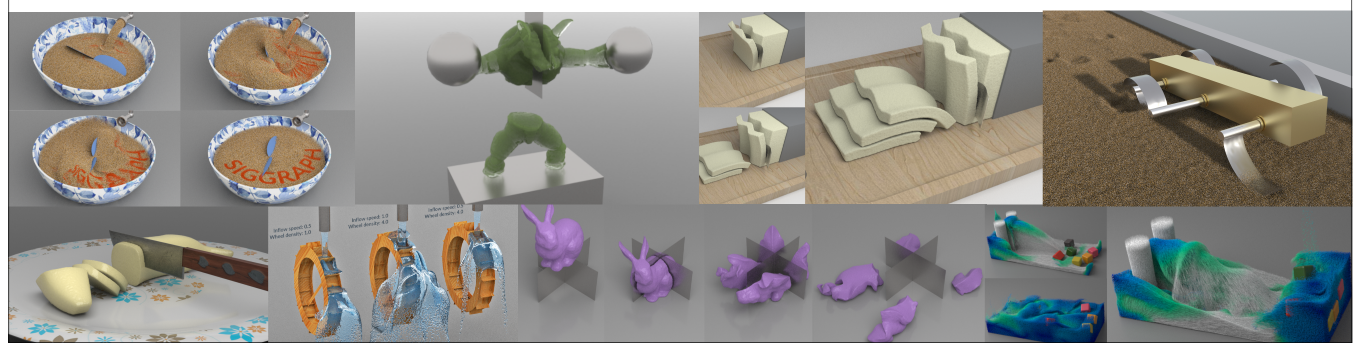
A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling

SIGGRAPH 2018

Yuanming Hu[1] Yu Fang[2] Ziheng Ge[3] Ziyin Qu[4] Yixin Zhu[5]
Andre Pradhana[4] Chenfanfu Jiang[4]

[1]MIT CSAIL [2]Tsinghua University [3]University of Science and Technology of China
[4]University of Pennsylvania [5]UCLA

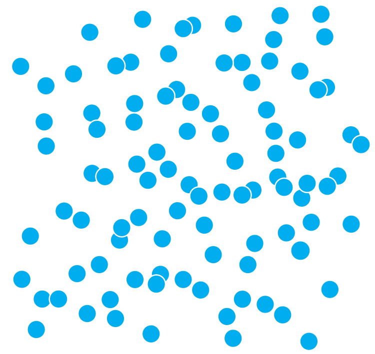


Thank you Florence for the introduction and thank you all for coming. I'm Yuanming from MIT.

The Material Point Method (MPM)

MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.

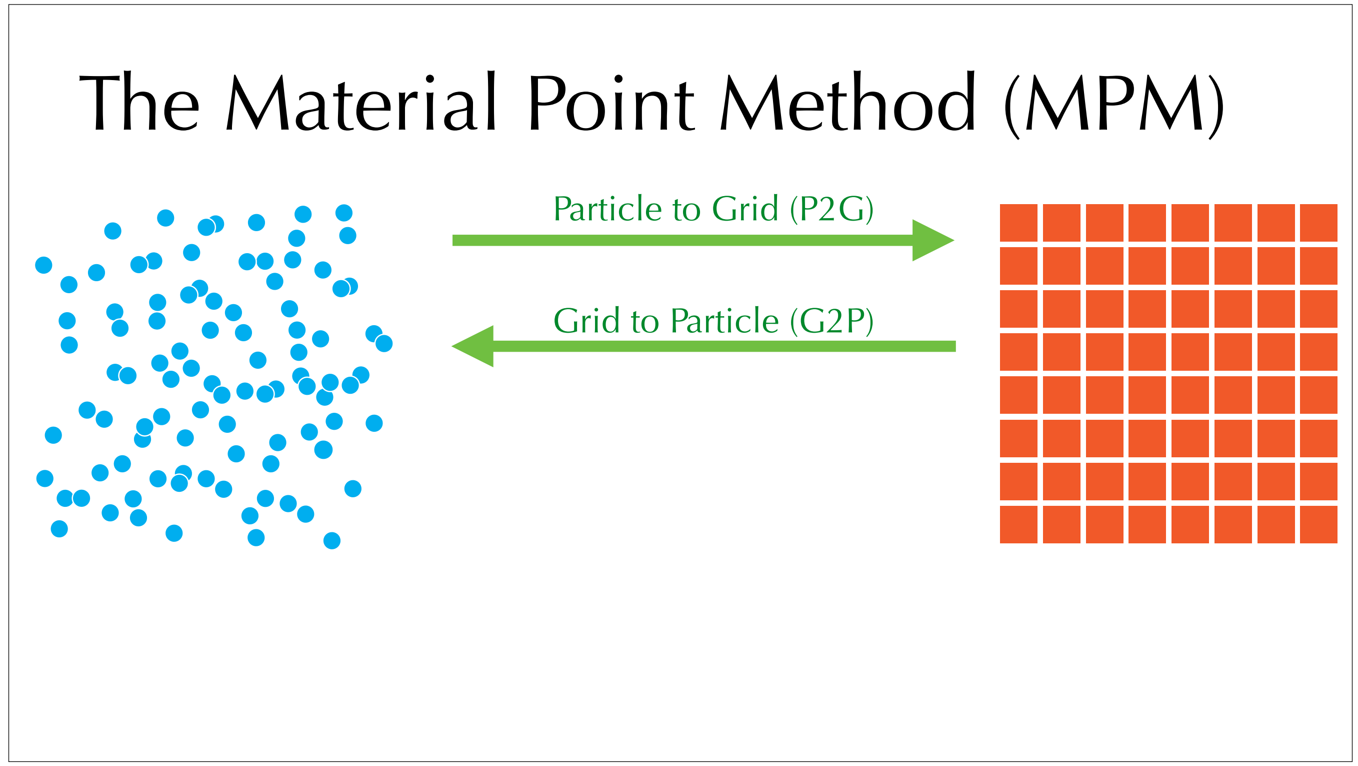
The Material Point Method (MPM)



MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.

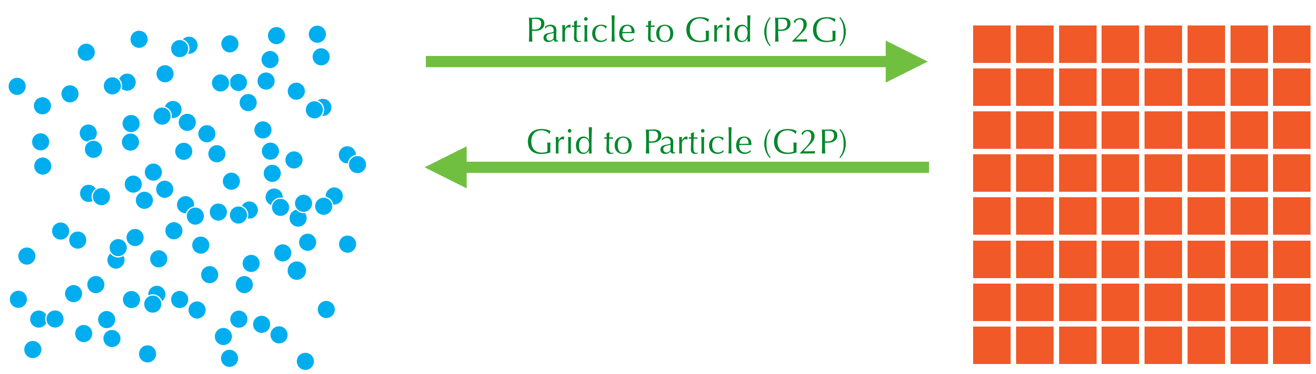
The Material Point Method (MPM)

MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.



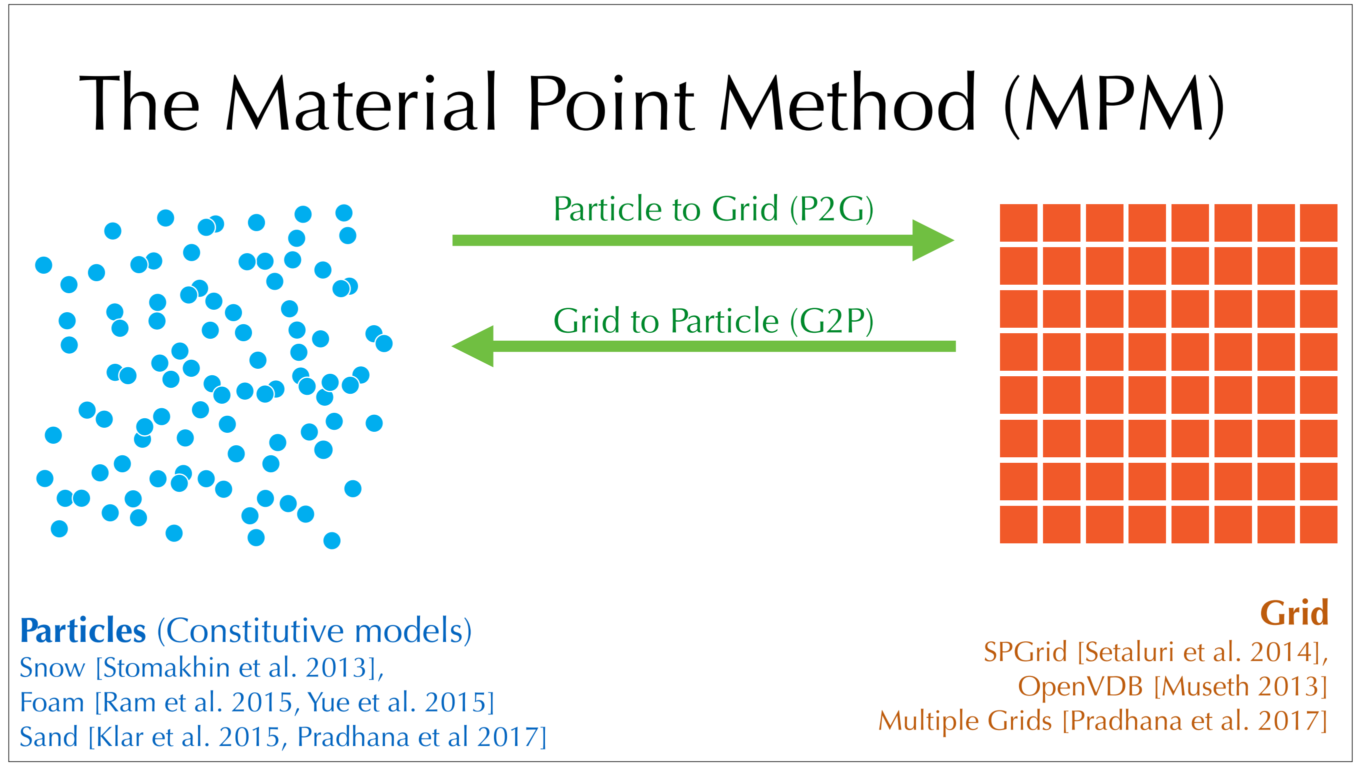
MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.

The Material Point Method (MPM)



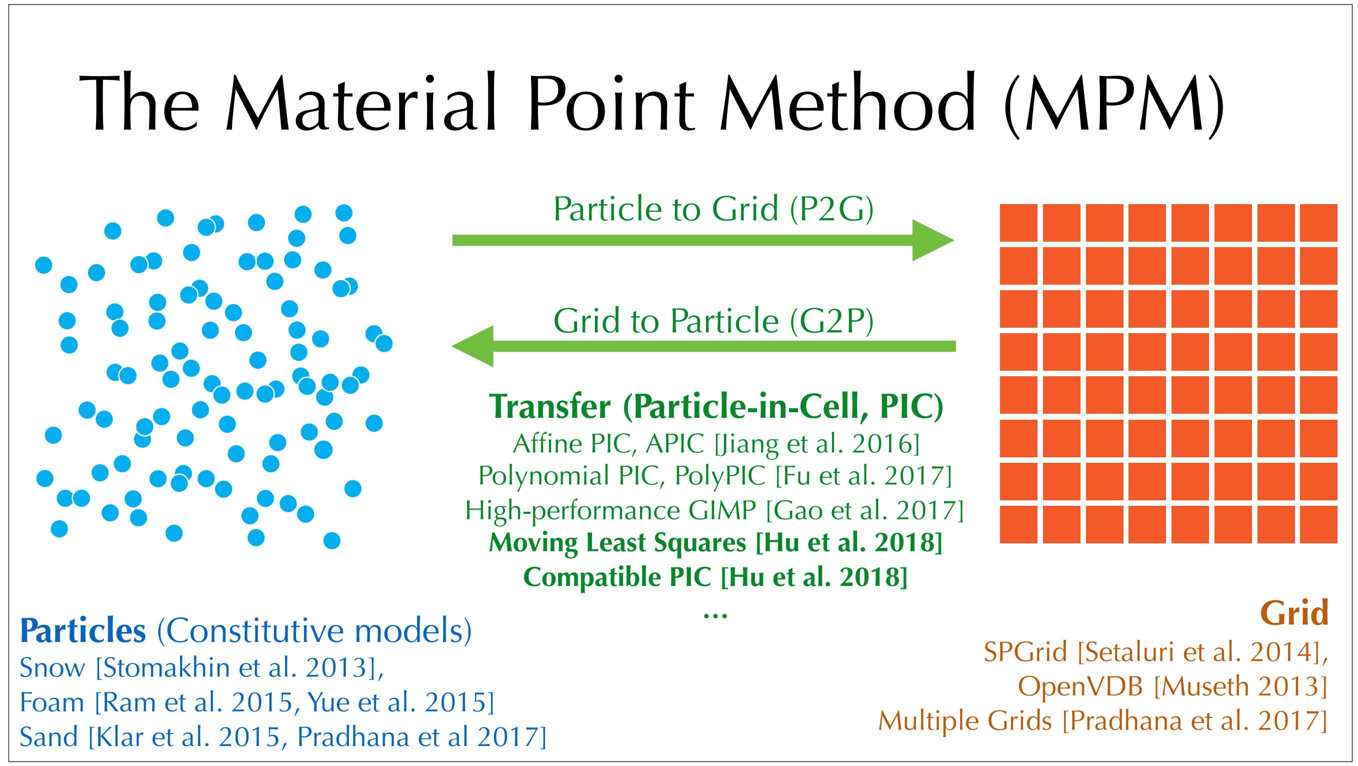
Particles (Constitutive models)
Snow [Stomakhin et al. 2013],
Foam [Ram et al. 2015, Yue et al. 2015]
Sand [Klar et al. 2015, Pradhana et al 2017]

MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.



MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.

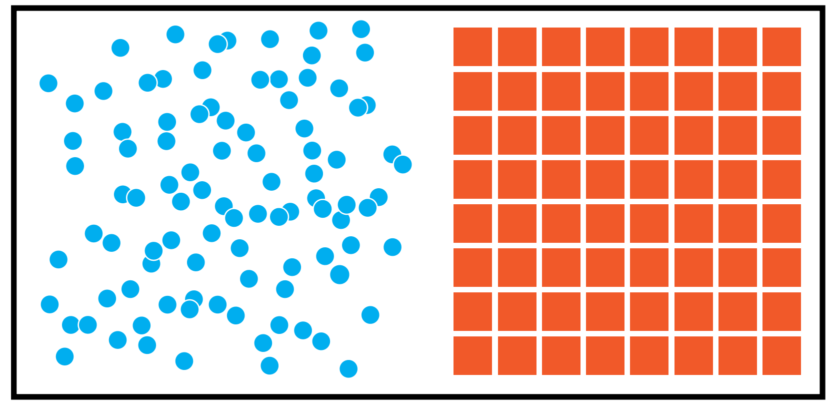
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.



MPM is a hybrid Eulerian-Lagrangian method, which means both particles[click] and grids[click] are used, and information is transferred [click] back-and-forth.

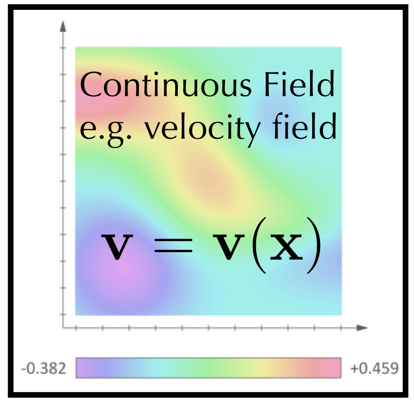
There have been a lot of recent work on MPM, on particles [click] and on grids [click]. Our work, like many previous work, is about how information is transferred between particles and grids.

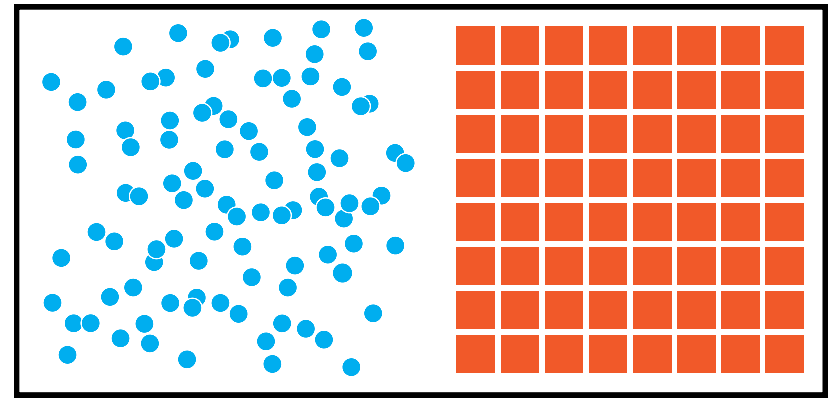
The Material Point Method (MPM)



When talking about MPM, everybody mentions particles and grids. However, there is another important concept usually missing in people's discussion on MPM. [Click] That is the continuous field approximated by discrete particles and nodes. In fact, the continuous field is where all physics happen. Essentially, MPM is a discretisation method that projects the continuous quantities to discrete particles and nodes.

The Material Point Method (MPM)





When talking about MPM, everybody mentions particles and grids. However, there is another important concept usually missing in people's discussion on MPM. [Click] That is the continuous field approximated by discrete particles and nodes. In fact, the continuous field is where all physics happen. Essentially, MPM is a discretisation method that projects the continuous quantities to discrete particles and nodes.

Contributions

- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and Easier
- **Part II: Compatible Particle-in-Cell (CPIC)**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

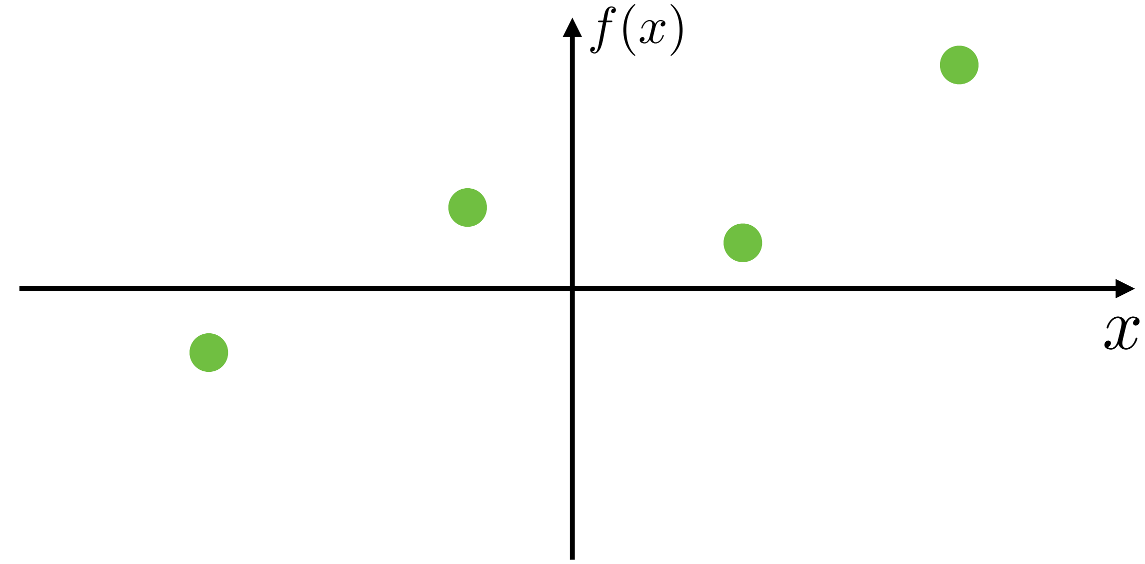
I will talk about two contributions in our work, namely the moving least squares discretisation and the compatible particle in cell method.

Contributions

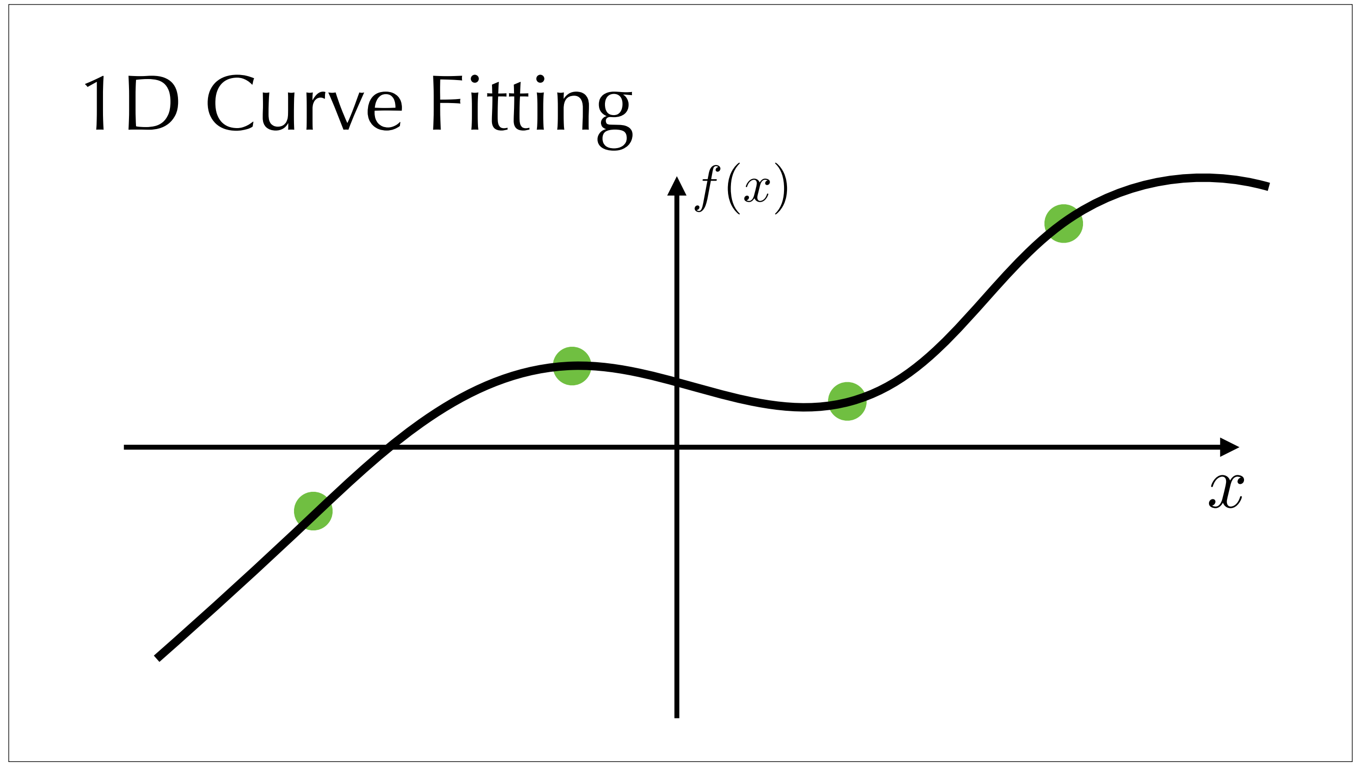
- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and easier
- **Part II: Compatible Particle-in-Cell**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

The first part introduces MLS-MPM.

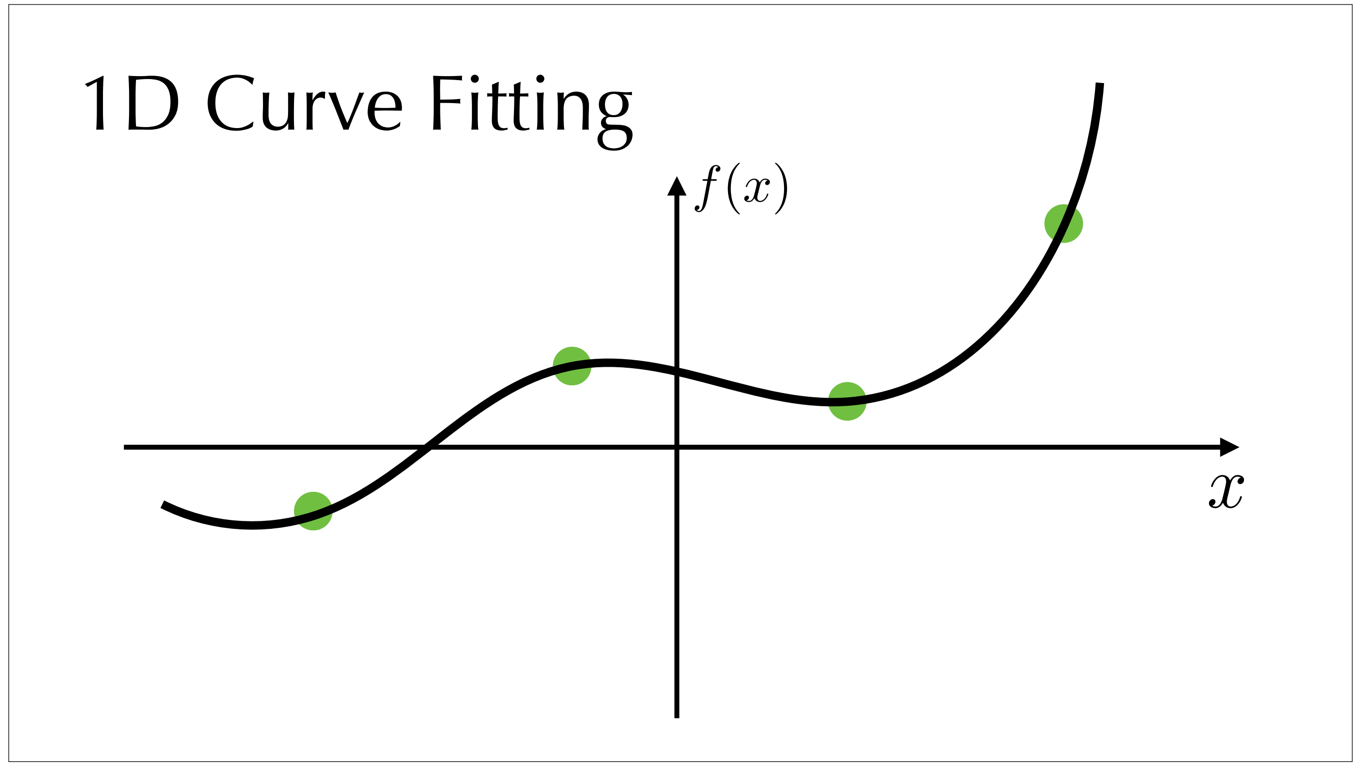
1D Curve Fitting



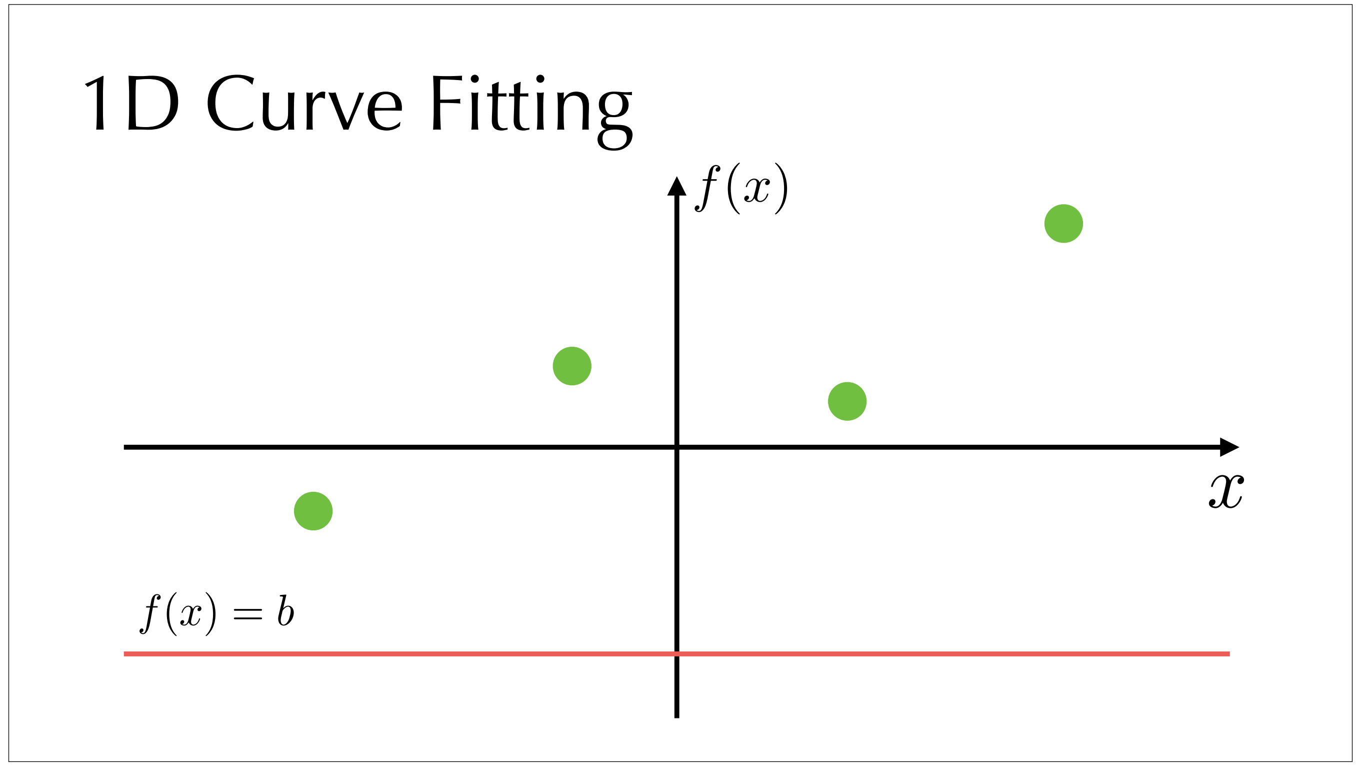
Let's start with some high-school math. Given several data points, we want to to reconstruct a smooth curve to fit them.



Apparently this is a under constraint problem and there are many possible solutions. One guess may look like this…

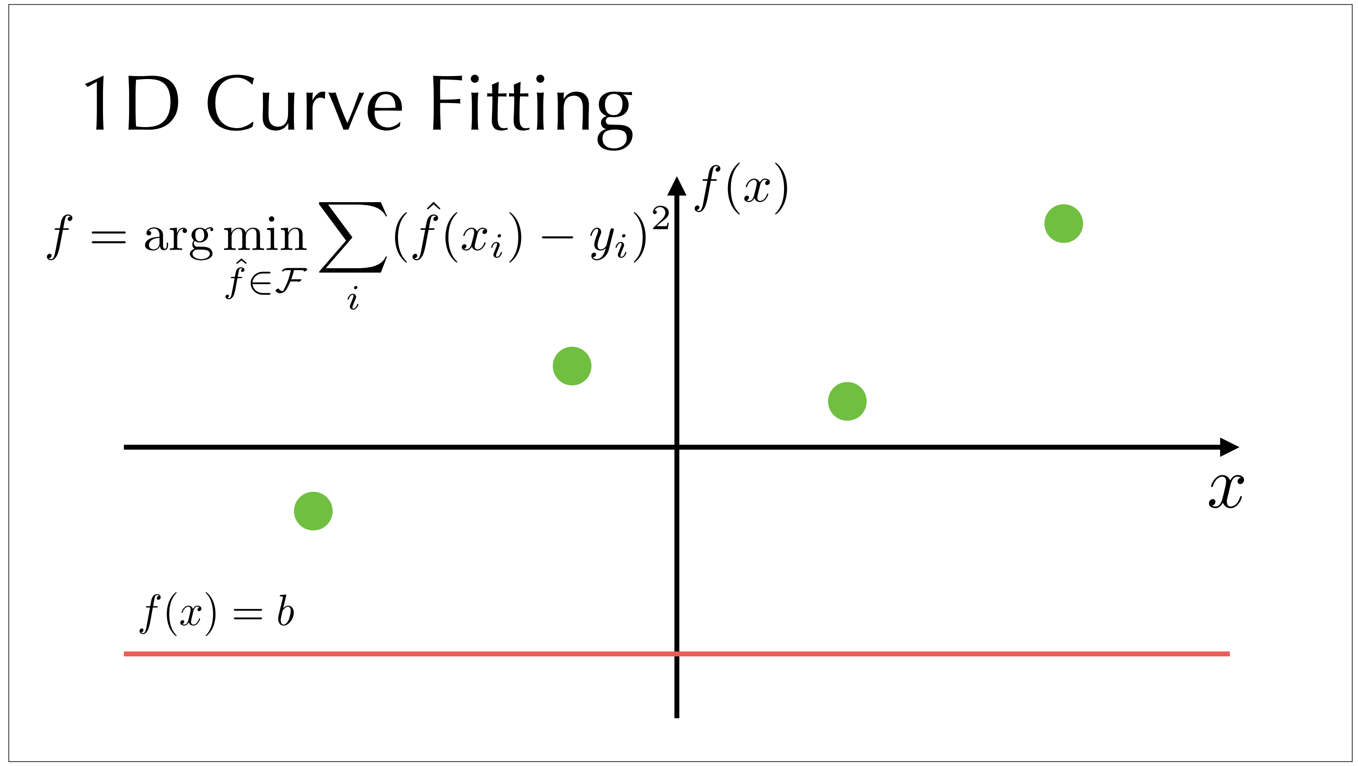


Another guess may look like this.



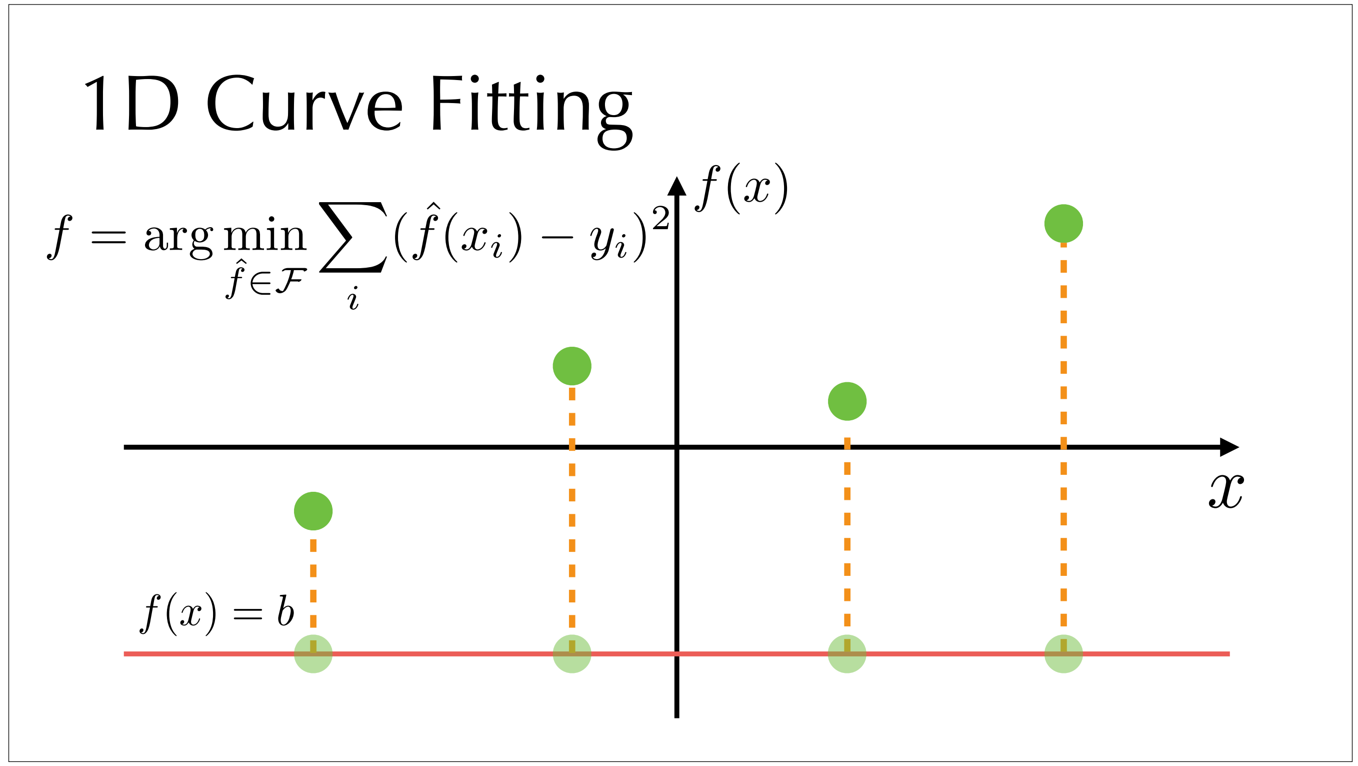
So lets regularise it a bit. Maybe it is good to constraint our choice in only constant functions, like f(x)=b.
Of course you want to pick one constant function [click] that minimise certain error metric. We use the least squares reconstruction error here [click], which is defined as the total squares distance from the data points to the projected points on the curve.

f=\text{arg}\min_{\hat{f}\in\mathcal{F}} \sum_i (\hat{f}(x_i)-y_i)^2



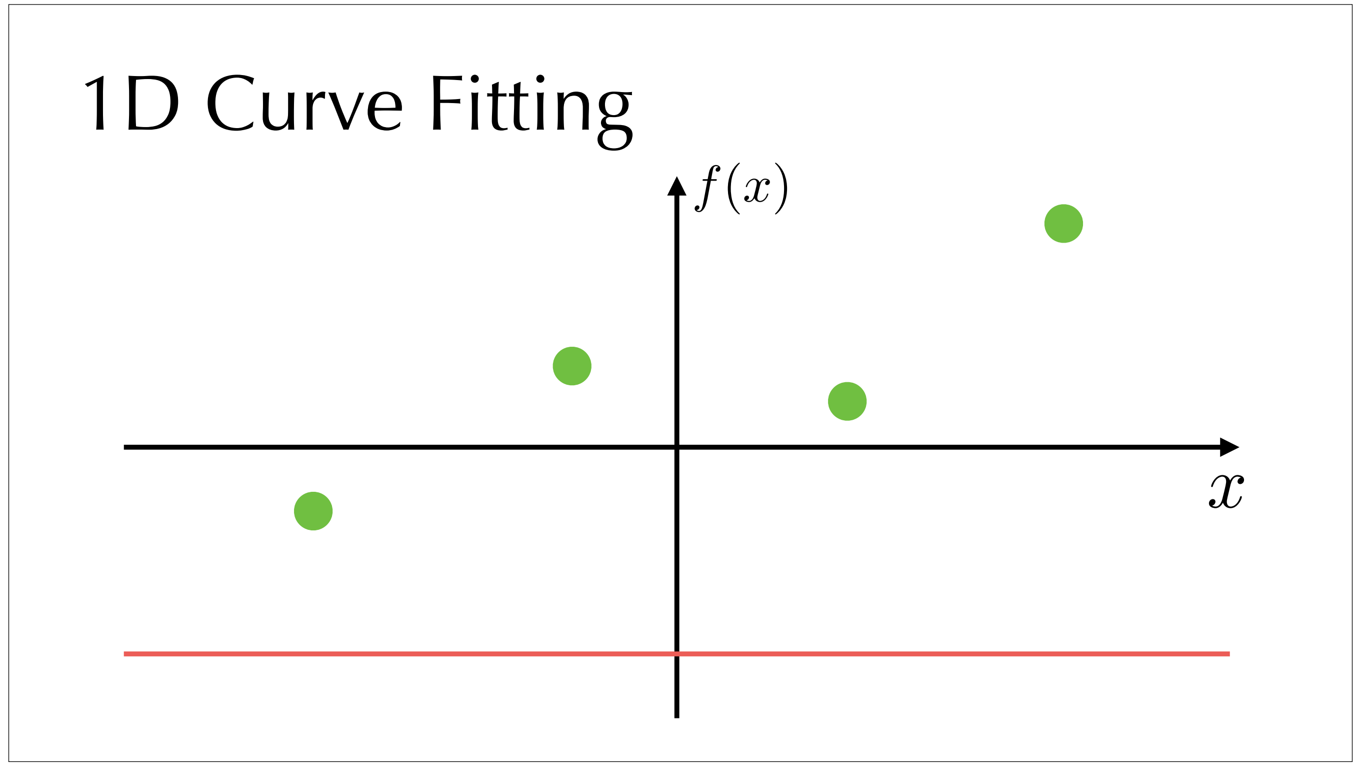
So lets regularise it a bit. Maybe it is good to constraint our choice in only constant functions, like f(x)=b.
Of course you want to pick one constant function [click] that minimise certain error metric. We use the least squares reconstruction error here [click], which is defined as the total squares distance from the data points to the projected points on the curve.

f=\text{arg}\min_{\hat{f}\in\mathcal{F}} \sum_i (\hat{f}(x_i)-y_i)^2

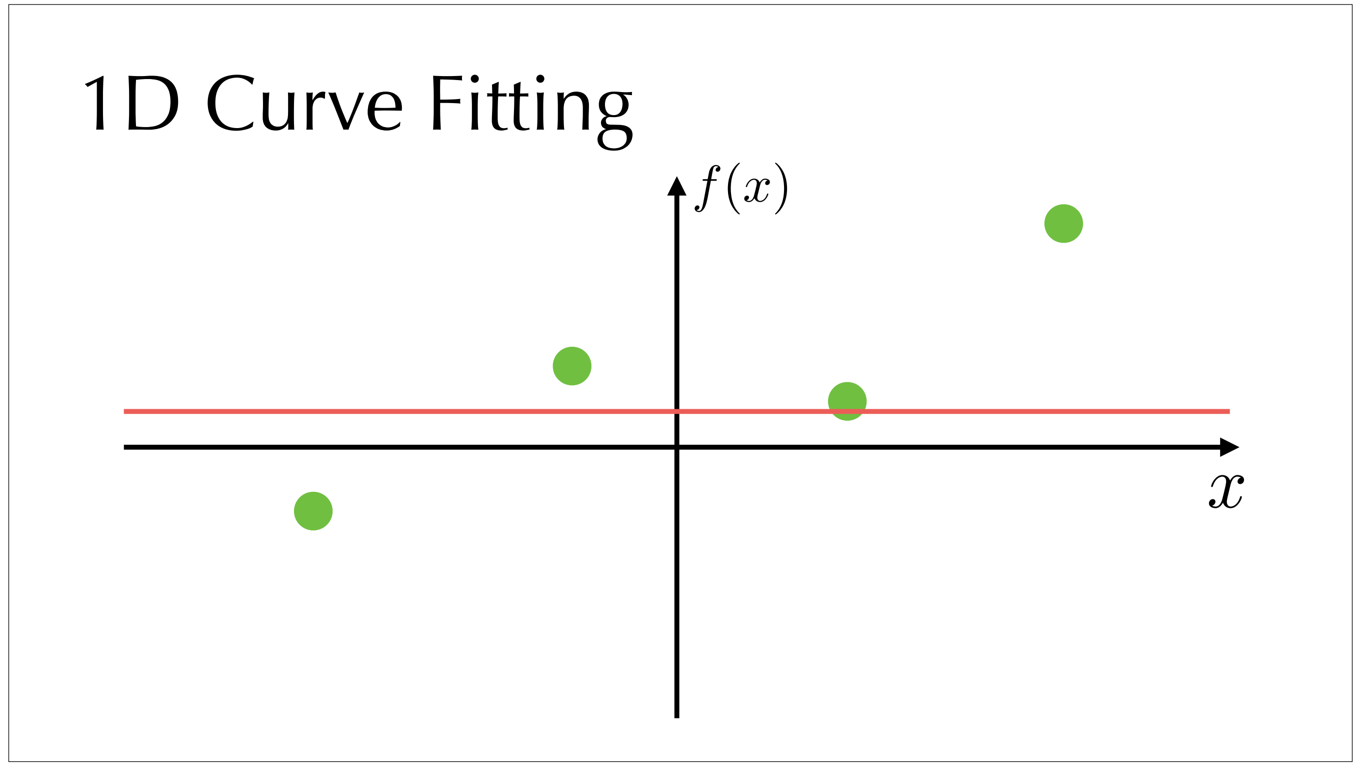


So lets regularise it a bit. Maybe it is good to constraint our choice in only constant functions, like f(x)=b.
Of course you want to pick one constant function [click] that minimise certain error metric. We use the least squares reconstruction error here [click], which is defined as the total squares distance from the data points to the projected points on the curve.

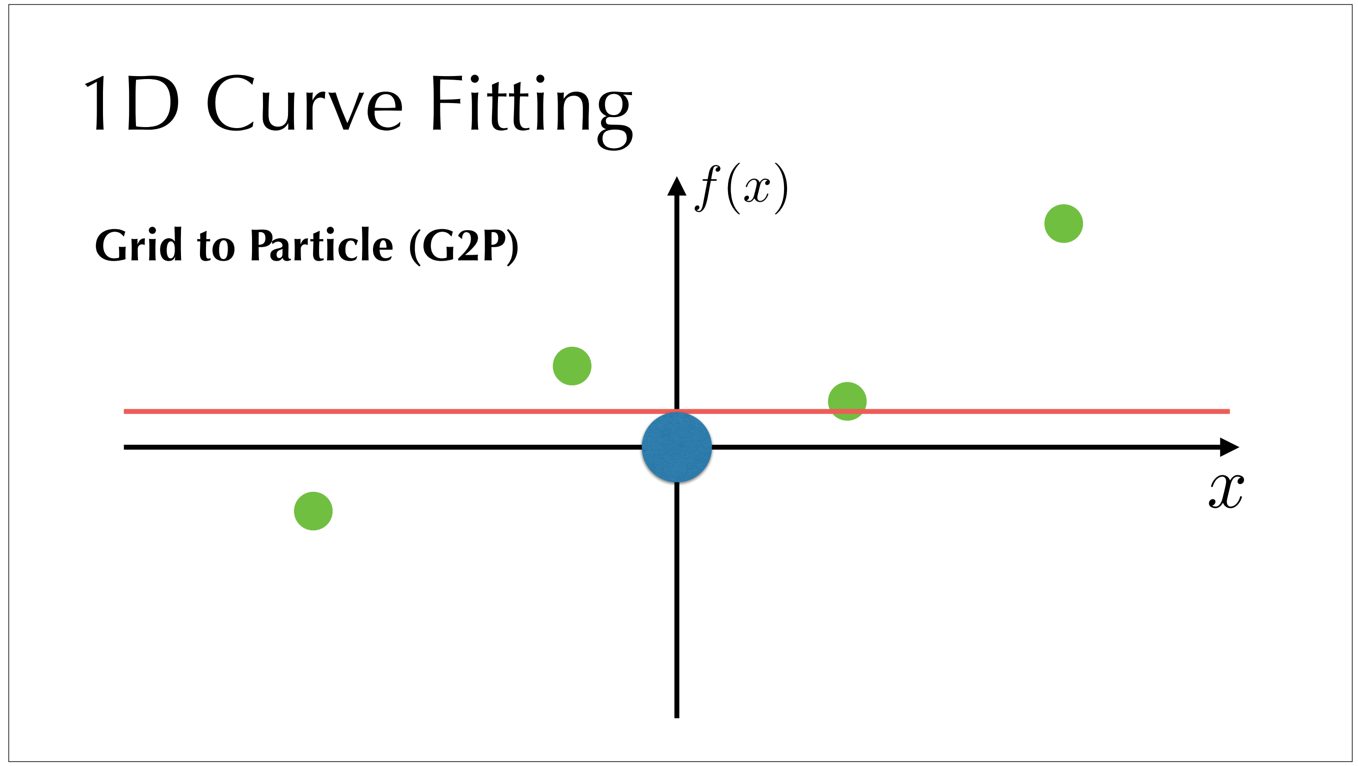
f=\text{arg}\min_{\hat{f}\in\mathcal{F}} \sum_i (\hat{f}(x_i)-y_i)^2



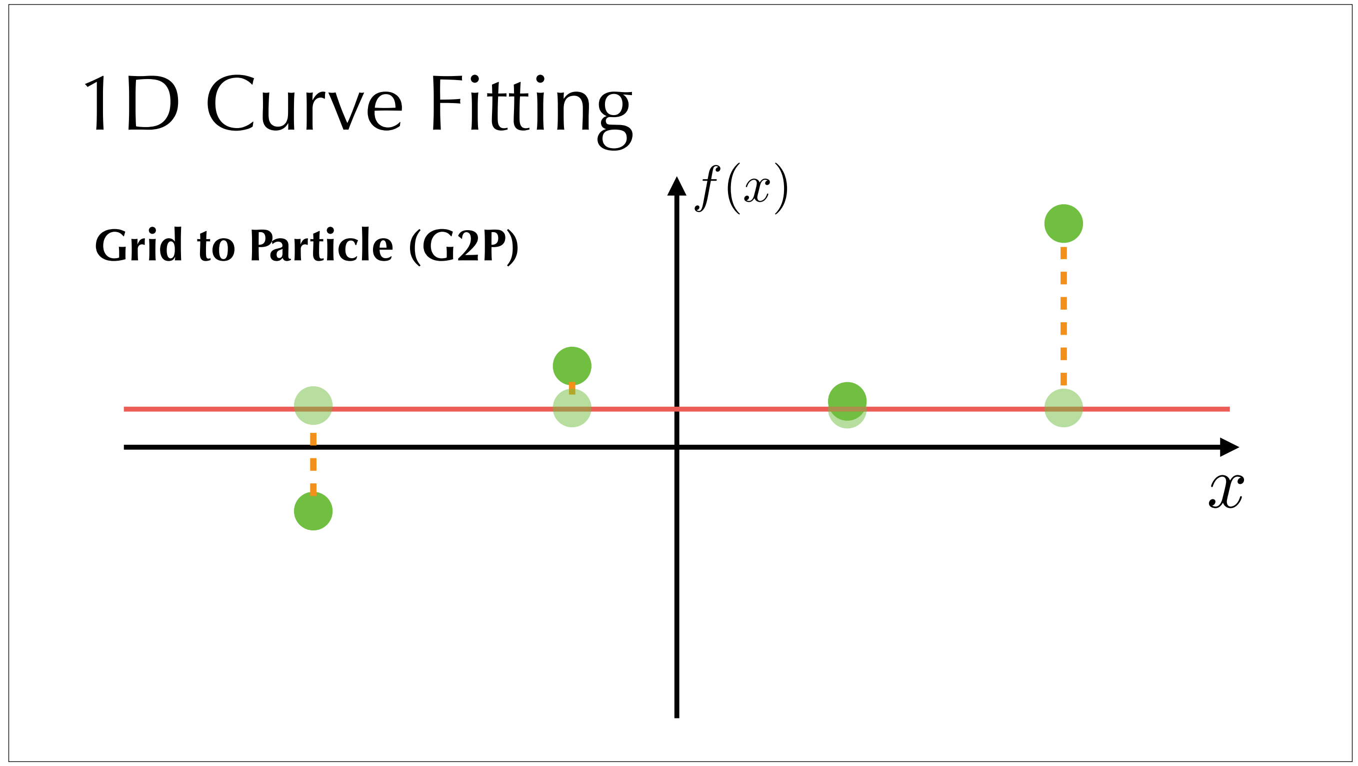
Apparently this function is not a good choice. Let's move it higher so that the distance gets minimized. If we consider the green dots to be the velocity values on the grid, this is exactly what happens during traditional particle-in-cell methods [click], when the particle is at the origin. Estimating the particle velocity is done by finding such function.



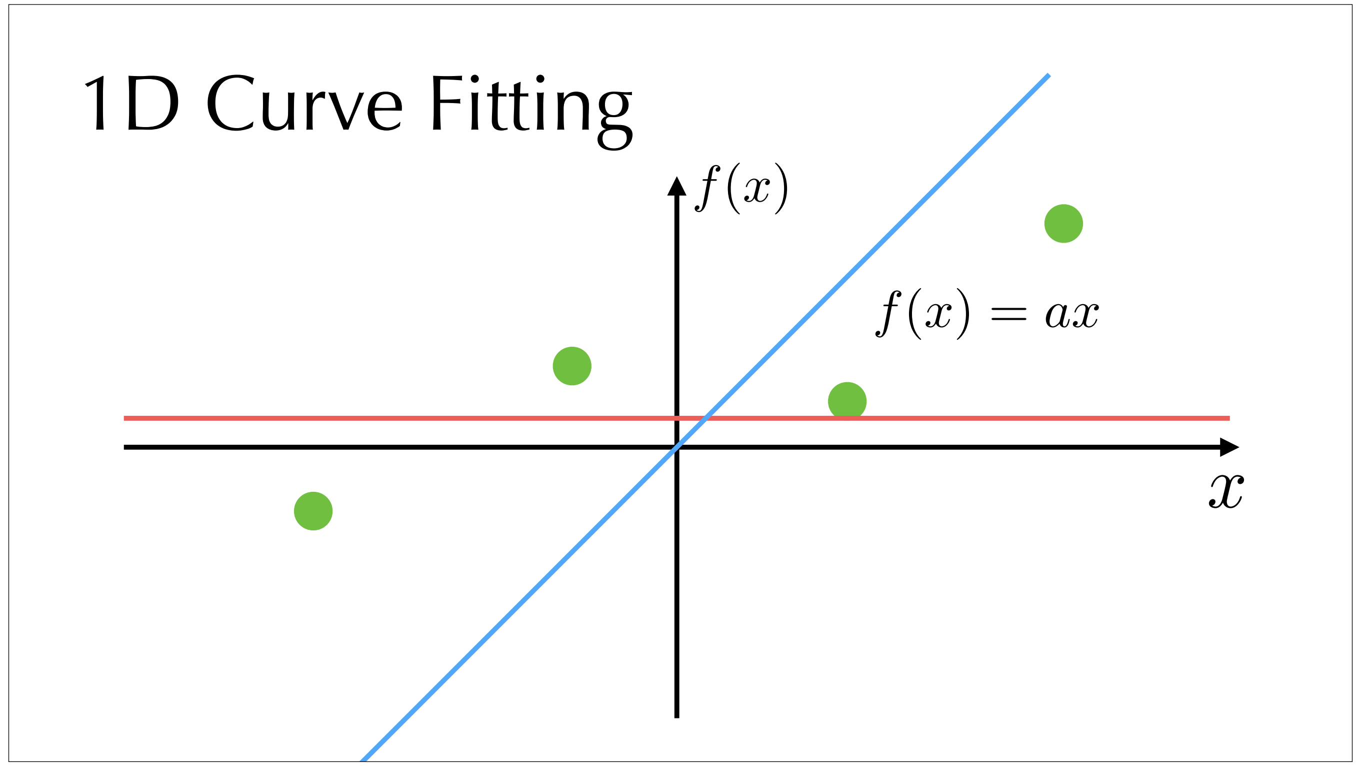
Apparently this function is not a good choice. Let's move it higher so that the distance gets minimized. If we consider the green dots to be the velocity values on the grid, this is exactly what happens during traditional particle-in-cell methods [click], when the particle is at the origin. Estimating the particle velocity is done by finding such function.



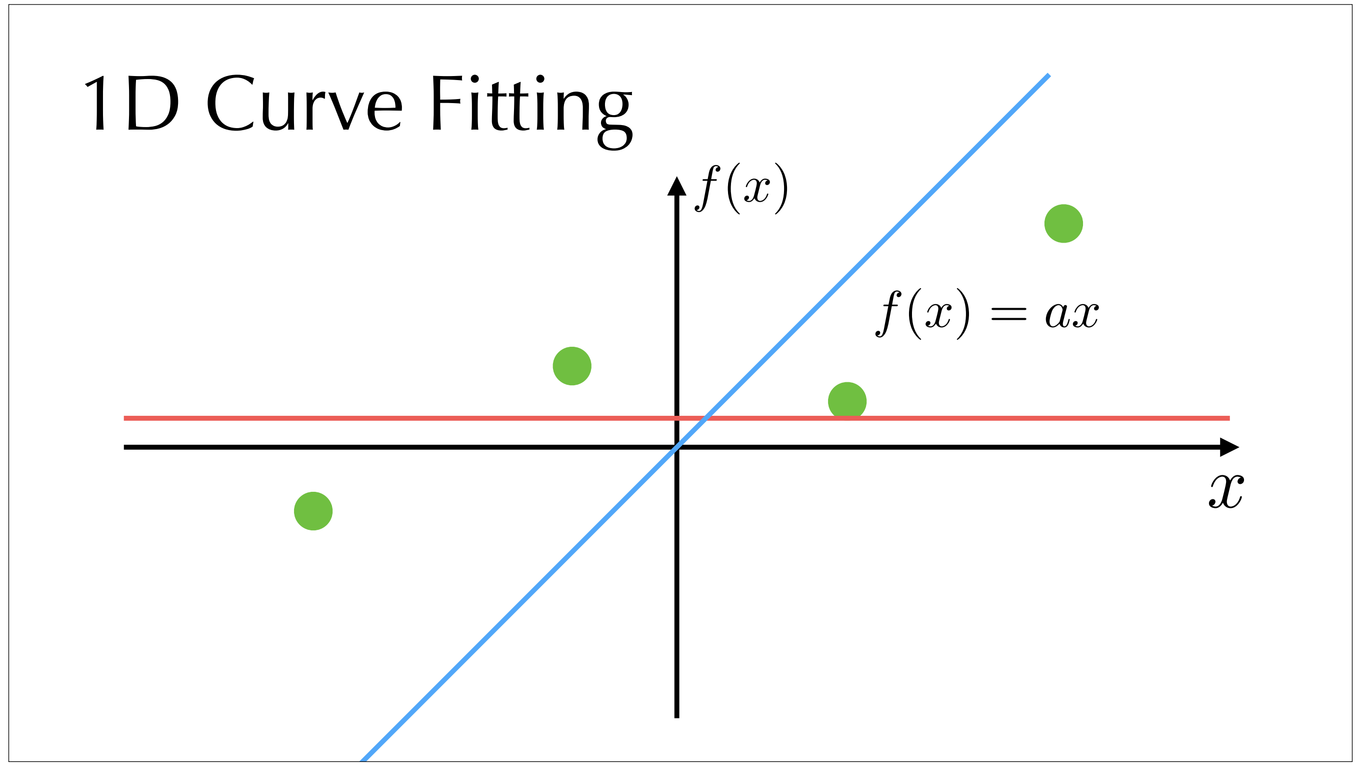
Apparently this function is not a good choice. Let's move it higher so that the distance gets minimized. If we consider the green dots to be the velocity values on the grid, this is exactly what happens during traditional particle-in-cell methods [click], when the particle is at the origin. Estimating the particle velocity is done by finding such function.



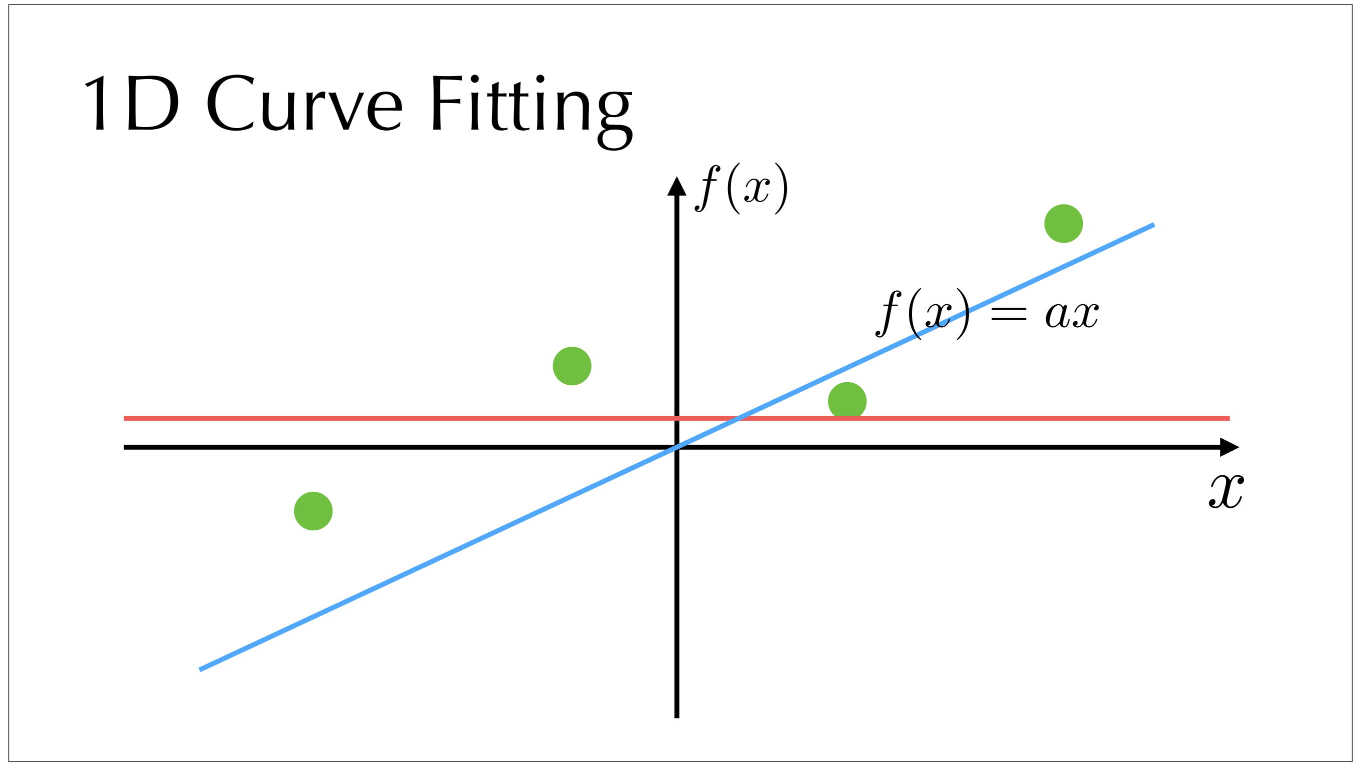
And during particle to grid transfer, the grid velocity will be overwritten by the reconstructed values. This is why we want to minimise the reconstruction distance: smaller distance means better conversation of energy and less dissipation.



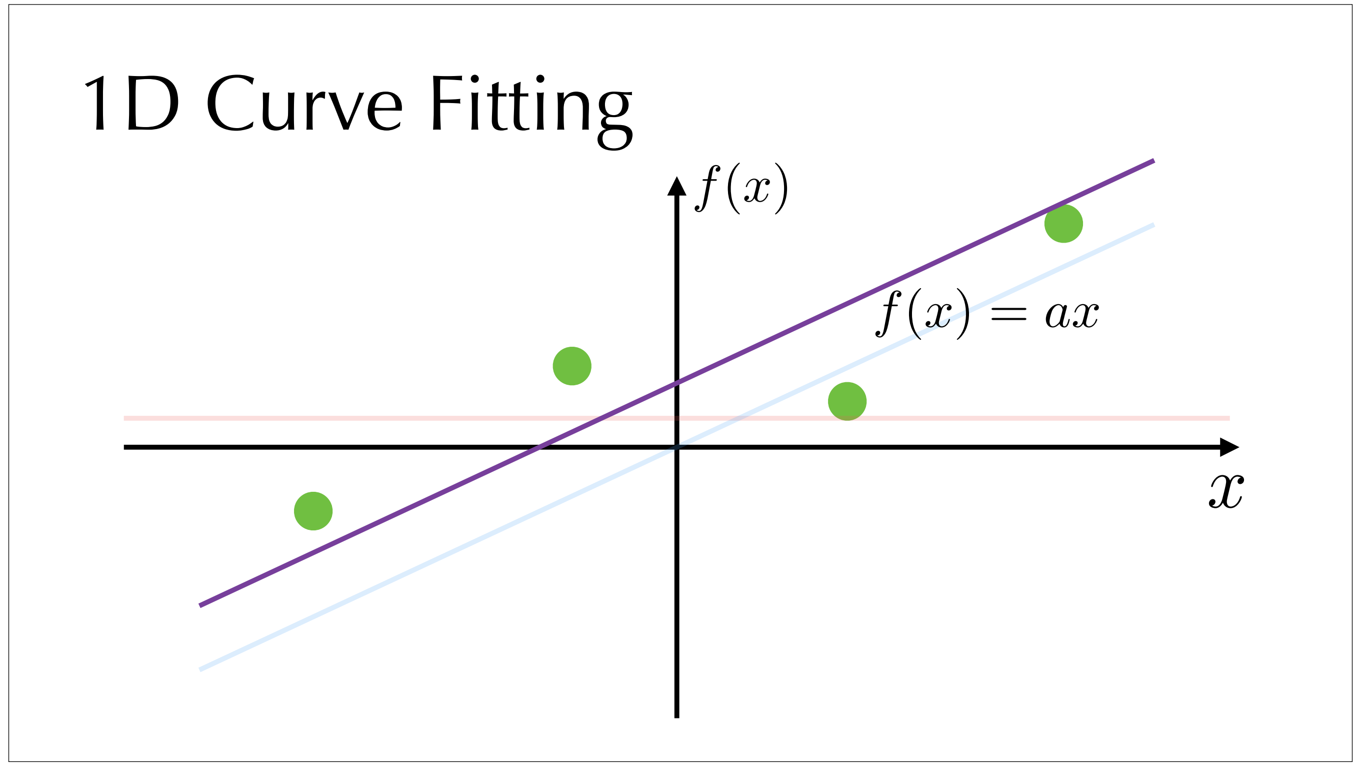
It’s clear that just using a constant function is usually not sufficient. Let’s include linear functions as well.



We can scale it [click] to make it match the slope of these points.

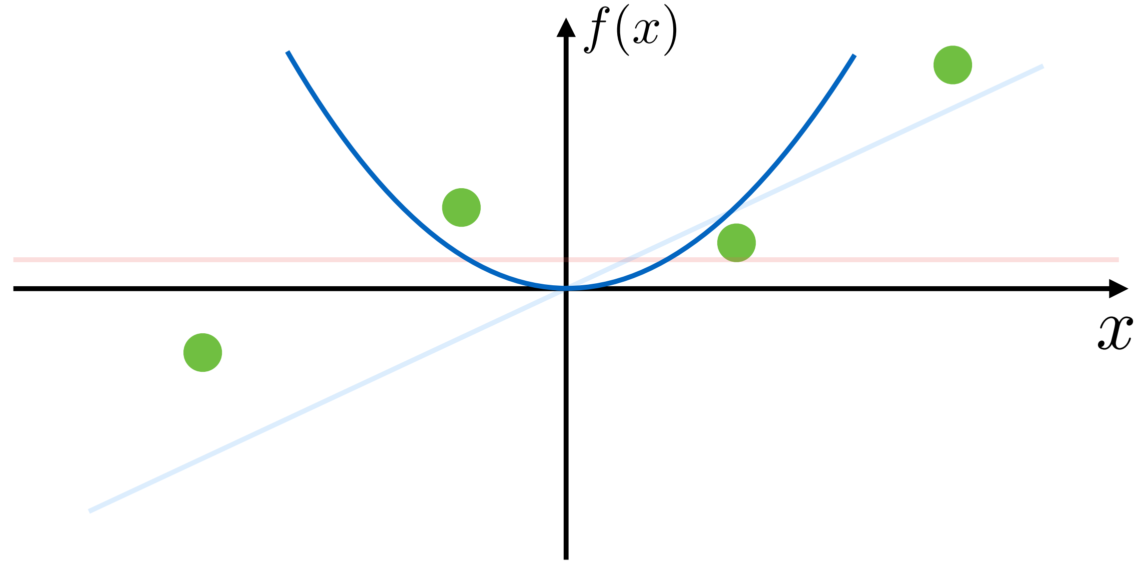


We can scale it [click] to make it match the slope of these points.



If we super-impose the constant and linear functions, we can get a much better reconstruction. This is what APIC does. Instead of storing a constant velocity field on particles, APIC additionally stores information about the velocity field gradient, or the slope.

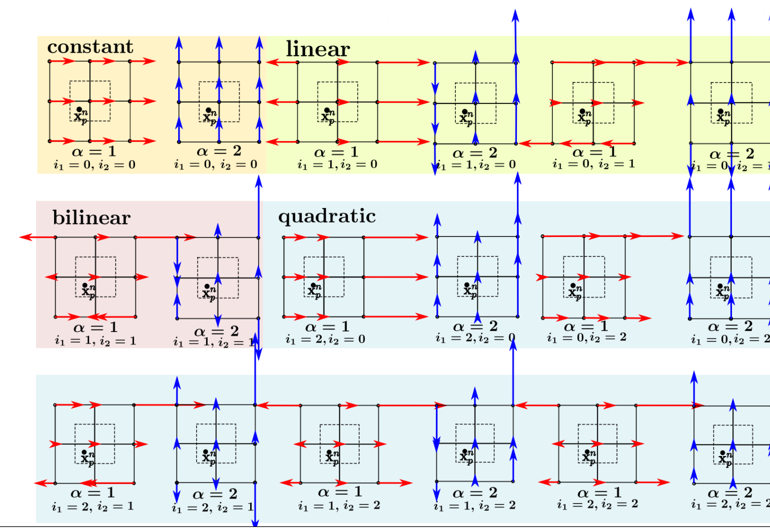
1D Curve Fitting



Of course we can introduce more basis, as in PolyPIC.

Least-Squares Transfers in 2D

Figure from A Polynomial Particle-In-Cell Method, Fu et al. 2017

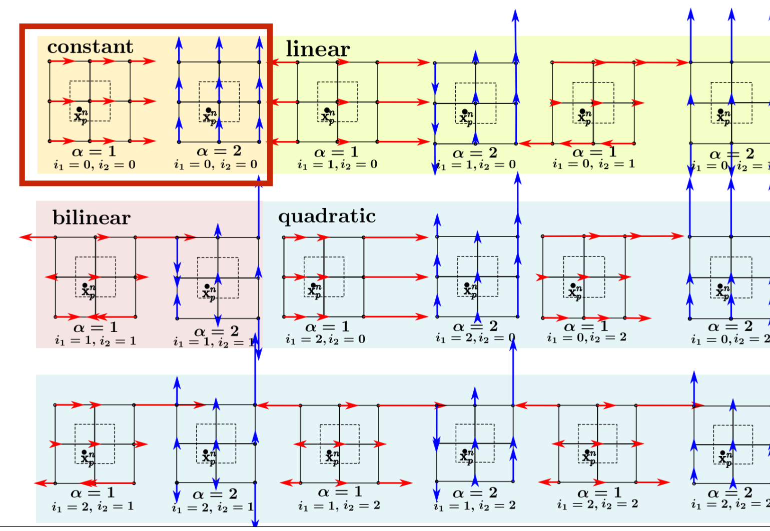


In 2D things get a bit more complex, but the idea is the same: use basis function to get least squares reconstructions.

Least-Squares Transfers in 2D

Figure from A Polynomial Particle-In-Cell Method, Fu et al. 2017

PIC

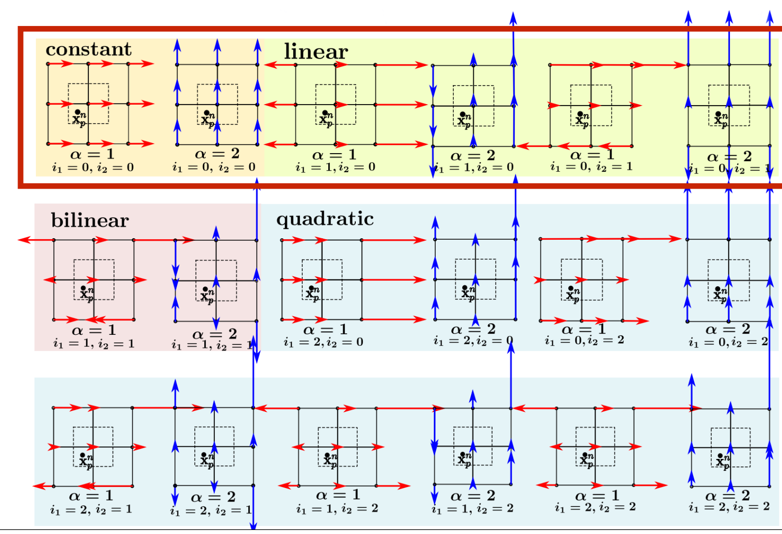


Traditional PIC uses only constant basis.

Least-Squares Transfers in 2D

Figure from A Polynomial Particle-In-Cell Method, Fu et al. 2017

APIC

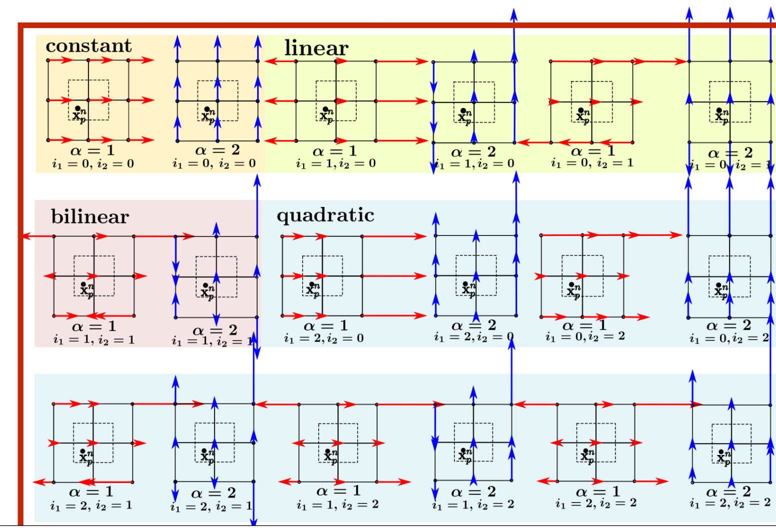


APIC uses linear basis

Least-Squares Transfers in 2D

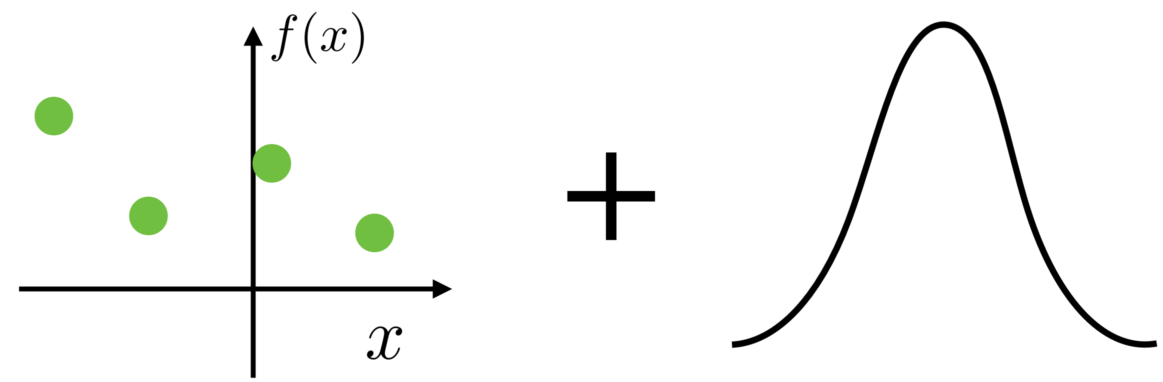
Figure from A Polynomial Particle-In-Cell Method, Fu et al. 2017
18 DoFs=9 nodes x 2 DoFs per node: Lossless transfer!

PolyPIC



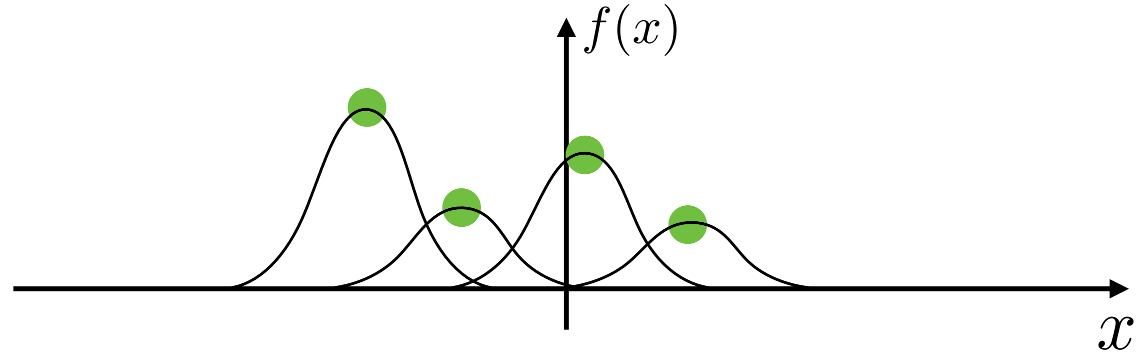
PolyPIC further introduces bilinear and quadratic basis functions, leading to lossless grid-particle transfer. The more basis we use, the less dissipation we have.

1D Curve Fitting: Spline Interpolation



Actually, there is another way to reconstruct the continuous function, which is spline interpolation.

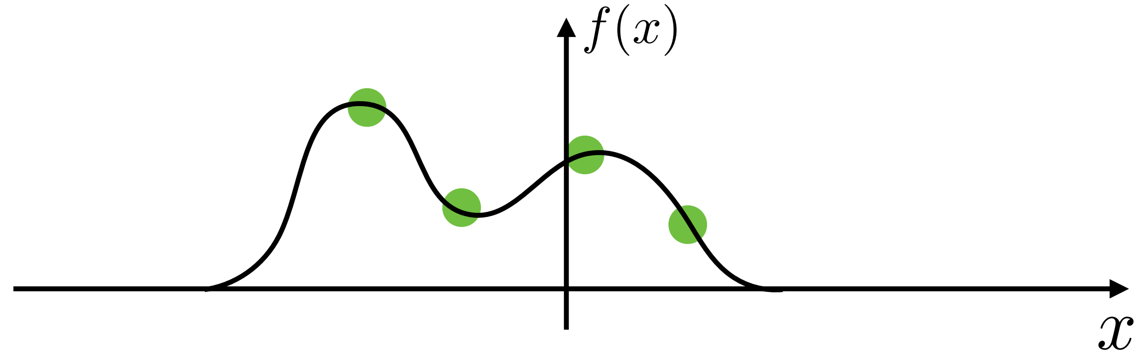
1D Curve Fitting: Spline Interpolation



“Shape functions” in FEM and MPM

Instead of using polynomials, we can use small splines at each node position. This is actually the “shape function” approach we use in FEM and MPM.

1D Curve Fitting: Spline Interpolation



Super Imposed Shape Functions:
Continuous Function from **Discrete** DoFs

By adding these splines together, we get something like this. The reconstructed function looks smooth and it has been the classical way to discrete PDEs in weak-form based methods such as FEM and MPM.

Which one to use?

	APIC/PolyPIC	MPM Discretization
Moving Least Squares Interpolation	✓	?
B-Spline Interpolation	?	✓

So we have talked about two ways of reconstructing a continuous field out of discrete samples. It seems that for some reason we have been using MLS interpolation for APIC, but B-Spline interpolation for MPM discretization.
One natural question here is, can we only use one single interpolation method for both? If we can do so, maybe we can simplify and optimise our algorithm.

Which one to use?

	APIC/PolyPIC	MPM Discretization
Moving Least Squares Interpolation	✓	?
B-Spline Interpolation	?	✓

So let's try using B-Splines for APIC.

Which one to use?

	APIC/PolyPIC	MPM Discretization
Moving Least Squares Interpolation	✓	?
B-Spline Interpolation	No Angular Momentum Conservation	✓

But after some attempts it turns out that doing so doesn't bring much advantage. In fact, it is not only hard to compute but also leads to no angular momentum conservation

Which one to use?

	APIC/PolyPIC	MPM Discretization
Moving Least Squares Interpolation	✓	?
B-Spline Interpolation	No Angular Momentum Conservation	✓

So the only hope we have is to MLS interpolation for MPM discretizaiton.

Which one to use?

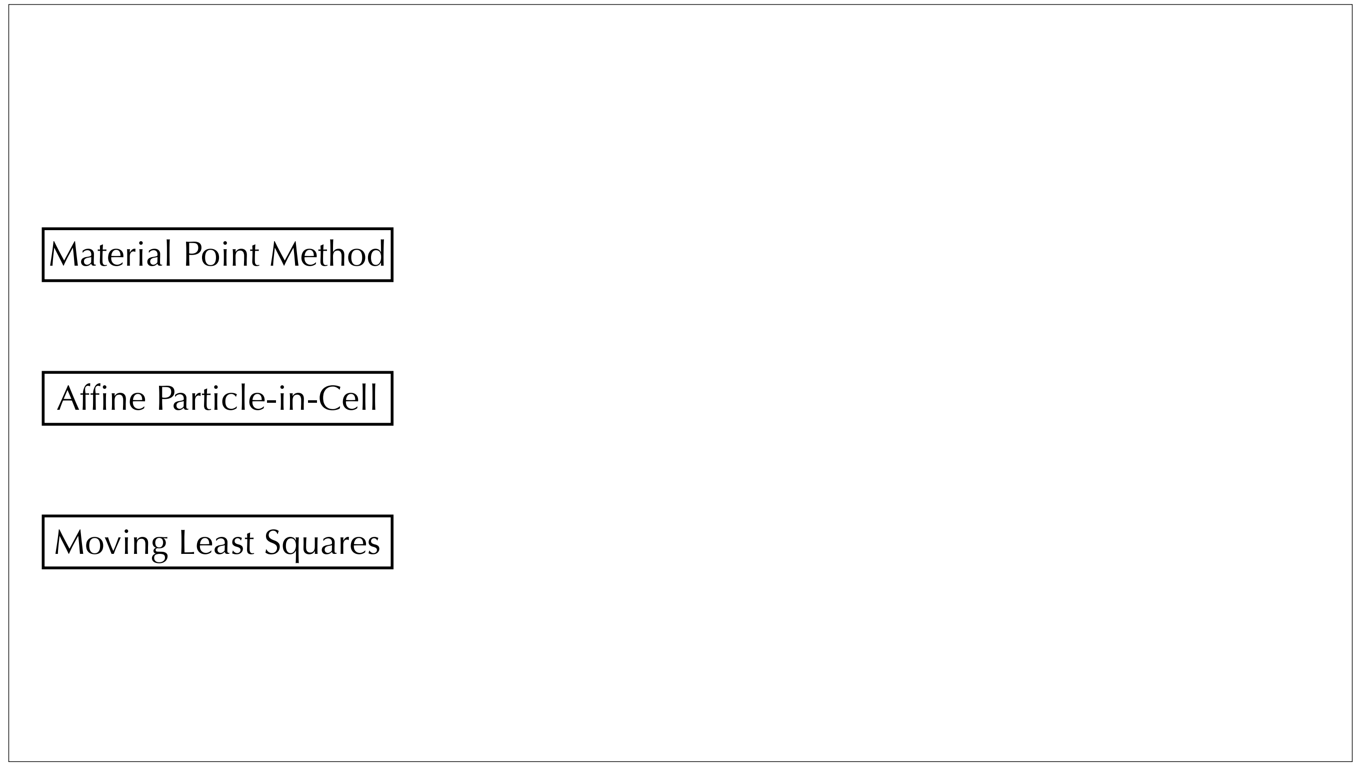
	APIC/PolyPIC	MPM Discretization
Moving Least Squares Interpolation	✓	**MLS-MPM!**
B-Spline Interpolation	**No Angular Momentum Conservation**	✓

Fortuantely, it works well! This leads to MLS-MPM.

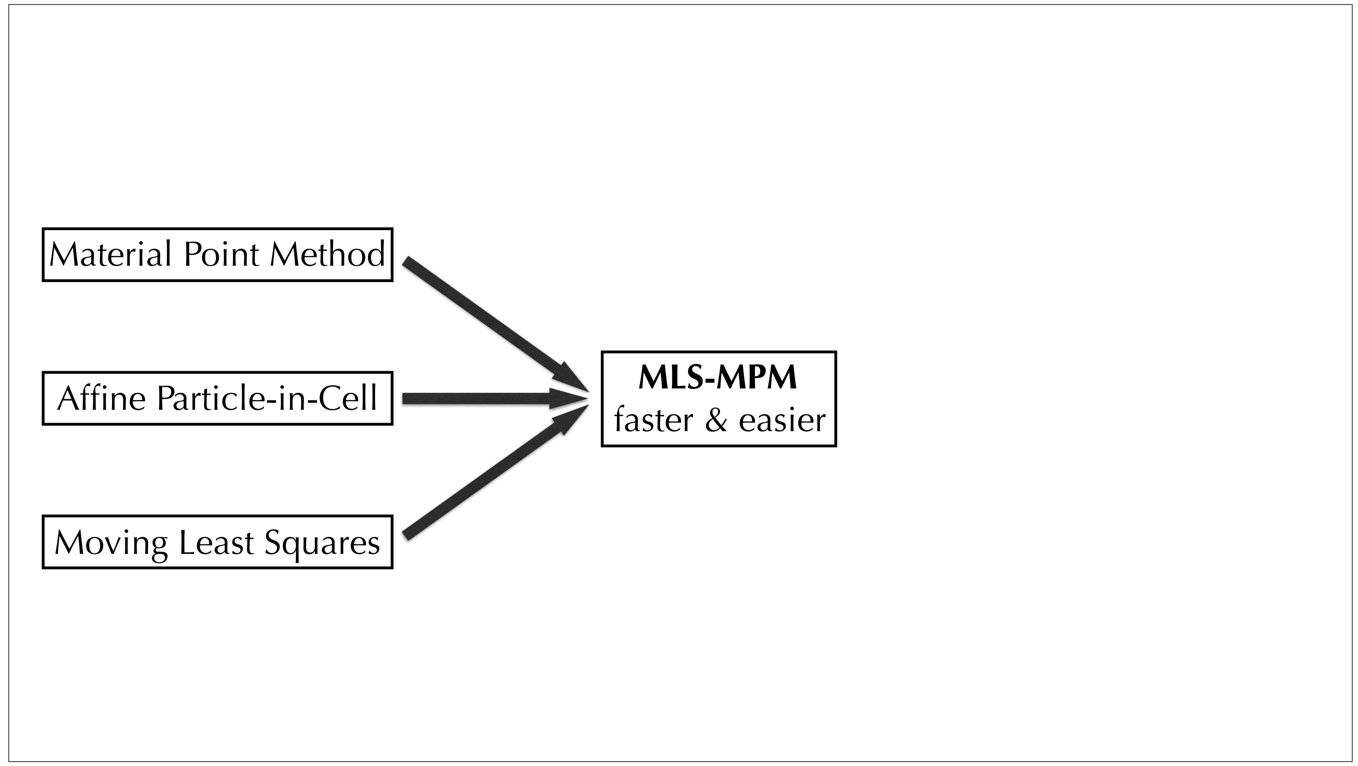
Material Point Method

Affine Particle-in-Cell

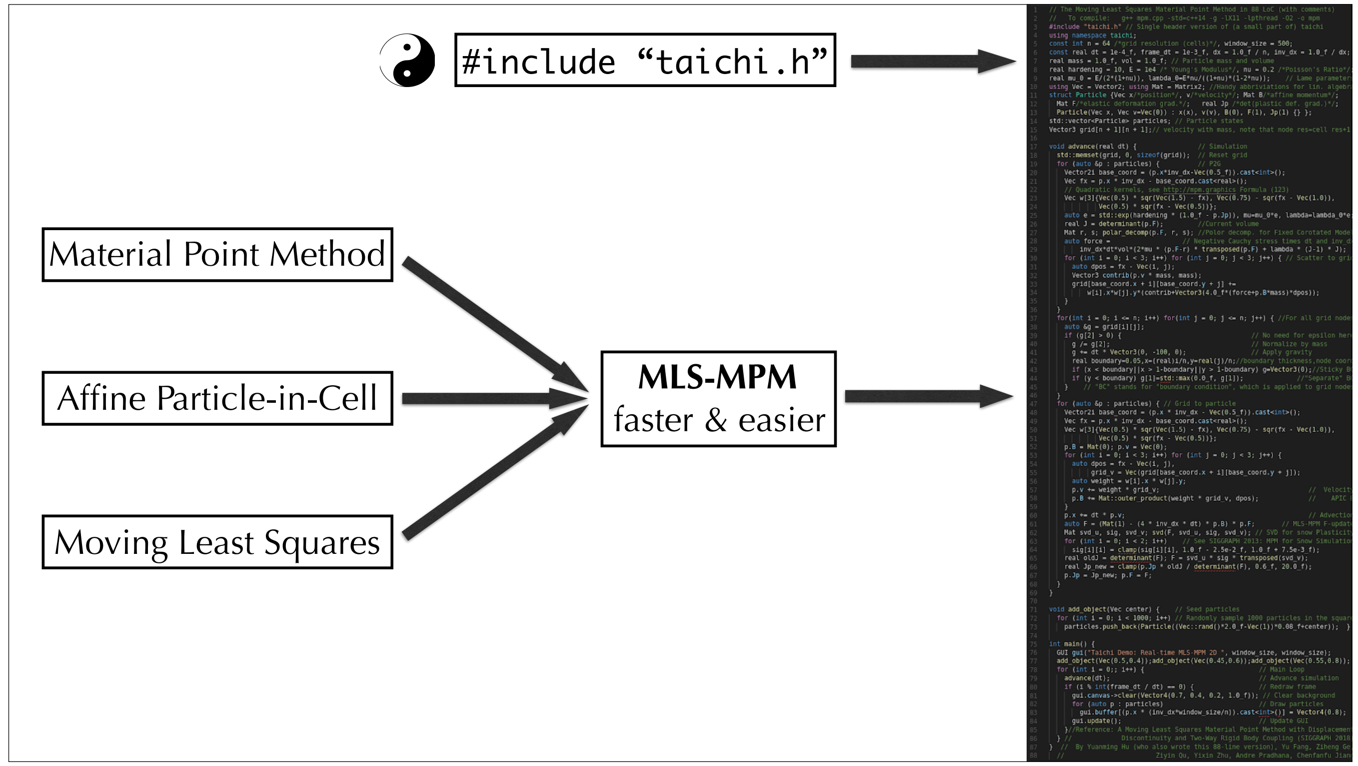
Moving least squares makes MPM work together with APIC better [click]. Before I dive into a little bit of math, I want to highlight that MLS-MPM is actually even simpler to implement than traditional MPM. [click] In fact, it can be implemented within 88 lines of code. [click] This is the first time to the best of our knowledge, when MPM becomes so simple to implement.



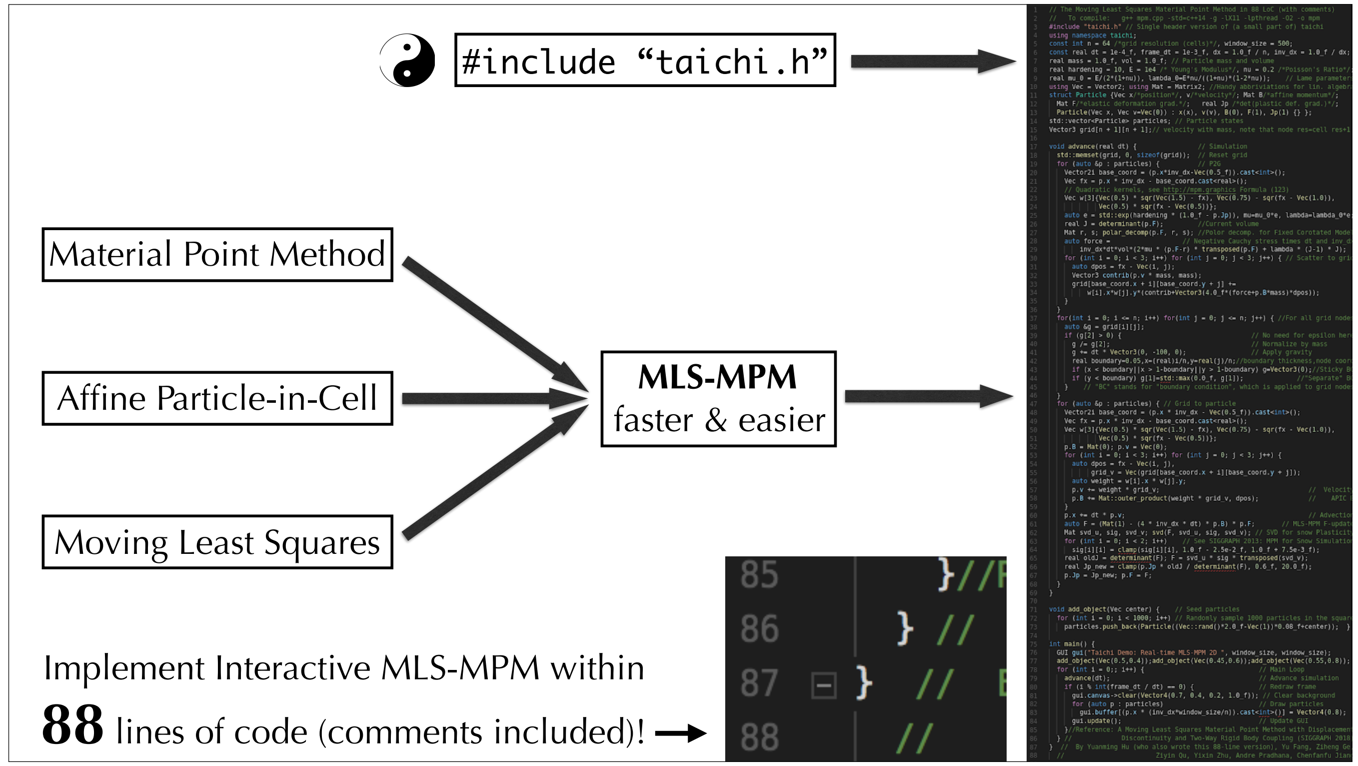
Moving least squares makes MPM work together with APIC better [click]. Before I dive into a little bit of math, I want to highlight that MLS-MPM is actually even simpler to implement than traditional MPM. [click] In fact, it can be implemented within 88 lines of code. [click] This is the first time to the best of our knowledge, when MPM becomes so simple to implement.



Moving least squares makes MPM work together with APIC better [click]. Before I dive into a little bit of math, I want to highlight that MLS-MPM is actually even simpler to implement than traditional MPM. [click] In fact, it can be implemented within 88 lines of code. [click] This is the first time to the best of our knowledge, when MPM becomes so simple to implement.



Moving least squares makes MPM work together with APIC better [click]. Before I dive into a little bit of math, I want to highlight that MLS-MPM is actually even simpler to implement than traditional MPM. [click] In fact, it can be implemented within 88 lines of code. [click] This is the first time to the best of our knowledge, when MPM becomes so simple to implement.



Moving least squares makes MPM work together with APIC better [click]. Before I dive into a little bit of math, I want to highlight that MLS-MPM is actually even simpler to implement than traditional MPM. [click] In fact, it can be implemented within 88 lines of code. [click] This is the first time to the best of our knowledge, when MPM becomes so simple to implement.

From **MPM** to **MLS-MPM**

Shape/Test function	B-spline	MLS Shape function weighted by B-spline
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	$m_i^n = \sum_p m_p \omega_{ip}$
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \nabla \omega_{ip}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
Velocity Gradient Evaluation	$\nabla \boldsymbol{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n}+1}$
Deformation Gradient Update	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}}) \right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}}) \right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

Here I summarize elements of traditional MPM and MLS-MPM. This table does look a bit scary, but I promise that I will only spend one minute on it, and fortunately most entries are actually the same for MPM and MLS-MPM.

\Delta t V_p^0 \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT}\nabla \omega_{ip}
\frac{4}{\Delta x^2}\Delta t V_p^0 \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT} (\mathbf{x}_i-\mathbf{x}_p) \omega_{ip}
m_p\mathbf{C}_p^{n}(\mathbf{x_i}-\mathbf{x_p})\omega_{ip}
m_i^n=\sum_p m_p \omega_{ip}
\mathbf{C}_p^{n+1}=\frac{4}{\Delta x^2}\sum_i{v_i(\mathbf{x}_i-\mathbf{x}_p)}\omega_{ip}

From **MPM** to **MLS-MPM**

Shape/Test function	B-spline	MLS Shape function weighted by B-spline
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	$m_i^n = \sum_p m_p \omega_{ip}$
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT} \nabla \omega_{ip}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT} (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
Velocity Gradient Evaluation	$\nabla \boldsymbol{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n+1}}$
Deformation Gradient Update	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n+1}} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n+1}} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

In fact, the only difference is the stress momentum contribution and the evaluation of velocity gradient. [click] MLS-MPM reuses the APIC affine velocity field, which makes G2P faster.

```
\Delta t V_p^0 \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT}\nabla \omega_{ip}
\frac{4}{\Delta x^2}\Delta t V_p^0 \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT} (\mathbf{x}_i-\mathbf{x}_p) \omega_{ip}
m_p\mathbf{C}_p^{n}(\mathbf{x_i}-\mathbf{x_p})\omega_{ip}
m_i^n=\sum_p m_p \omega_{ip}
\mathbf{C}_p^{n+1}=\frac{4}{\Delta x^2}\sum_i{v_i(\mathbf{x}_i-\mathbf{x}_p)}\omega_{ip}
```

From **MPM** to **MLS-MPM**

Shape/Test function	B-spline	MLS Shape function weighted by B-spline
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	$m_i^n = \sum_p m_p \omega_{ip}$
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \nabla \omega_{ip}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	$\boxed{\mathbf{C}_p^{n+1}} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
Velocity Gradient Evaluation	$\nabla \boldsymbol{v}_p^{n+1} = \sum_i \boldsymbol{v}_i^{n+1} (\nabla w_{ip}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \boxed{\mathbf{C}_p^{n+1}}$
Deformation Gradient Update	$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_p^n)\right) \mathbf{F}_p^n$	$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_p^n)\right) \mathbf{F}_p^n$

In fact, the only difference is the stress momentum contribution and the evaluation of velocity gradient. [click] MLS-MPM reuses the APIC affine velocity field, which makes G2P faster.

\Delta t V_p^0 \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT}\nabla \omega_{ip}
\frac{4}{\Delta x^2}\Delta t V_p^0 \frac{\partial\Psi}{\partial\mathbf{F}}(\mathbf{F}_p^n)\mathbf{F}_p^{nT} (\mathbf{x}_i-\mathbf{x}_p) \omega_{ip}
m_p\mathbf{C}_p^{n}(\mathbf{x_i}-\mathbf{x_p})\omega_{ip}
m_i^n=\sum_p m_p \omega_{ip}
\mathbf{C}_p^{n+1}=\frac{4}{\Delta x^2}\sum_i{v_i(\mathbf{x}_i-\mathbf{x}_p)}\omega_{ip}

From **MPM** to **MLS-MPM**

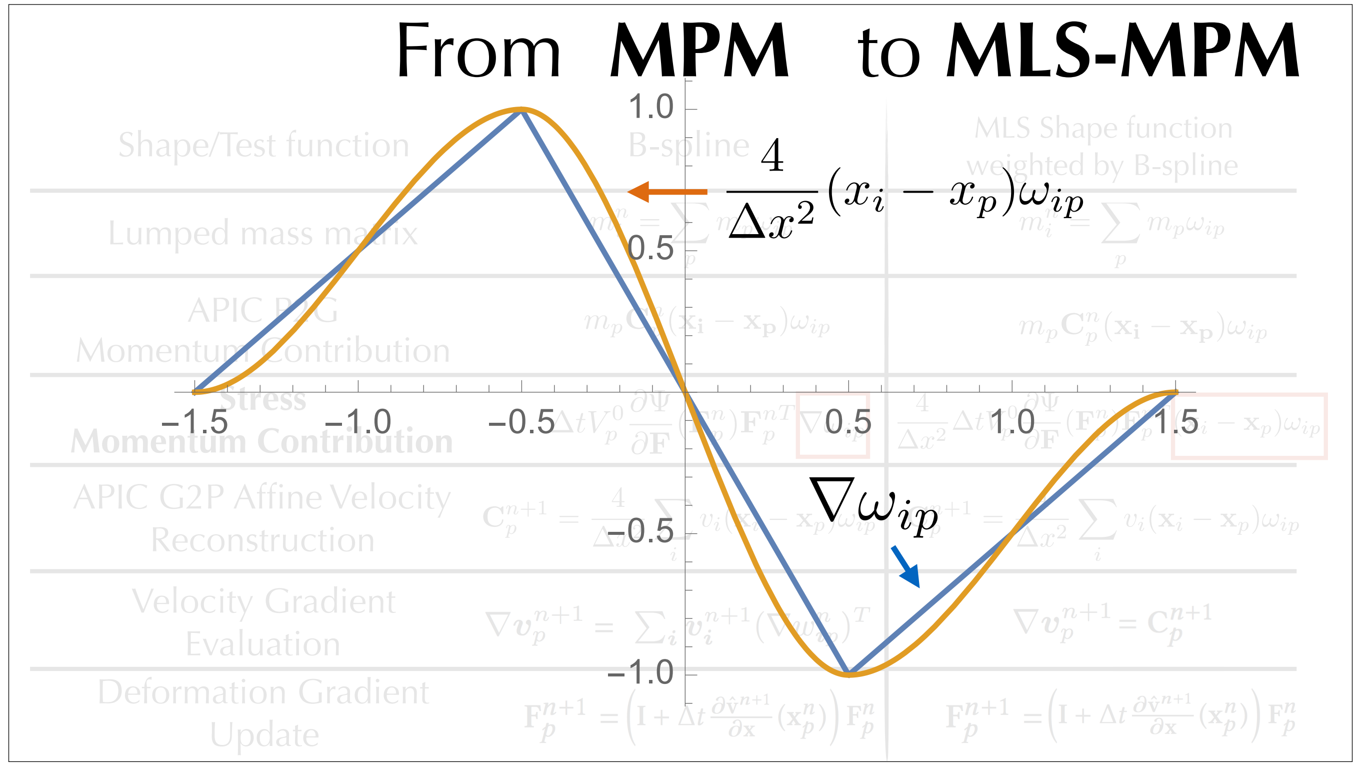
Shape/Test function	B-spline	MLS Shape function weighted by B-spline
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	$m_i^n = \sum_p m_p \omega_{ip}$
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \boxed{\nabla \omega_{ip}}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \boxed{(\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
Velocity Gradient Evaluation	$\nabla \boldsymbol{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n}+1}$
Deformation Gradient Update	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

Since the B-spline gradient is replaced by the simple MLS shape function gradient, MLS-MPM avoids the costly B-spline kernel gradients.

From **MPM** to **MLS-MPM**

Shape/Test function	B-spline	MLS Shape function weighted by B-spline
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	$m_i^n = \sum_p m_p \omega_{ip}$
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \boxed{\nabla \omega_{ip}}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \boxed{(\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
Velocity Gradient Evaluation	$\nabla \boldsymbol{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n}+1}$
Deformation Gradient Update	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

It is interesting to see how these two gradient kernels look like. In fact, they are very close to each other and are both at least first-order accurate.



It is interesting to see how these two gradient kernels look like. In fact, they are very close to each other and are both at least first-order accurate.

From **MPM** to **MLS-MPM**

Shape/Test function	B-spline	MLS Shape function weighted by B-spline
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	$m_i^n = \sum_p m_p \omega_{ip}$
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	$m_p \mathbf{C}_p^n \boxed{(\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}}$
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \nabla \omega_{ip}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \boxed{(\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
Velocity Gradient Evaluation	$\nabla \boldsymbol{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n}+1}$
Deformation Gradient Update	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

MLS-MPM fuses the computation of APIC and stress momentum contribution into a single matrix-vector multiplication, this halves the required FLOPs.

Performance

Timing (ms)	Reference	Ours (MPM)	Ours* (MPM)	Ours* (MLS-MPM)
P2G (1 thread)	4760 (1×)	5744 (0.83×)	2685 (1.77×)	1283 (3.71×)
P2G (4 threads)	1220 (1×)	1525 (0.80×)	688 (1.77×)	328 (3.72×)
G2P (1 thread)	8255 (1×)	7476 (1.10×)	1144 (7.21×)	589 (14.01×)
G2P (4 threads)	2070 (1×)	2011 (1.03×)	313 (6.61×)	163 (12.70×)

Reference: Tampubolon et al. 2017.
Multi-species simulation of porous sand and water mixtures

We benchmarked our solver against a reliable implementation from Tampubolon et al.

Performance

Timing (ms)	Reference	Ours (MPM)	Ours* (MPM)	Ours* (MLS-MPM)
P2G (1 thread)	4760 (1×)	5744 (0.83×)	2685 (1.77×)	1283 (3.71×)
P2G (4 threads)	1220 (1×)	1525 (0.80×)	688 (1.77×)	328 (3.72×)
G2P (1 thread)	8255 (1×)	7476 (1.10×)	1144 (7.21×)	589 (14.01×)
G2P (4 threads)	2070 (1×)	2011 (1.03×)	313 (6.61×)	163 (12.70×)

Baseline: Traditional MPM

Our unoptimized implementation has a comparable performance to theirs.

Performance

Timing (ms)	Reference	Ours (MPM)	Ours* (MPM)	Ours* (MLS-MPM)
P2G (1 thread)	4760 (1×)	5744 (0.83×)	2685 (1.77×)	1283 (3.71×)
P2G (4 threads)	1220 (1×)	1525 (0.80×)	688 (1.77×)	328 (3.72×)
G2P (1 thread)	8255 (1×)	7476 (1.10×)	1144 (7.21×)	589 (14.01×)
G2P (4 threads)	2070 (1×)	2011 (1.03×)	313 (6.61×)	163 (12.70×)

Optimized Traditional MPM

(Low-level performance engineering)

Then we do some low-level performance engineering on traditional MPM. This makes P2G 1.6x faster and G2P 7x faster.

Performance

Timing (ms)	Reference	Ours (MPM)	Ours* (MPM)	Ours* (MLS-MPM)
P2G (1 thread)	4760 (1×)	5744 (0.83×)	2685 (1.77×)	1283 (3.71×)
P2G (4 threads)	1220 (1×)	1525 (0.80×)	688 (1.77×)	328 (3.72×)
G2P (1 thread)	8255 (1×)	7476 (1.10×)	1144 (7.21×)	589 (14.01×)
G2P (4 threads)	2070 (1×)	2011 (1.03×)	313 (6.61×)	163 (12.70×)

Optimized MLS-MPM

(algorithmic improvement)

Finally, we implement MLS-MPM.

Performance

2.10x faster P2G
1.94x faster G2P

Timing (ms)	Reference	Ours (MPM)	Ours* (MPM)	Ours* (MLS-MPM)
P2G (1 thread)	4760 (1×)	5744 (0.83×)	2685 (1.77×)	1283 (3.71×)
P2G (4 threads)	1220 (1×)	1525 (0.80×)	688 (1.77×)	328 (3.72×)
G2P (1 thread)	8255 (1×)	7476 (1.10×)	1144 (7.21×)	589 (14.01×)
G2P (4 threads)	2070 (1×)	2011 (1.03×)	313 (6.61×)	163 (12.70×)

Optimized MLS-MPM

This algorithmic improvement gives us 2x further speed-up.

Contributions

- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and easier
- **Part II: Compatible Particle-in-Cell**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

So that's the first part, where we unified APIC and MPM force discretization. The result is elegant and efficient, and more importantly, easier to implement.
Ladies and gentlemen, it's not one thousand lines of code, not one hundred lines of code, not even ten lines of code. [click] It's negative 100 lines of code. At the same time, it is mathematically sound and runs faster. The more you use, the less code you write, and the faster your code runs. There is literally no reason not to use it.

Contributions

- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faste

-100 lines of code!

- **Part II: Compatible Particle-In-Cell**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

So that's the first part, where we unified APIC and MPM force discretization. The result is elegant and efficient, and more importantly, easier to implement.
Ladies and gentlemen, it's not one thousand lines of code, not one hundred lines of code, not even ten lines of code. [click] It's negative 100 lines of code. At the same time, it is mathematically sound and runs faster. The more you use, the less code you write, and the faster your code runs. There is literally no reason not to use it.

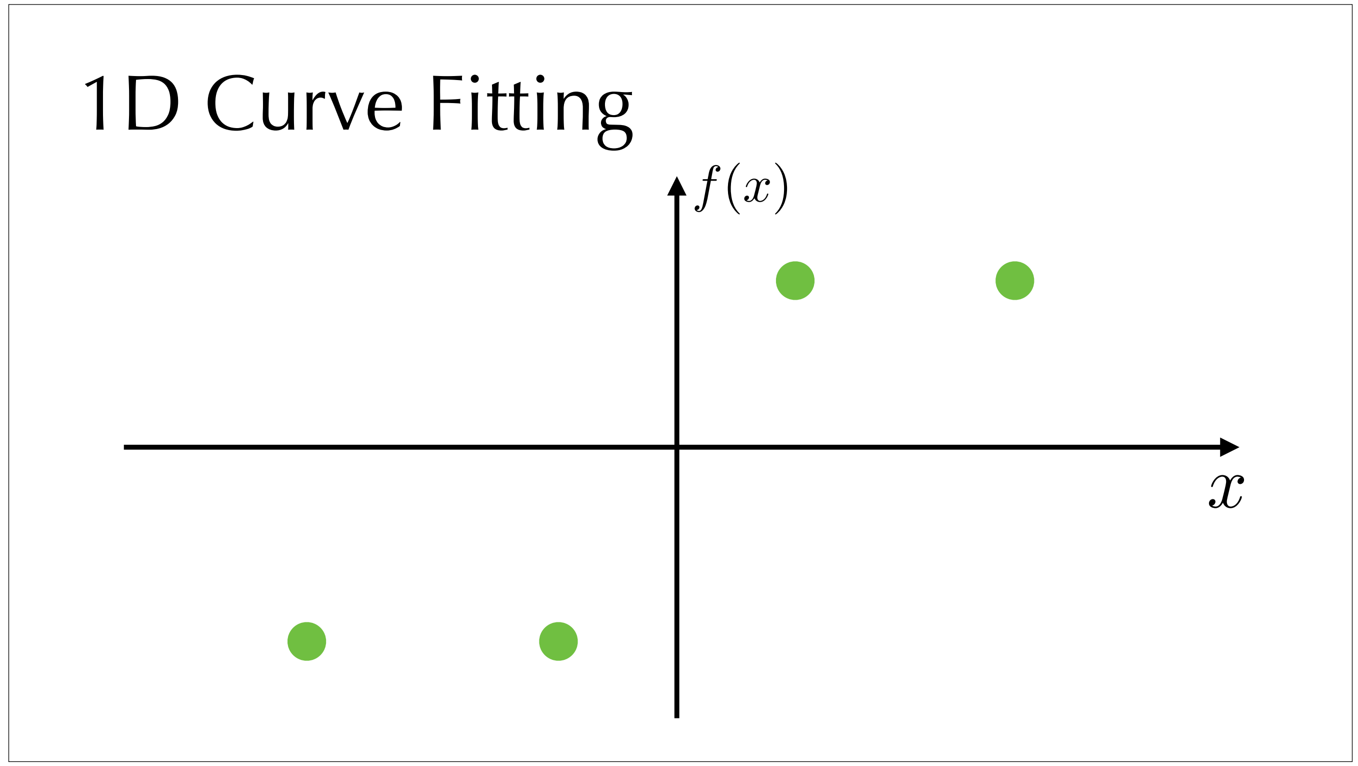
Contributions

- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and Easier
- **Part II: Compatible Particle-in-Cell**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

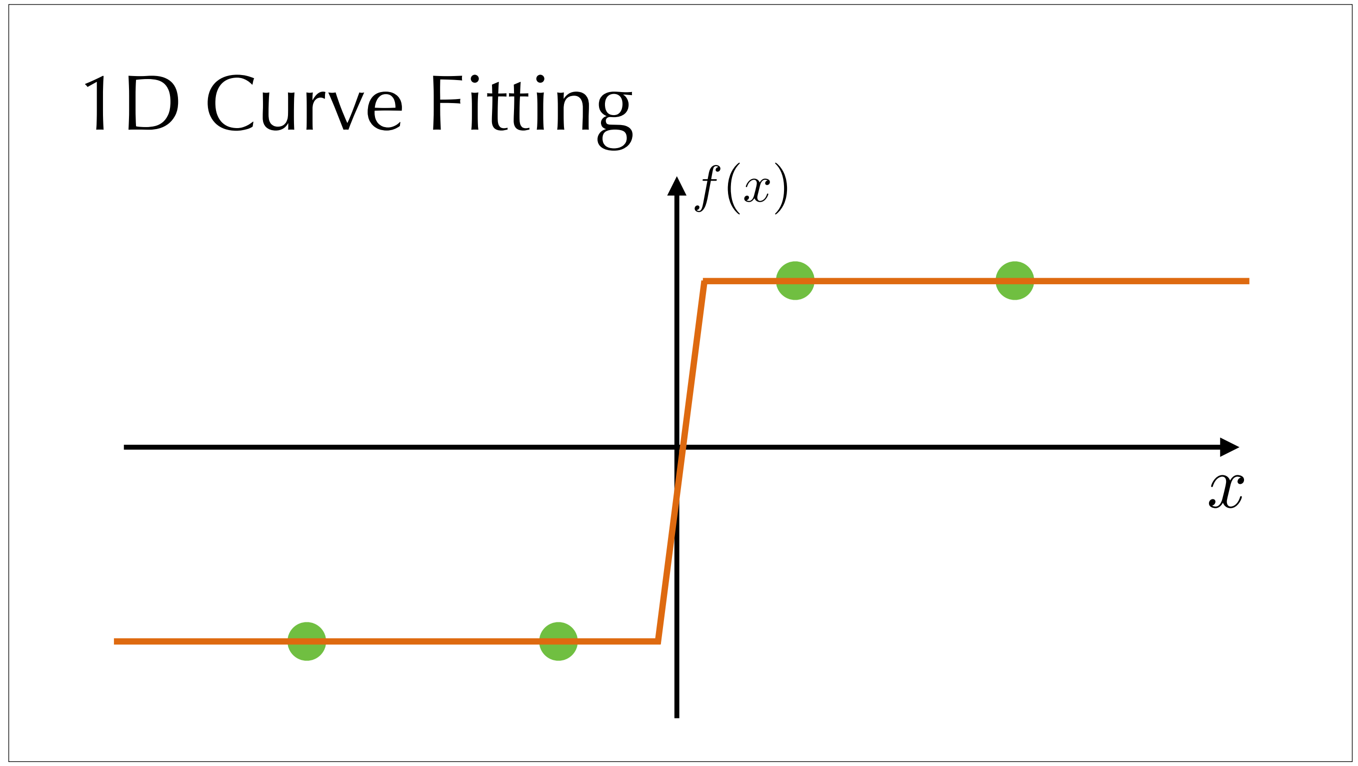
Hopefully I have waken people up. Let’s move on to the second part, the compatible particle-in-cell method.

Suppose you get an armadillo. As a simulation guy you can’t wait to do something bad on it. [click] For example, you may want to cut it. However, this turns out to be very hard to MPM.

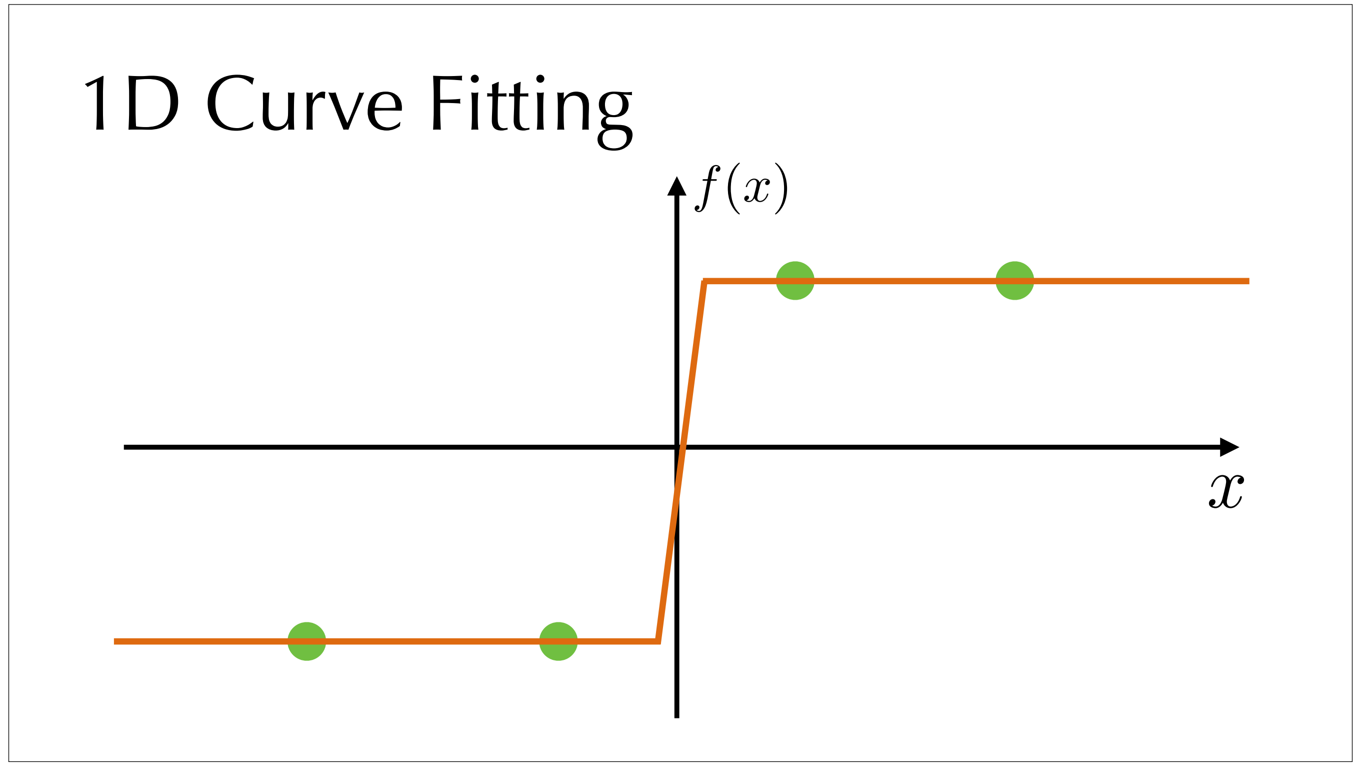
Suppose you get an armadillo. As a simulation guy you can’t wait to do something bad on it. [click] For example, you may want to cut it. However, this turns out to be very hard to MPM.



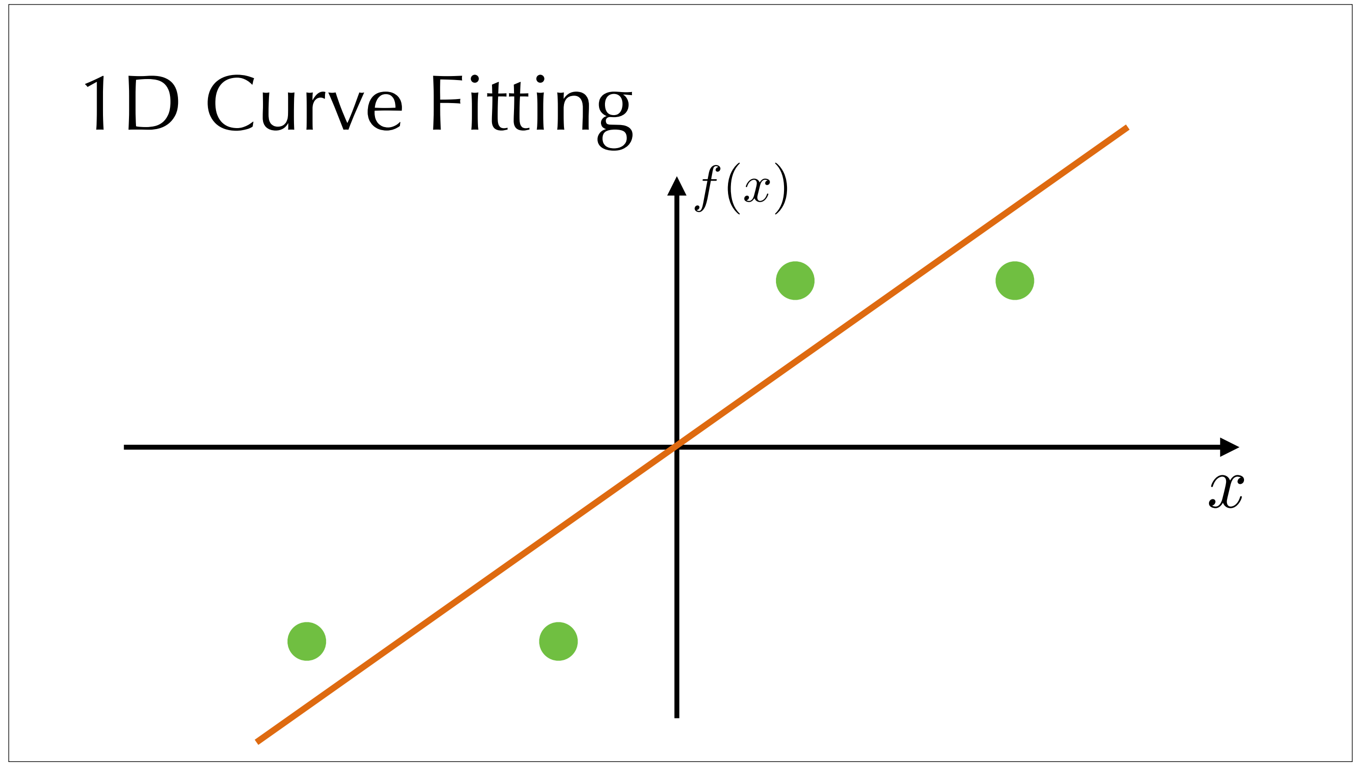
Let's ask ourselves what is cutting. Clearly the desired behaviour is material separation after cutting. Suppose we are cutting the material at the origin, we want the left part moving to the left while the right part moving to the right.



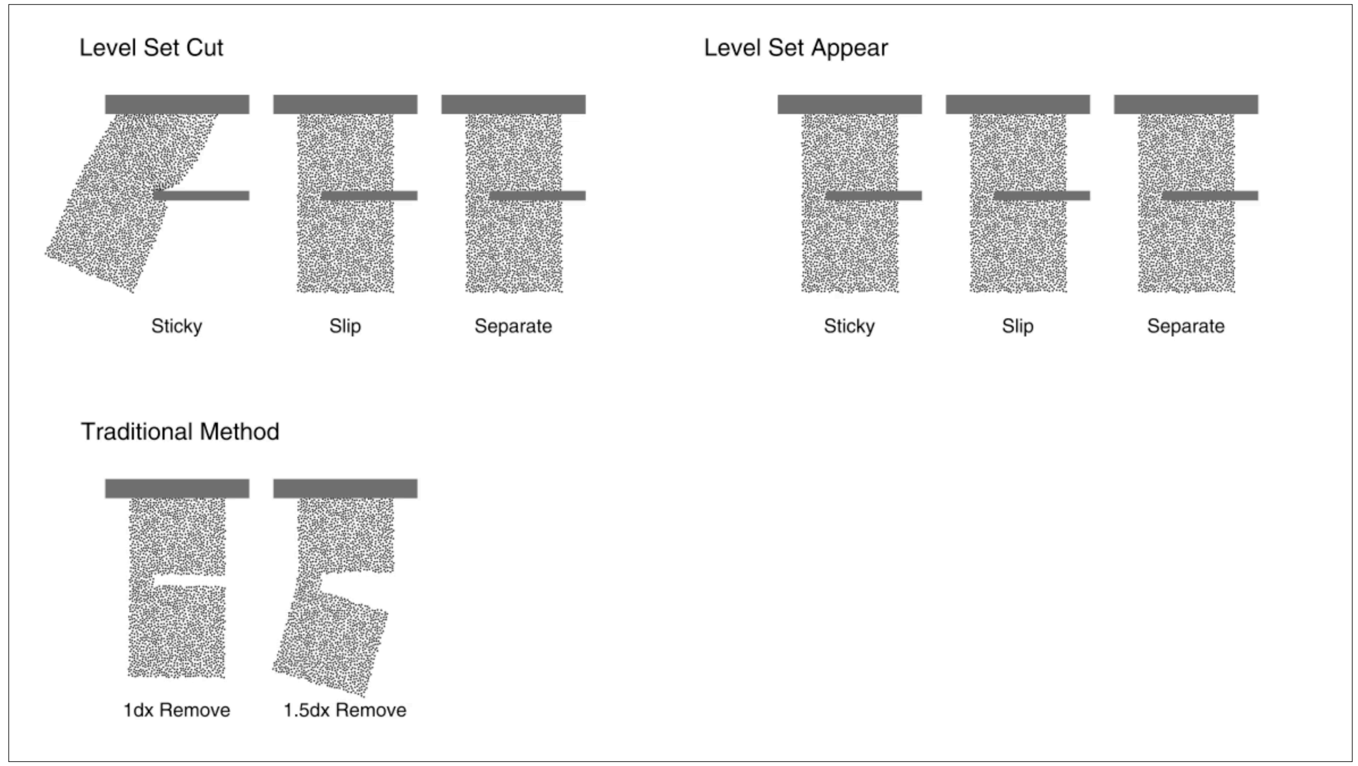
This means we want the reconstructed velocity field to look like this: a sharp discontinuity as the cut point.



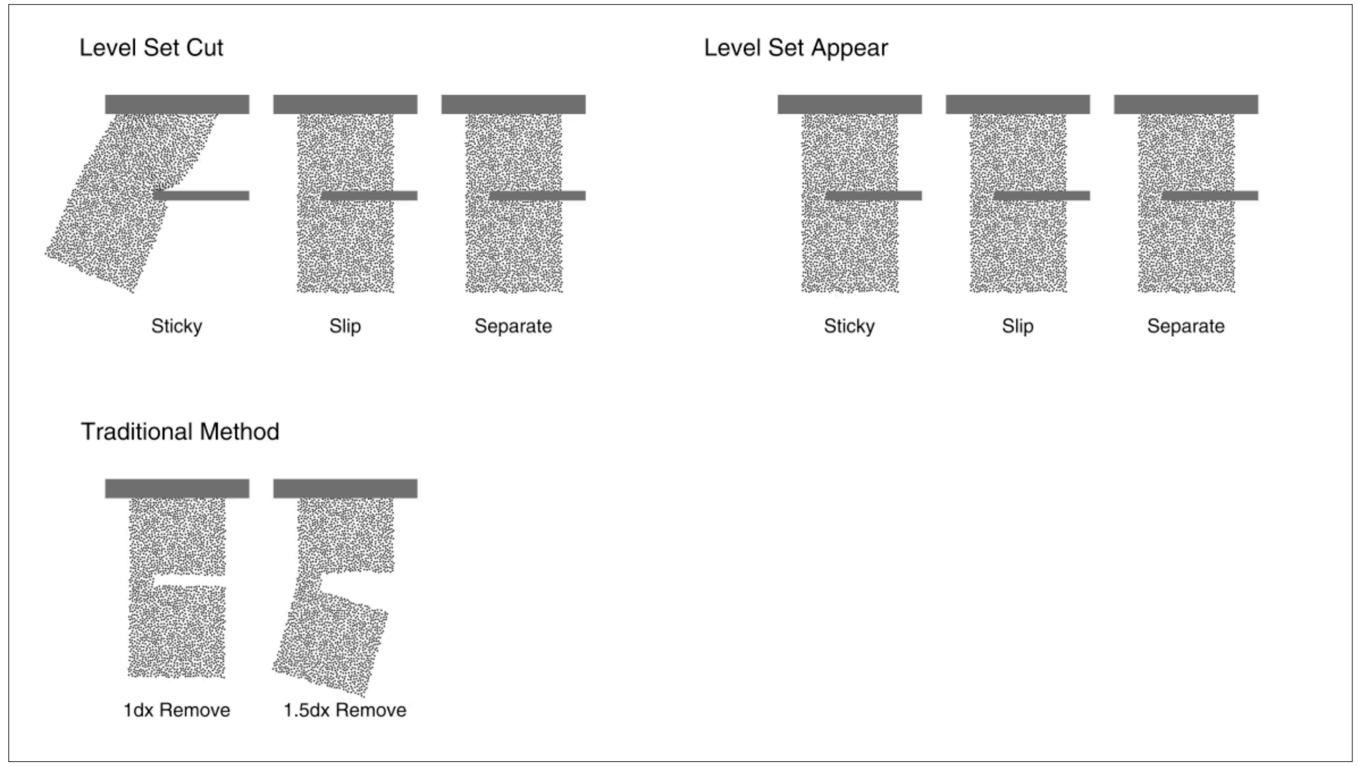
Unfortunately this is not supported by PIC, APIC and even PolyPIC. They only have continuous basis functions.



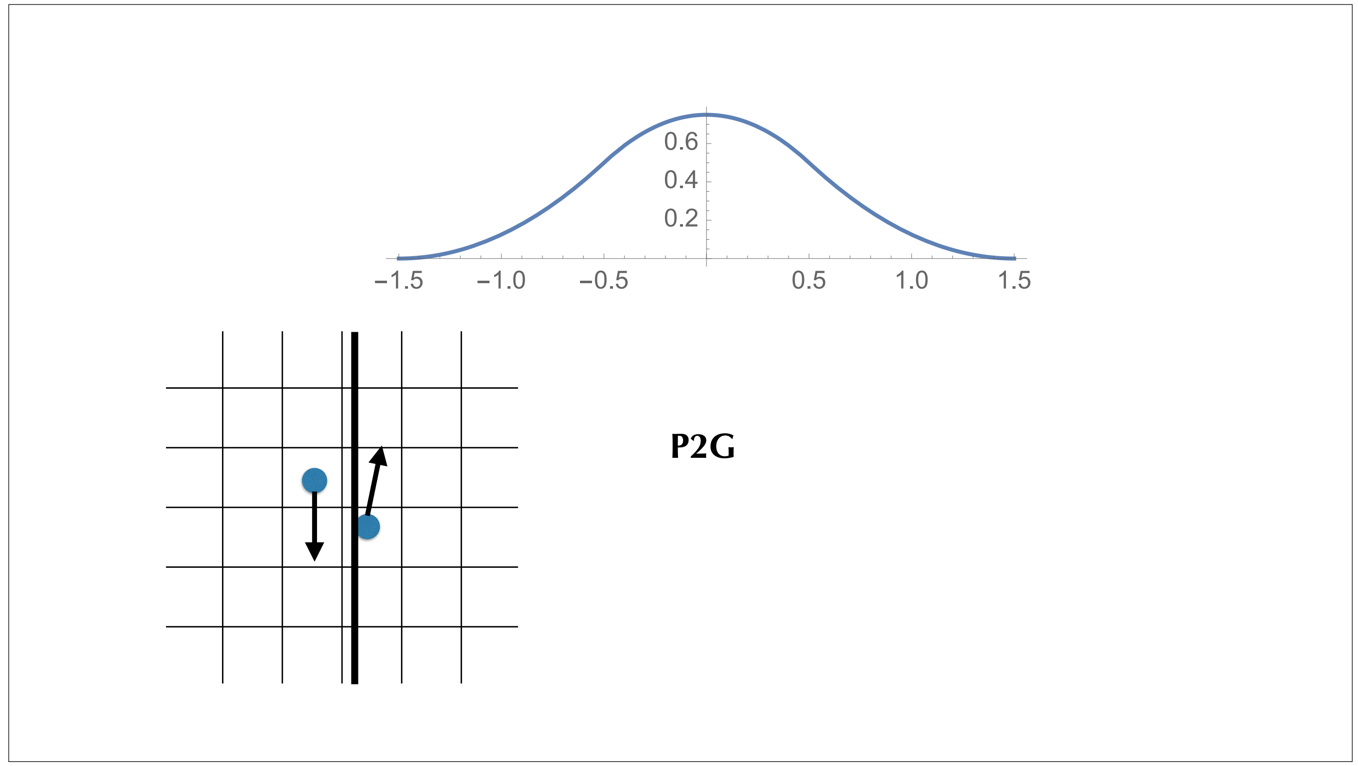
If we use APIC, we will get this.



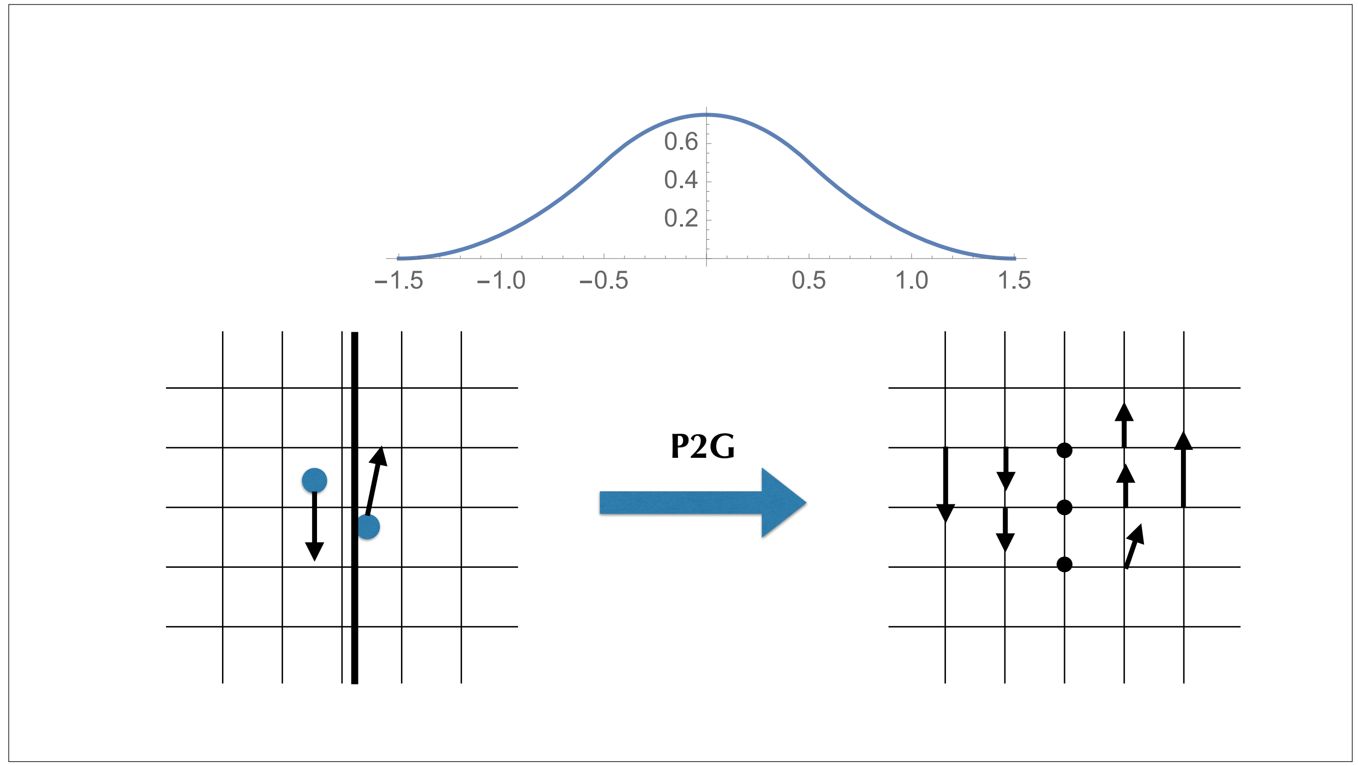
In traditional MPM we cannot actually simulate cutting. [click]
A moving thin level set of any boundary conditions will either be treated as a collider or completely ignored by the material.
Putting the level set directly inside also doesn't work, since the interpolated velocity field is still continuous.
Other common approaches include particle deletion and softening. However, due to the fuzzy nature of PIC kernels, a significant amount of particles must be deleted for separation. [Wait]
Softening creates unpleasant artifacts.



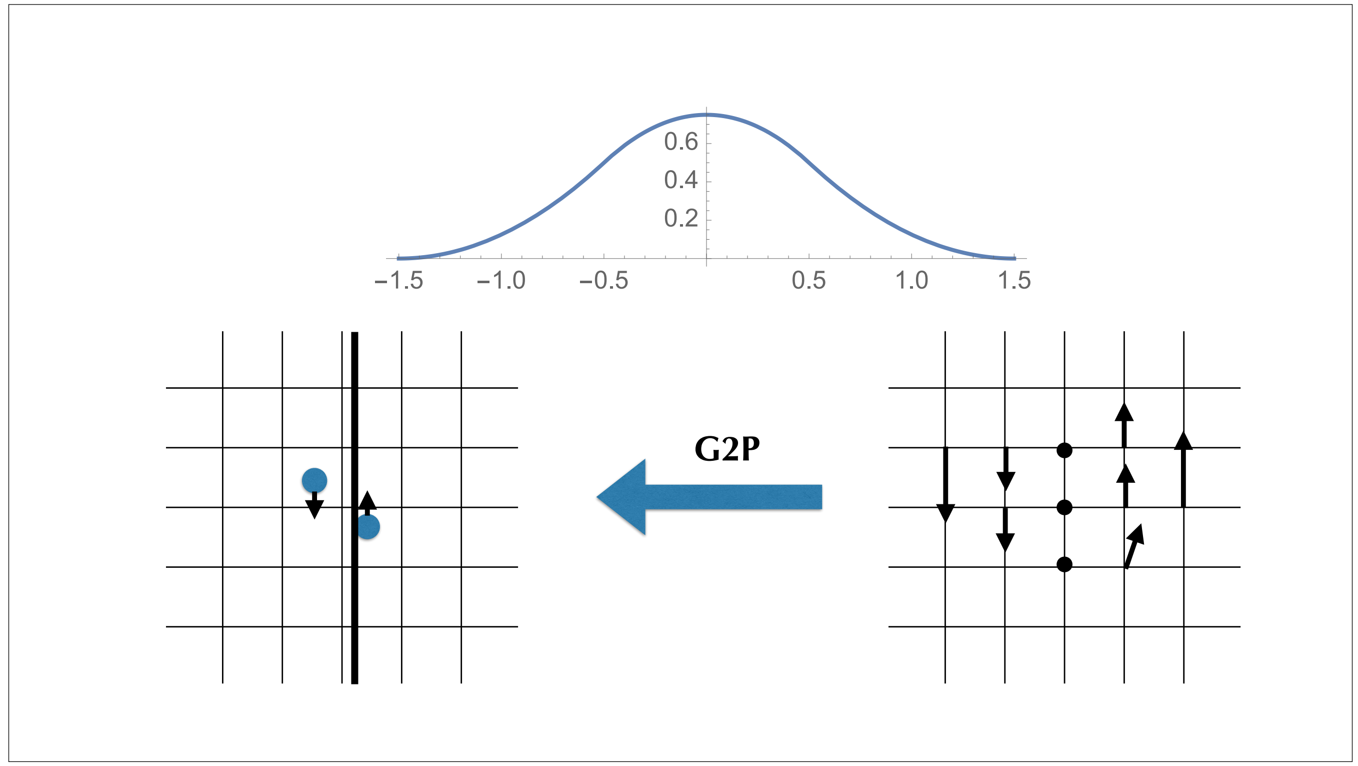
In traditional MPM we cannot actually simulate cutting. [click]
A moving thin level set of any boundary conditions will either be treated as a collider or completely ignored by the material.
Putting the level set directly inside also doesn't work, since the interpolated velocity field is still continuous.
Other common approaches include particle deletion and softening. However, due to the fuzzy nature of PIC kernels, a significant amount of particles must be deleted for separation. [Wait]
Softening creates unpleasant artifacts.



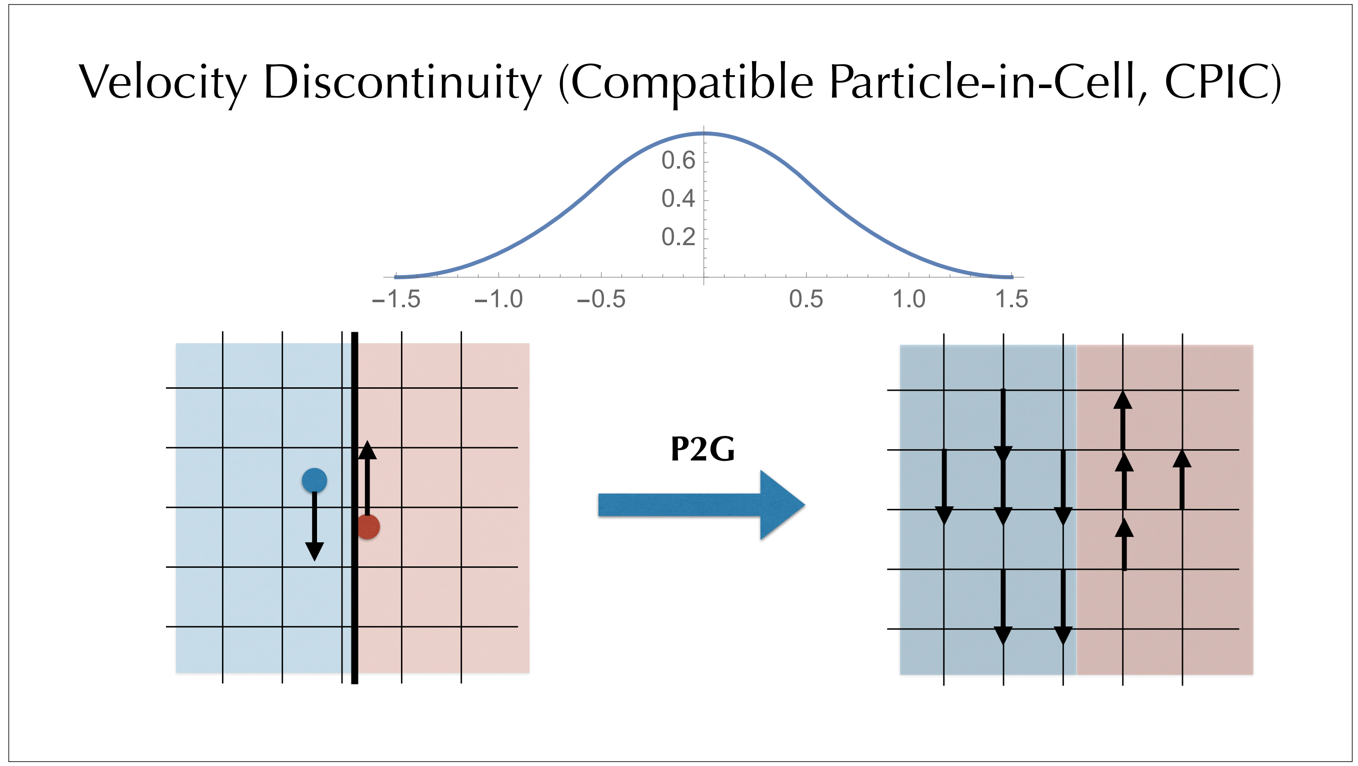
From the MPM point of view, the relative motion of two particles will be smoothed out during P2G.



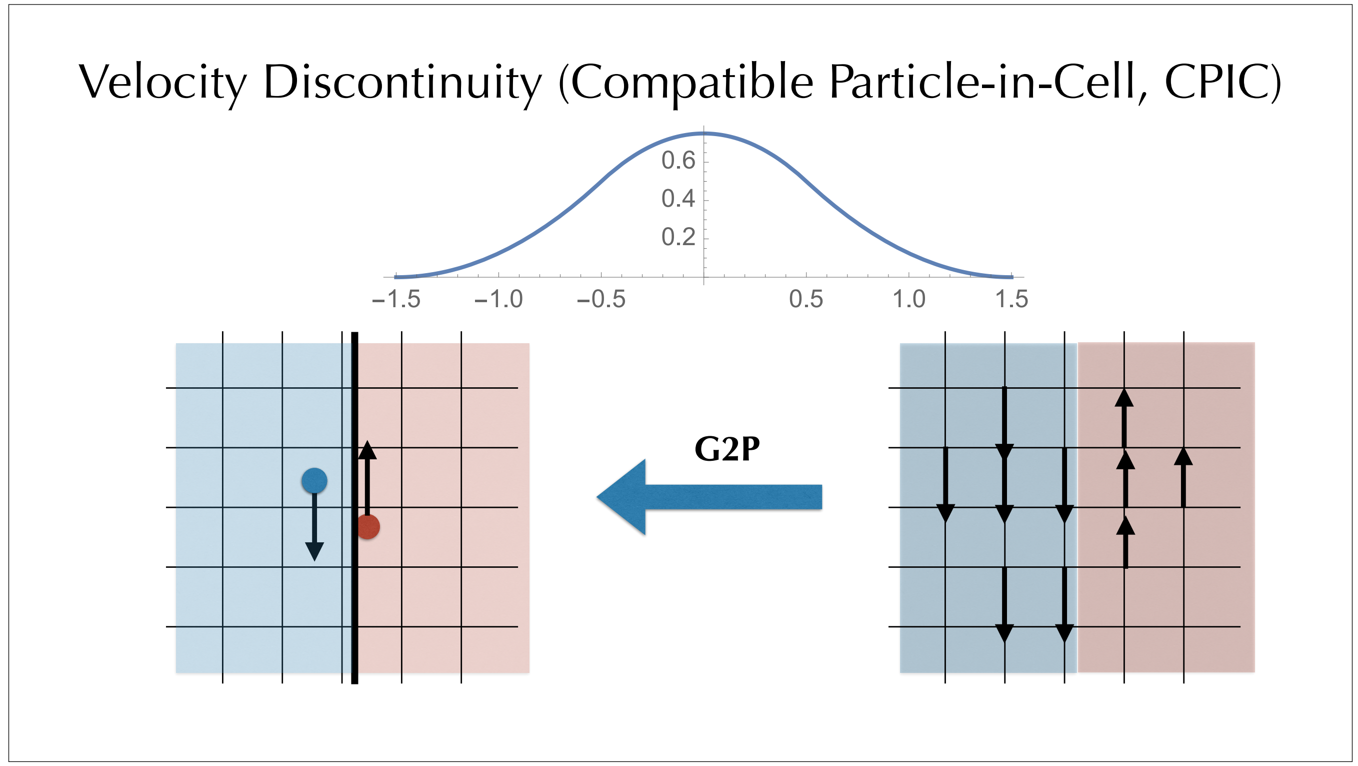
From the MPM point of view, the relative motion of two particles will be smoothed out during P2G.



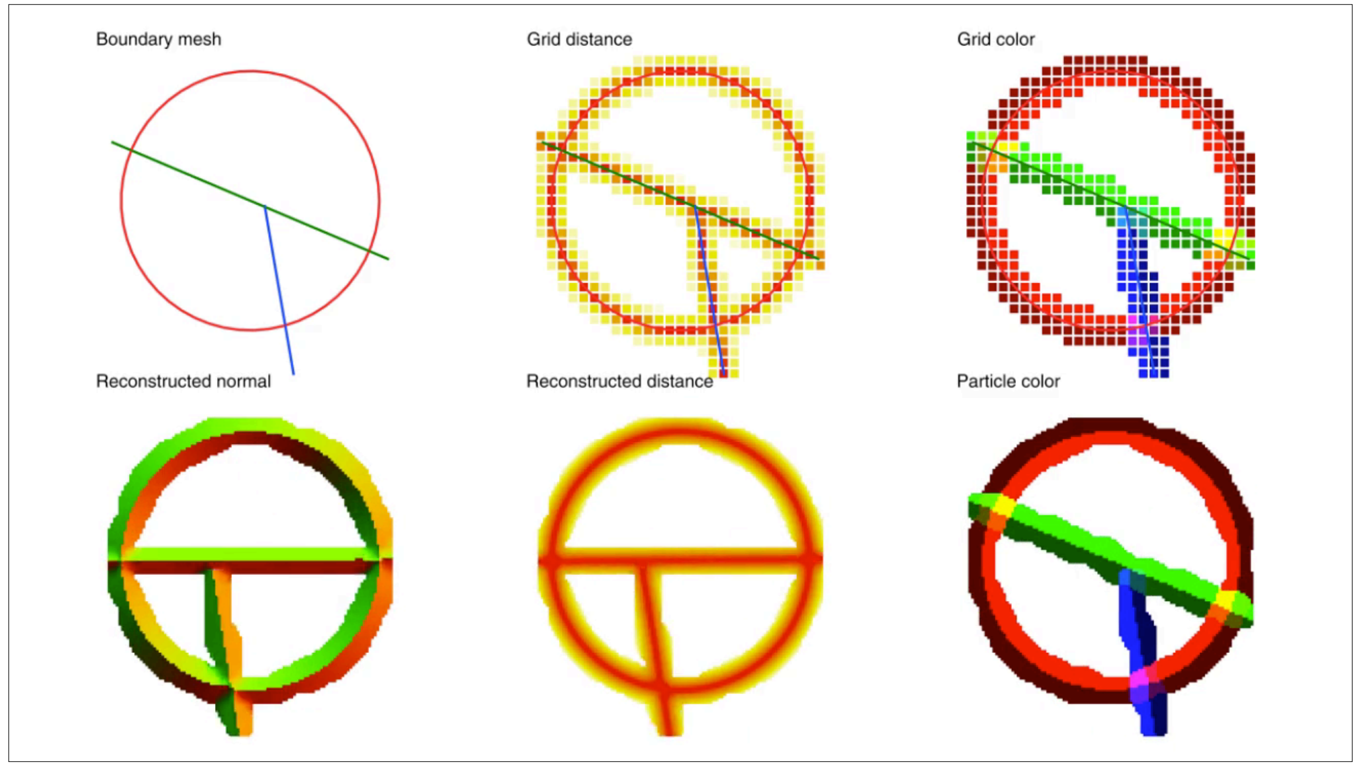
During resample, because the relative motion on grid is smoothed out by the kernel, the particles will gather almost a rigid velocity field. In other words, a collision happens.



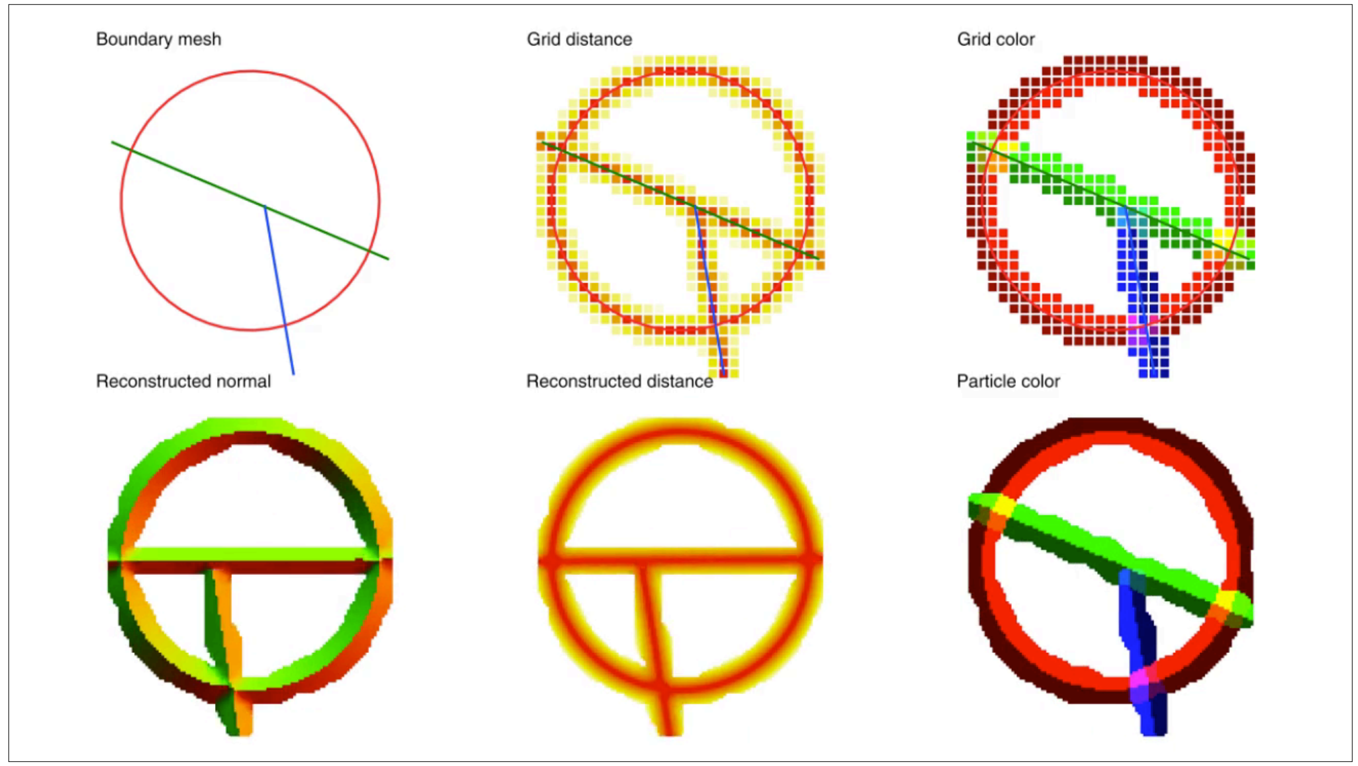
Unfortunately there is no existing way that can avoid such smoothing. We propose a method to solve this issue. This is achieved by assigning colours to both particles and grids, and particle only interact with nodes with the same color.



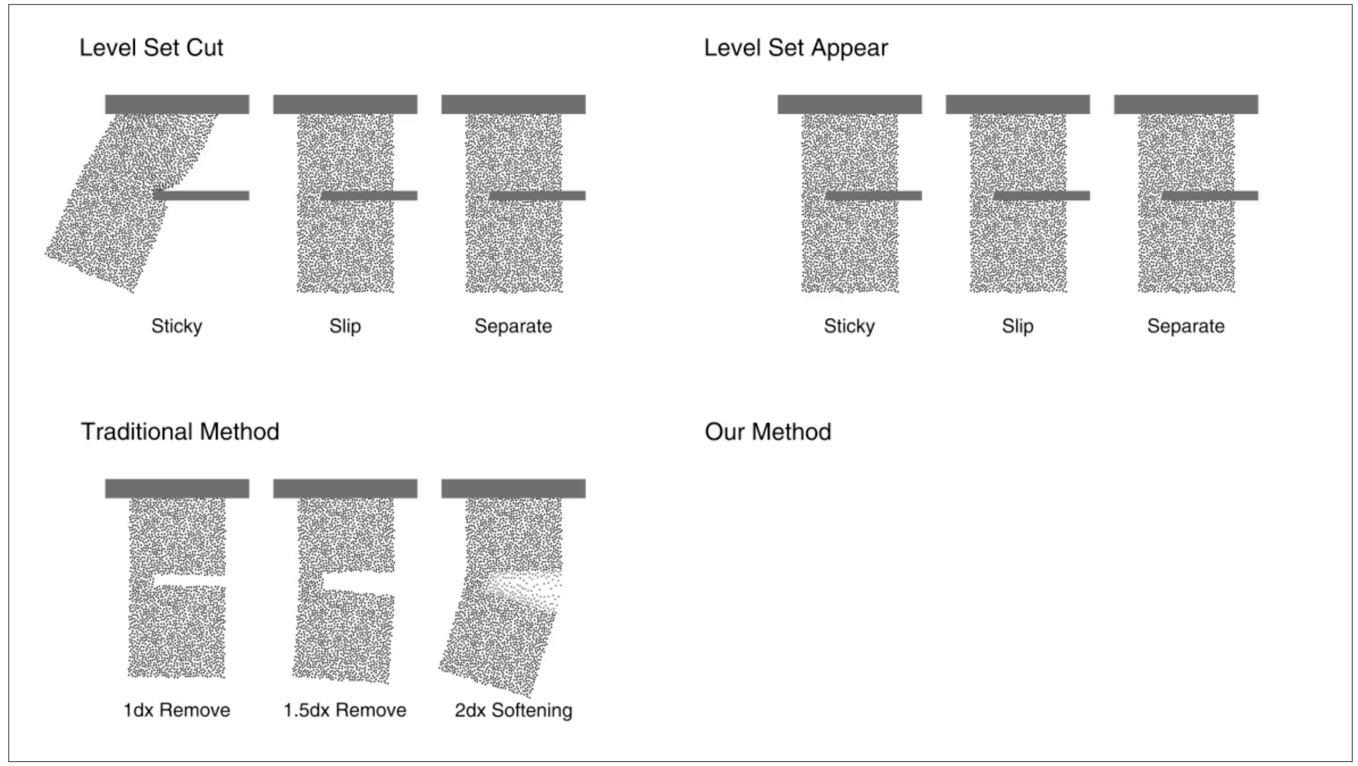
By doing this, no smoothing happens at the boundary.



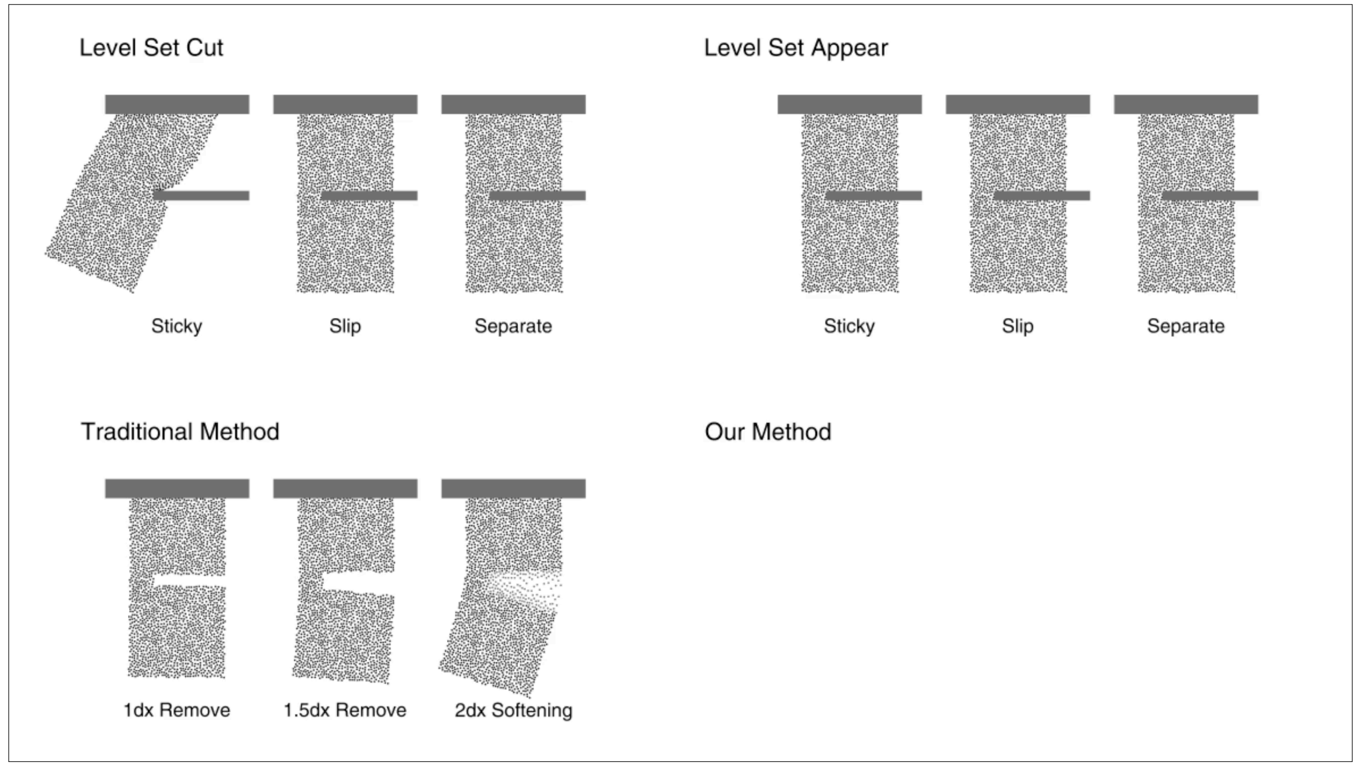
We introduce the coloured distance field, which generalises the traditional signed distance field to represent self-intersecting and open boundary. We start with a boundary mesh [wait], then rasterise the distance to mesh to grid nodes [wait]. Then we rasterise the color of meshes to classify the grid nodes. [wait] On particles, we can reconstruct the distance and normal, again, using moving least squares.



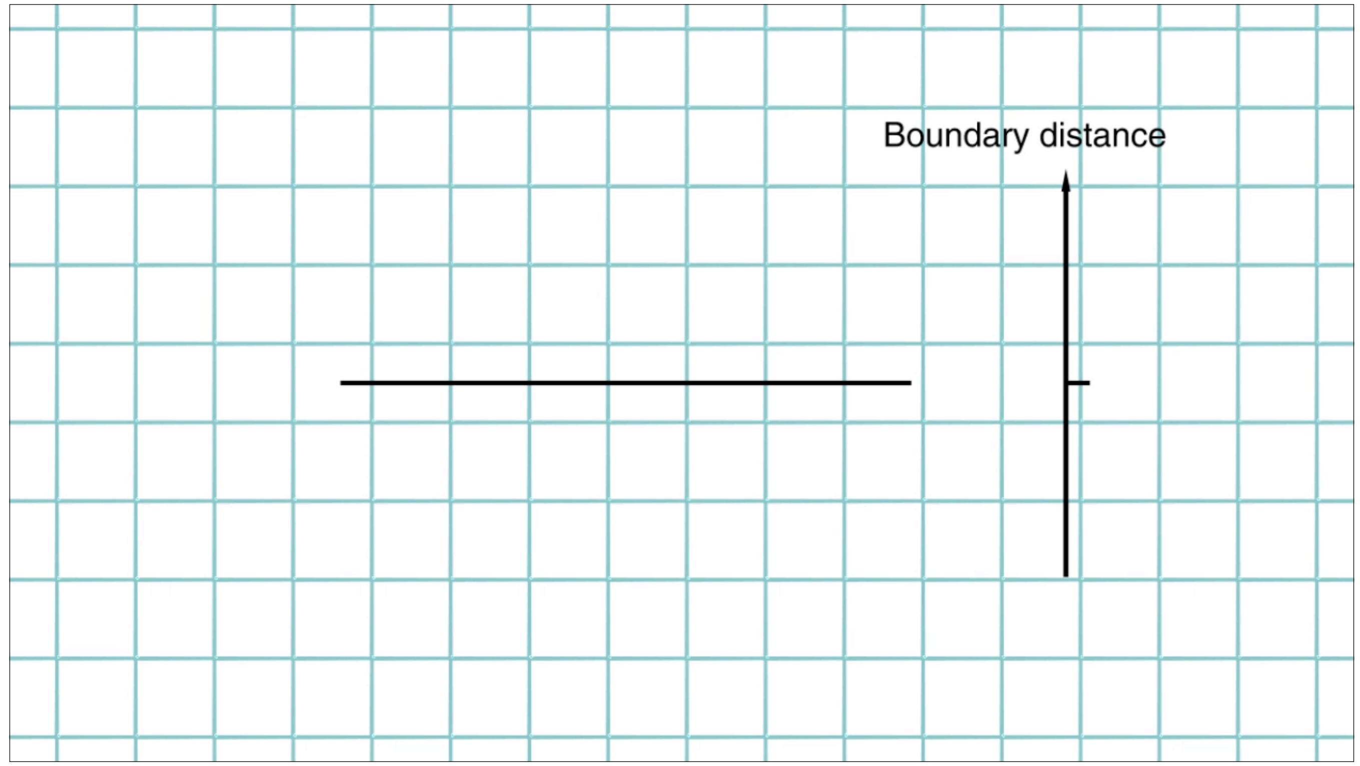
We introduce the coloured distance field, which generalises the traditional signed distance field to represent self-intersecting and open boundary. We start with a boundary mesh [wait], then rasterise the distance to mesh to grid nodes [wait]. Then we rasterise the color of meshes to classify the grid nodes. [wait] On particles, we can reconstruct the distance and normal, again, using moving least squares.



[click] With all these included, our MPM world becomes much sharper. We can incrementally cut a piece a jelly [wait], or cut it instantly. [wait] We can even do more than one cuts.

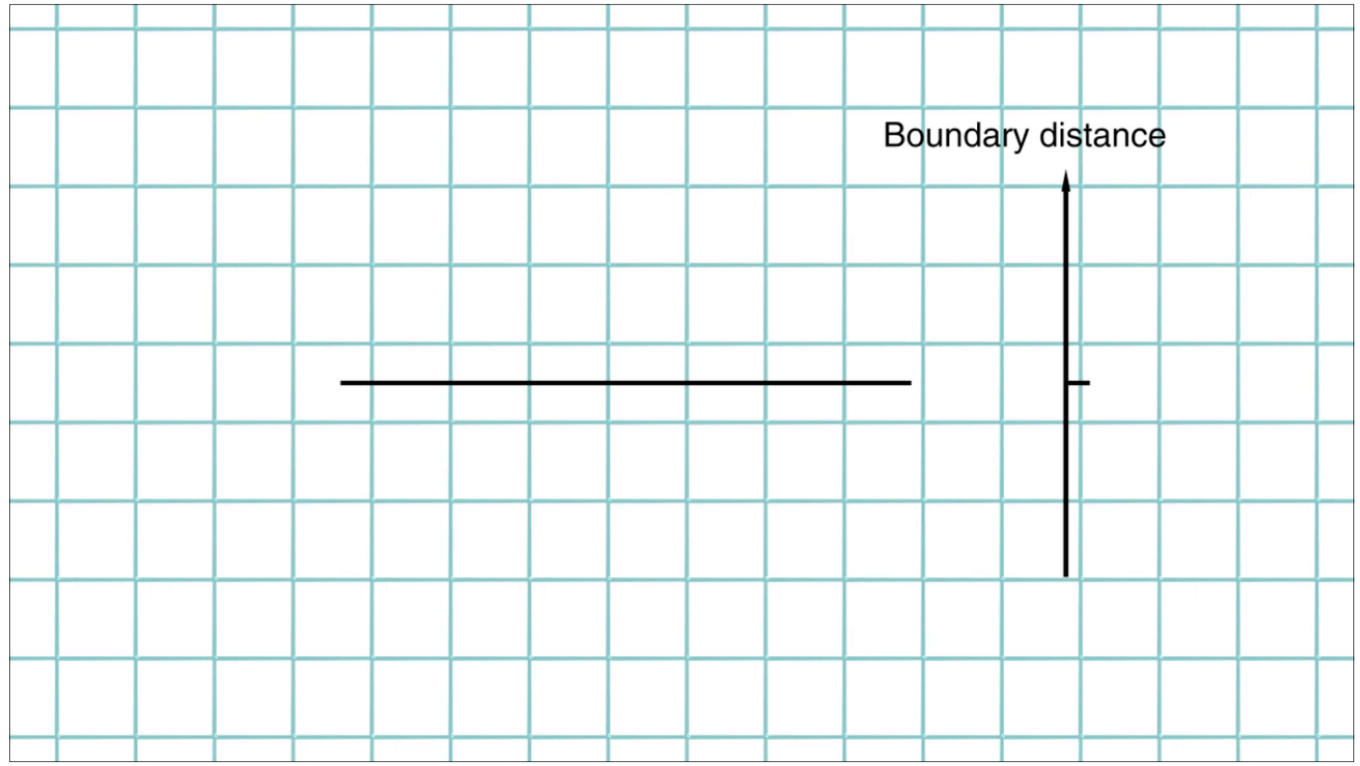


[click] With all these included, our MPM world becomes much sharper. We can incrementally cut a piece a jelly [wait], or cut it instantly. [wait] We can even do more than one cuts.



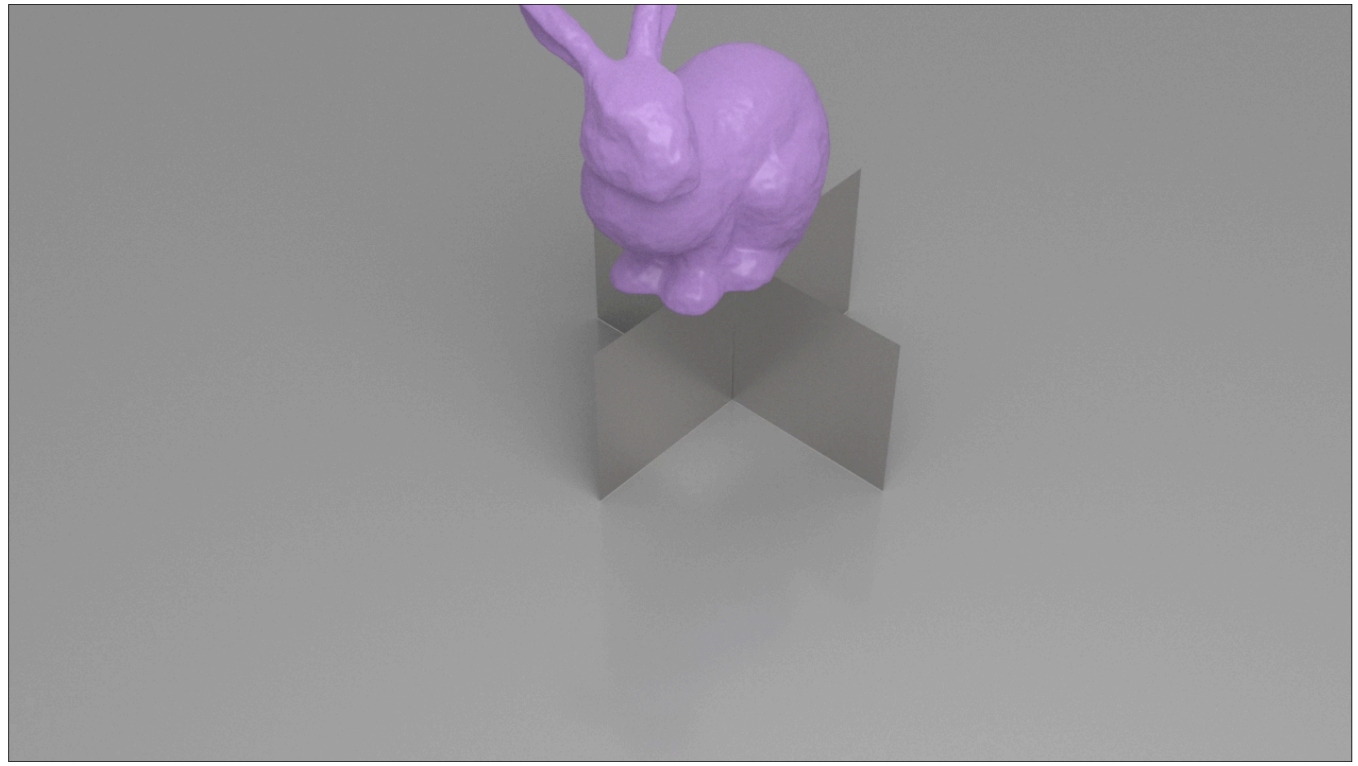
The remaining problem is, how can we assign a colours to particles? Particles gain corresponding color when it moves close to the boundary. [DO not wait]

note that the color will be persevered even if it moves across the boundary.

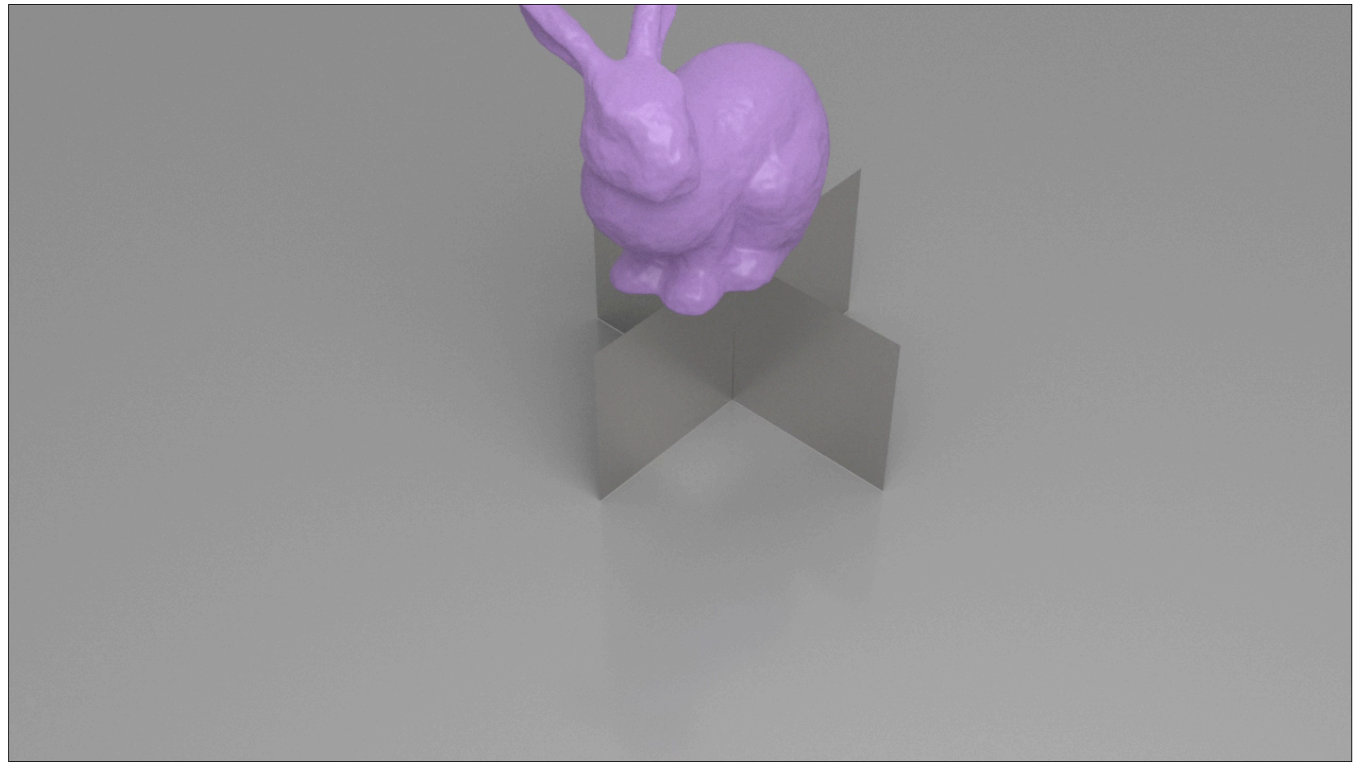


The remaining problem is, how can we assign a colours to particles? Particles gain corresponding color when it moves close to the boundary. [DO not wait]

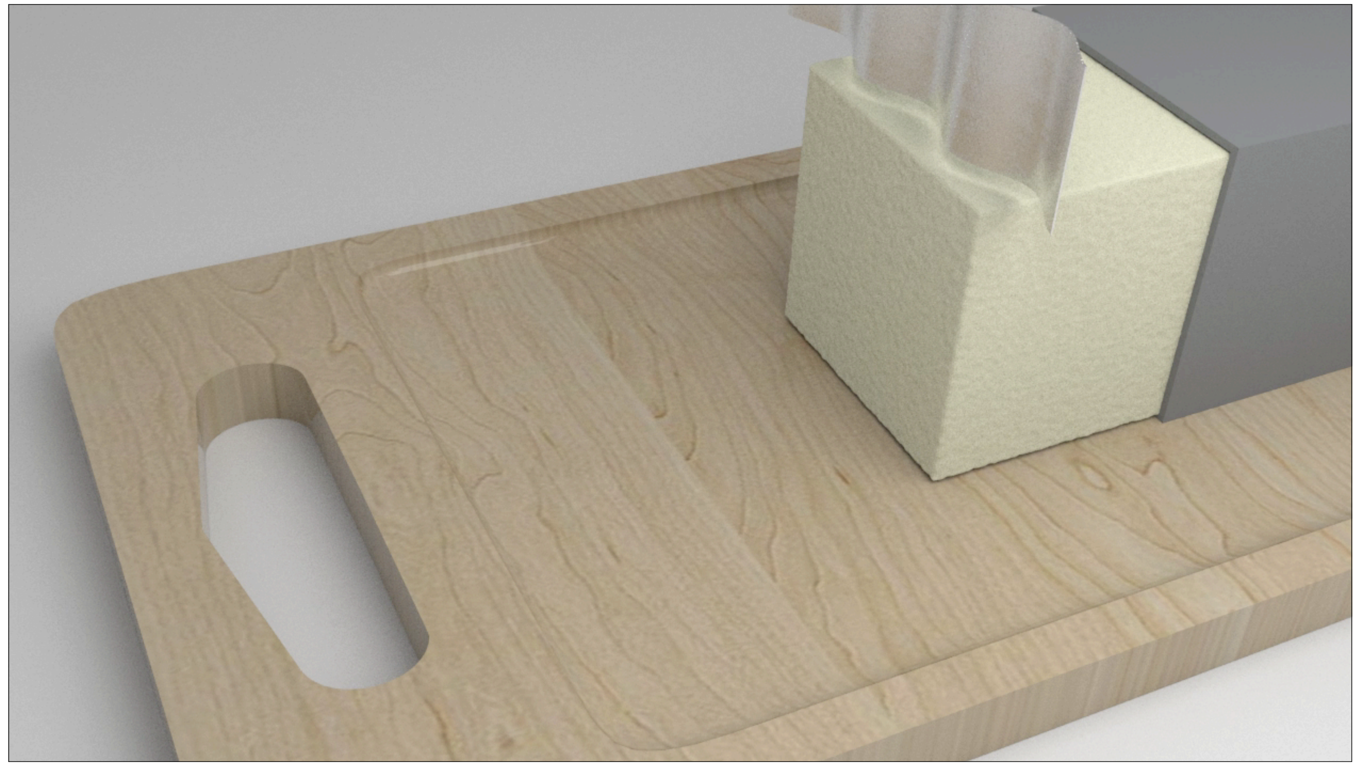
note that the color will be persevered even if it moves across the boundary.



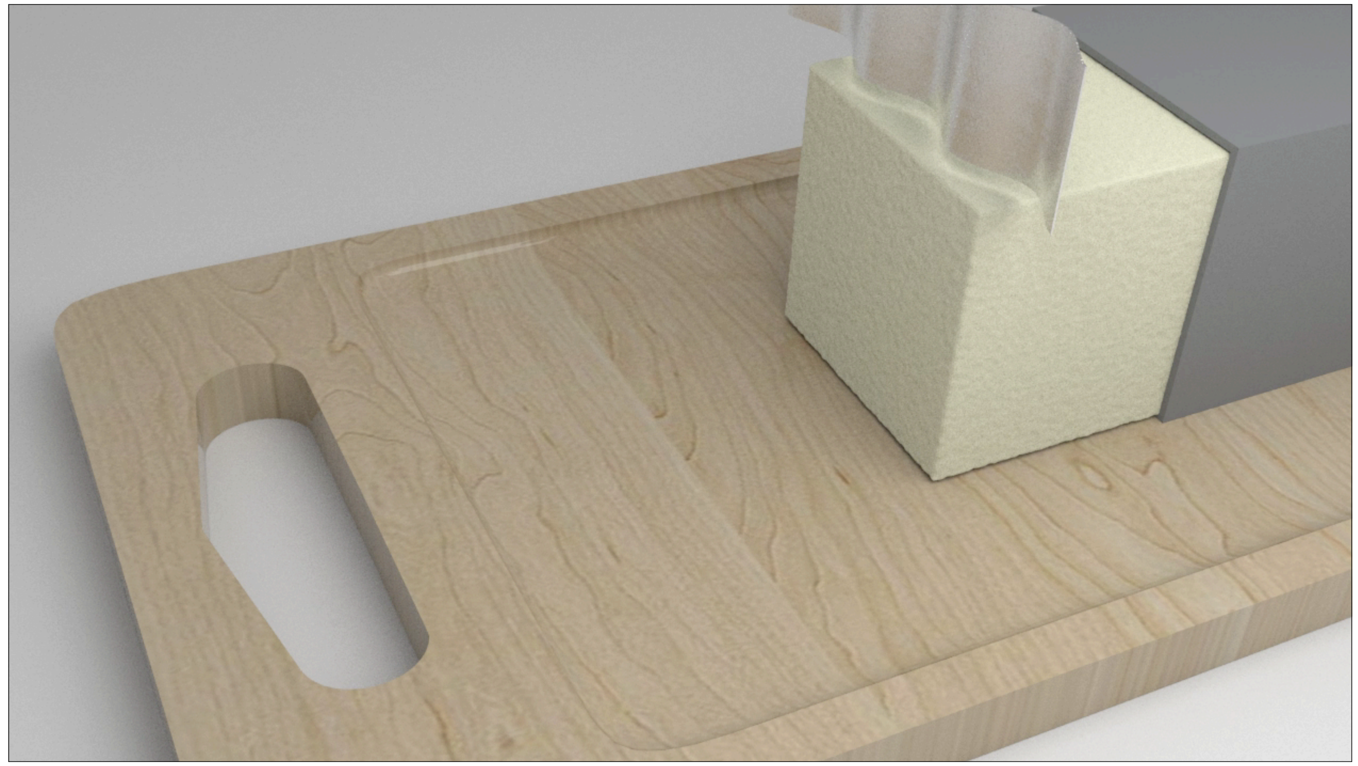
Now we are ready to cut a bunny.



Now we are ready to cut a bunny.



and some cheese

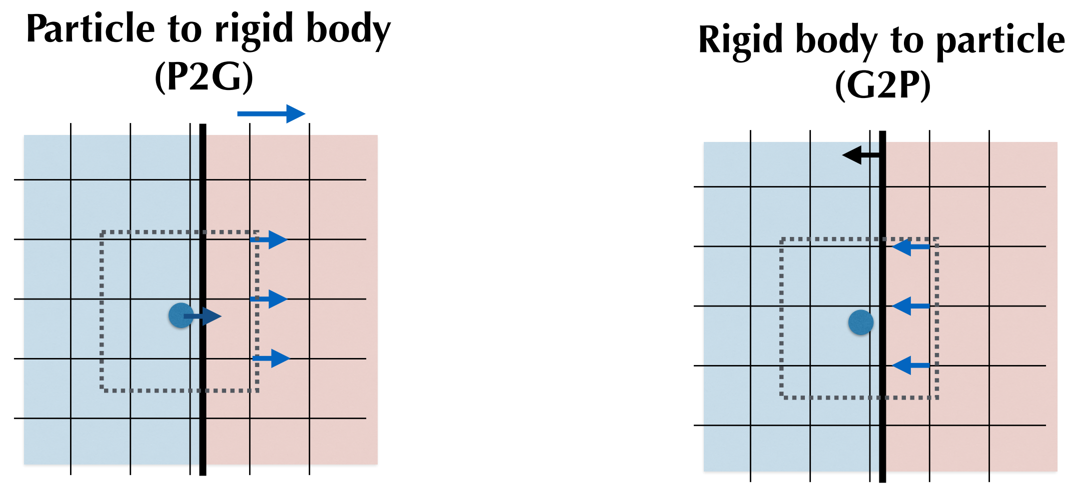


and some cheese

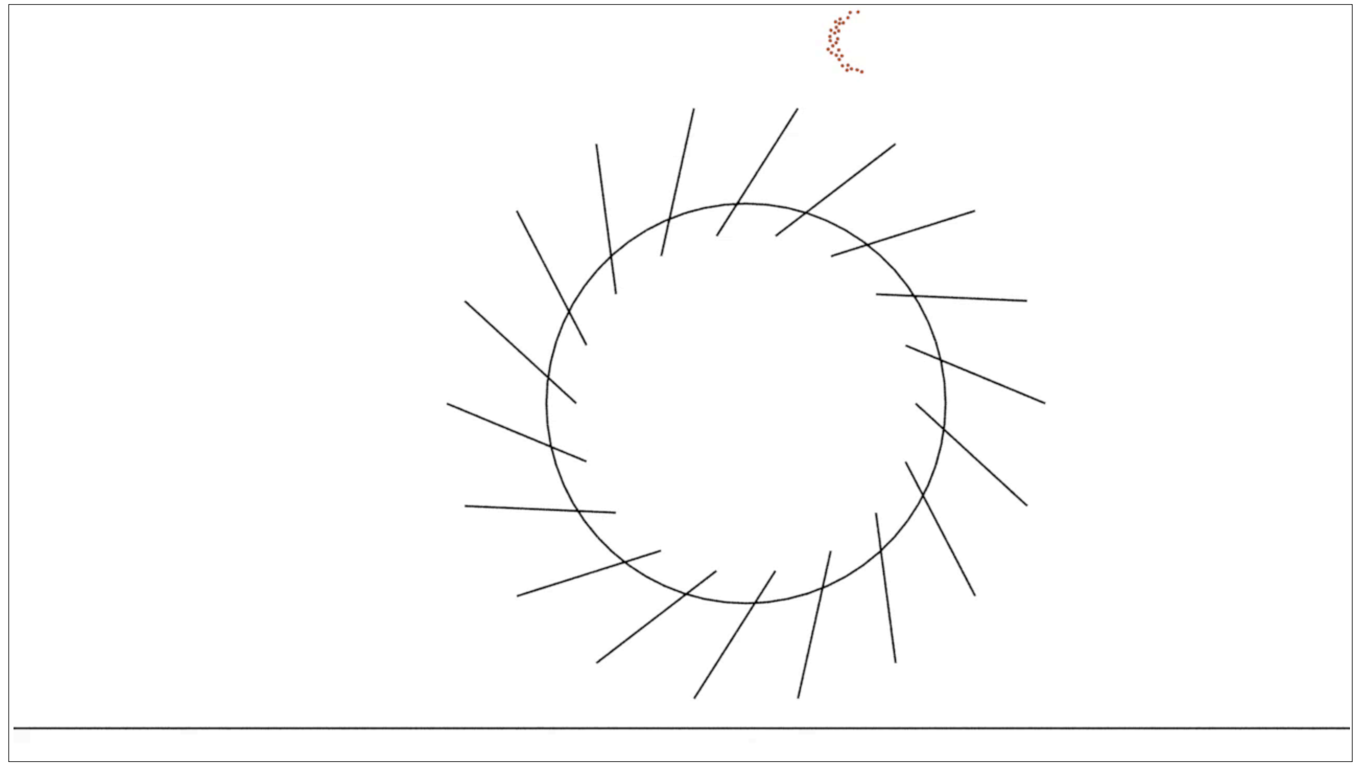
Cutting actually cover more natural phenomena. For example, blending.

Cutting actually cover more natural phenomena. For example, blending.

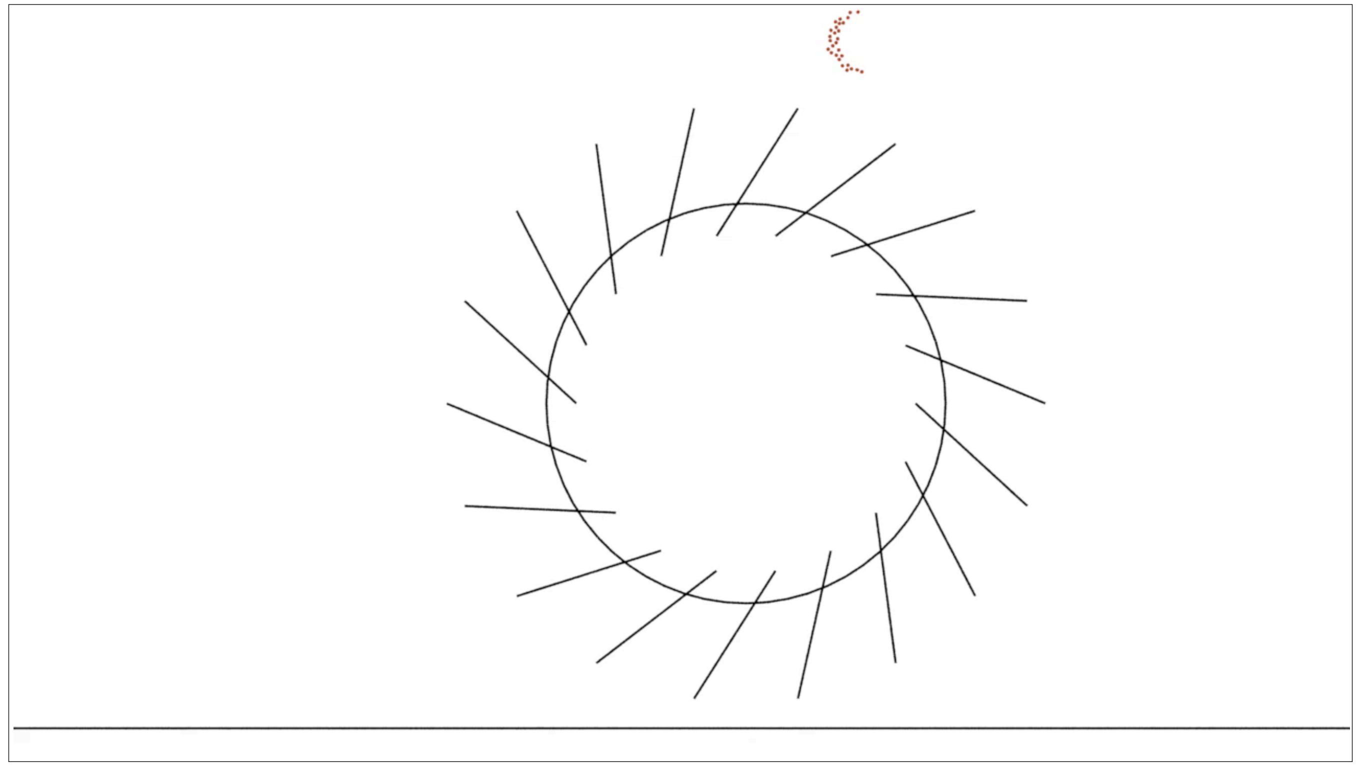
Two-way Rigid Body Coupling



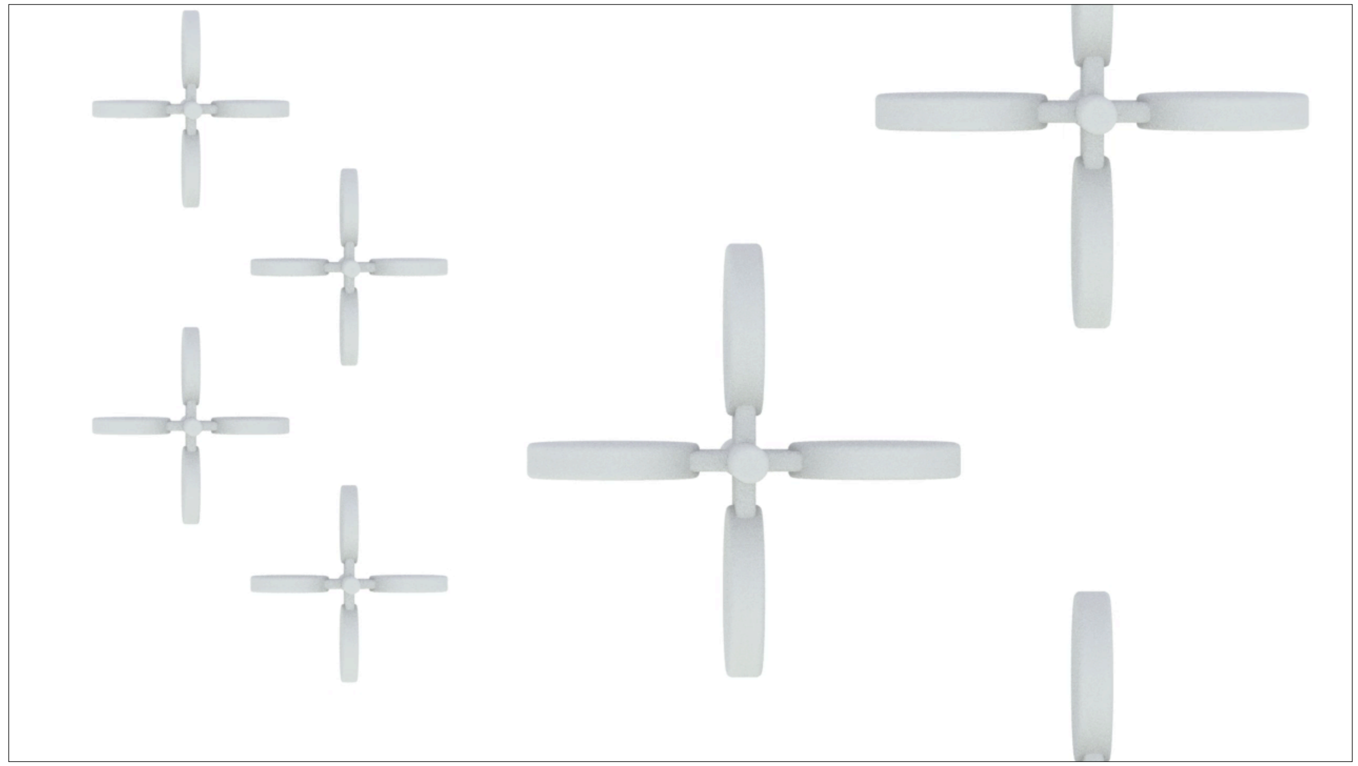
If we enable interaction between the boundary mesh and the particles, we can simply implement two-way rigid body coupling.



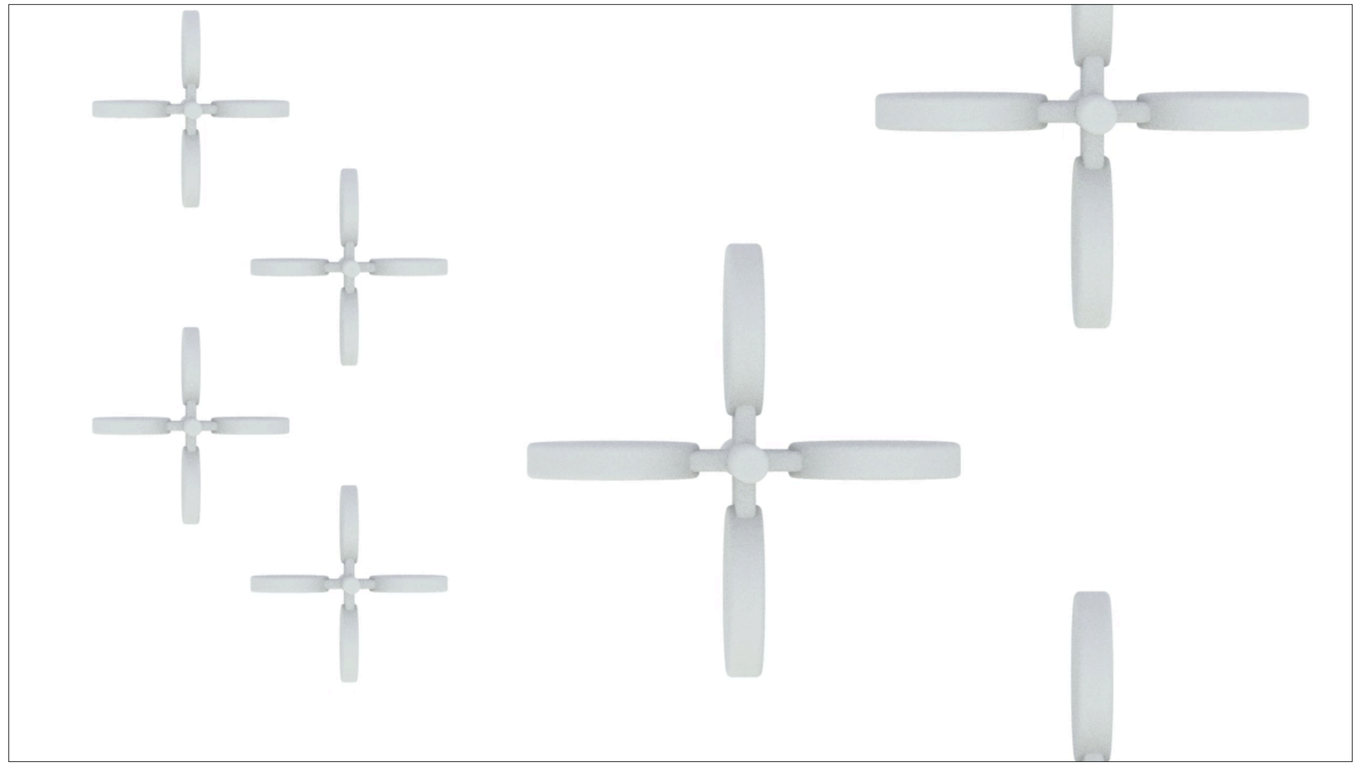
Here’s one coupling exmaple.



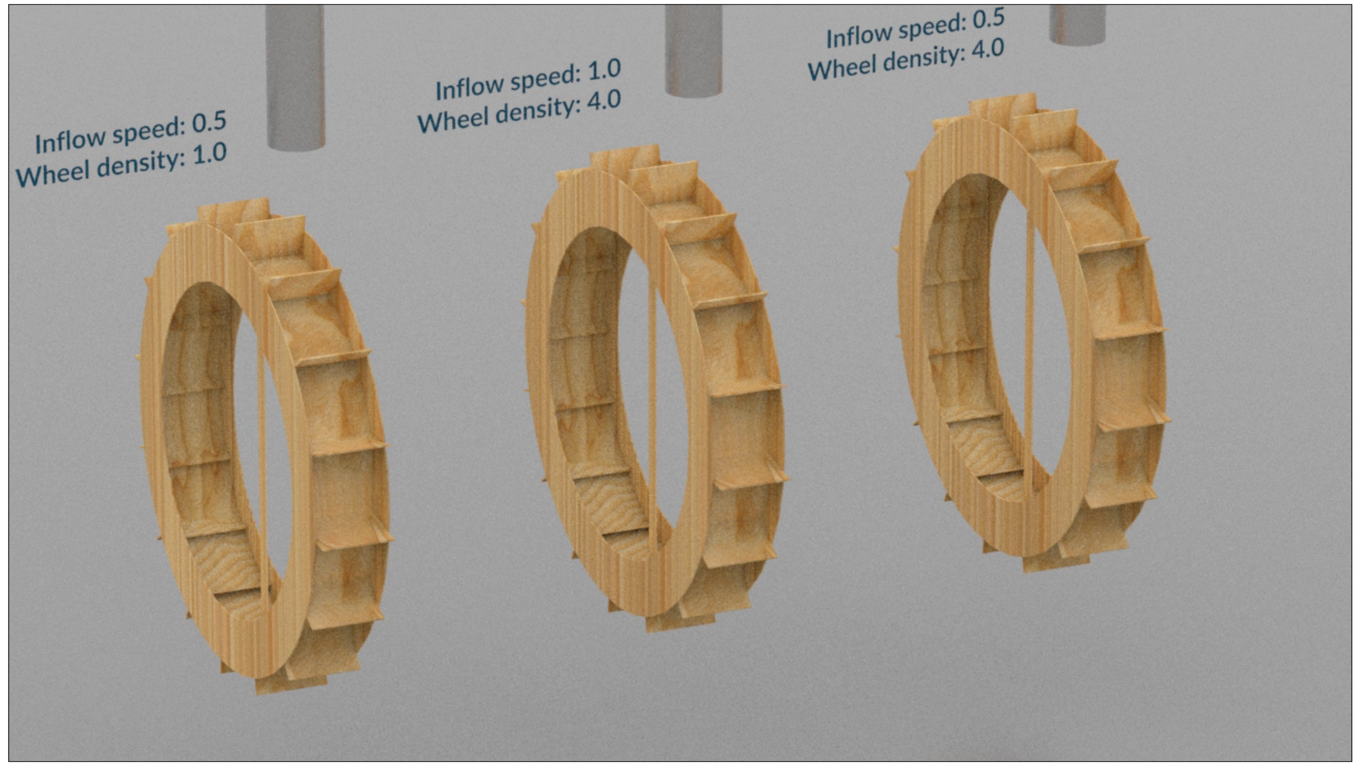
Here’s one coupling exmaple.



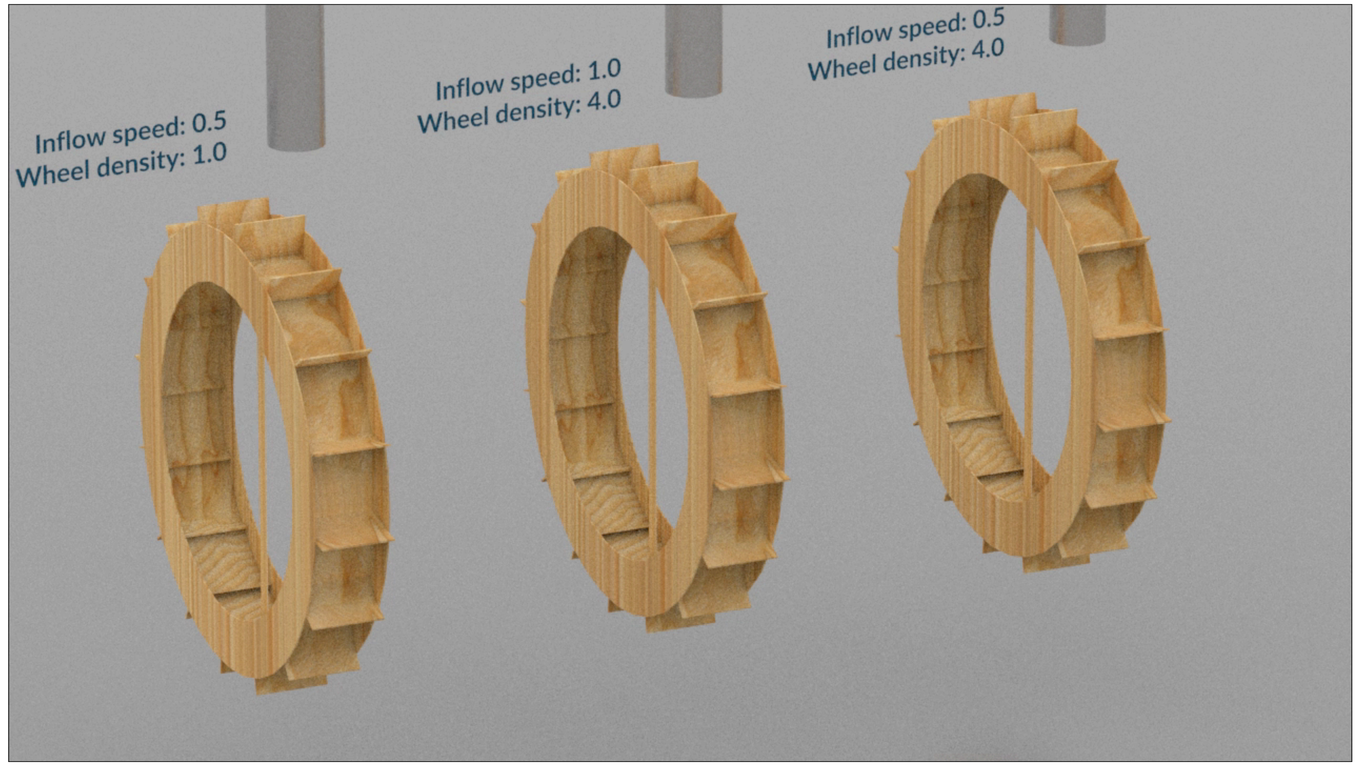
A 3D version.



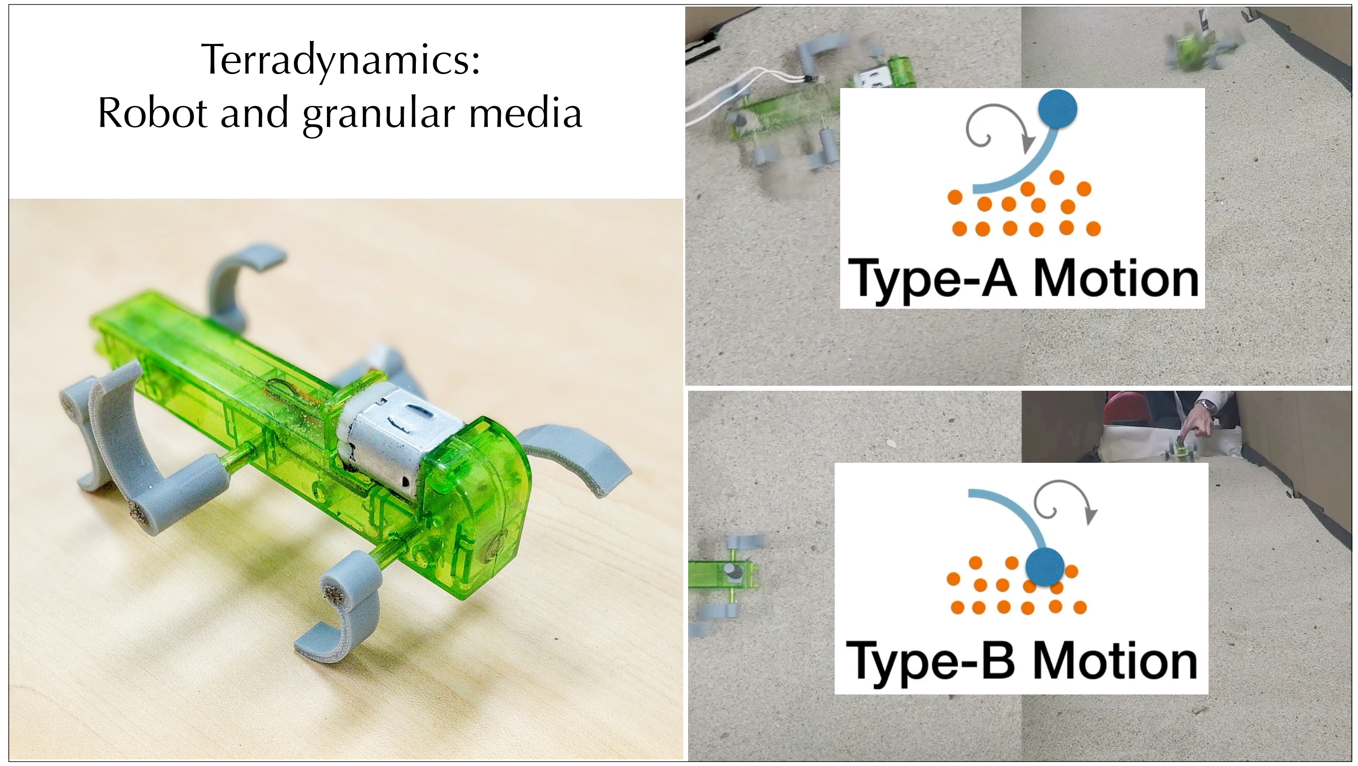
A 3D version.



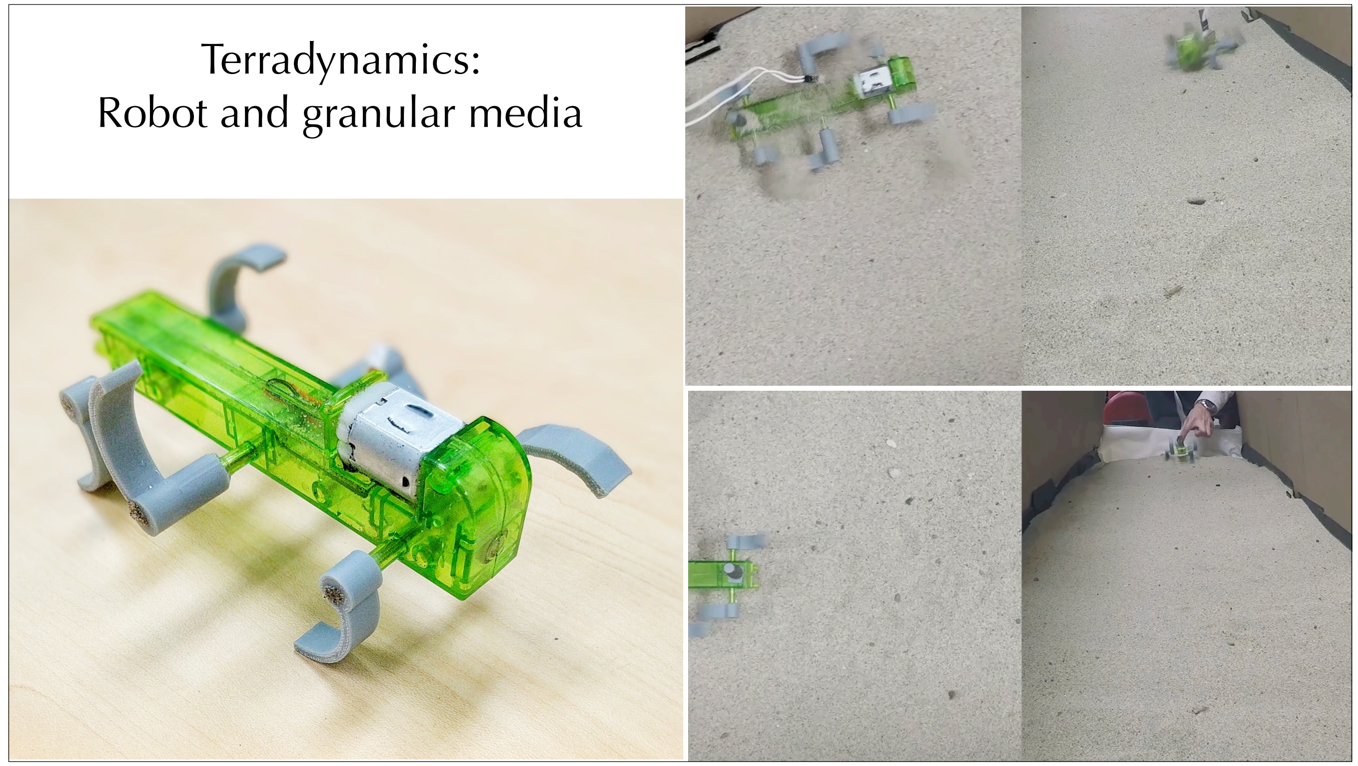
Inflow speed: 0.5
Wheel density: 1.0
Inflow speed: 1.0
Wheel density: 4.0
Inflow speed: 0.5
Wheel density: 4.0



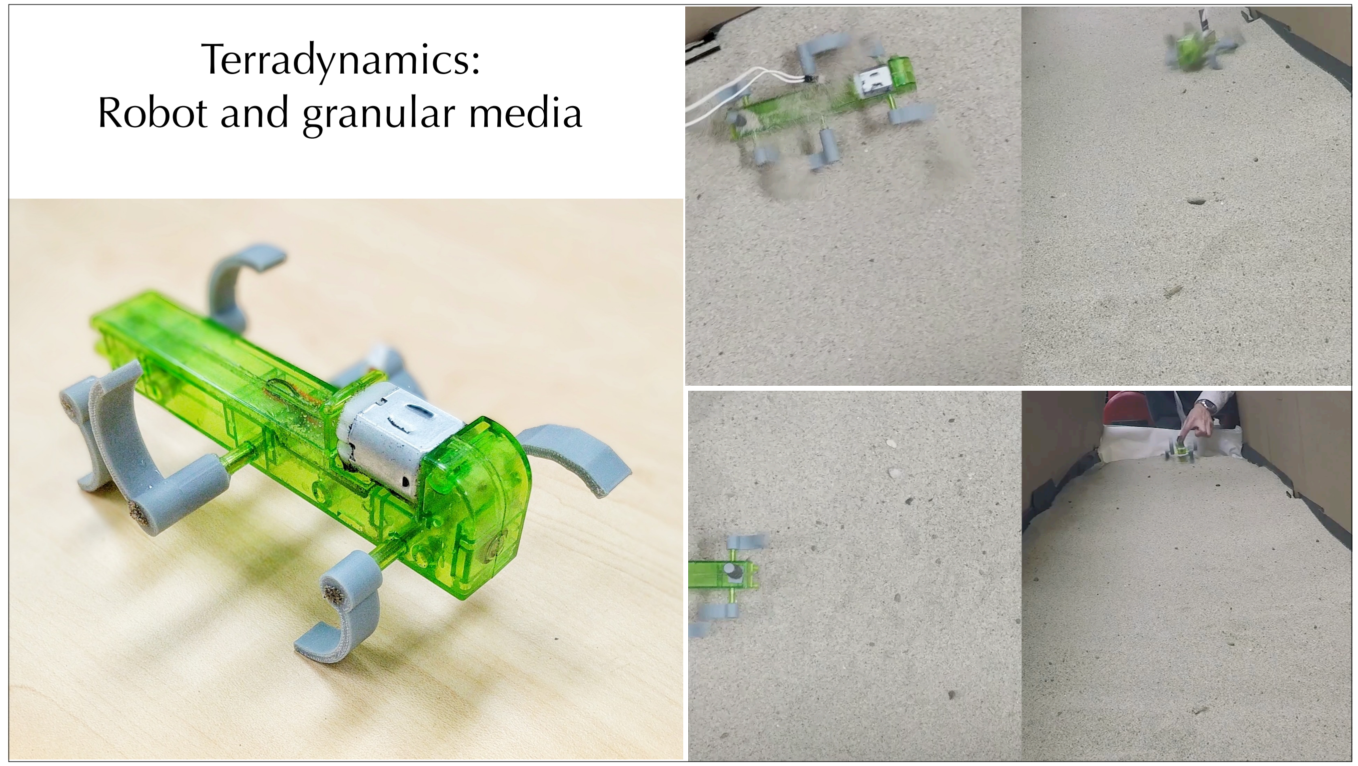
Inflow speed: 0.5
Wheel density: 1.0
Inflow speed: 1.0
Wheel density: 4.0
Inflow speed: 0.5
Wheel density: 4.0



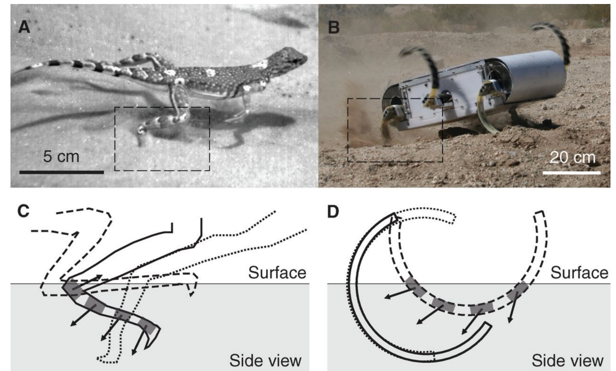
We can even simulate crawling robots entirely in MPM now, and study the motion of the robot.



We can even simulate crawling robots entirely in MPM now, and study the motion of the robot.

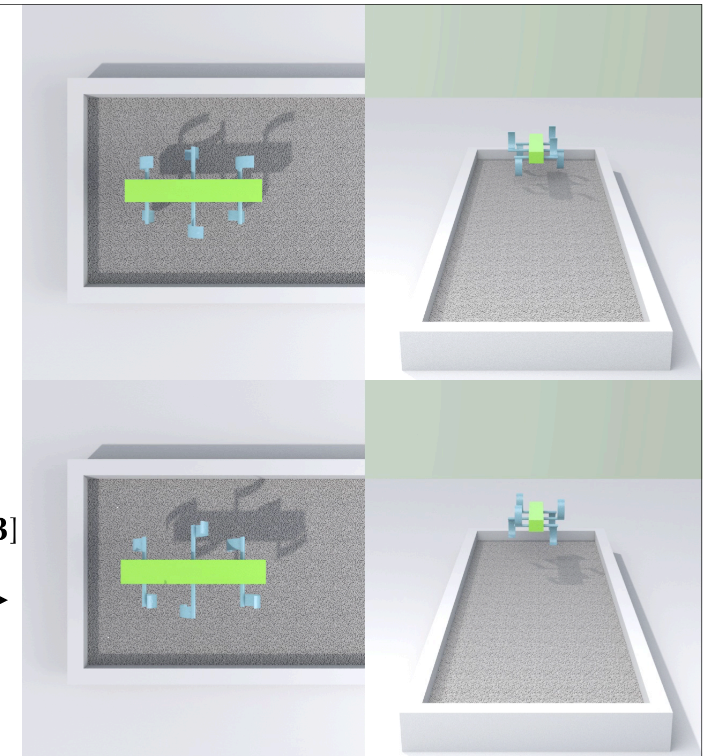


We can even simulate crawling robots entirely in MPM now, and study the motion of the robot.

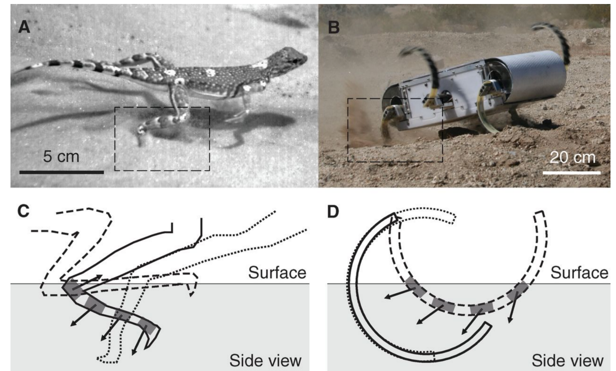


[Li et al., A terradynamics of legged locomotion on granular media. **Science 2013**]

This direction should be faster →

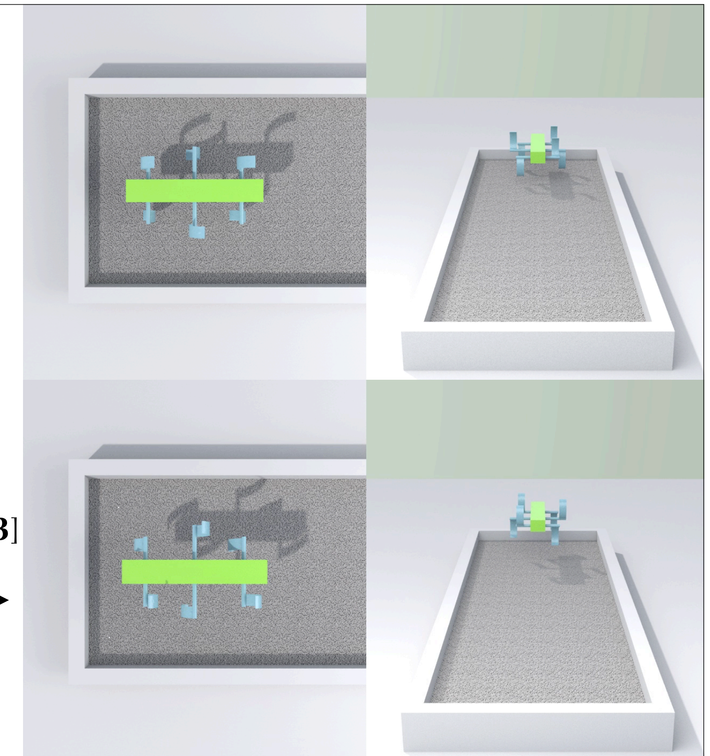


There is actually a science paper claiming that the bottom motion will be faster.



[Li et al., A terradynamics of legged locomotion on granular media. **Science 2013**]

This direction should be faster



There is actually a science paper claiming that the bottom motion will be faster.

Contributions

- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and Easier
- **Part II: Compatible Particle-in-Cell**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

So that’s it. CPIC allows velocity field discontinuity to exist in the reconstructed velocity field, and easily enables cutting and rigid body coupling.

Contributions

- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and Easier
- **Part II: Compatible Particle-in-Cell**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

So we have introduced MLS-MPM and CPIC. Both of them are simple ideas and are easy to implement.

Reproducible every demo with a python script:

git clone https://github.com/yuanming-hu/taichi_mpm

Apart from the 88-line version of MLS-MPM in 2D, we have also released code and data for the fully functioning high-performance 3D version based on taichi and you are welcome to try it.

[click] If you already have taichi installed on Linux, you can just do ‘ti install mpm’, and it will automatically deploy our MPM solver. We hope that this can low the barrier for everybody to start playing MPM.

[click] I would like to thank everyone who made this project possible, and thank you all for listening! That concludes my talk and I’m happy to take some questions.

Reproducible every demo with a python script:

git clone https://github.com/yuanming-hu/taichi_mpm

or use the **taichi project manager**: ***ti install mpm***

Apart from the 88-line version of MLS-MPM in 2D, we have also released code and data for the fully functioning high-performance 3D version based on taichi and you are welcome to try it.

[click] If you already have taichi installed on Linux, you can just do ‘ti install mpm’, and it will automatically deploy our MPM solver. We hope that this can low the barrier for everybody to start playing MPM.

[click] I would like to thank everyone who made this project possible, and thank you all for listening! That concludes my talk and I’m happy to take some questions.

Reproducible every demo with a python script:

git clone https://github.com/yuanming-hu/taichi_mpm

or use the **taichi project manager**: ***ti install mpm***

Thank you!

Questions are welcome!

Apart from the 88-line version of MLS-MPM in 2D, we have also released code and data for the fully functioning high-performance 3D version based on taichi and you are welcome to try it.

[click] If you already have taichi installed on Linux, you can just do ‘ti install mpm’, and it will automatically deploy our MPM solver. We hope that this can low the barrier for everybody to start playing MPM.

[click] I would like to thank everyone who made this project possible, and thank you all for listening! That concludes my talk and I’m happy to take some questions.

From **MPM** to **MLS-MPM**

Shape/Test function	**B-spline**	**MLS Shape function weighted by B-spline**
Lumped mass matrix	$m_i^n = \sum_p m_p \omega_{ip}$	
APIC P2G Momentum Contribution	$m_p \mathbf{C}_p^n (\mathbf{x_i} - \mathbf{x_p}) \omega_{ip}$	
Stress Momentum Contribution	$\Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} \nabla \omega_{ip}$	$\frac{4}{\Delta x^2} \Delta t V_p^0 \frac{\partial \Psi}{\partial \mathbf{F}} (\mathbf{F}_p^n) \mathbf{F}_p^{nT} (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$
APIC G2P Affine Velocity Reconstruction	$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i v_i (\mathbf{x}_i - \mathbf{x}_p) \omega_{ip}$	
Velocity Gradient Evaluation	$\nabla \mathbf{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \mathbf{v}_p^{n+1} = \mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n+1}}$
Deformation Gradient Update	$\mathbf{F}_p^{n+1} = \left(\mathbf{F} + \Delta t \nabla \mathbf{v}_p^{n+1}\right) \mathbf{F}_p^n$	

Since the B-spline gradient is replaced by the simple MLS shape function gradient, MLS-MPM avoids the costly B-spline kernel gradients.

\mathbf{F}_p^{n+1}=\left(\mathbf{F} + \Delta t\nabla\mathbf{v}_p^{n+1} \right) \mathbf{F}_p^{n}