 \quad 8 \quad 10 \quad 13 \quad 14] \\
10 &\rightarrow [1 \quad 3 \quad 4 \quad 5 \quad 7 \quad 9 \quad 12 \quad 13] \\
11 &\rightarrow [3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 13 \quad 14] \\
12 &\rightarrow [4 \quad 7 \quad 8 \quad 10 \quad 13 \quad 14] \\
13 &\rightarrow [0 \quad 1 \quad 6 \quad 9 \quad 10 \quad 11 \quad 12 \quad 15] \\
14 &\rightarrow [0 \quad 1 \quad 2 \quad 4 \quad 9 \quad 11 \quad 12 \quad 15] \\
15 &\rightarrow [2 \quad 3 \quad 5 \quad 6 \quad 13 \quad 14]
\end{aligned}
$$

Non-Adjacent Array List

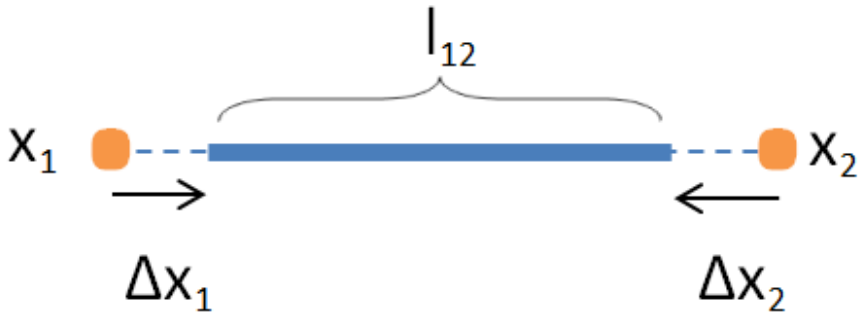
$$
\begin{aligned}
\text{Group } 1 &\rightarrow [0 \quad 4 \quad 5 \quad 10] \\
\text{Group } 2 &\rightarrow [1 \quad 2 \quad 7 \quad 11] \\
\text{Group } 3 &\rightarrow [3 \quad 6 \quad 9] \\
\text{Group } 4 &\rightarrow [12 \quad 8] \\
\text{Group } 5 &\rightarrow [13] \\
\text{Group } 6 &\rightarrow [14 \quad 15]
\end{aligned}
$$

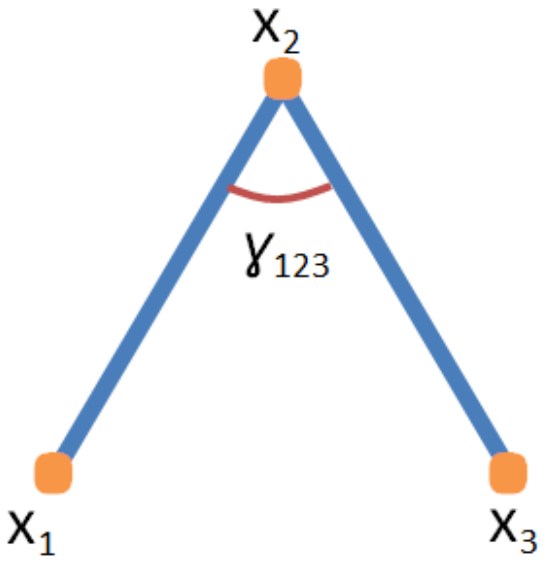
Colored Groups

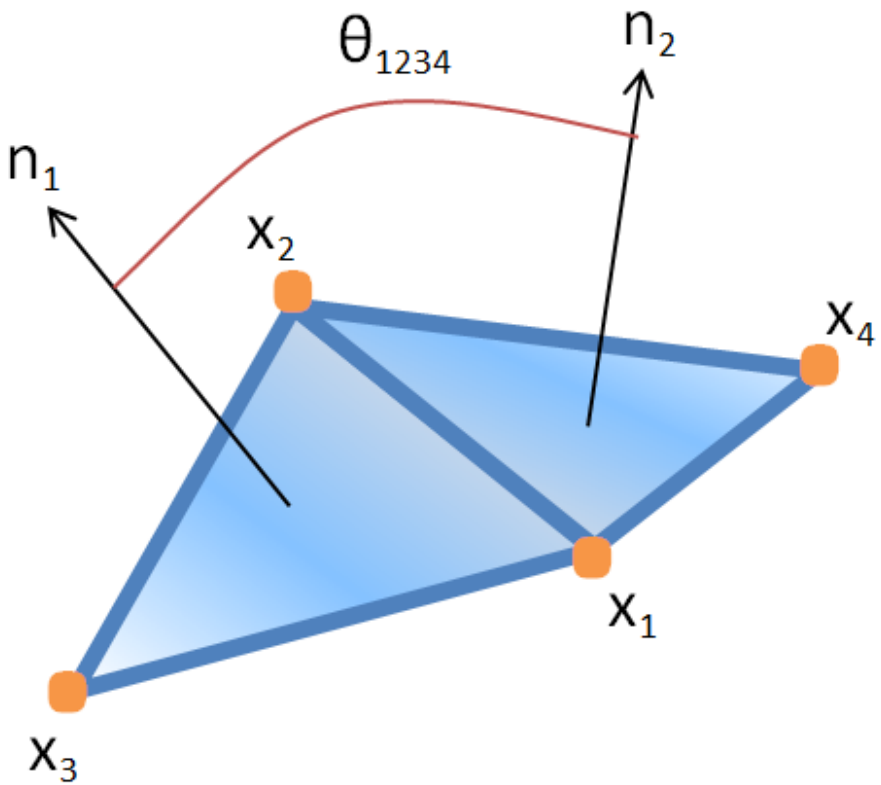


Mesh – After Coloring

Constraints





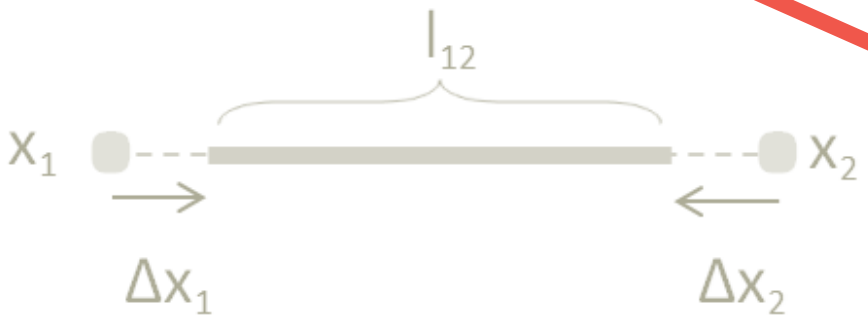


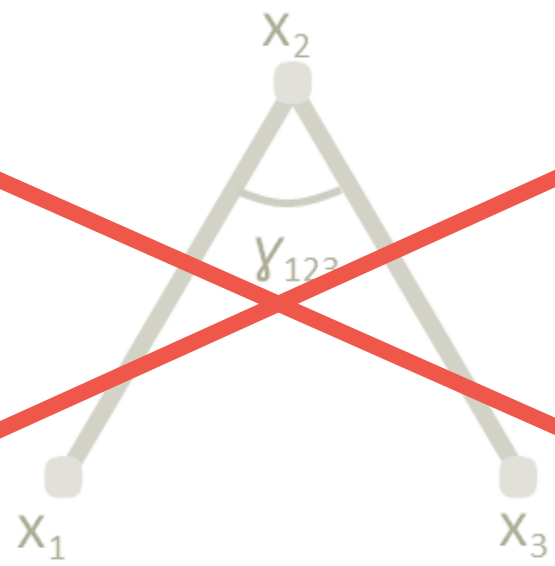
$$C_{stretch}(x_1, x_2) = |x_2 - x_1| - l_{12}$$

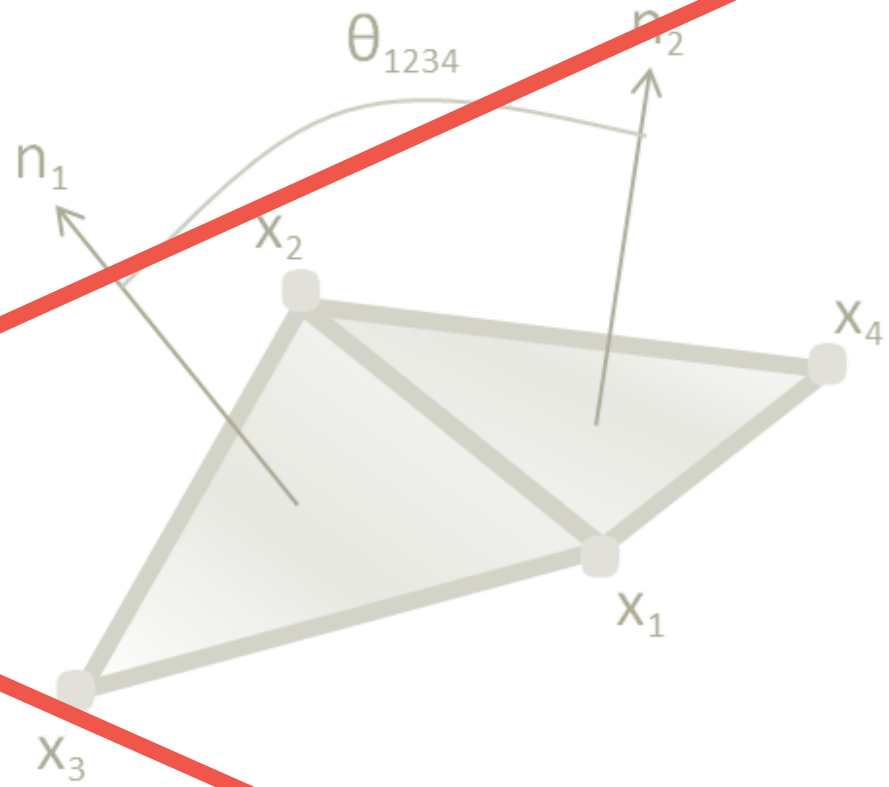
$$C_{bend}(x_1, x_2, x_3, x_4) = cos^{-1}(N_{123} \cdot N_{124}) - \theta_{1234}$$

$$C_{shear}(x_1, x_2, x_3) = cos^{-1}(M_{12} \cdot M_{13}) - \gamma_{123}$$

Constraints







$$C_{stretch}(x_1, x_2) = |x_2 - x_1| - l_{12}$$

$$C_{bend}(x_1, x_2, x_3, x_4) = cos^{-1}(N_{123} \cdot N_{124}) - \theta_{1234}$$

$$C_{shear}(x_1, x_2, x_3) = cos^{-1}(M_{12} \cdot M_{13}) - \gamma_{123}$$

Constraints

2D Meshes:

Hookean Spring:

$$C_{Hookean}(x_1, x_2) = \frac{1}{2}k(||x_2 - x_1|| - l_0)^2$$

Liu et al. 13

STVK Spring:

$$C_{StVKSp.}(x_1, x_2) = \frac{1}{2}k(||x_2 - x_1||^2 - l_0^2)^2$$

Junior et al. 18

Ani. Cont. Cloth:

$$C = \begin{pmatrix} \frac{k_{\text{weft}}}{1-\nu_{\text{weft}}\nu_{\text{warp}}} & \frac{k_{\text{weft}}\nu_{\text{warp}}}{1-\nu_{\text{weft}}\nu_{\text{warp}}} & 0 \\ \frac{k_{\text{warp}}\nu_{\text{weft}}}{1-\nu_{\text{weft}}\nu_{\text{warp}}} & \frac{k_{\text{warp}}}{1-\nu_{\text{weft}}\nu_{\text{warp}}} & 0 \\ 0 & 0 & k_{\text{shear}} \end{pmatrix}$$

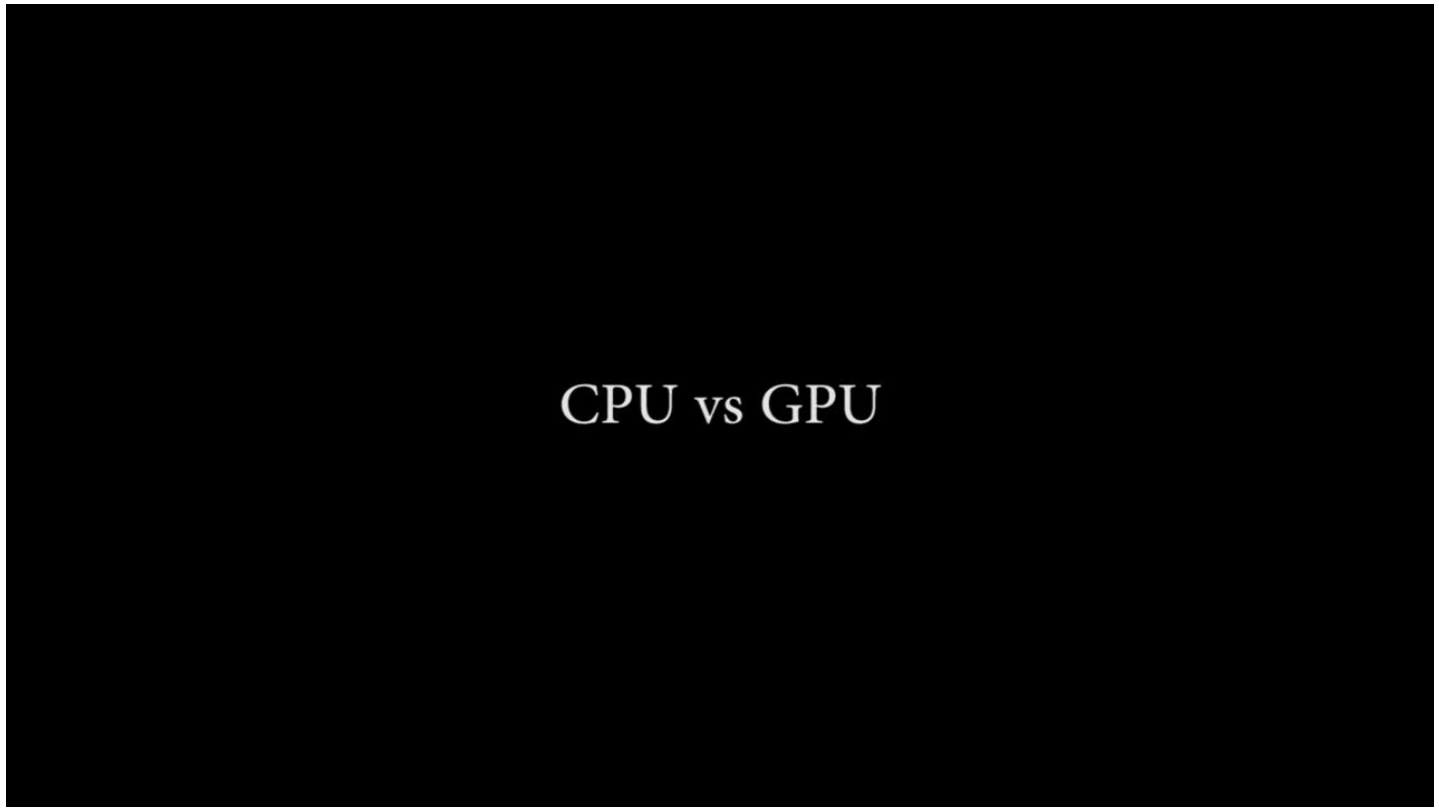
Bender et al. 14

Constraints

3D Meshes:

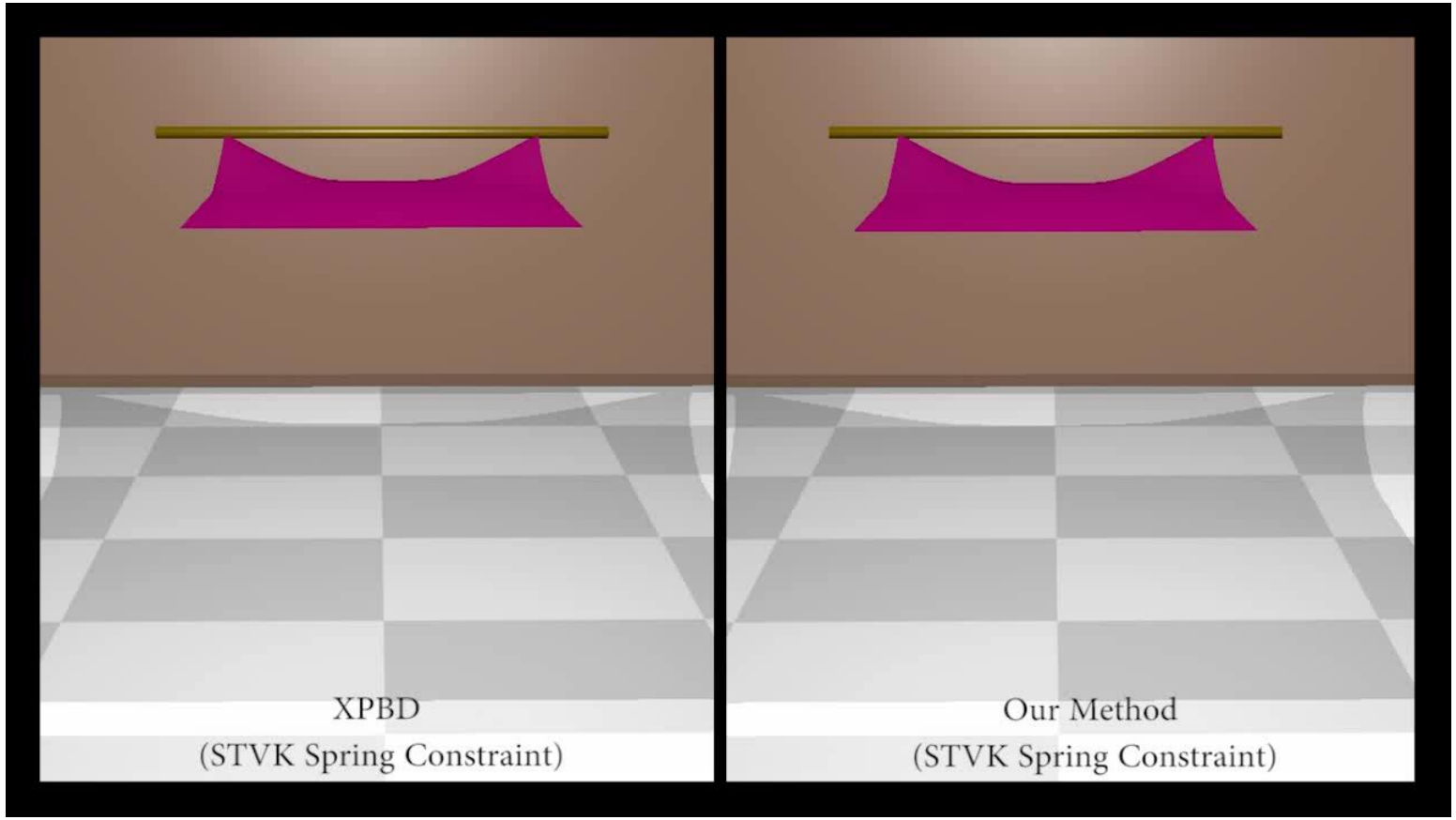
Linear Elastic Material:	$\Psi_{LinElas} = \mu\xi : \xi + \frac{\Lambda}{2} tr^2(\xi)$ Sifakis & Barbic 12
STVK Elastic Material:	$\Psi_{StVKEl.} = \mu G : G + \frac{\Lambda}{2} tr^2(G)$ Bender et al. 14
Neo-Hookean Elastic Material:	$\Psi_{NeoH} = \frac{\mu}{2}(I_1 - log(I_3) - 3) + \frac{\Lambda}{8} log^2(I_3)$ Bender et al. 14
Mooney-Rivlin Elastic Material:	$\Psi_{MooRiv} = \frac{\mu_{01}}{2}(\frac{{I_1}^2 - I_2}{I_3^{2/3}} - 6) + \mu_{10}(\frac{I_1}{I_3^{1/3}} - 3) + \nu_1(I_3^{1/2} - 1)^2$ Sin et al. 13
Arruda-Boyce Elastic Material:	$\Psi_{ArrBo} = \sum_{i=1}^{n} \alpha_i({I_1}^i - 3^i)$ Smith et al. 18

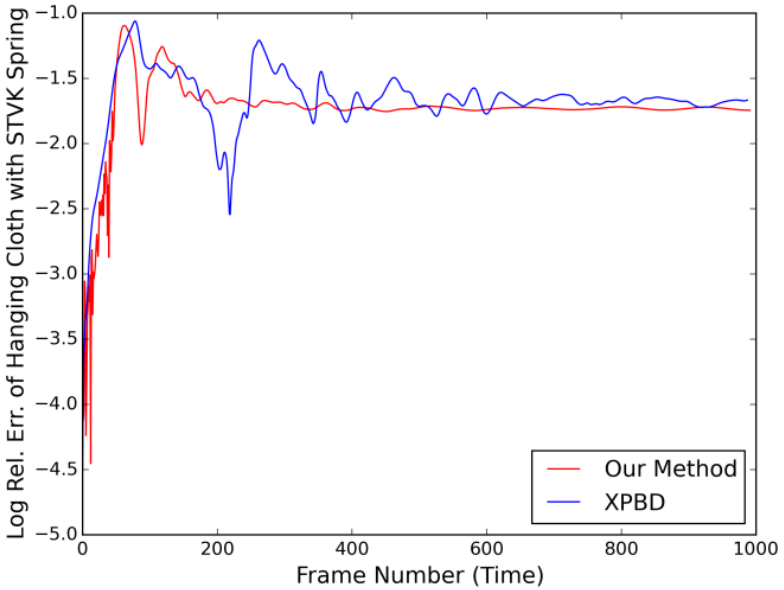
Results



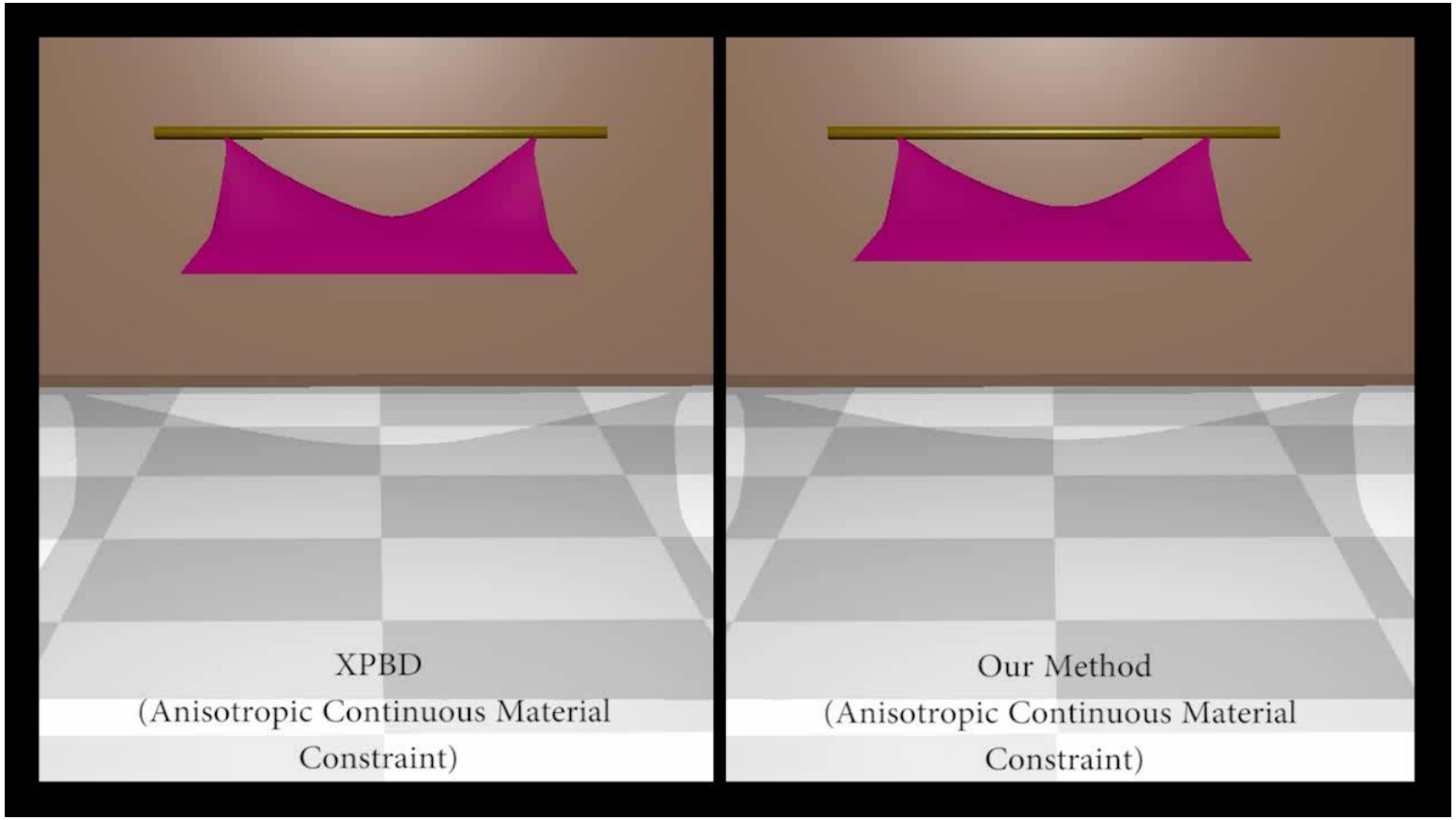
Model	# of Edges	# of Faces	# of Iteration	CPU (FPS)	GPU (FPS)
Cloth	4880	3220	30	17-18	50-51
Bunny	12288	8192	15	2-3	37-38

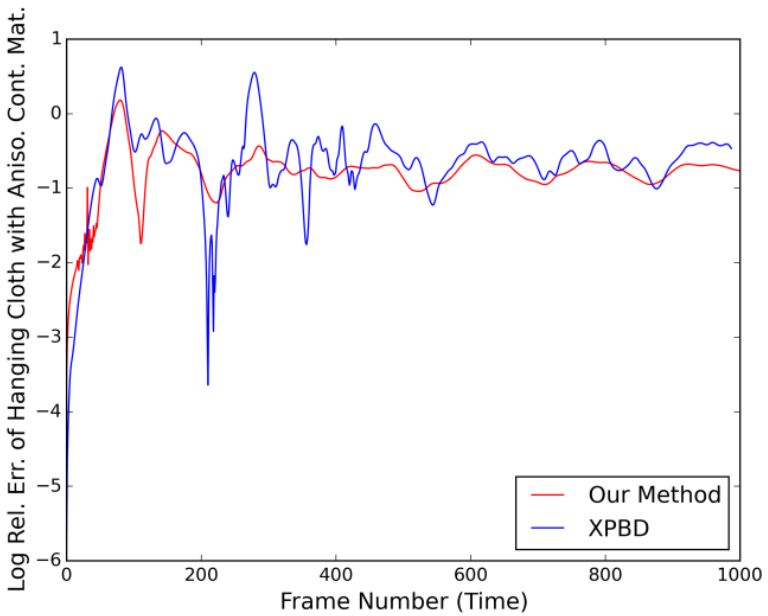
Results


XPBD
(STVK Spring Constraint)
Our Method
(STVK Spring Constraint)

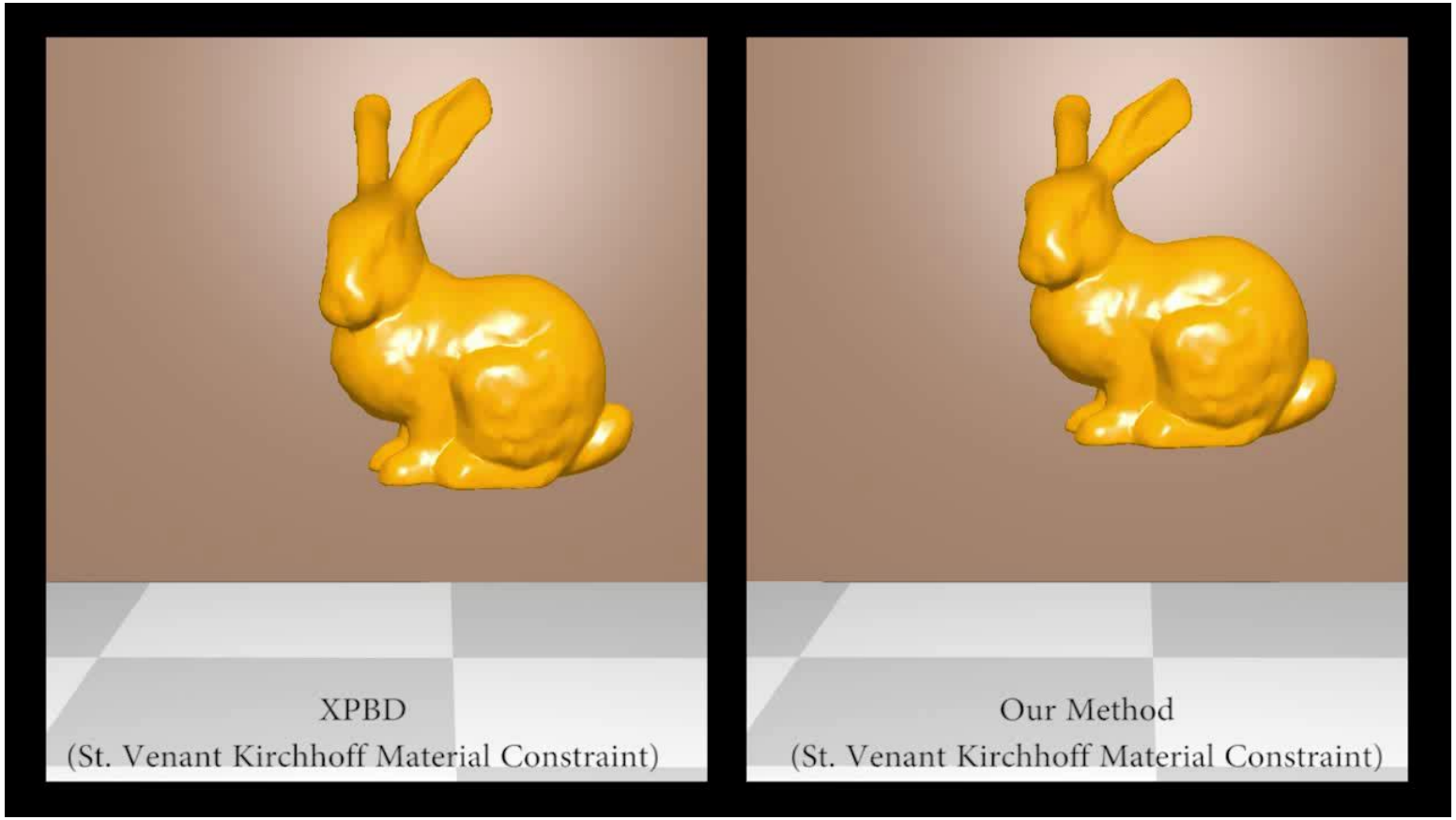

Log Rel. Err. of Hanging Cloth with STVK Spring
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
−4.5
−5.0
0
200
400
600
800
1000
Frame Number (Time)
Our Method
XPBD

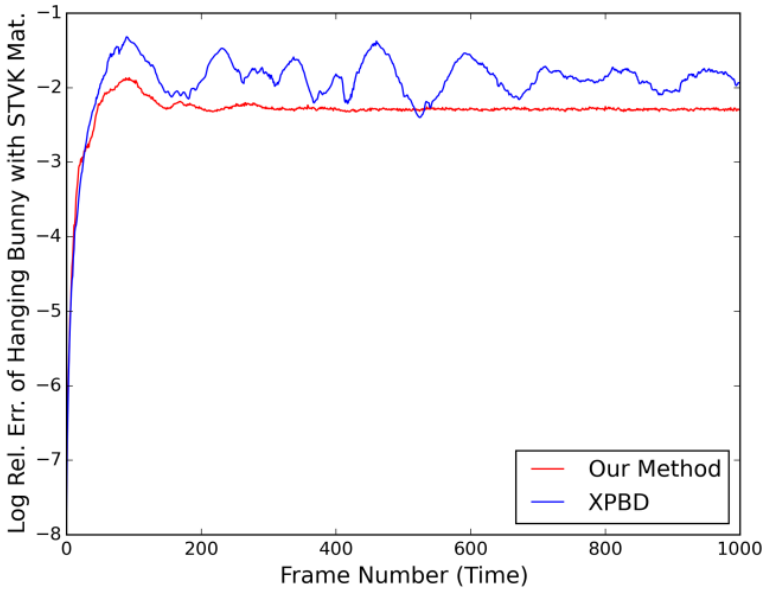
Results


XPBD
(Anisotropic Continuous Material
Constraint)
Our Method
(Anisotropic Continuous Material
Constraint)


Log Rel. Err. of Hanging Cloth with Aniso. Cont. Mat.
1
0
−1
−2
−3
−4
−5
−6
0
200
400
600
800
1000
Frame Number (Time)
Our Method
XPBD

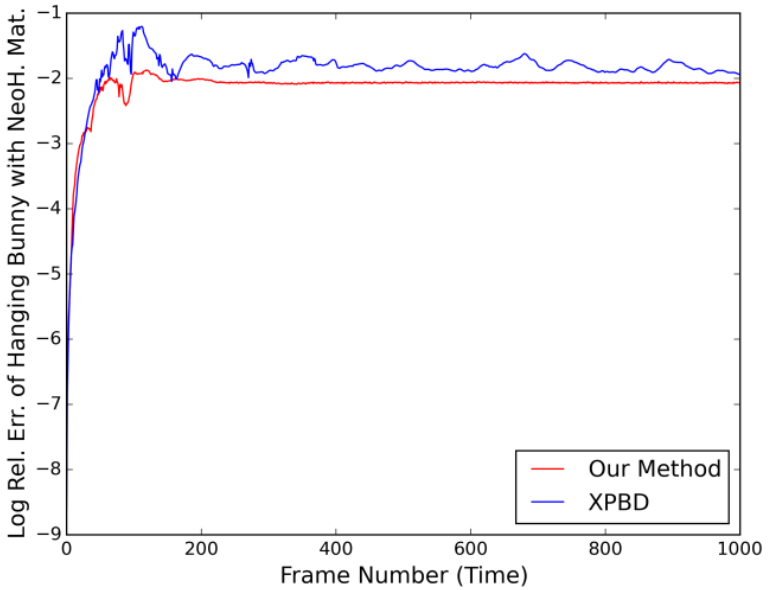
Results


XPBD
(St. Venant Kirchhoff Material Constraint)
Our Method
(St. Venant Kirchhoff Material Constraint)

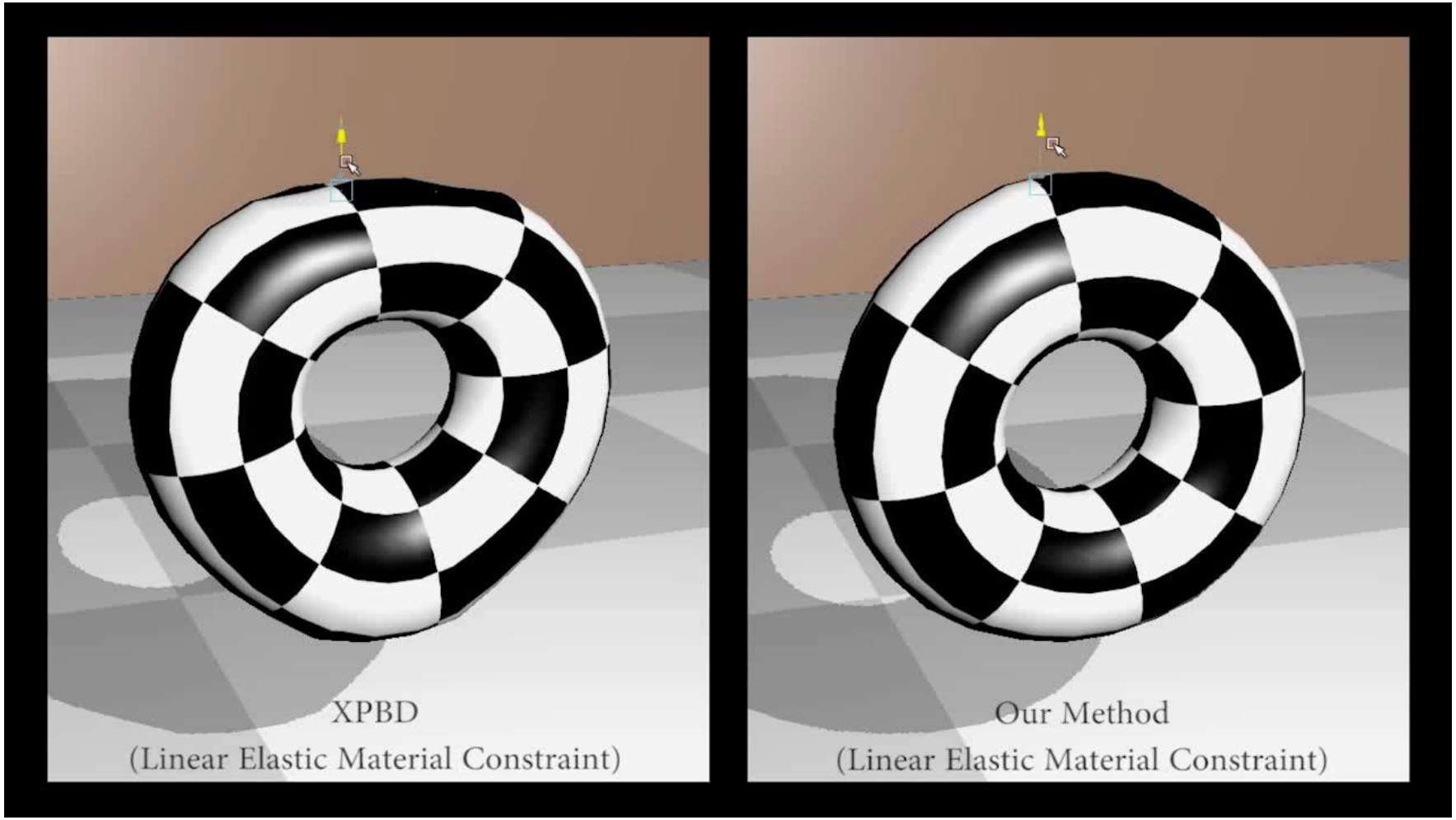

Log Rel. Err. of Hanging Bunny with STVK Mat.
−1
−2
−3
−4
−5
−6
−7
−8
0
200
400
600
800
1000
Frame Number (Time)
Our Method
XPBD

Results

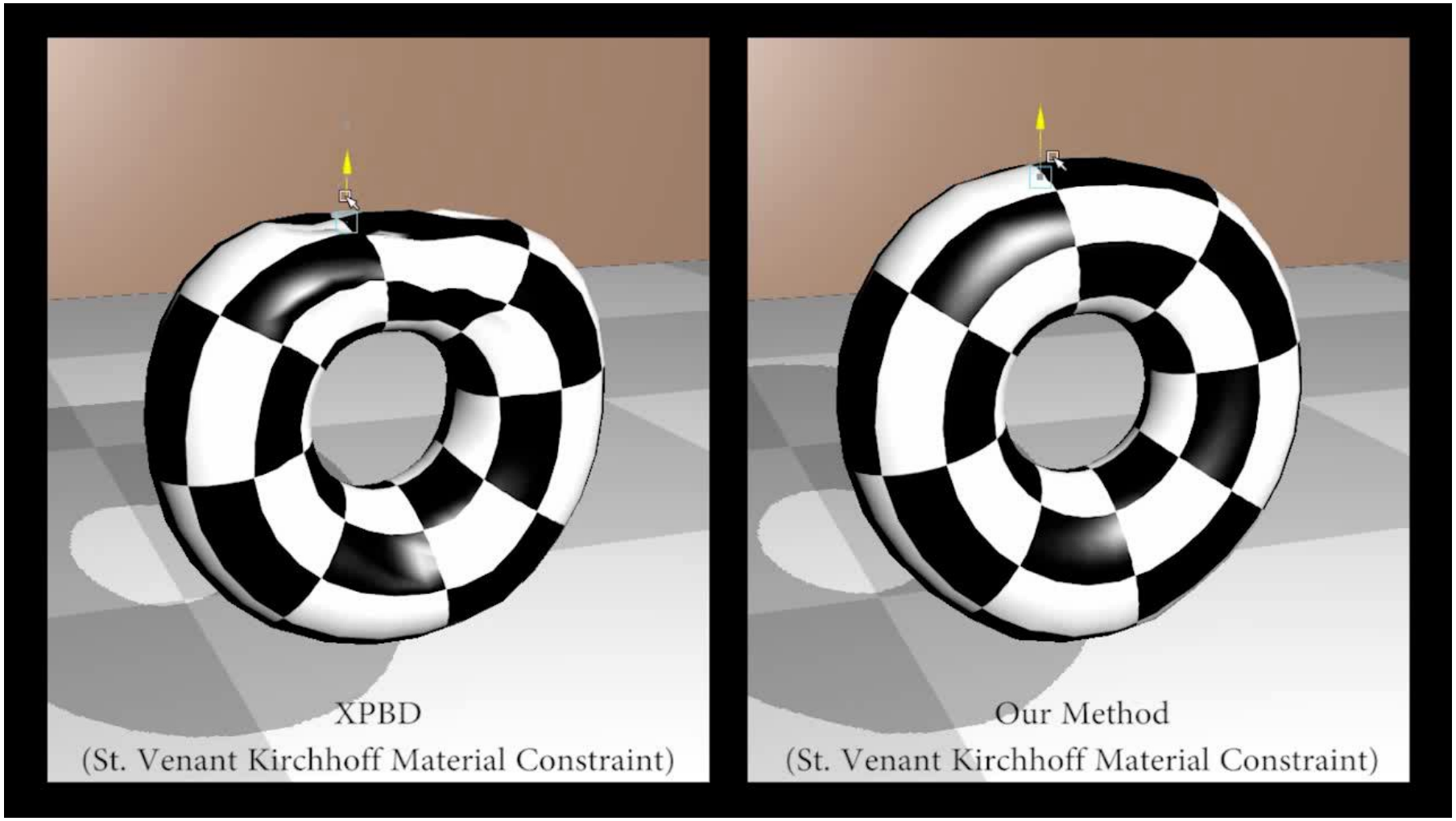

XPBD
(Neo - Hookean Material Constraint)
Our Method
(Neo - Hookean Material Constraint)


Log Rel. Err. of Hanging Bunny with NeoH. Mat.
−1
−2
−3
−4
−5
−6
−7
−8
−9
0
200
400
600
800
1000
Frame Number (Time)
Our Method
XPBD

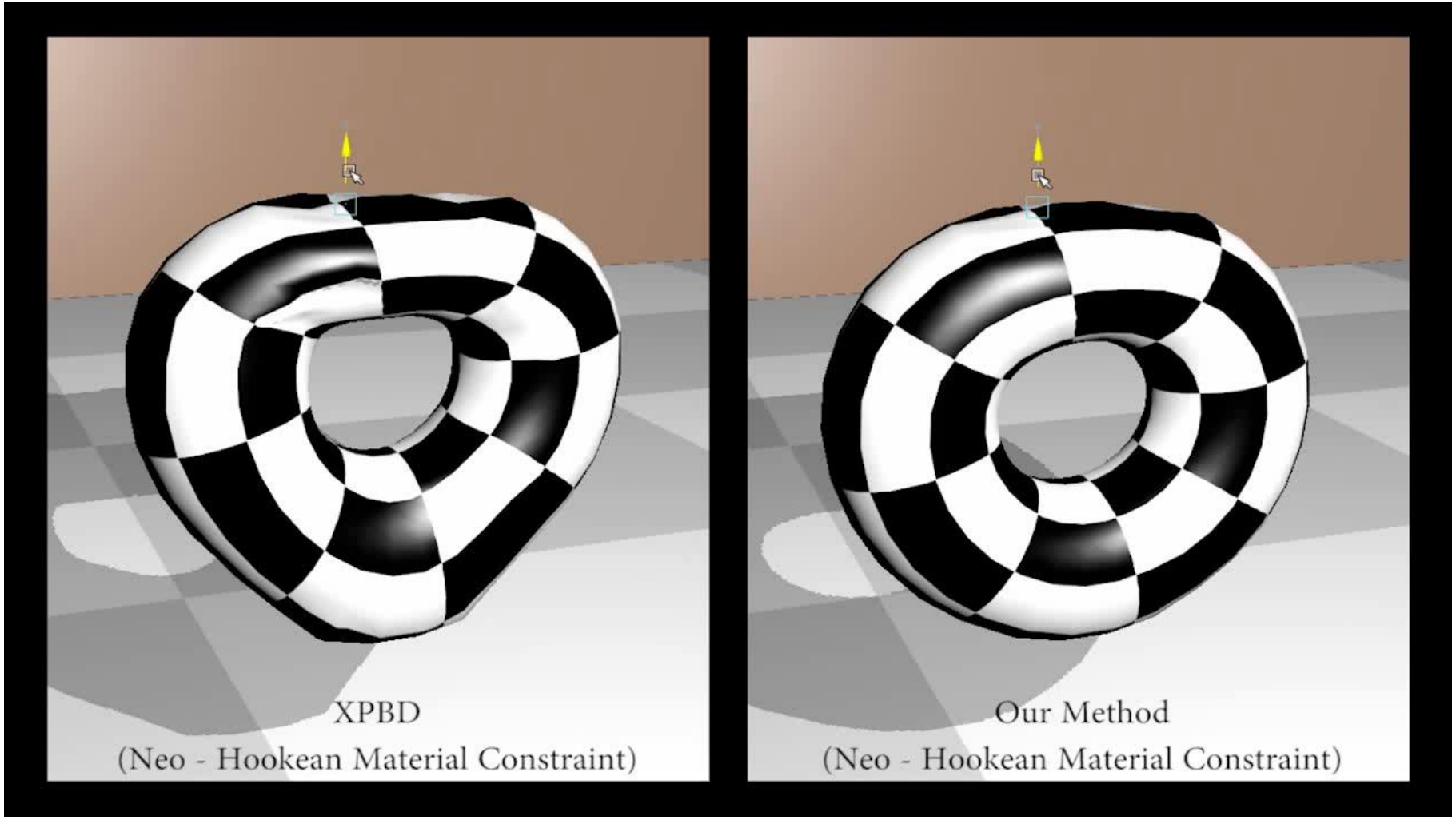
Results


XPBD
(Linear Elastic Material Constraint)
Our Method
(Linear Elastic Material Constraint)

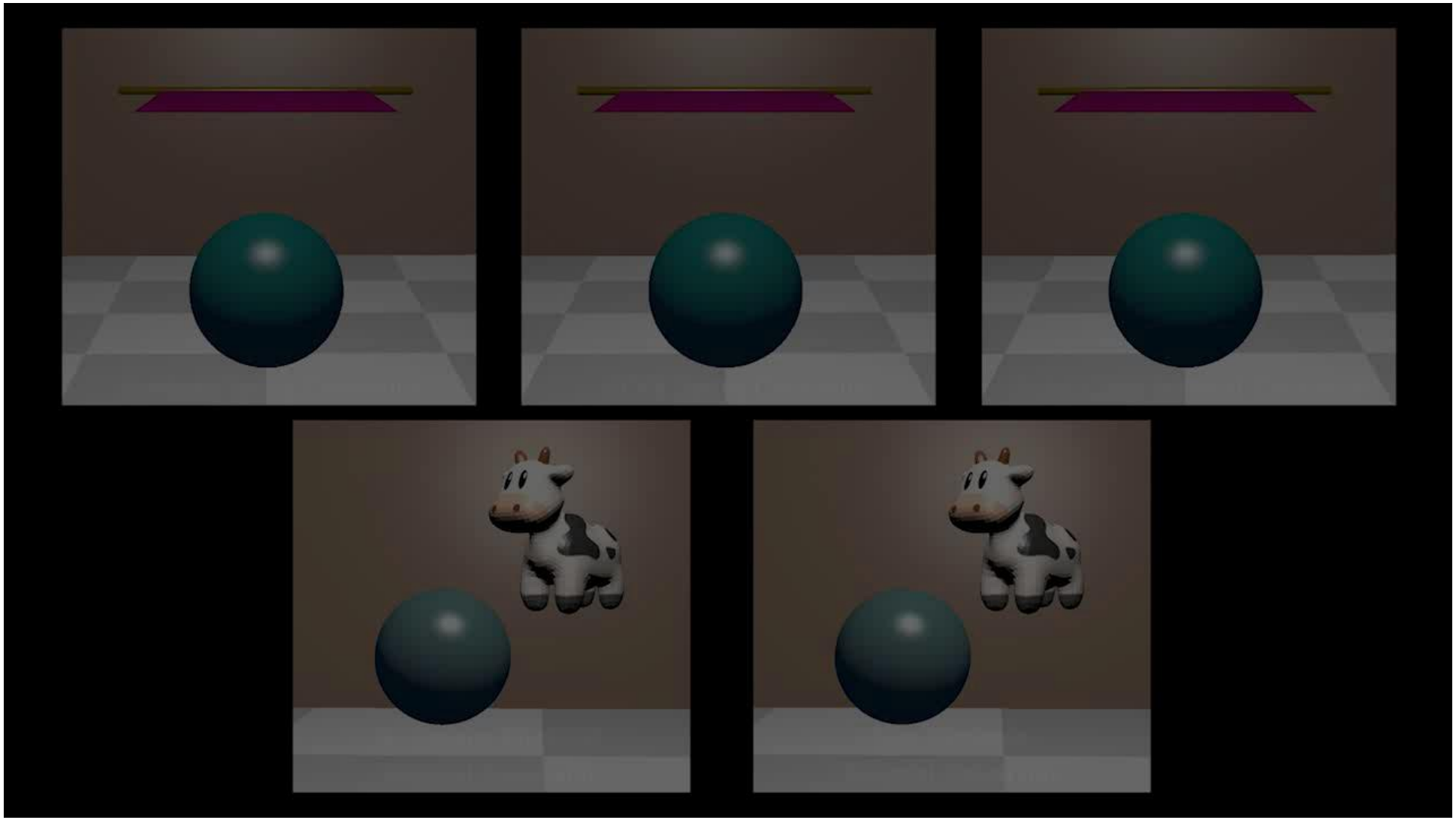
Results


XPBD
(St. Venant Kirchhoff Material Constraint)
Our Method
(St. Venant Kirchhoff Material Constraint)

Results


XPBD
(Neo - Hookean Material Constraint)
Our Method
(Neo - Hookean Material Constraint)

Results



Summary

- In this paper, we:
  - extend the Gauss-Seidel iteration of (X)PBD algorithm with the energy balances

Summary

- In this paper, we:
  - extend the Gauss-Seidel iteration of (X)PBD algorithm with the energy balances
  - present our intuitive Mesh Coloring Alg. for parallel processing on GPU

Summary

- In this paper, we:
  - extend the Gauss-Seidel iteration of (X)PBD algorithm with the energy balances
  - present our intuitive Mesh Coloring Alg. for parallel processing on GPU
  - present the results by using many spring and constitutive material potentials

Summary

- In this paper, we:
  - extend the Gauss-Seidel iteration of (X)PBD algorithm with the energy balances
  - present our intuitive Mesh Coloring Alg. for parallel processing on GPU
  - present the results by using many spring and constitutive material potentials
  - show that our method is:
    - straightforward, stable and easy to adapt to existing frameworks.

THANK YOU!