Numerical Fracture in the Material Point Method with Convective Particle Domain Interpolation

Michael A. Homel

Material Point Method Workshop

March 15, 2013

UNIVERSITY OF UTAH
COMPUTATIONAL SOLID MECHANICS

CHPC

Schlumberger

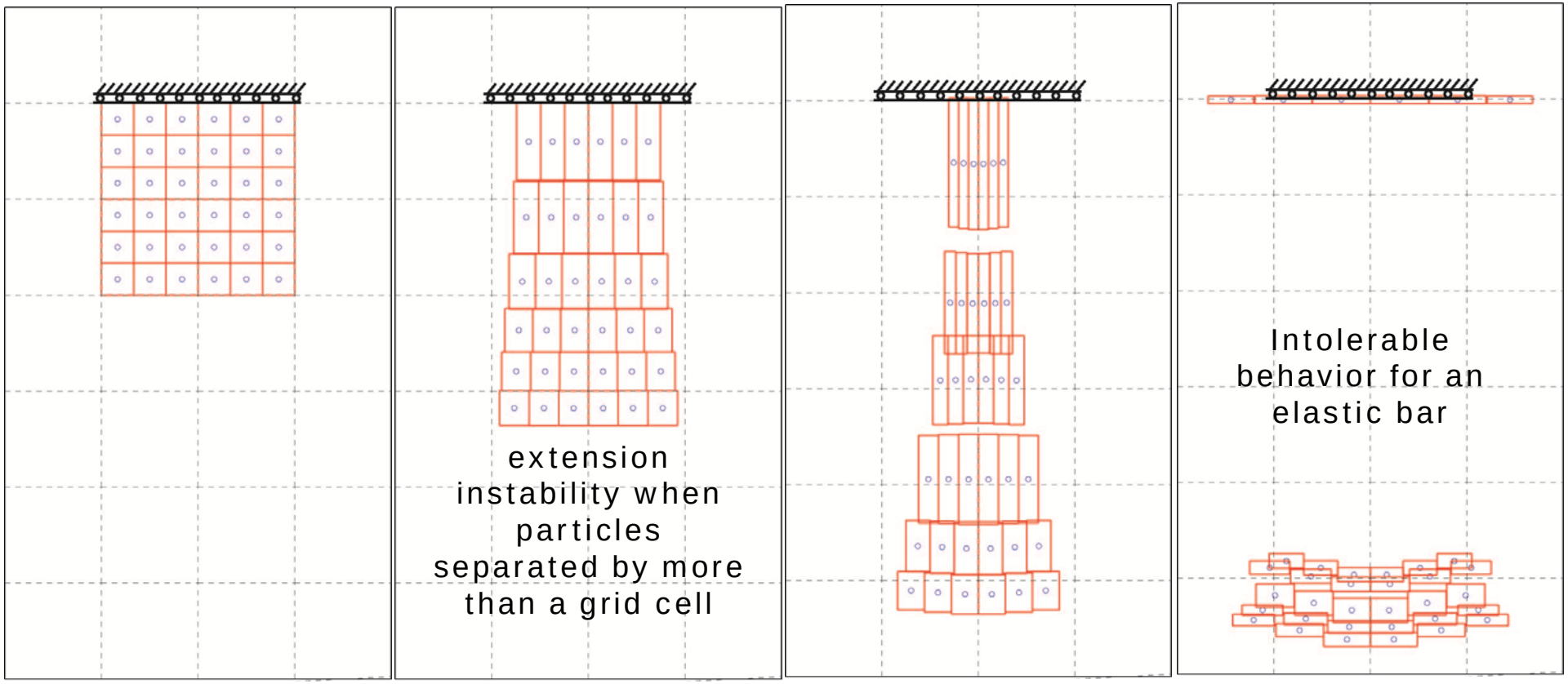
Outline

- Numerical Fracture in the Material Point Method
- Convective Particle Domain Interpolation (CPDI)
- Controlling Numerical Fracture with CPDI
- Implementation and Results

NUMERICAL FRACTURE IN MPM

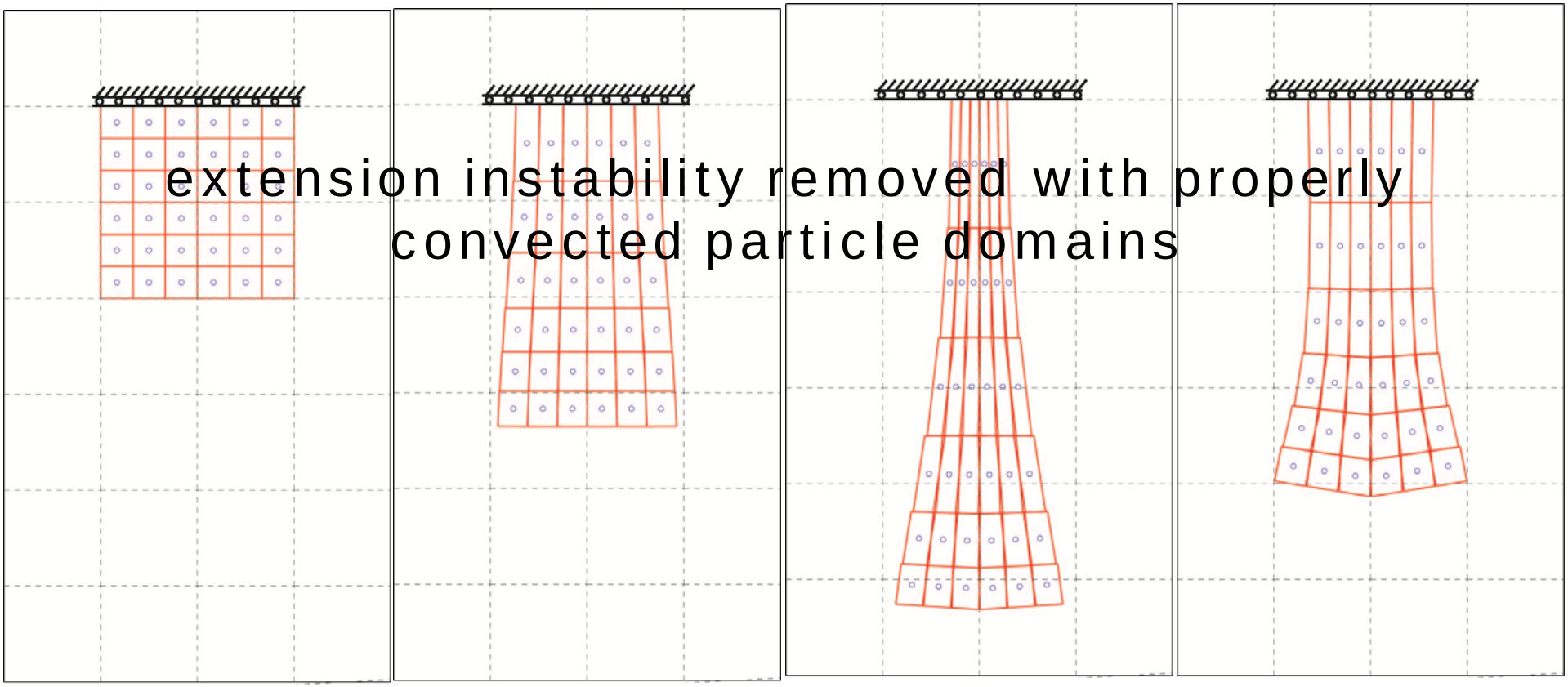
Extension Instability in a Self-Weighted Elastic Bar

PREVIOUSLY STATE-OF-THE-ART MPM METHODS



The cpGIMP method has extension instability despite particle stretching.
(other methods were even worse)

Extension Instability Eliminated with CPDI

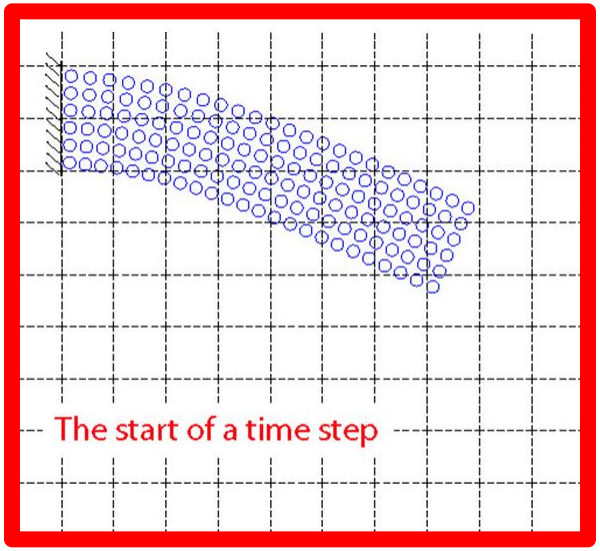


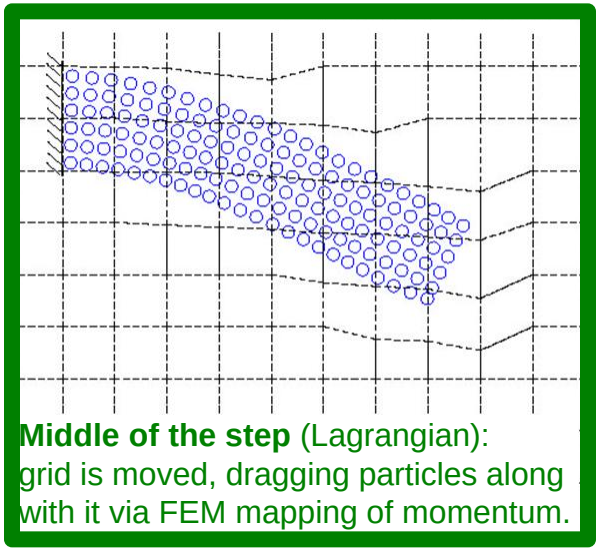
CPDI method

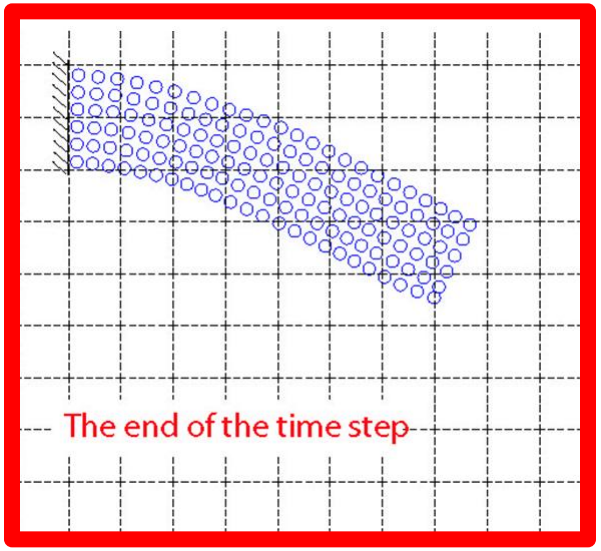
CONVECTIVE PARTICLE DOMAIN INTERPOLATION

Developed at the University of Utah by A. Sadeghirad, R.M. Brannon, J. Burghardt and J. Guilkey

MPM can be viewed as an arbitrary Lagrange-Eulerian finite-element method (ALE-FEM)



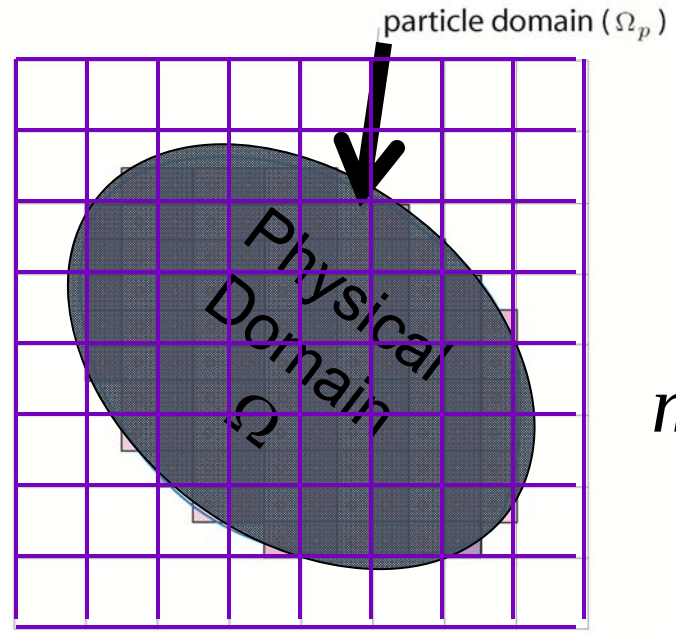




Rough algorithm:

- Map data (mass, momentum, and loads) from MPM particles to FEM grid nodes
- Solve discretized FEM momentum equations on the predefined background grid
- Use standard FEM mapping to map data from FEM grid nodes to MPM particles
- (optional) reset the FEM grid if you want particles to flow through the grid

Overview



A non-overlapping physical domain exists for each particle. These domains could be found by Voronoi tessellation. The tessellation exists, but an advantage of the MPM is that no precious CPU time is spent actually finding it.

Standard weak formulation applies on the grid, which is best regarded as a conventional FEM grid using conventional FEM nodal basis functions, $S_i(\mathbf{x})$:

$$\sum_{j=1}^{n} m_{ij}\boldsymbol{a}_j(t) = \boldsymbol{f}_i^{\text{int}}(t) + \boldsymbol{f}_i^{\text{ext}}(t)$$

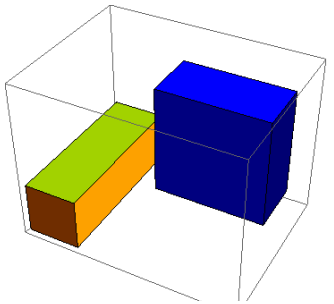
$$m_i = \int_{\Omega} S_i(\mathbf{x})\rho dV\,, \qquad \boldsymbol{f}_i^{\text{int}}(t) = -\int_{\Omega} \boldsymbol{\sigma}\cdot\nabla S_i(\boldsymbol{x})\ dV$$

These FEM integrals are all of the form: $\int_{\Omega} h(\boldsymbol{x})\rho dV$

FEM evaluates them (without loss) by a sum over element domains.
MPM evaluates them (without loss) by a sum over particle domains.
Simplest "standard" MPM unwisely uses single-point integration:

$$\int_{\Omega} h(\boldsymbol{x})\rho dV \equiv \sum_{p=1}^{N_p} \int_{\Omega_p} h(\boldsymbol{x})\rho dV$$

Existing material point methods are distinguished by how the FEM integrals are evaluated



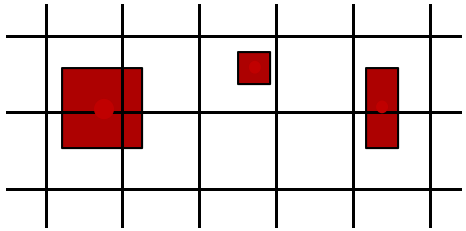
Contribution of p^{th} particle to i^{th} grid node:

$$m_{ip} = \int_{\Omega_p} S_i(\mathbf{x})\rho dV$$

$$\mathbf{f}_{ip} = \int_{\Omega_p} \boldsymbol{\sigma}\cdot\nabla S_i(\mathbf{x}) dV$$

- The "standard" MPM unwisely uses single point integration (controlled exclusively from value of integrand at the material point):

$$\varphi_{ip} \equiv S_i(\boldsymbol{x}_p) \qquad \nabla\varphi_{ip} \equiv \nabla S_i(\boldsymbol{x}_p)$$

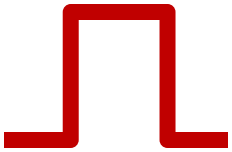


- The generalized interpolation material point (GIMP) method [Bardenhagen & Kober] uses a weighted average ("standard" MPM recovered via Dirac weight function):

$$\varphi_{ip} \equiv \overline{S_{ip}} = \frac{1}{V_p}\int_{\Omega_\chi} \chi_p(\boldsymbol{x}-\boldsymbol{x}_p) S_i(\boldsymbol{x}) d\boldsymbol{x}$$

$$\nabla\varphi_{ip} \equiv \overline{\nabla S_{ip}} = \frac{1}{V_p}\int_{\Omega_\chi} \chi_p(\boldsymbol{x}-\boldsymbol{x}_p) \nabla S_i(\boldsymbol{x}) d\boldsymbol{x}$$

Usually, the χ_p weight function is taken to be a "top hat" function.



GIMP: Has used rectangles (to date) Merits and drawbacks

Two existing (unsatisfactory) strategies

- uGIMP: the particle domains are assumed to be unchanged for all time.
- cpGIMP: particle domain sizes evolve with time such that they continue to be rectangular or cuboid which are aligned with the global coordinate system.

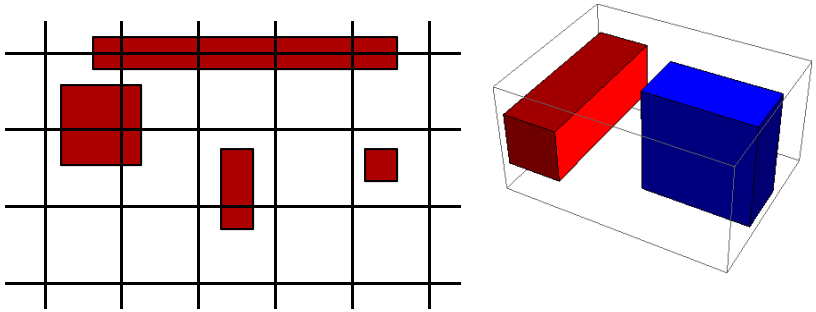
Advantage: GIMP prevents the so-called ringing instability.

Disadvantages:

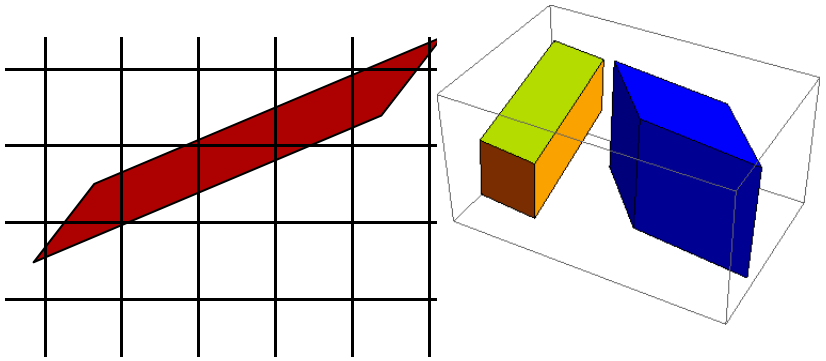
Existing GIMP formulations suffer extension instabilities (and cpGIMP is even unstable in small deformations with large rotations), which can be prevented by convecting the particle domain with deformation.

Exact evaluation of GIMP integrals is algorithmically complicated (takes a long time to program and debug) because of large variation of shape functions over particle domains

Motivation for an approximation of GIMP integrals



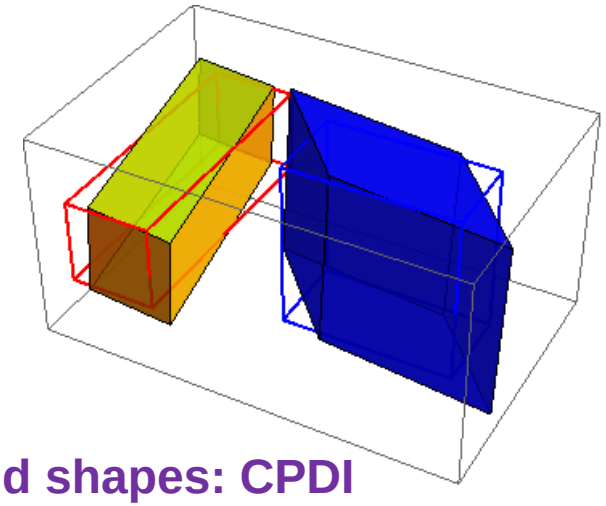
Limitation of existing GIMP implementations: Only rectangular or cuboid particle shapes are allowed.



The particle shape in the **convected particle domain** interpolation (**CPDI**) deforms with the material across many grid elements.

Goals of the CPDI method:

- More accurately track particle domains
- Recover stability in highly tensile deformation
- Relax assumptions about *initial* particle domain shape



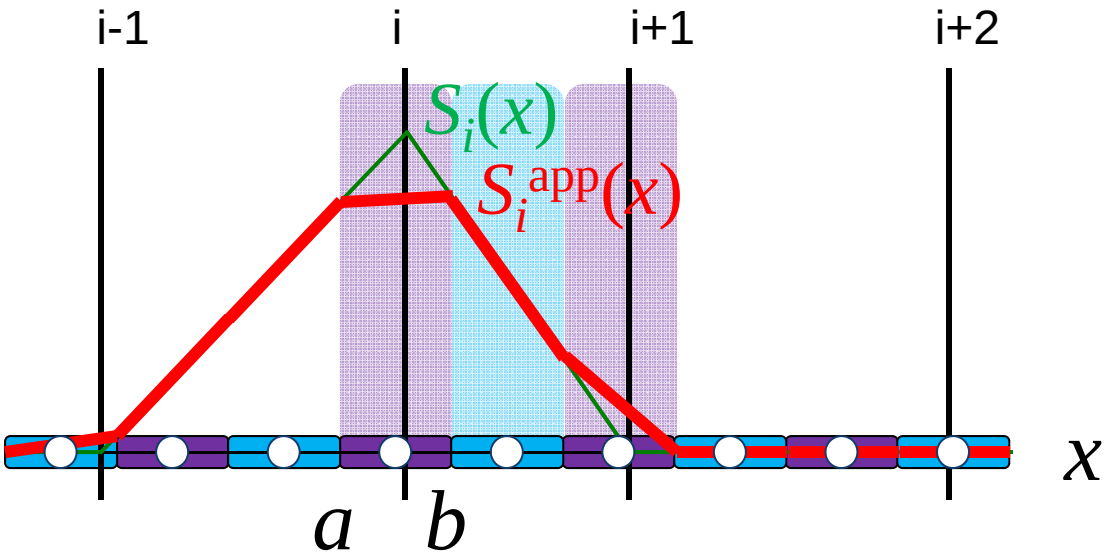
Solid shapes: CPDI
Wireframes: cpGIMP

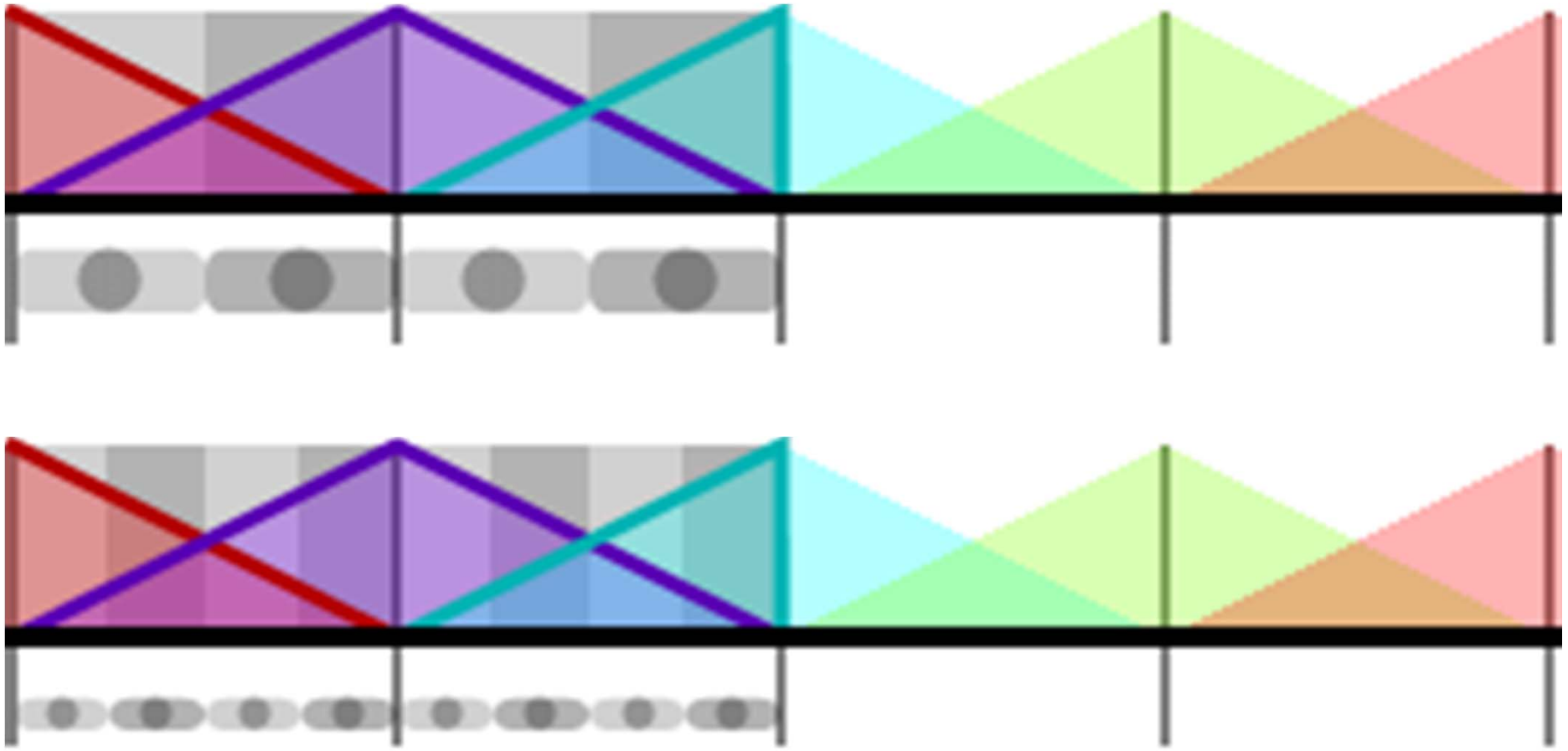
CPDI in 1D: Approximate GIMP integrals by approximating the shape function in the integrand – use shape function interpolation.

$$\underline{\mathbf{f}}_{ip} = \underline{\underline{\boldsymbol{\sigma}}}_p \cdot \underline{\mathbf{G}}_{ip} V_p \quad \text{where} \quad \underline{\mathbf{G}}_{ip} = \frac{1}{V_p} \int_{\Omega_p} \nabla S_i(\mathbf{x}) dV$$

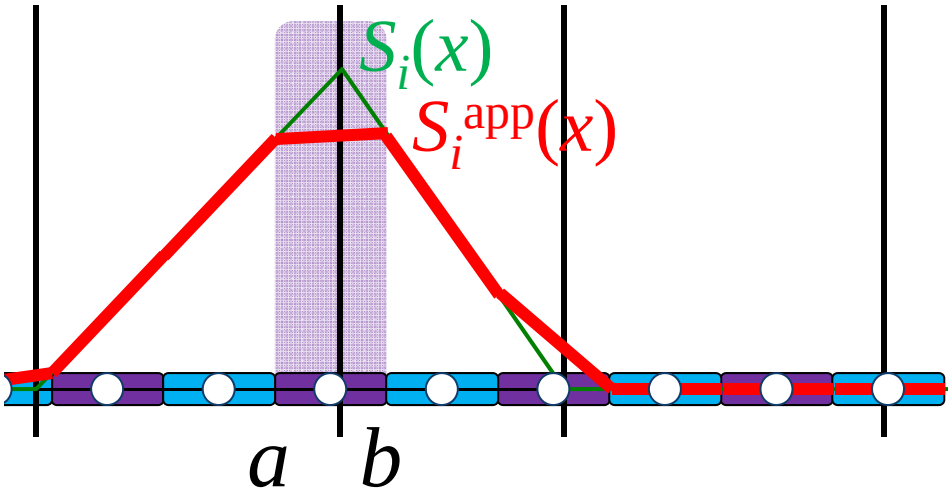
In 1-D, this reduces to $f_{ip} = \sigma_p G_{ip} V_p$ where

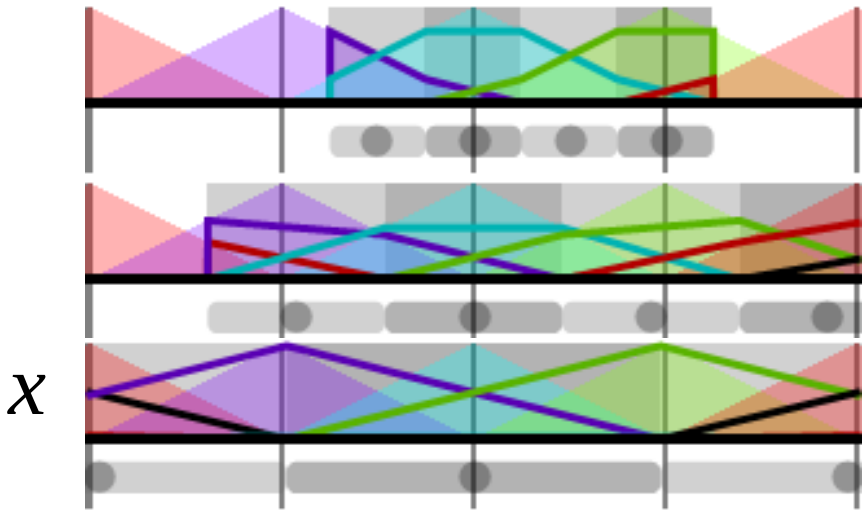
$$G_{ip} = \frac{1}{(b-a)} \int_a^b \frac{dS_i}{dx} dx$$



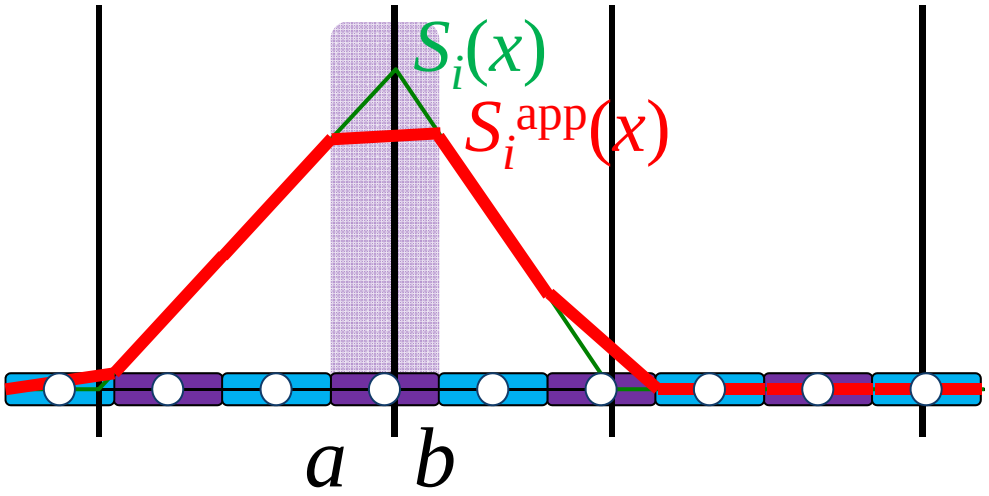


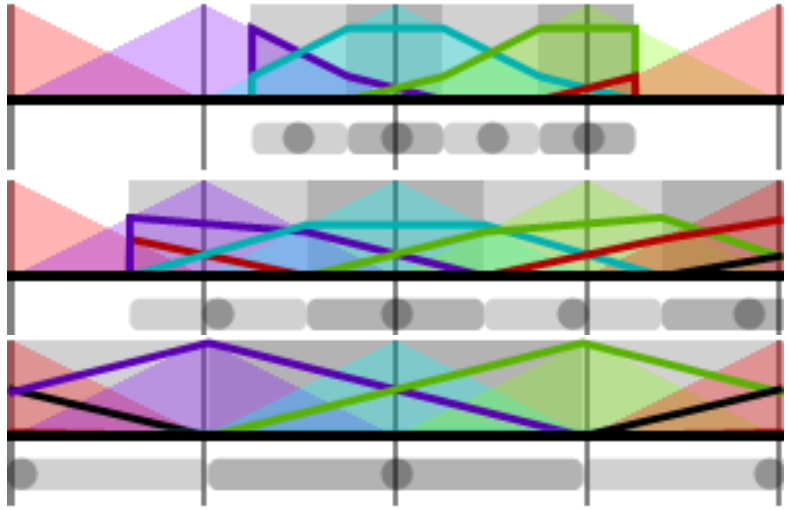
Dynamic approximate basis functions: Pure stretch

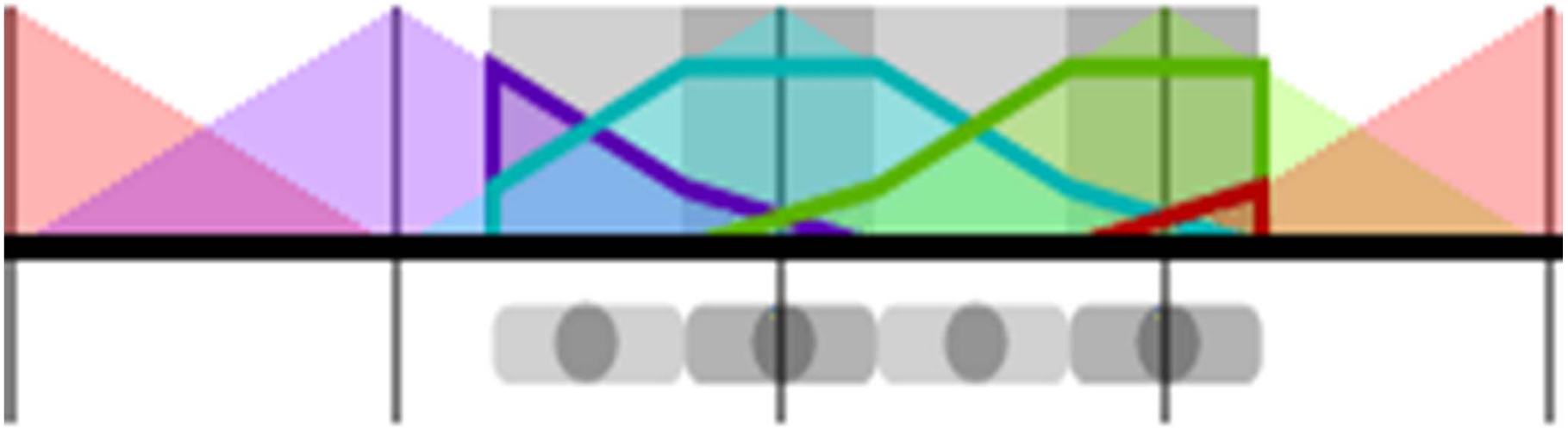




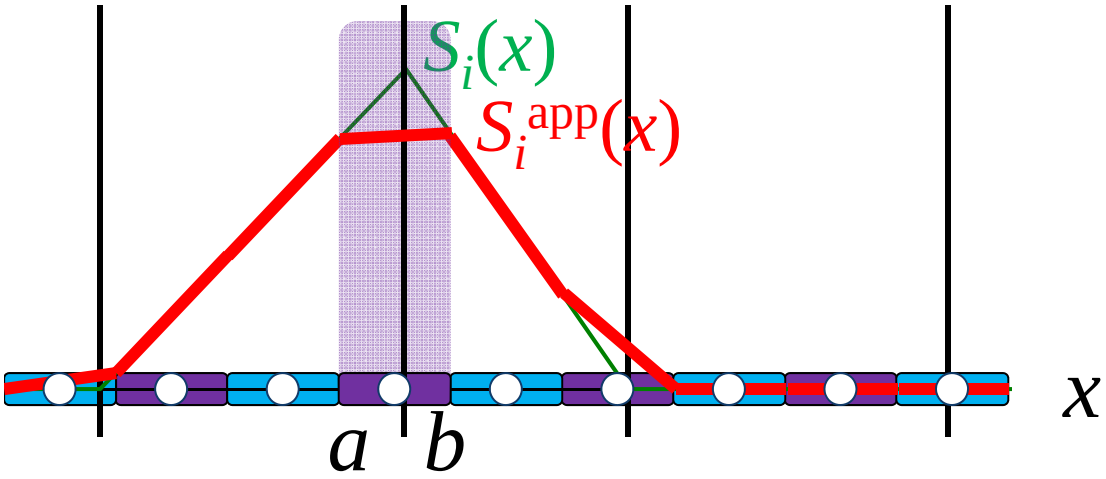
Dynamic approximate basis functions: Pure stretch

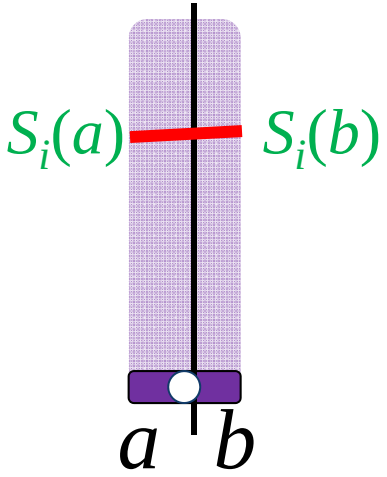


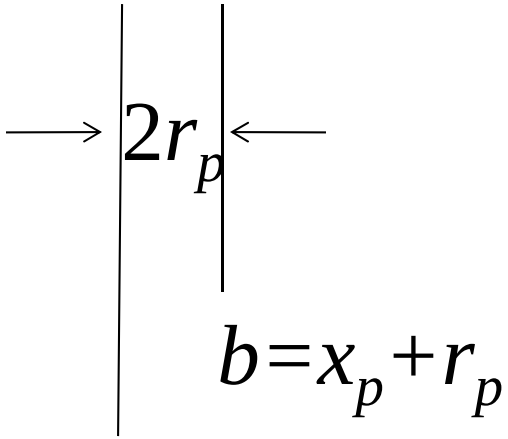




Particle domain interpolation is described using FEM basis functions Q(x) *on the particle*







Equation of straight segment :

$$S_i^{app}(x) = Q_a^p(x)S_i(a) + Q_b^p(x)S_i(b)$$

where

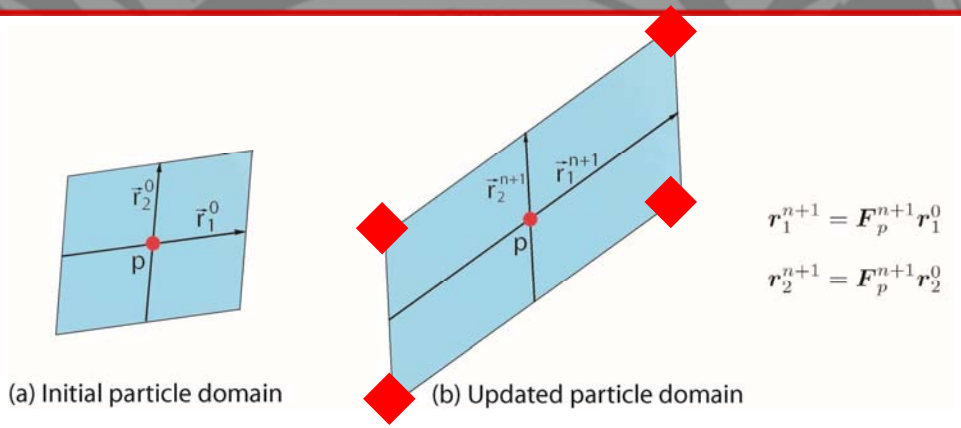
$$Q_a^p(x) = \frac{x-b}{a-b} = \frac{x-x_p-r_p}{-2r_p}$$

$$Q_b(x) = \frac{x-a}{b-a} = \frac{x-x_p+r_p}{2r_p}$$

CPDI method in 2-D

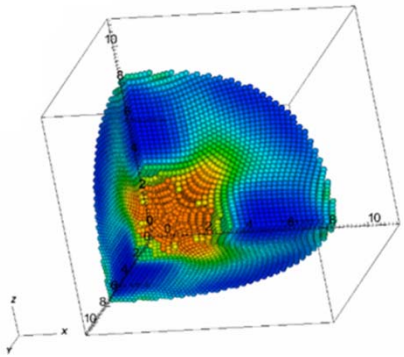
Optional parallelogram *initial* particle domains avoid unsavory "stair-steps" at angled boundaries.

Accurate *deformed* particle domains are found using the (already available) deformation tensor.



- CPDI uses approximate grid shape functions that are ordinary FEM-style interpolations of the actual shape functions over each particle domain:

$$S_i^{\text{app}}(\boldsymbol{x}) = \sum_{\alpha=1}^{4} Q_\alpha^p(\boldsymbol{x}) S_i(\boldsymbol{x}_\alpha^p) \quad \text{on} \quad \Omega_p$$



- Corresponding approximate FEM integrals:

$$\varphi_{ip} \equiv \overline{S_{ip}} \cong \frac{1}{V_p} \int_{\Omega_p} S_i^{\text{app}}(\boldsymbol{x}) d\boldsymbol{x} = \frac{1}{V_p} \sum_{\alpha=1}^{4} \left(\int_{\Omega_p} Q_\alpha^p(\boldsymbol{x}) d\Omega \right) S_i(\boldsymbol{x}_\alpha^p)$$

$$= \frac{1}{4} \left\{ S_i(\boldsymbol{x}_1^p) + S_i(\boldsymbol{x}_2^p) + S_i(\boldsymbol{x}_3^p) + S_i(\boldsymbol{x}_4^p) \right\}$$

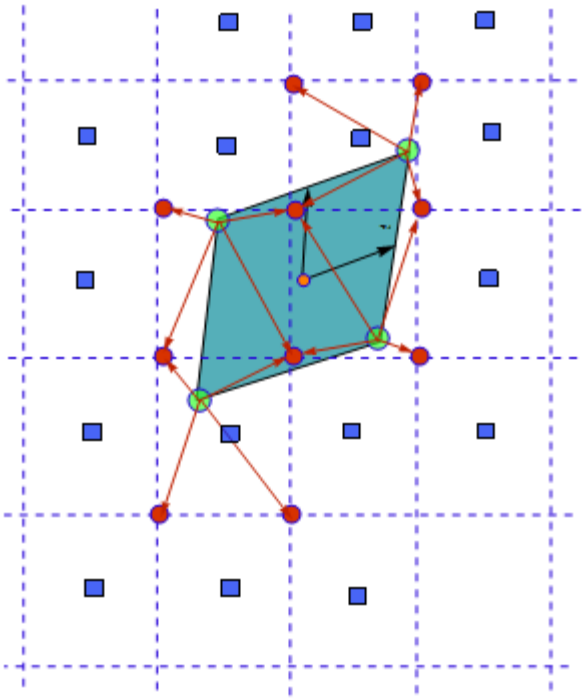
$$\nabla \varphi_{ip} \equiv \overline{\nabla S_{ip}} \cong \frac{1}{V_p} \int_{\Omega_p} \nabla S_i^{\text{app}}(\boldsymbol{x}) d\boldsymbol{x} = \frac{1}{V_p} \sum_{\alpha=1}^{4} \left(\int_{\Omega_p} \nabla Q_\alpha^p(\boldsymbol{x}) d\Omega \right) S_i(\boldsymbol{x}_\alpha^p)$$

CONTROLLING NUMERICAL FRACTURE

Motivation

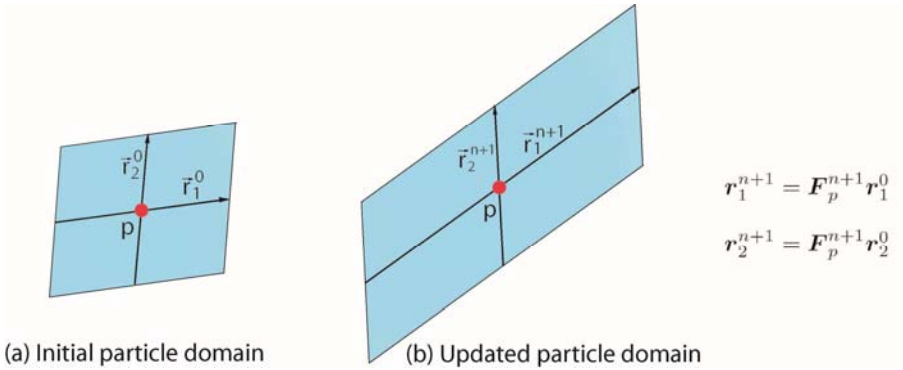
CPDI eliminates the extension instability, allowing particles to stretch across many grid cells without inducing numerical fracture. While this behavior may be desirable, there are times when it is beneficial to allow the material to separate.

- If the constitutive model allows the material to accumulate damage and a corresponding significant loss of strength, unrealistically large particle stretches may evolve. Ideally, a constitutive model could trigger fracture in the host code, inducing separation of particles under tensile stress when the damage has reached a critical value.
- Even without a damage model, it may be desirable to avoid extreme deformation states, for which a constitutive model may not be reasonably expected to converge.
- For a parallel implementation of CPDI, there is typically a boundary of ghost cells around each patch that is used to pass data between processors. The data structure may not support a particle whose domain extends beyond one or two grid cells, in particular if that particle is passing between patches.



CPDI Domain Freezing

- The CPDI method constructs an alternative shape function by evaluating the grid shape function at the corners of the particle domain.
- The corner locations are computed from the **r** vectors, transformed by the deformation gradient.
- By freezing the CPDI domain in some specified state, so that it is no longer tied to the deformation gradient, it becomes possible for the domain of adjacent particles to become separated by a background grid cell, thus inducing fracture.
- For example, if the CPDI domain is the initial particle domain, the GIMP method is recovered, with the fracture properties previously described.
- Thus implementing a method to freeze or scale the CPDI domains provides a means to control the numerical fracture behavior.



Algorithm Considerations

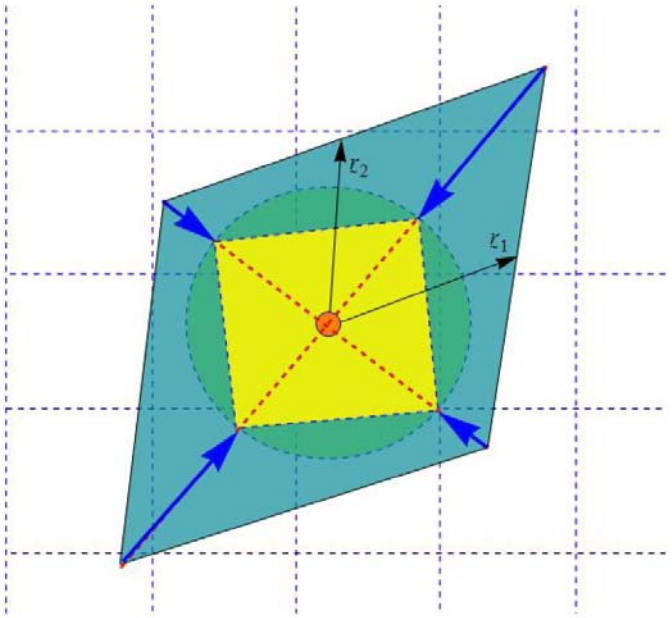
Desirable characteristics for the domain freezing algorithm

- Computationally efficient, without significant memory or requirements.
- Preserve information about material stretch and rotation, even if the material is rotated after the domain freeze is triggered.
- Ability to comply with the parallelization requirements of the host code.
- Ability to be triggered by the constitutive model if material is damaged.

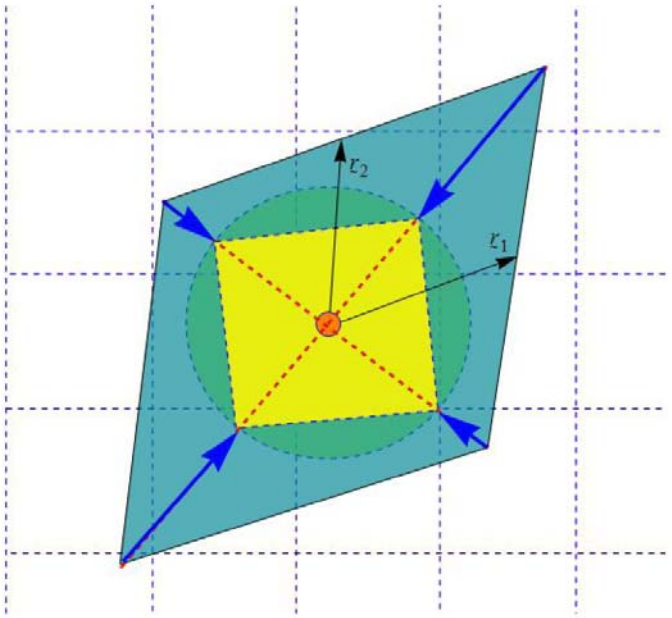
Approach

One approach is to scale CPDI domain so the particle is contained within a specified radius from cell center, preserving the character of the rotation and deformation state without the expense of a polar decomposition.

1. The relative position of the domain corners is computed from combining the **r** vectors.
2. If the length of these center-to-corner vectors exceeds an allowable limit, they are scaled back to lie within the allowable distance.
3. A new deformation gradient is constructed to match the new corner position, for use in the CPDI calculations, but this does not affect the material deformation gradient used to compute the stress in the constitutive model.



2D Algorithm



1. Compute critical length for parallelization

$$l_{\text{cr}} = 2\,\text{Min}(\Delta x,\ \Delta y)$$

2. Compute vectors to domain corners

$$\underset{\sim}{l}_a = \underset{\sim}{r}_1 + \underset{\sim}{r}_2 \qquad \underset{\sim}{l}_b = \underset{\sim}{r}_1 - \underset{\sim}{r}_2$$

3. Scale corner vectors if needed

$$\text{If}\left(\| l_a \| > l_{\text{cr}},\ \underset{\sim}{l}'_a = \underset{\sim}{l}_a \frac{l_{\text{cr}}}{\| l_a \|} \right)$$

$$\text{If}\left(\| l_b \| > l_{\text{cr}},\ \underset{\sim}{l}'_b = \underset{\sim}{l}_b \frac{l_{\text{cr}}}{\| l_b \|} \right)$$

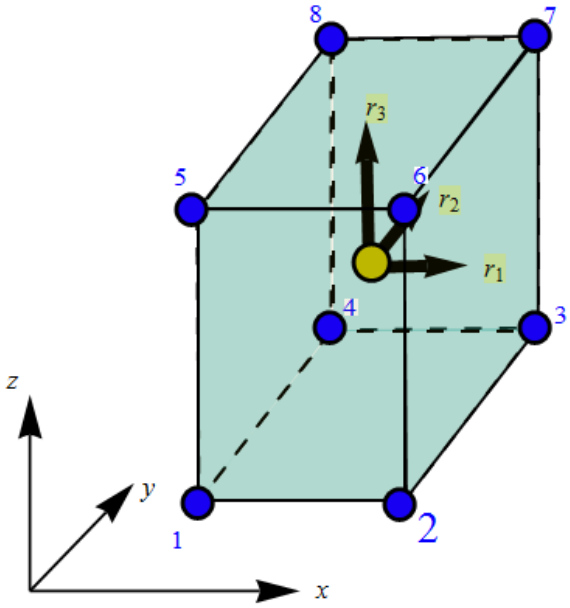
4. Reconstructed the $\boldsymbol{r}'$ vectors are from the $\boldsymbol{l}'$ vectors

$$\underset{\sim}{r}'_1 = \frac{1}{2}\left(\underset{\sim}{l}'_a + \underset{\sim}{l}'_b\right) \qquad \underset{\sim}{r}'_2 = \frac{1}{2}\left(\underset{\sim}{l}'_a - \underset{\sim}{l}'_b\right)$$

5. Reconstruct the deformation gradient

$$\underset{\approx}{F}' = \left\{ \underset{\sim}{r}'_1 \mid \underset{\sim}{r}'_2 \right\}$$

3D Algorithm



1. Compute critical length for parallel implementation

$$l_{\text{cr}} = 2\,\text{Min}(\Delta x,\ \Delta y,\ \Delta z)$$

2. Compute vectors to domain corners

$$\underset{\sim}{l}_a = \underset{\sim}{r}_1 + \underset{\sim}{r}_2 + \underset{\sim}{r}_3 \qquad \underset{\sim}{l}_b = \underset{\sim}{r}_1 - \underset{\sim}{r}_2 + \underset{\sim}{r}_3$$

$$\underset{\sim}{l}_c = -\underset{\sim}{r}_1 + \underset{\sim}{r}_2 + \underset{\sim}{r}_3 \qquad \underset{\sim}{l}_d = -\underset{\sim}{r}_1 - \underset{\sim}{r}_2 + \underset{\sim}{r}_3$$

3. Scale corner vectors if needed

$$\text{If}\left(\| l_i \| > l_{\text{cr}},\ \underset{\sim}{l}'_i = \underset{\sim}{l}_i \frac{l_{\text{cr}}}{\| l_i \|} \right),\ i = \{a, b, c, d\}$$

4. Reconstructed the $\boldsymbol{r}'$ vectors are from the $\boldsymbol{l}'$ vectors, however, here we must use a pseudo inverse to compute the solution.

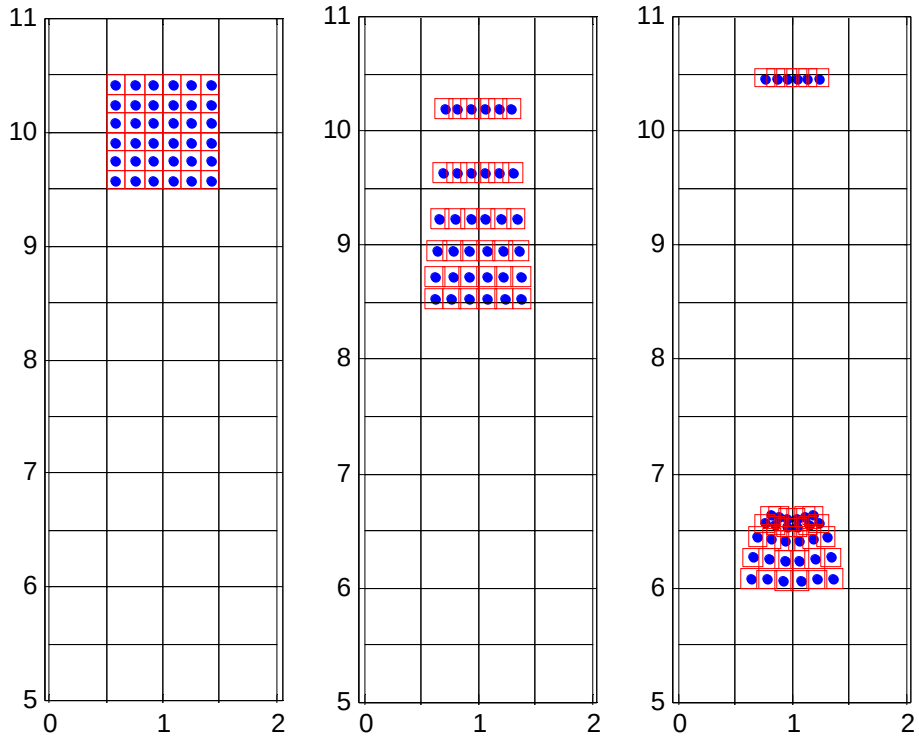
$$\underset{\sim}{r}'_1 = \frac{1}{4}\left(\underset{\sim}{l}_a + \underset{\sim}{l}_b - \underset{\sim}{l}_c - \underset{\sim}{l}_d\right) \qquad \underset{\sim}{r}'_2 = \frac{1}{4}\left(\underset{\sim}{l}_a - \underset{\sim}{l}_b + \underset{\sim}{l}_c - \underset{\sim}{l}_d\right) \qquad \underset{\sim}{r}'_3 = \frac{1}{4}\left(\underset{\sim}{l}_a + \underset{\sim}{l}_b + \underset{\sim}{l}_c + \underset{\sim}{l}_d\right)$$

5. Reconstruct the deformation gradient

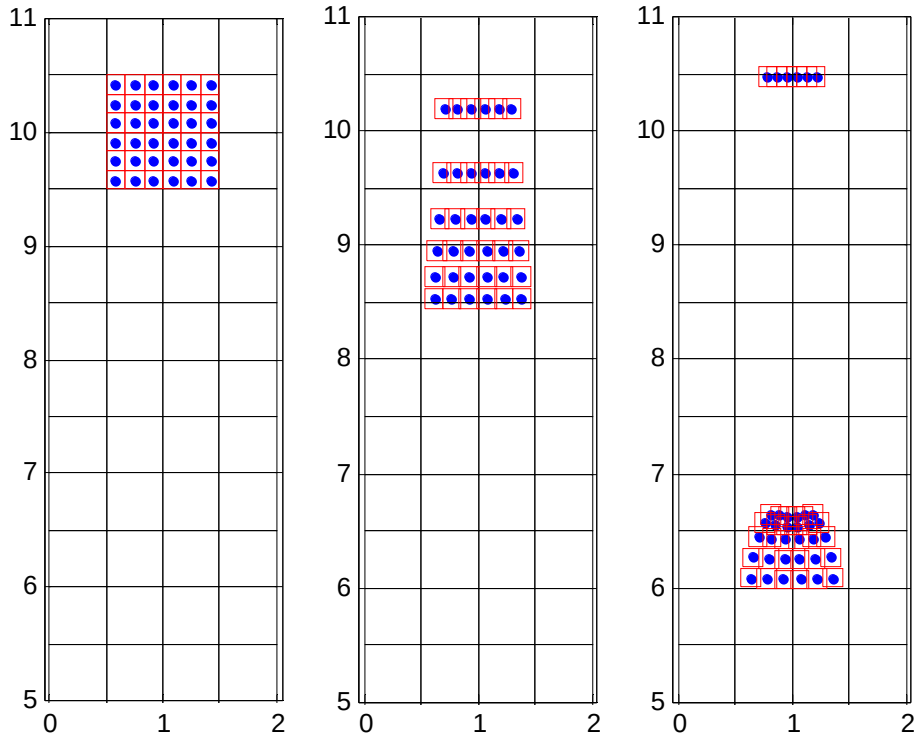
$$\underset{\approx}{F}' = \left\{ \underset{\sim}{r}'_1 \mid \underset{\sim}{r}'_2 \mid \underset{\sim}{r}'_3 \right\}$$

IMPLEMENTATION RESULTS

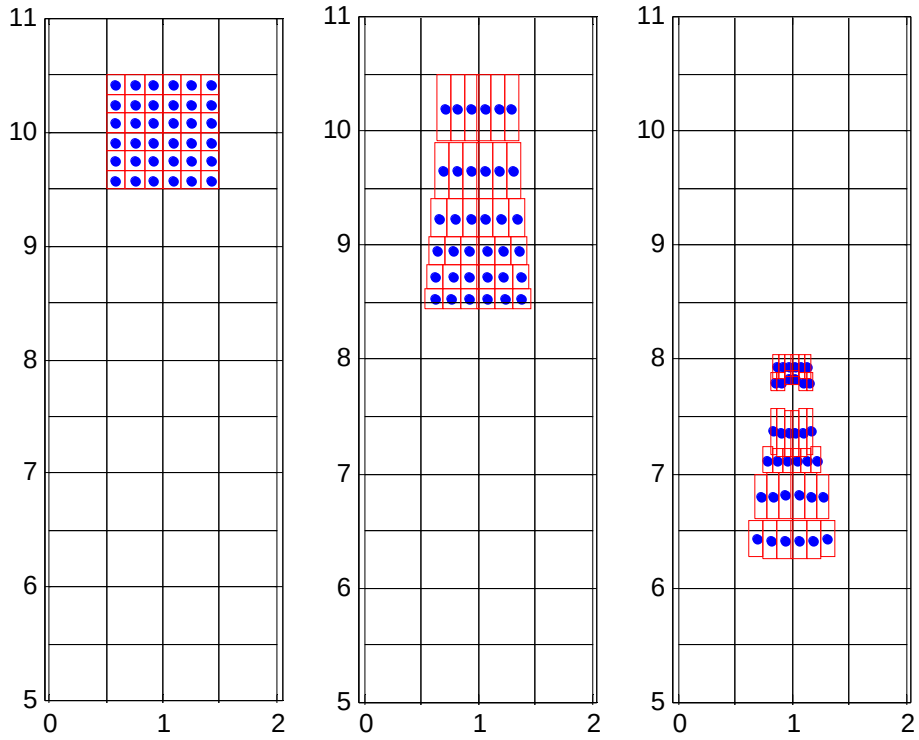
MPM Extension Instability



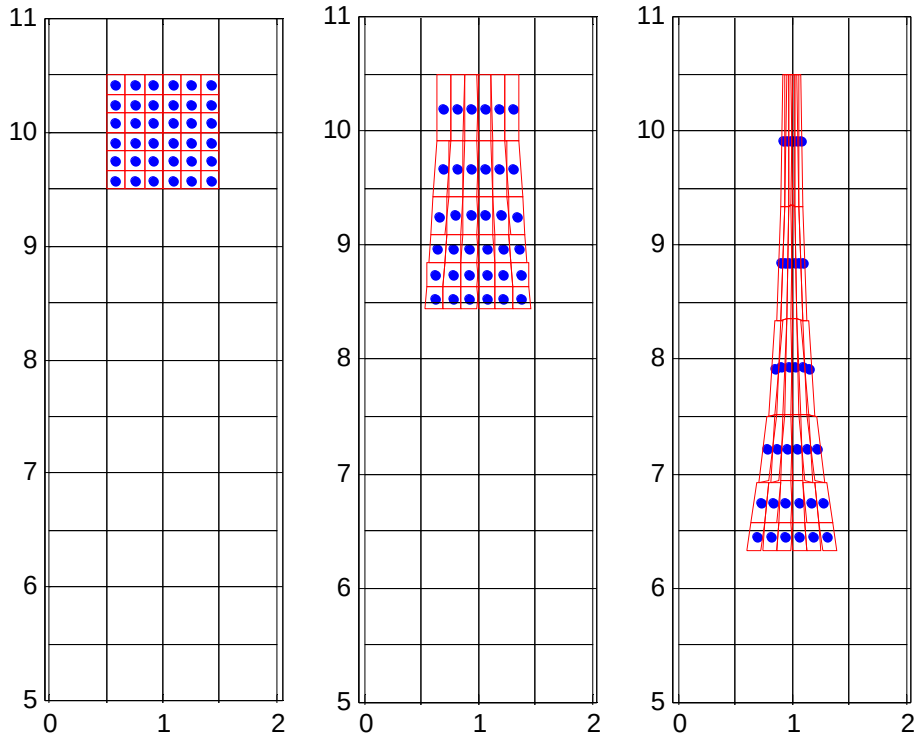
uGIMP Extension Instability



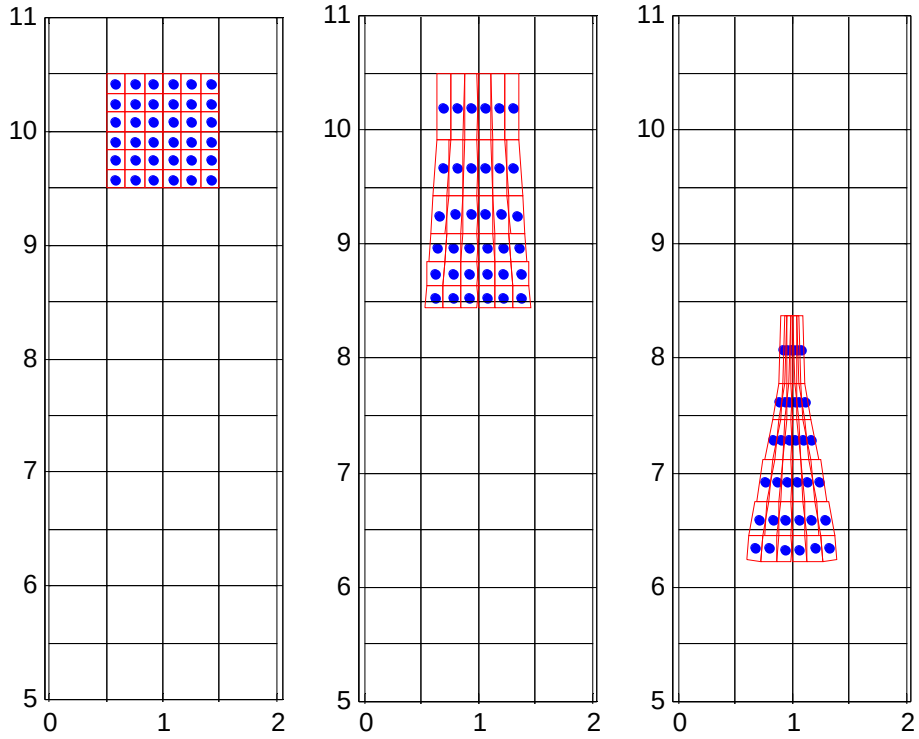
cpGIMP Extension Instability



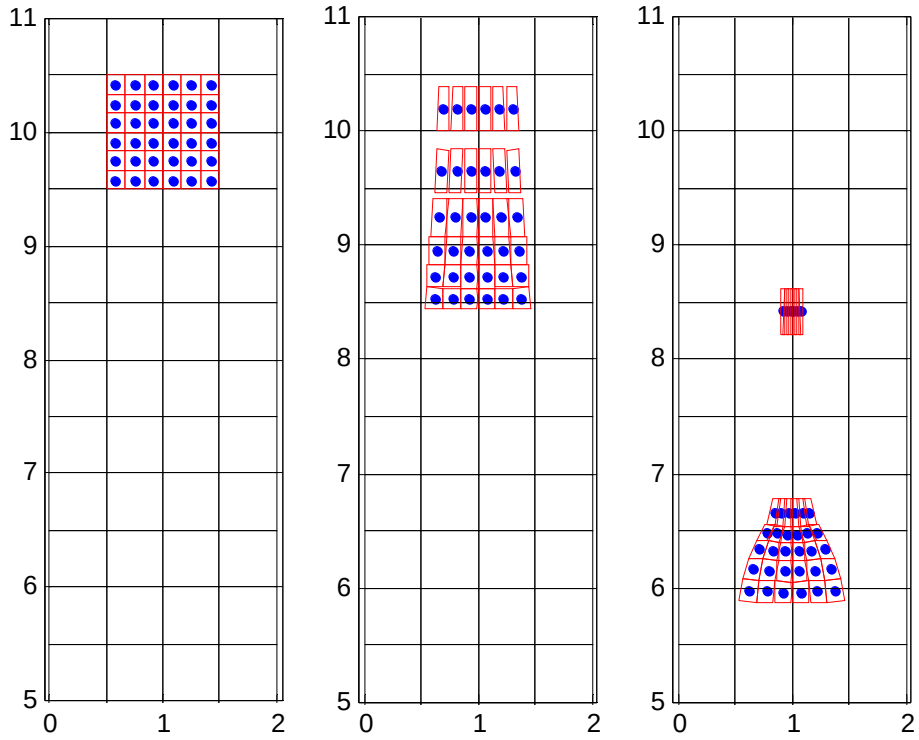
CPDI Extension Instability with $l_{cr} = 5.0\frac{\Delta x}{3}$



CPDI Extension Instability with $l_{cr} = 1.8\frac{\Delta x}{3}$



CPDI Extension Instability with $l_{cr} = 1.2\frac{\Delta x}{3}$



Summary

- It can be desirable to allow numerical fracture either to simulate material failure or to satisfy requirements of the host code data structure in parallelization
- CPDI can be modified to enforce an integration domain freezing/scaling that will lead to numerical fracture under large tensile strain
- Fracture behavior is dependent on the alignment of the particles with the background grid
- Minimum stretch at separation is still quite large, but coupled with a damage model (loss of strength), physically justified fracture behavior can be created.
- Overall, this approach enables the improved accuracy of CPDI in parallel simulations.

Questions