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

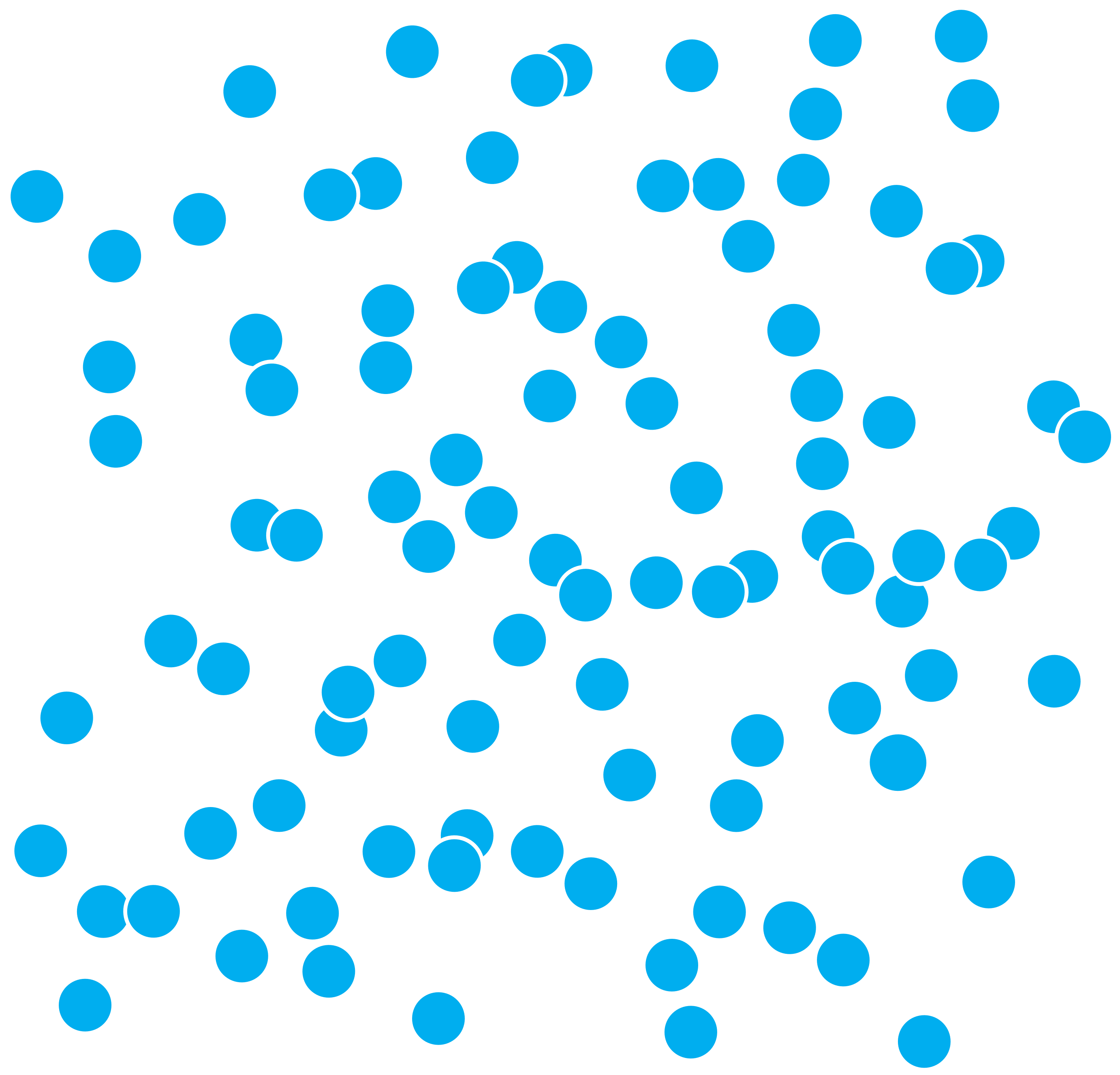
SIGGRAPH 2018

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

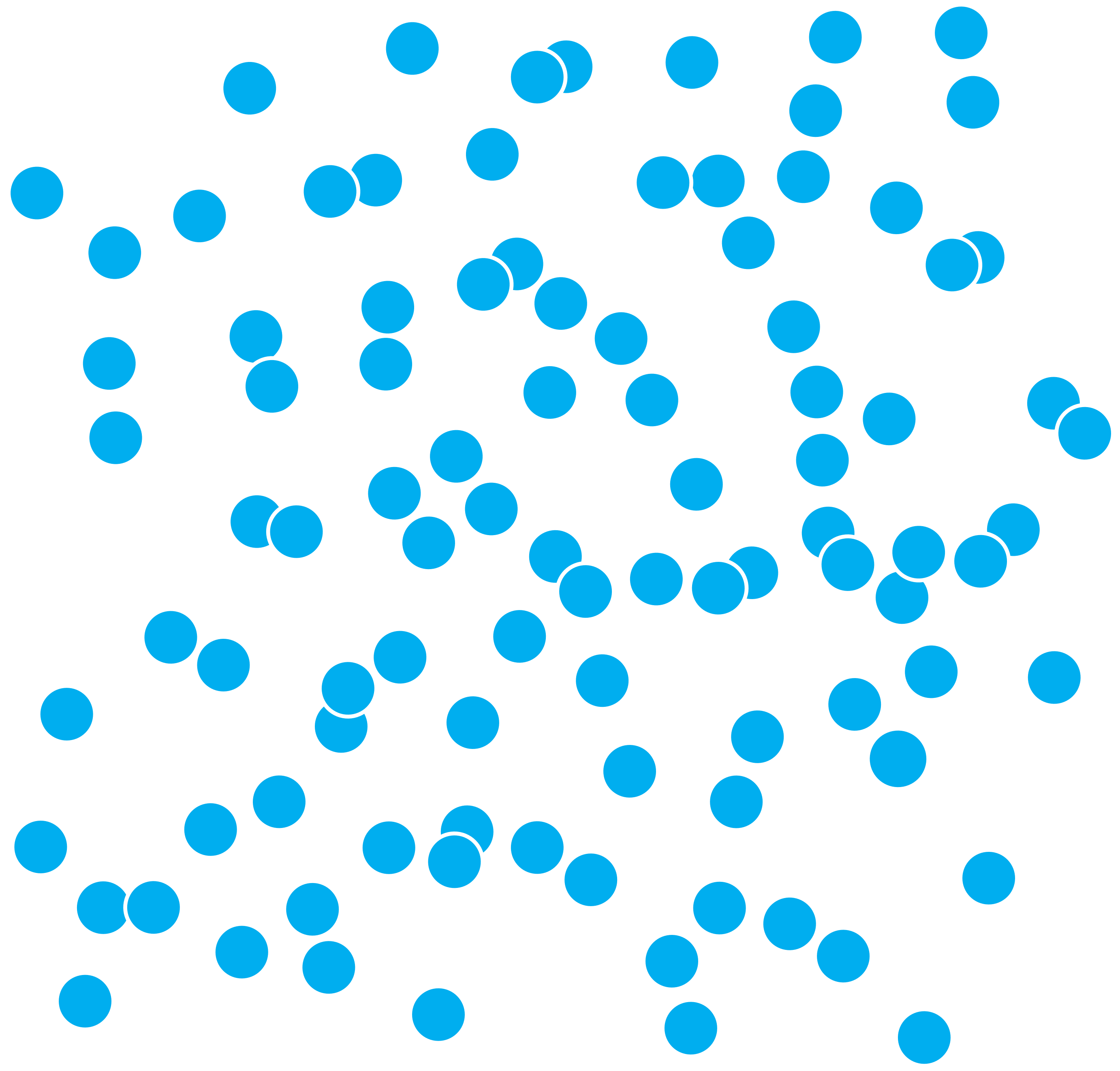
[1]MIT CSAIL [2]Tsinghua University [3]University of Science and Technology of China
[4]University of Pennsylvania [5]UCLA

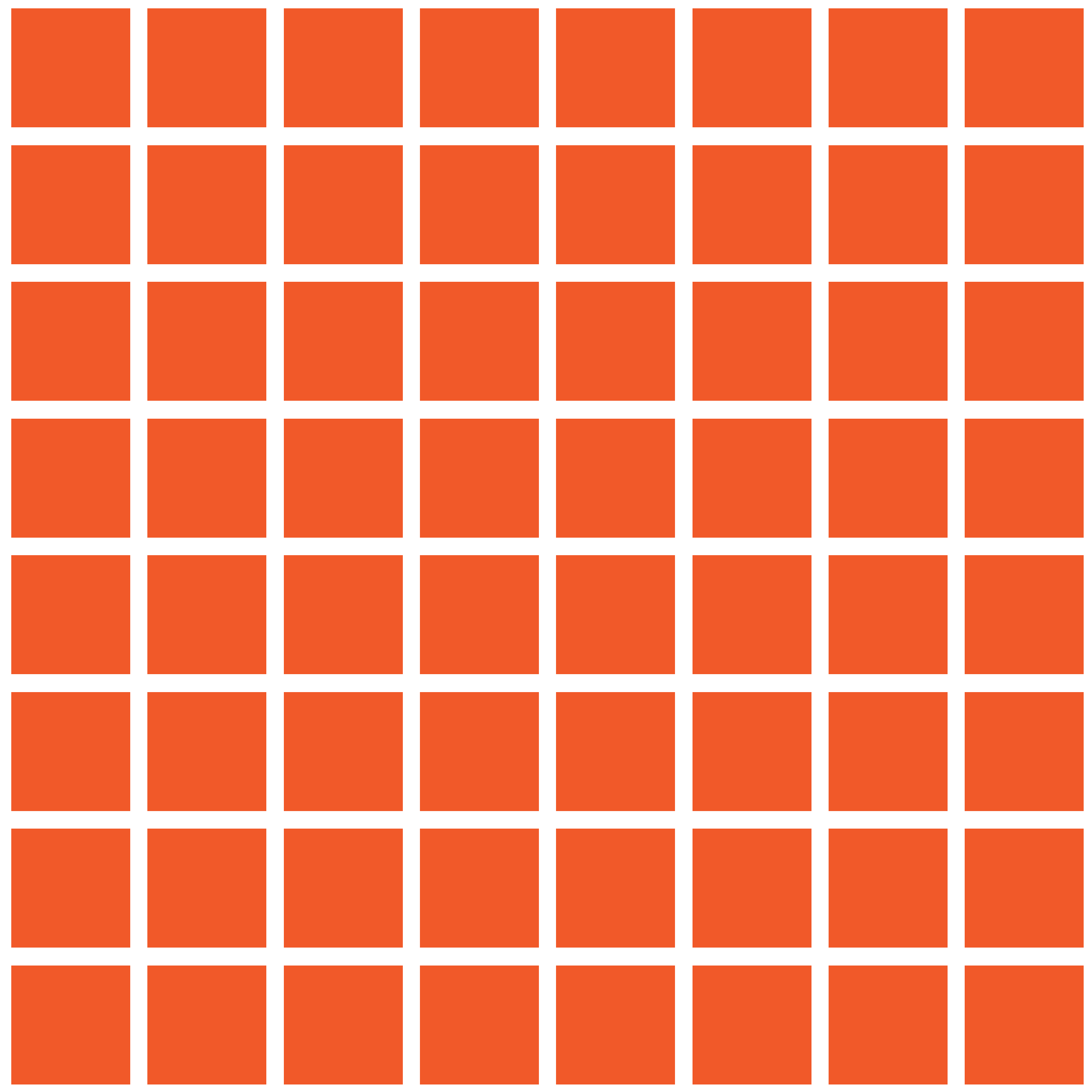
The Material Point Method (MPM)

The Material Point Method (MPM)

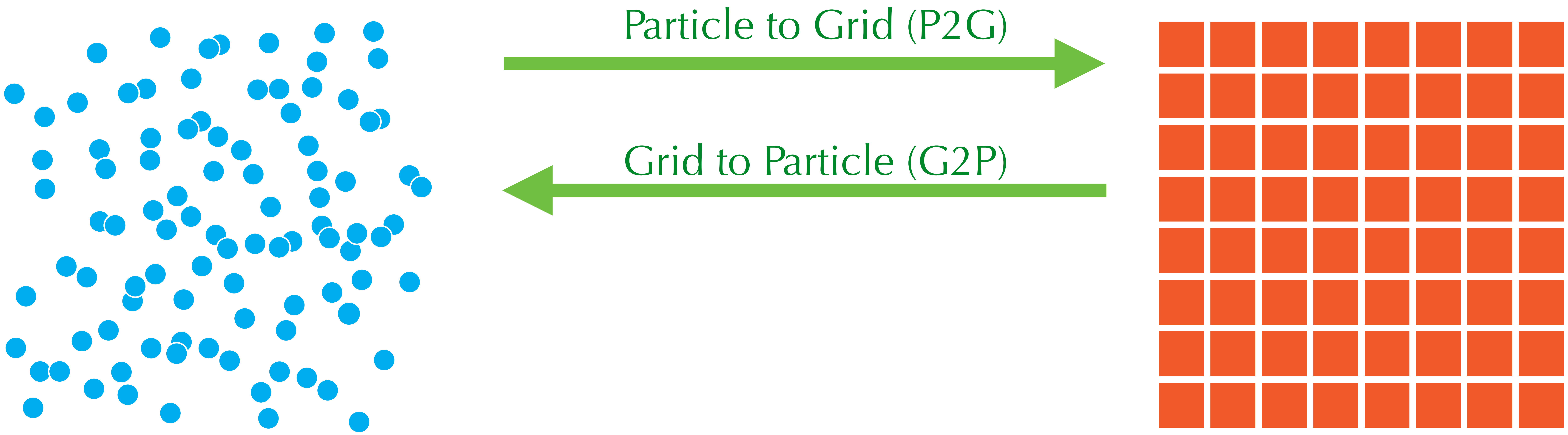


The Material Point Method (MPM)

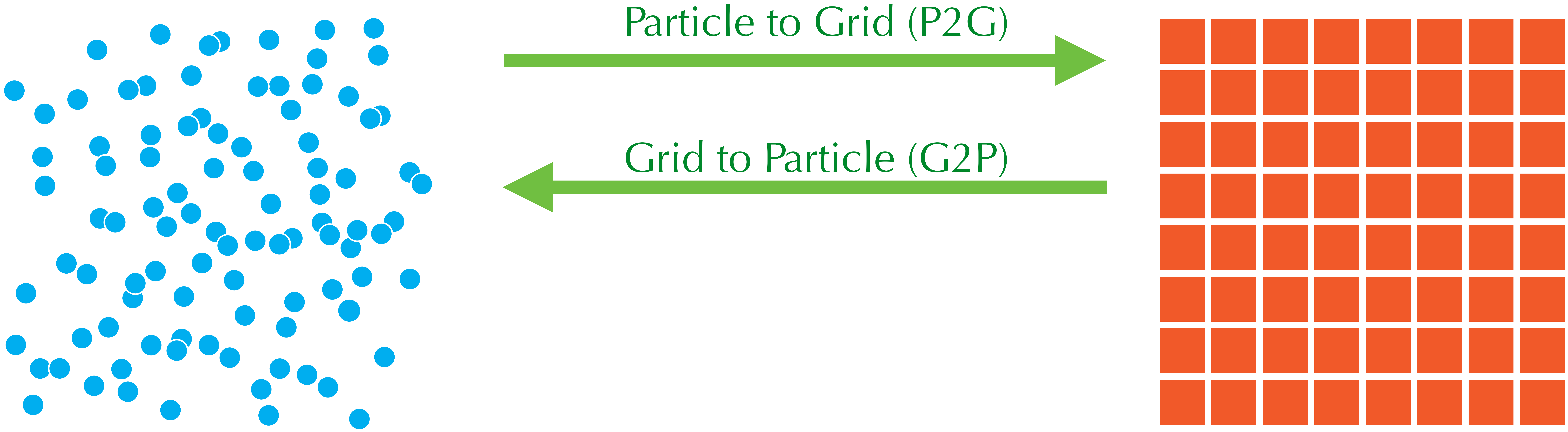




The Material Point Method (MPM)



The Material Point Method (MPM)



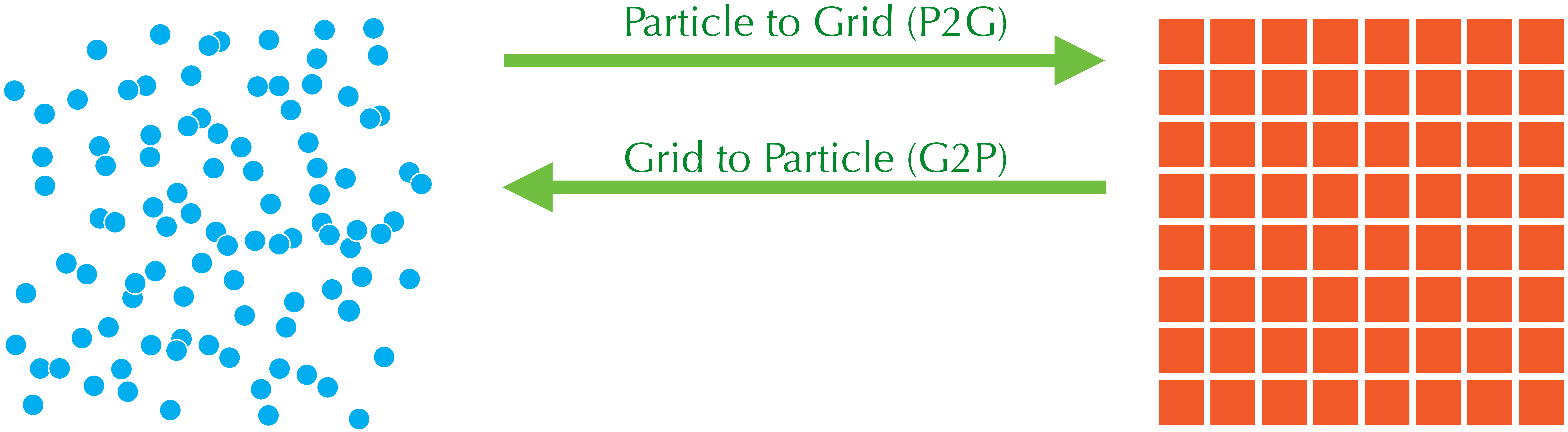
Particles (Constitutive models)

Snow [Stomakhin et al. 2013],

Foam [Ram et al. 2015, Yue et al. 2015]

Sand [Klar et al. 2015, Pradhana et al 2017]

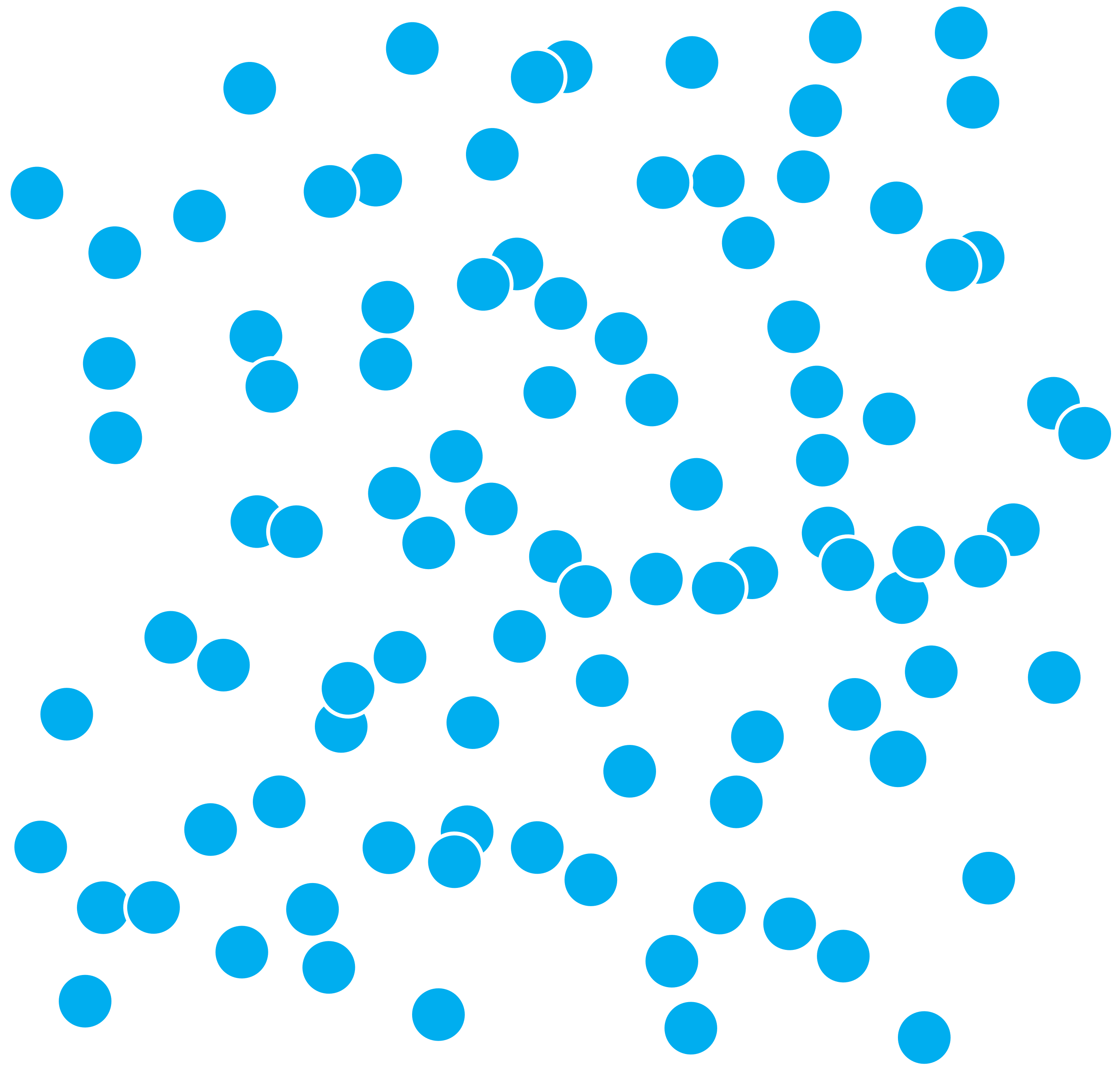
The Material Point Method (MPM)



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]

Grid
SPGrid [Setaluri et al. 2014],
OpenVDB [Museth 2013]
Multiple Grids [Pradhana et al. 2017]

The Material Point Method (MPM)



Particle to Grid (P2G)

Grid to Particle (G2P)

Transfer (Particle-in-Cell, PIC)

Affine PIC, APIC [Jiang et al. 2016]

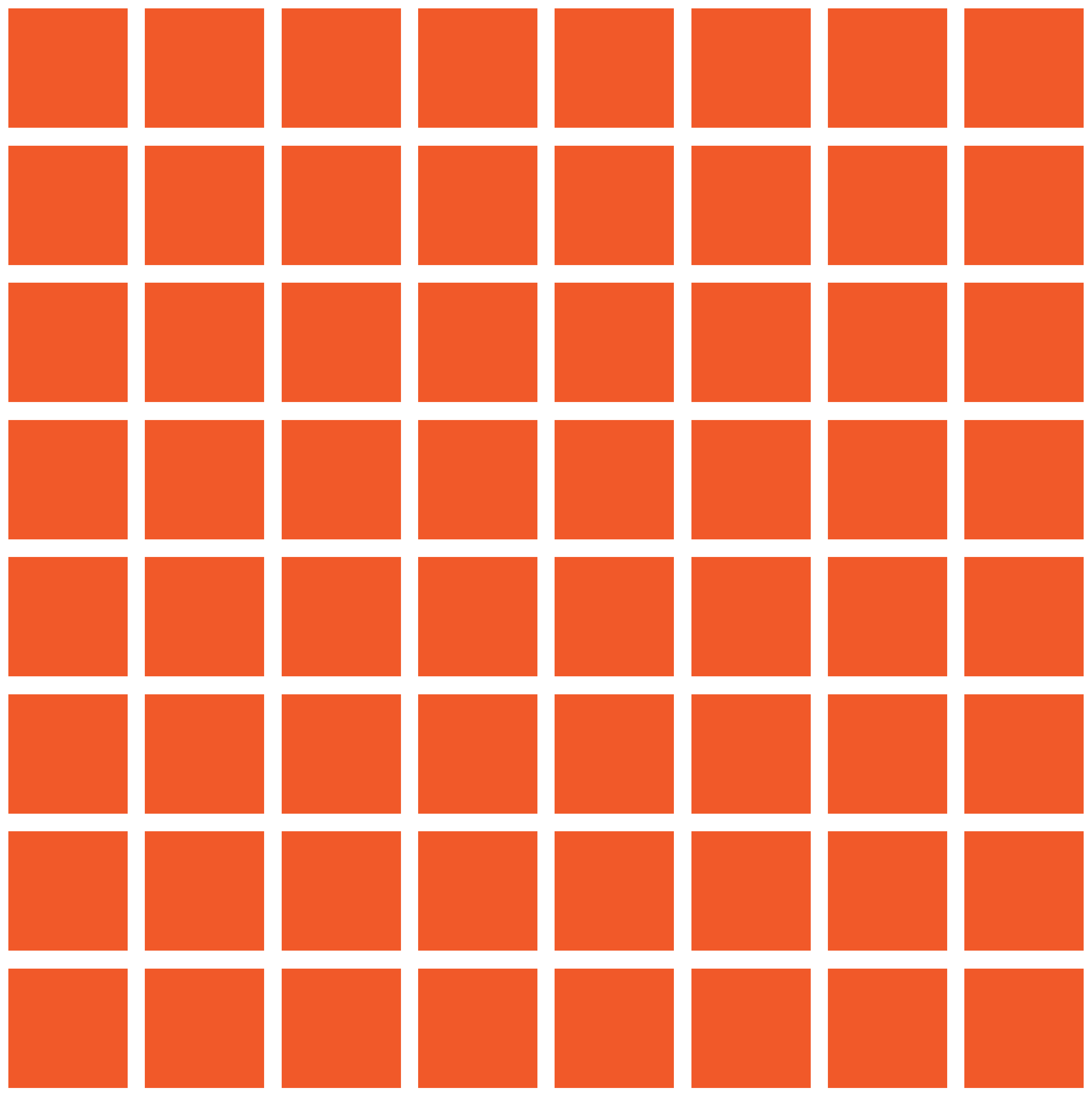
Polynomial PIC, PolyPIC [Fu et al. 2017]

High-performance GIMP [Gao et al. 2017]

Moving Least Squares [Hu et al. 2018]

Compatible PIC [Hu et al. 2018]

...



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]

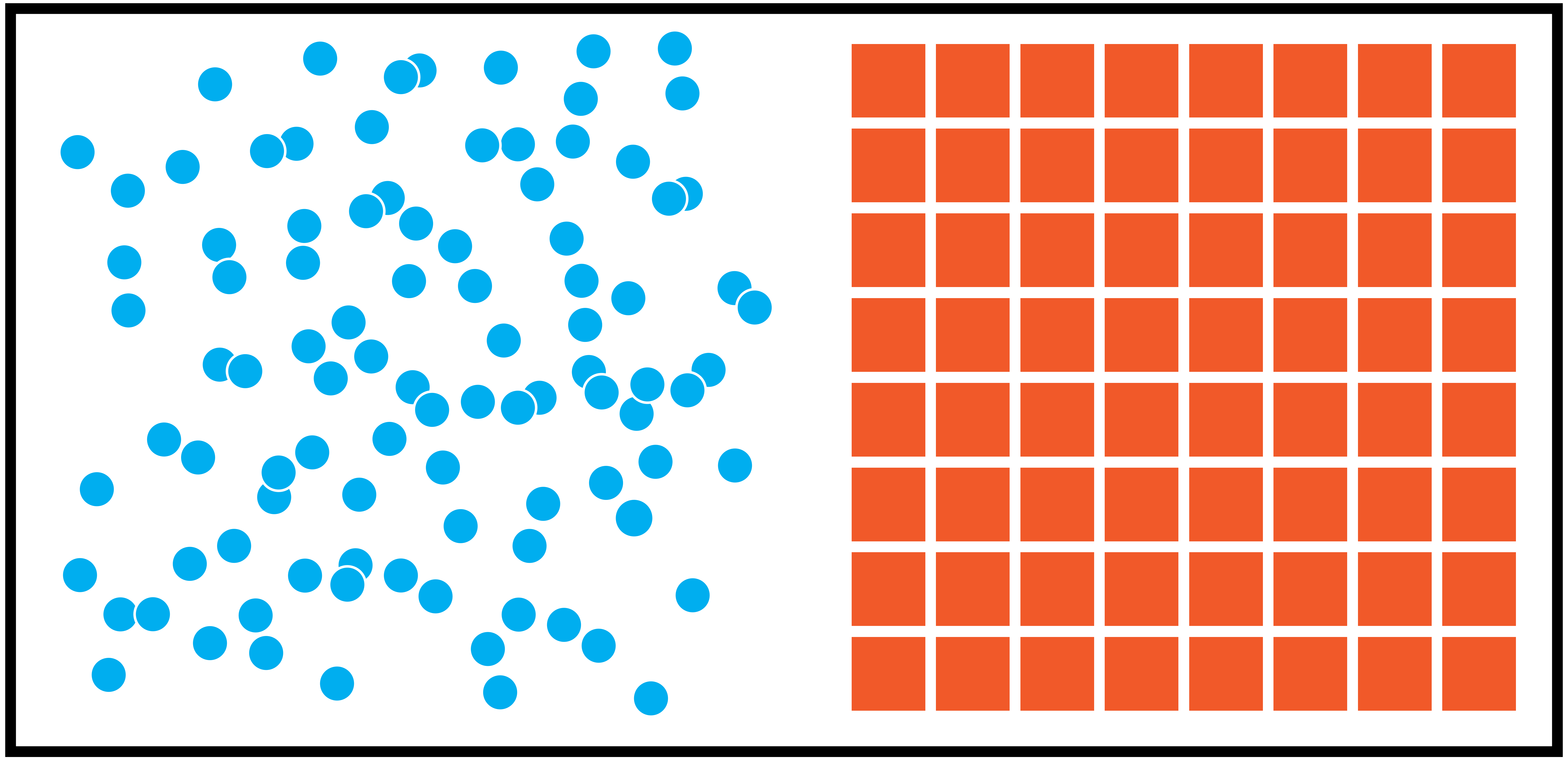
Grid

SPGrid [Setaluri et al. 2014],

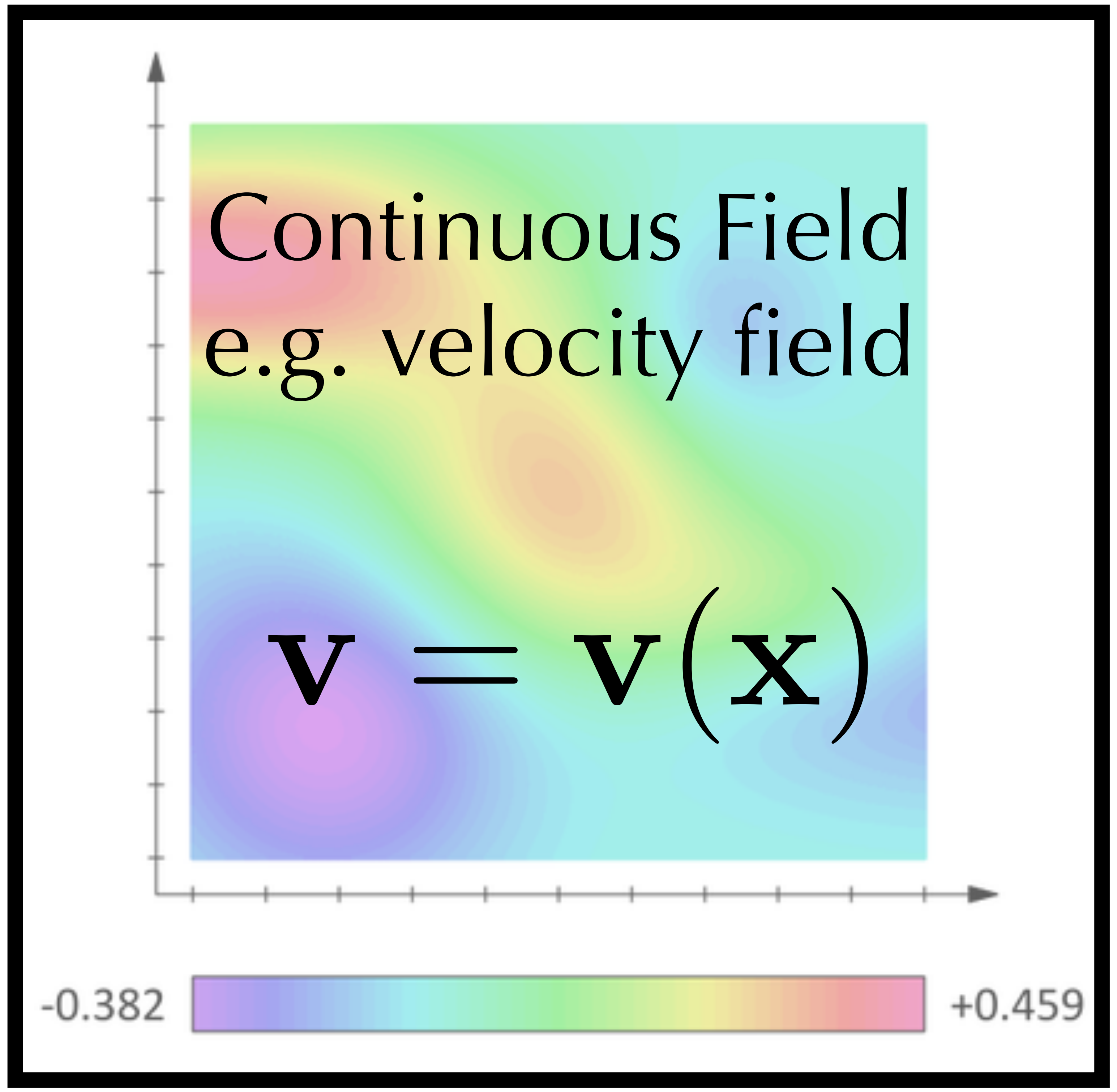
OpenVDB [Museth 2013]

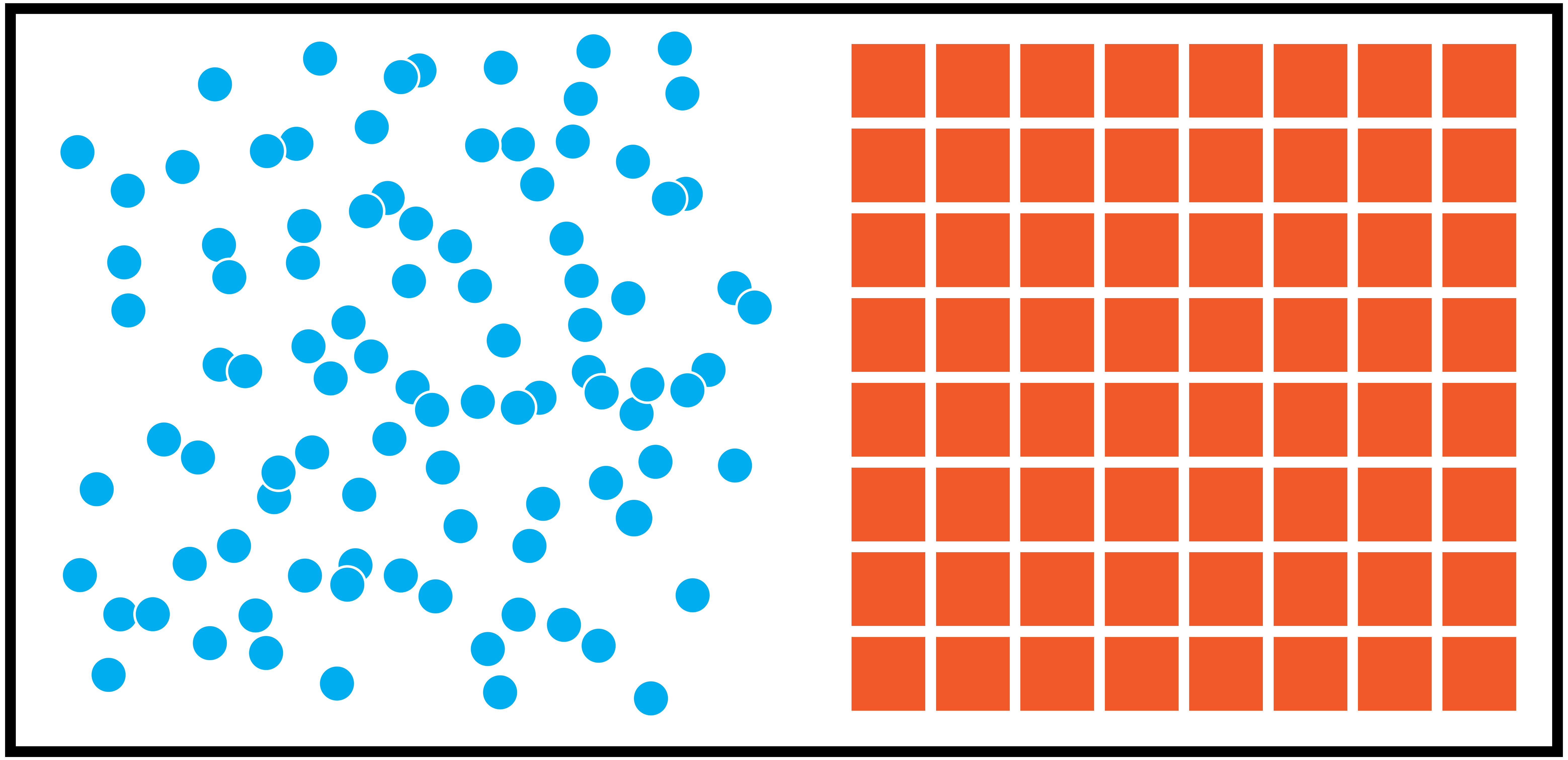
Multiple Grids [Pradhana et al. 2017]

The Material Point Method (MPM)



The Material Point Method (MPM)





Contributions

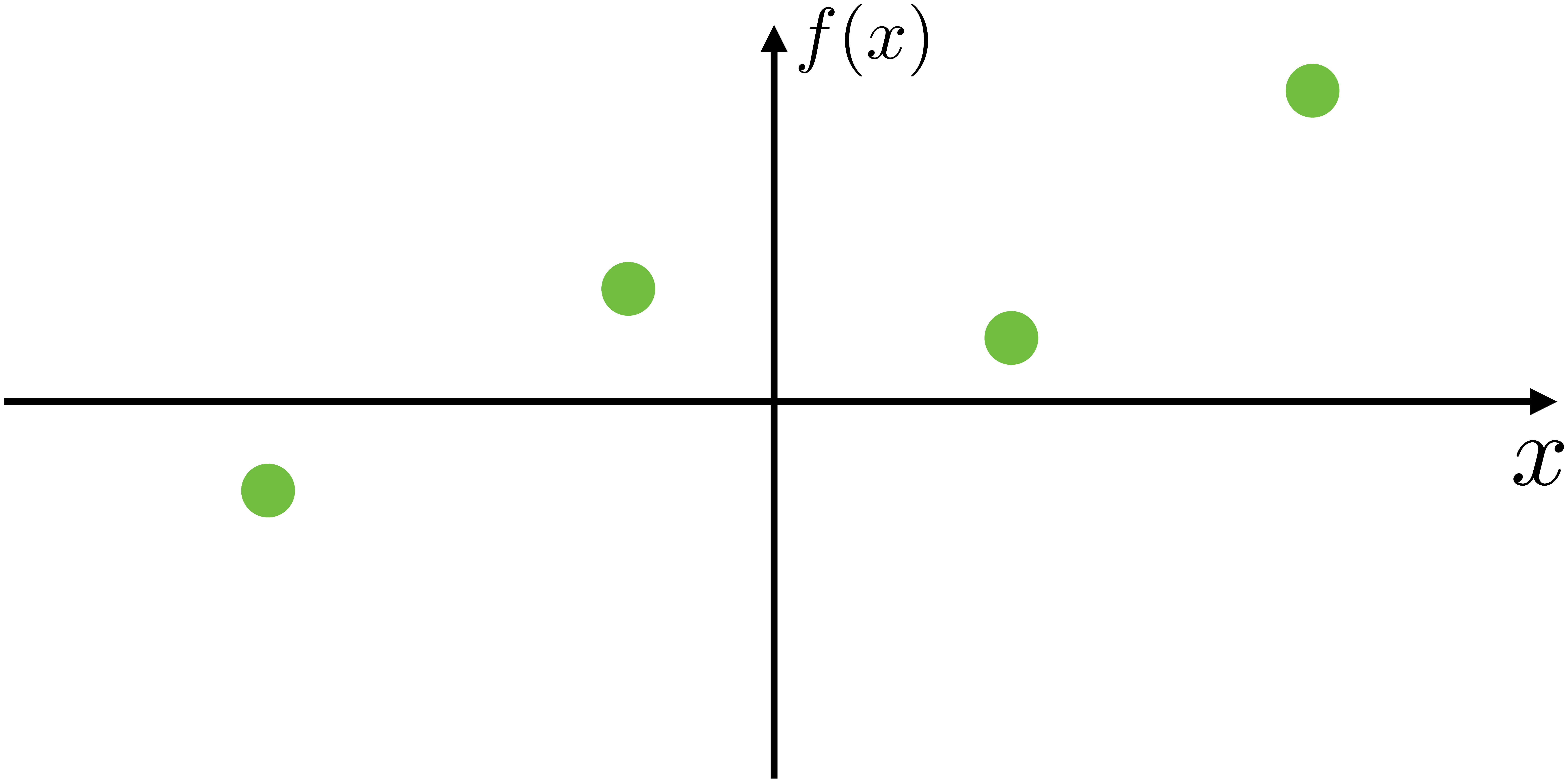
- **Part I: Moving Least Squares Discretization (MLS-MPM)**
  - Unifying Affine Particle-In-Cell and MPM force discretization
  - Weak-form consistent
  - Faster and Easier

- **Part II: Compatible Particle-in-Cell (CPIC)**
  - Velocity field discontinuity
  - Enables cutting and rigid body coupling

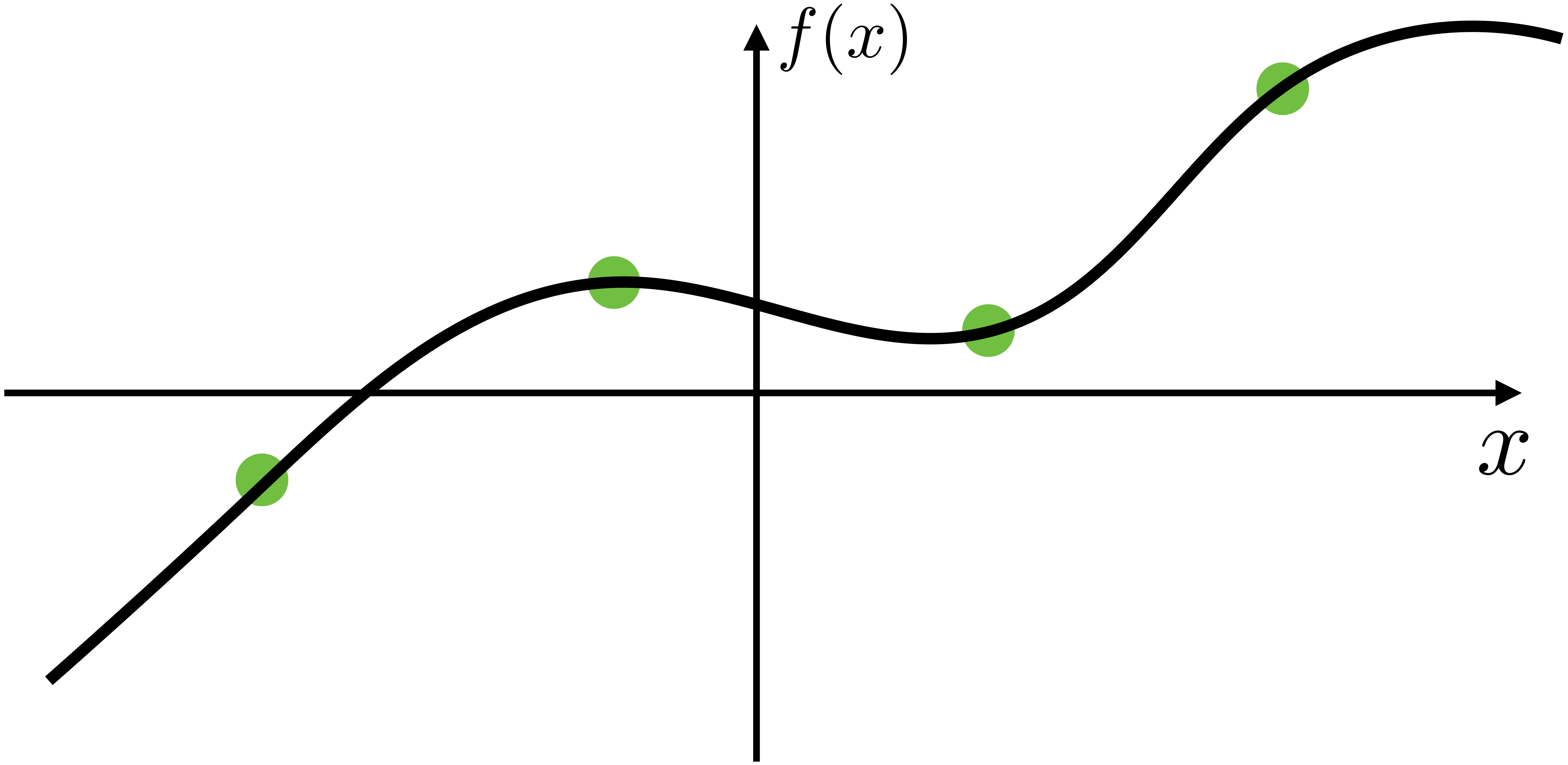
Contributions

- **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

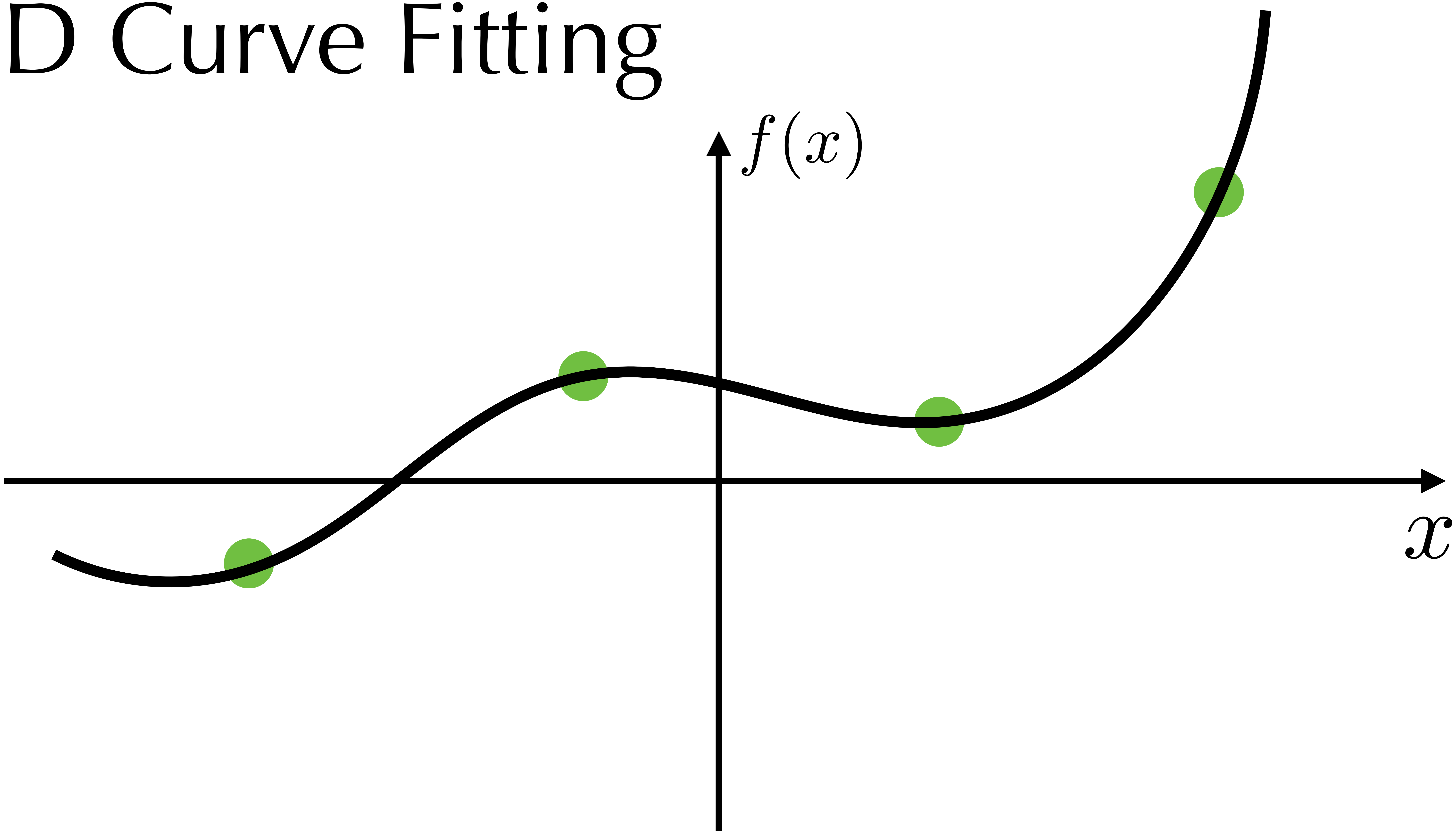
1D Curve Fitting



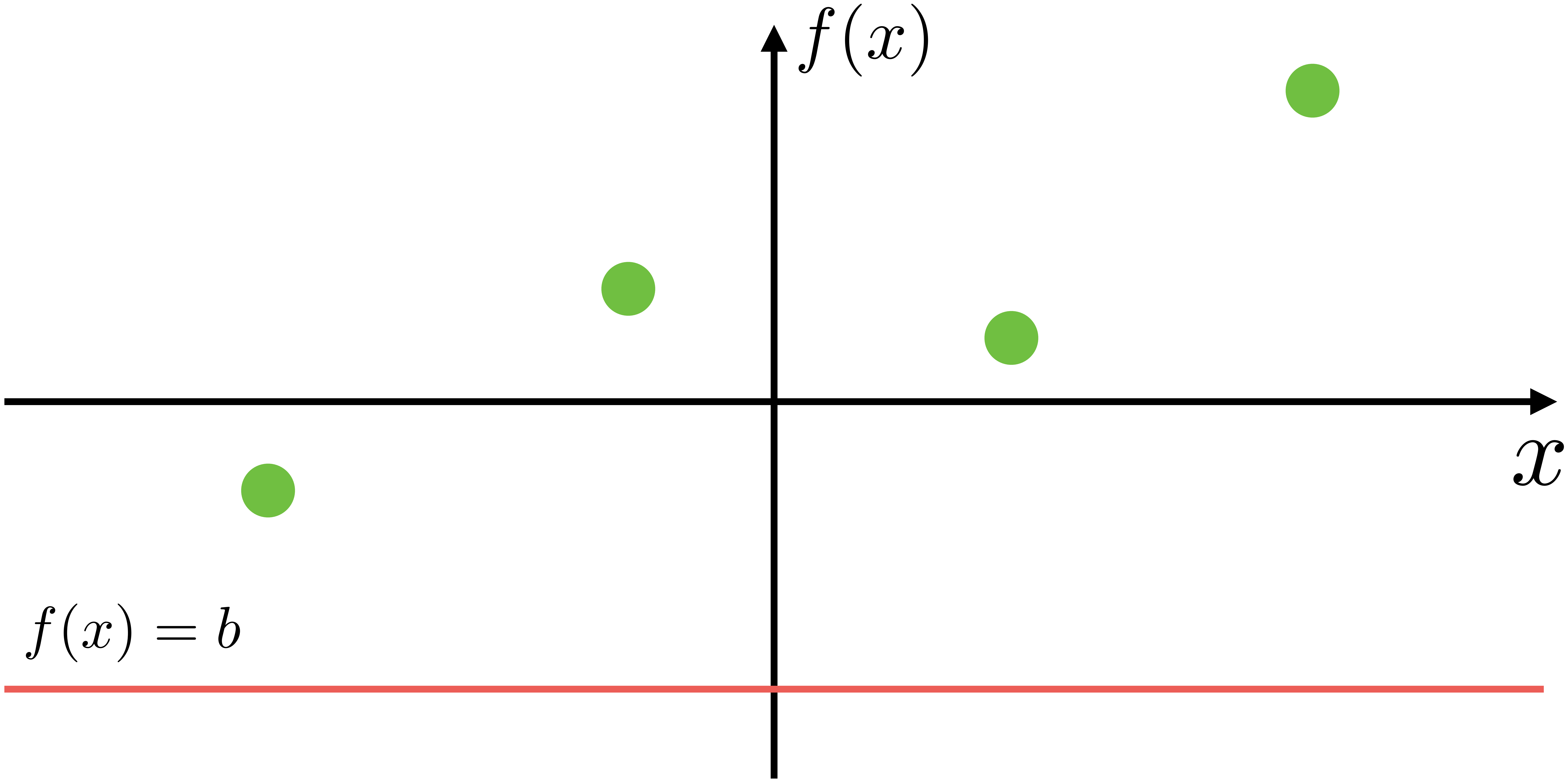
1D Curve Fitting



1D Curve Fitting

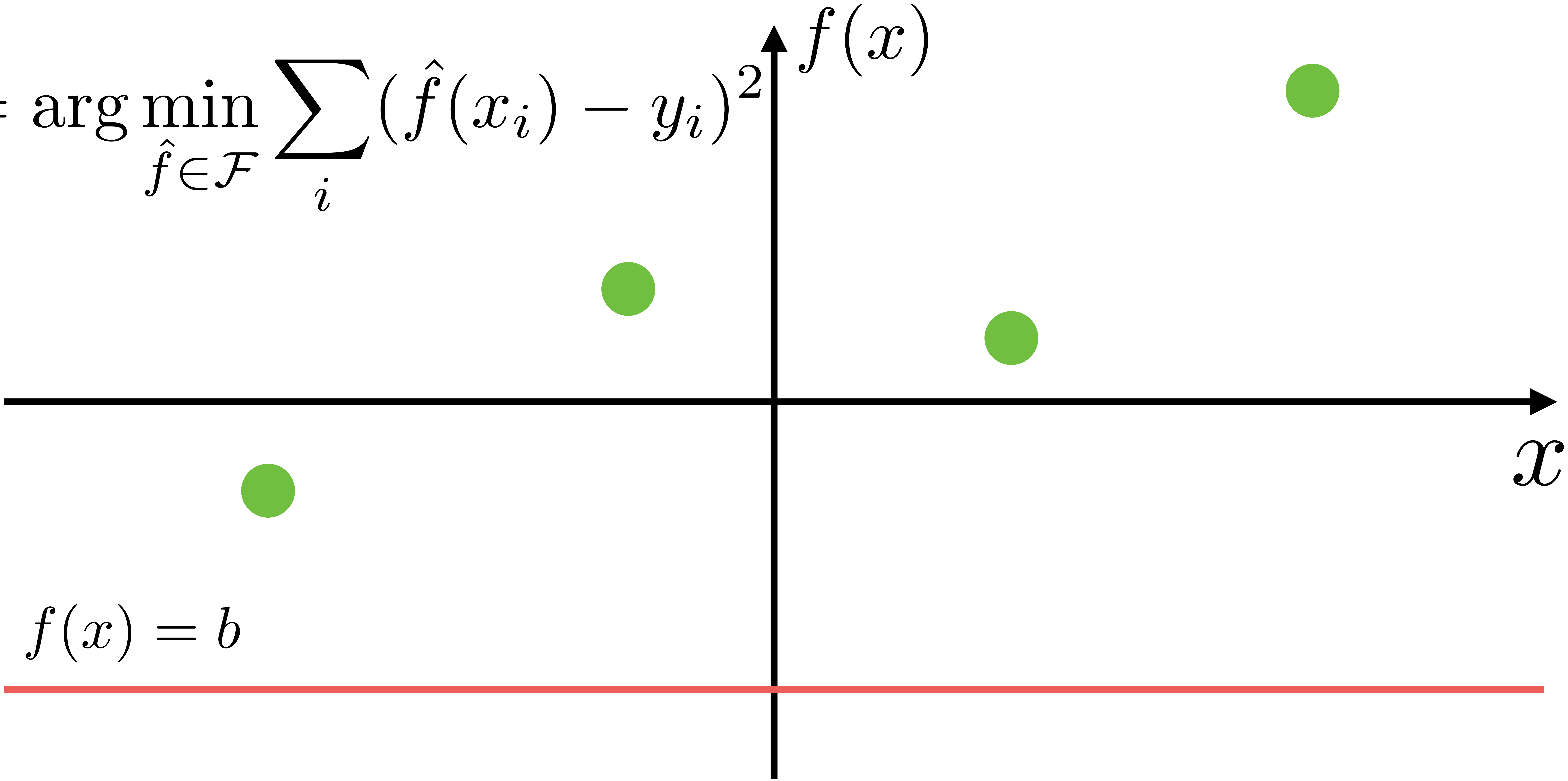


1D Curve Fitting



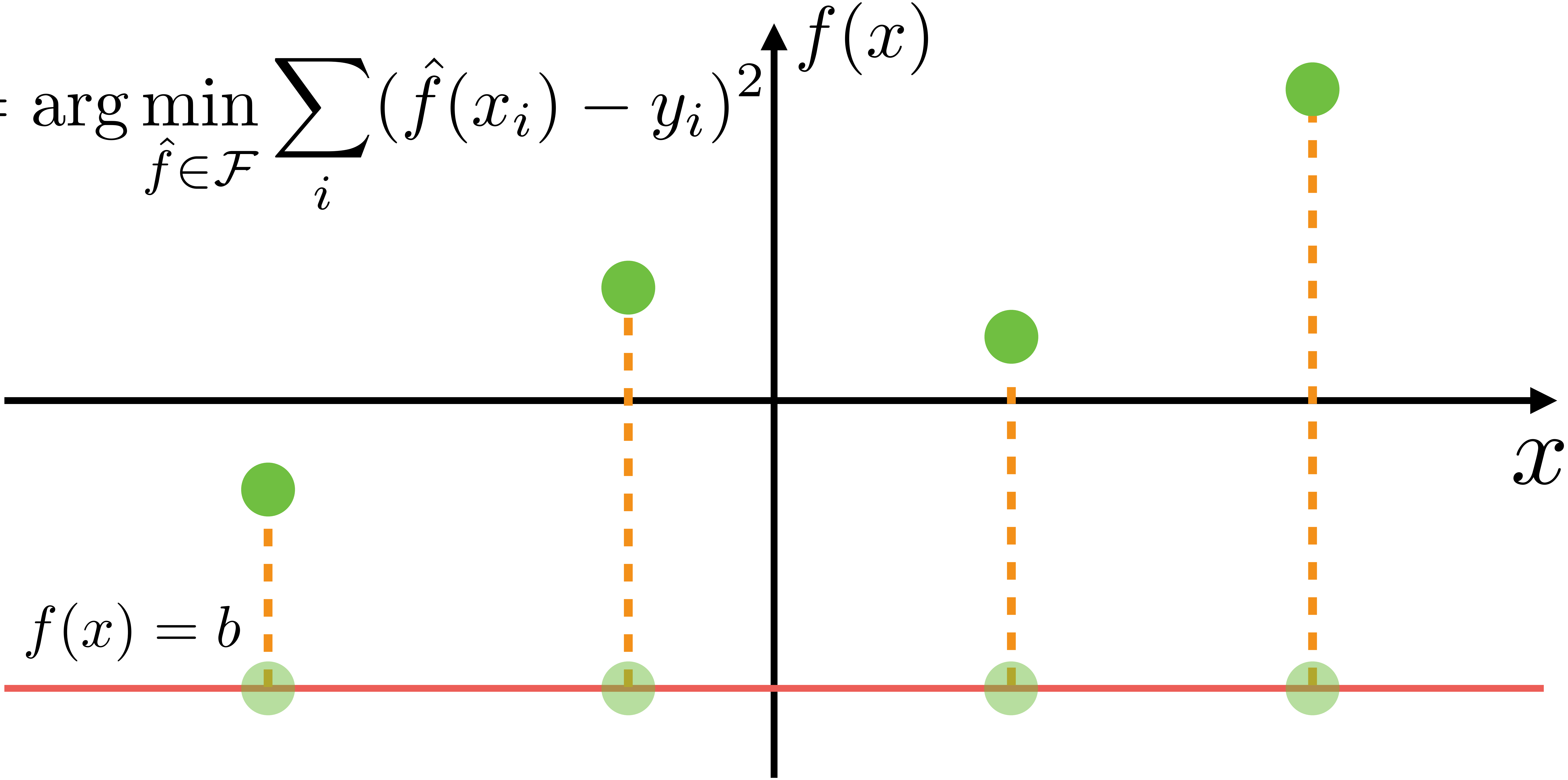
1D Curve Fitting

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

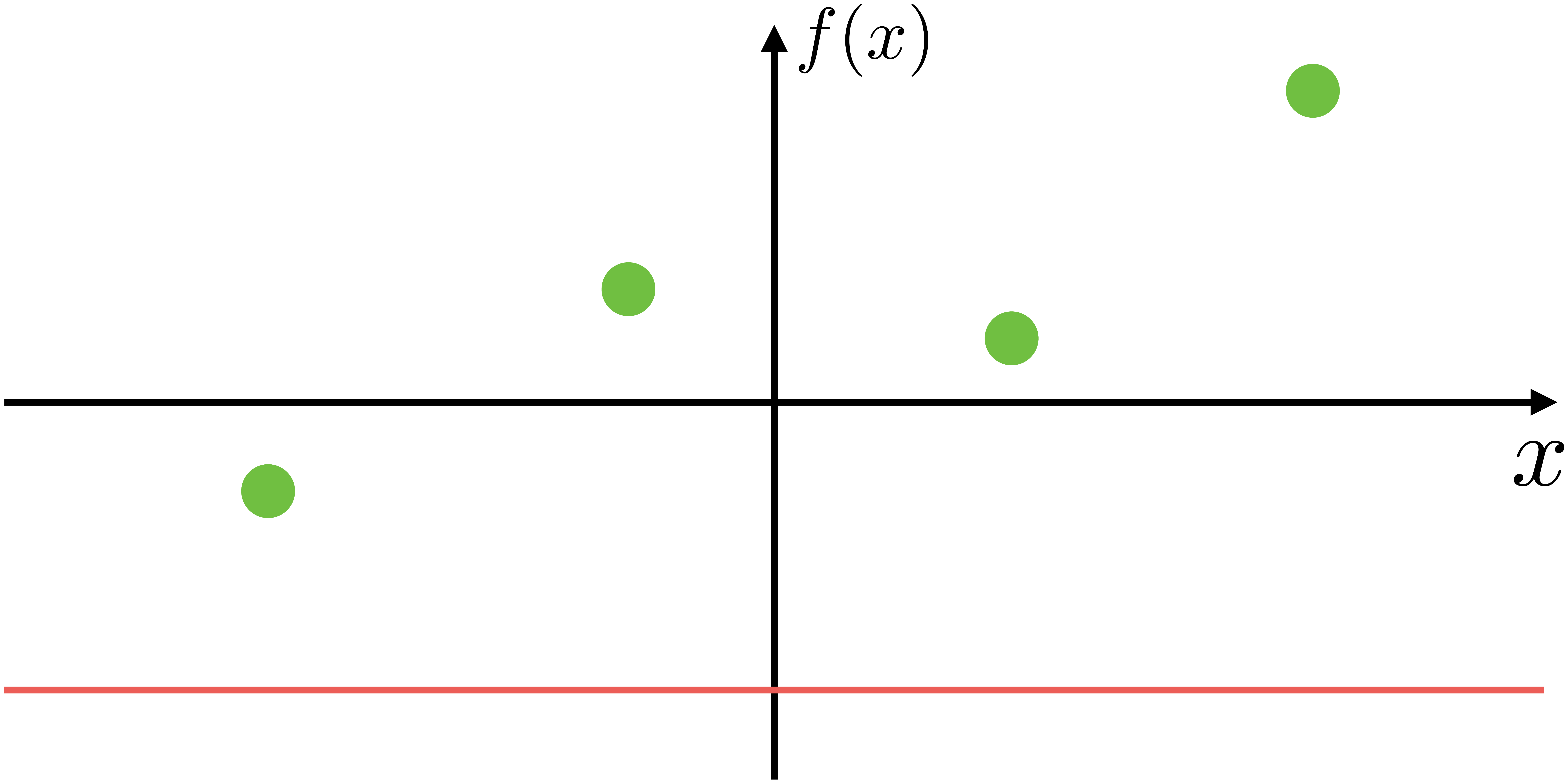


1D Curve Fitting

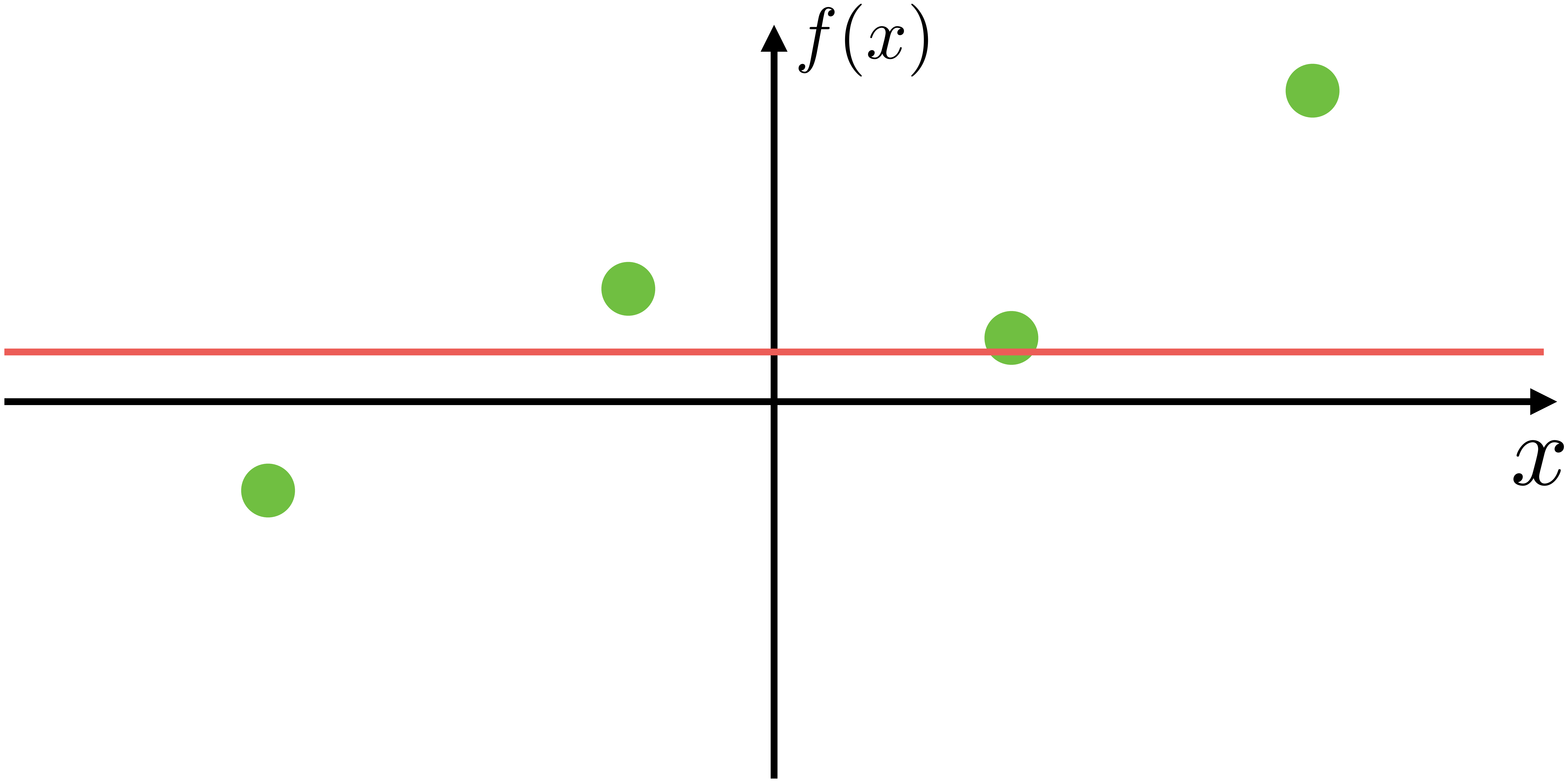
$$f = \arg\min_{\hat{f} \in \mathcal{F}} \sum_i (\hat{f}(x_i) - y_i)^2$$



1D Curve Fitting

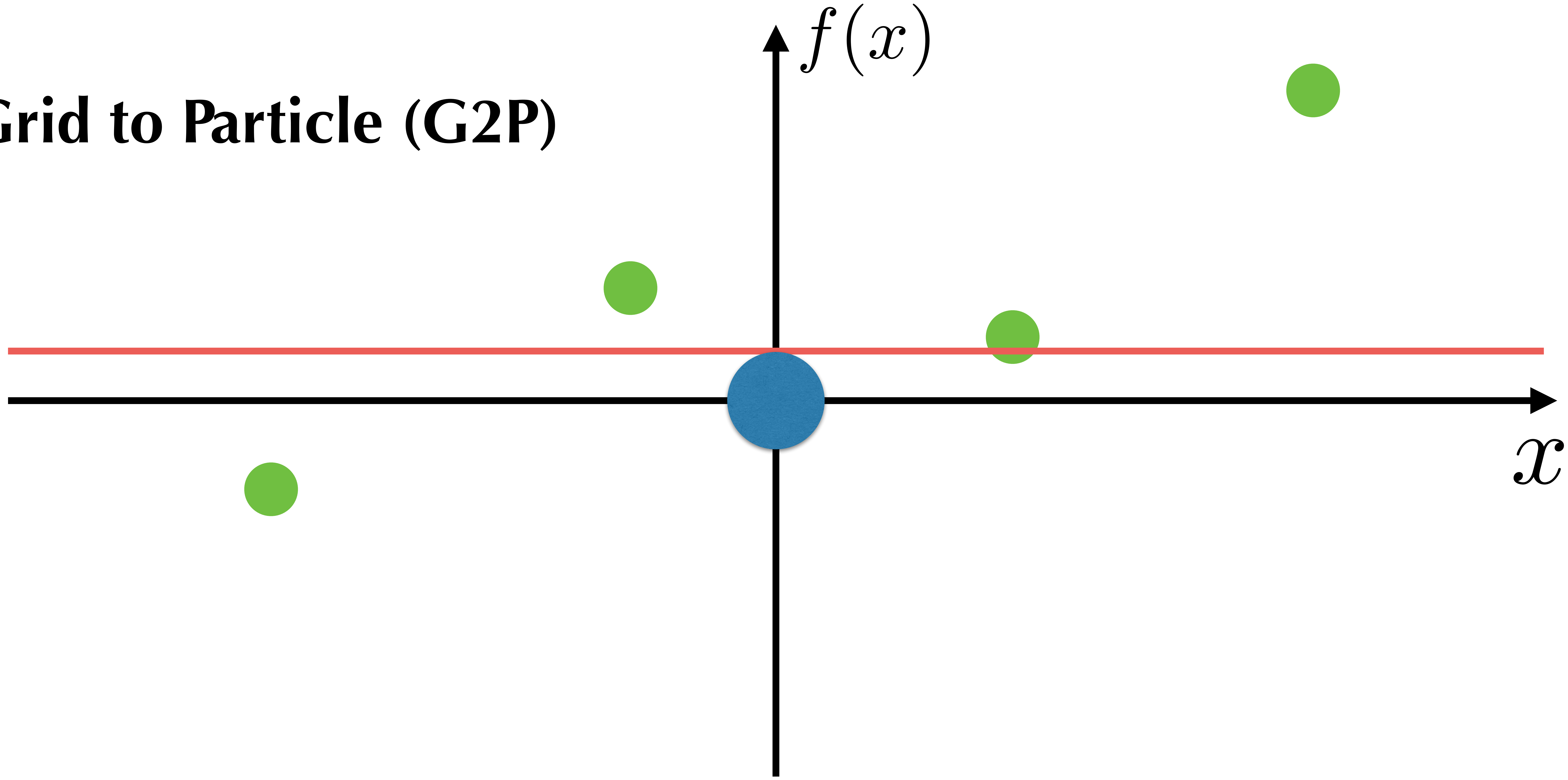


1D Curve Fitting



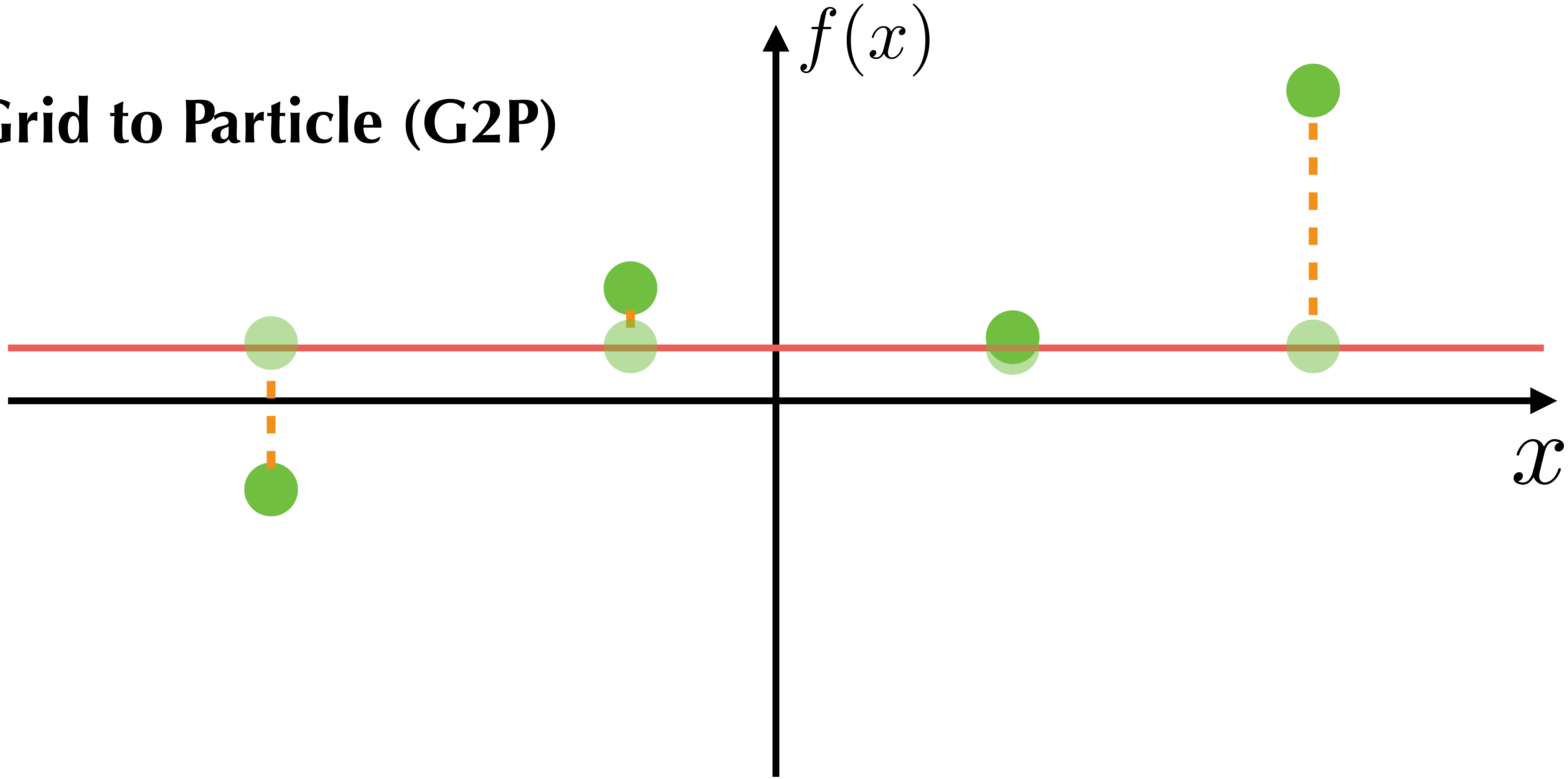
1D Curve Fitting

Grid to Particle (G2P)

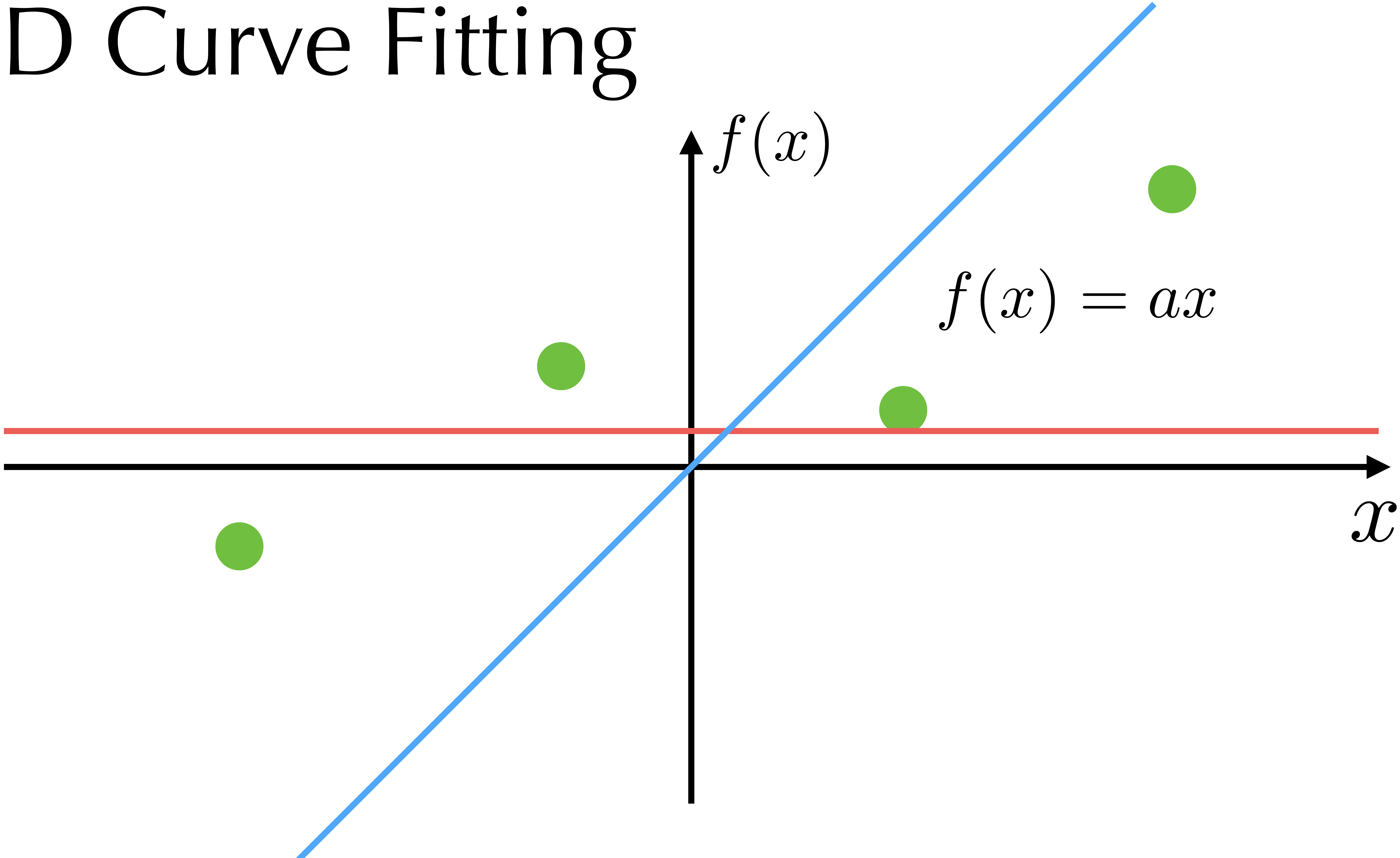


1D Curve Fitting

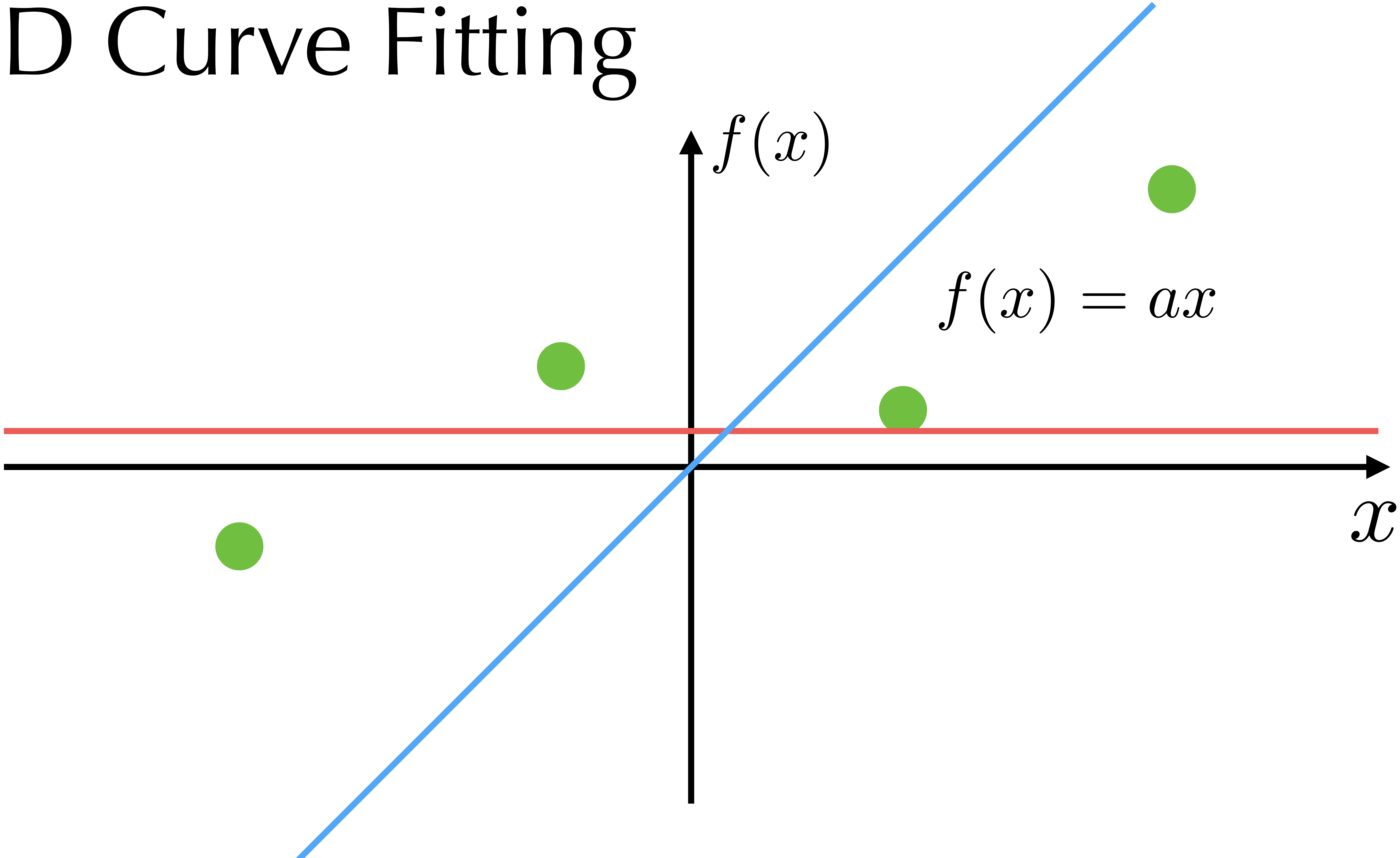
Grid to Particle (G2P)



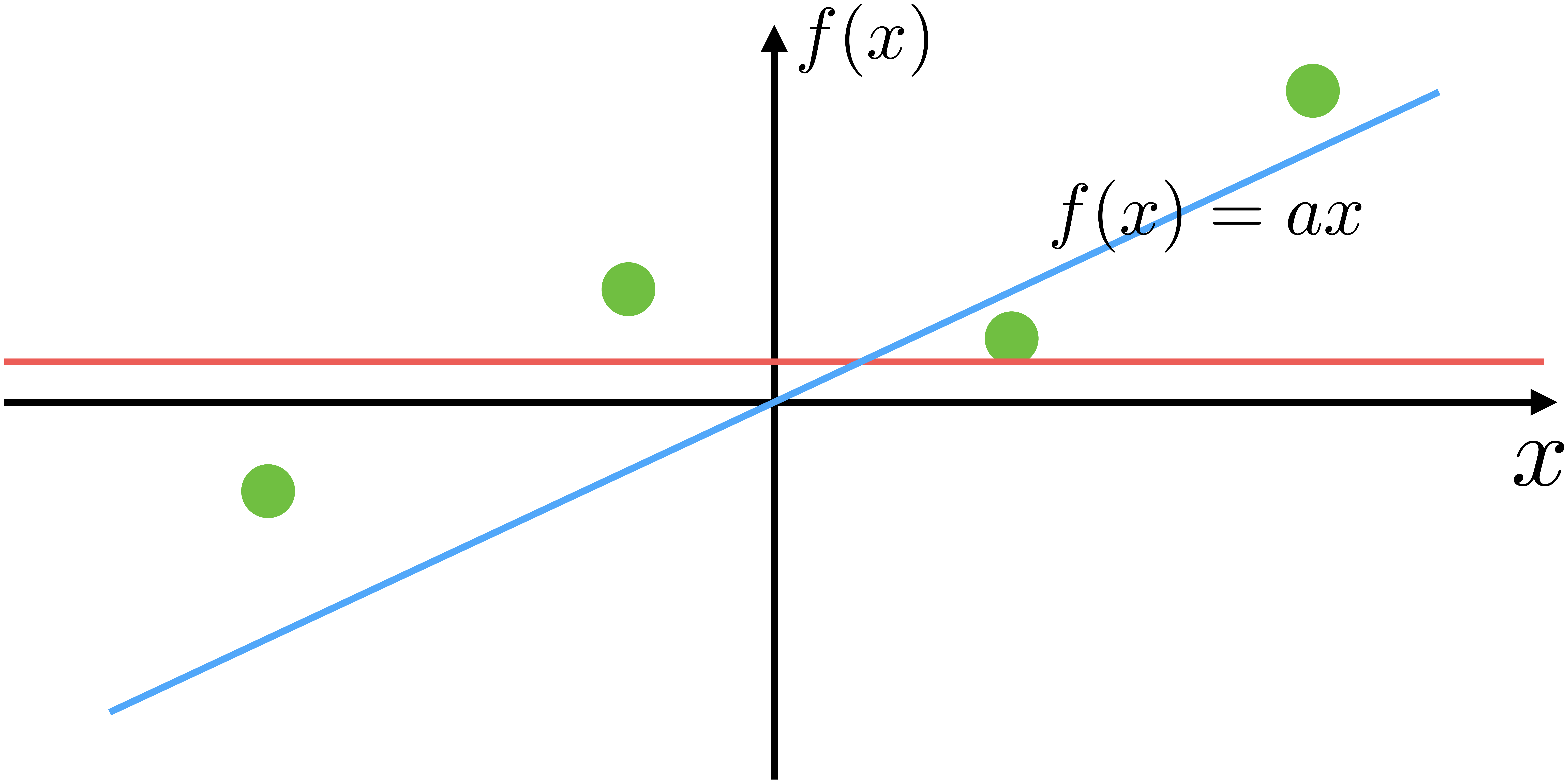
1D Curve Fitting



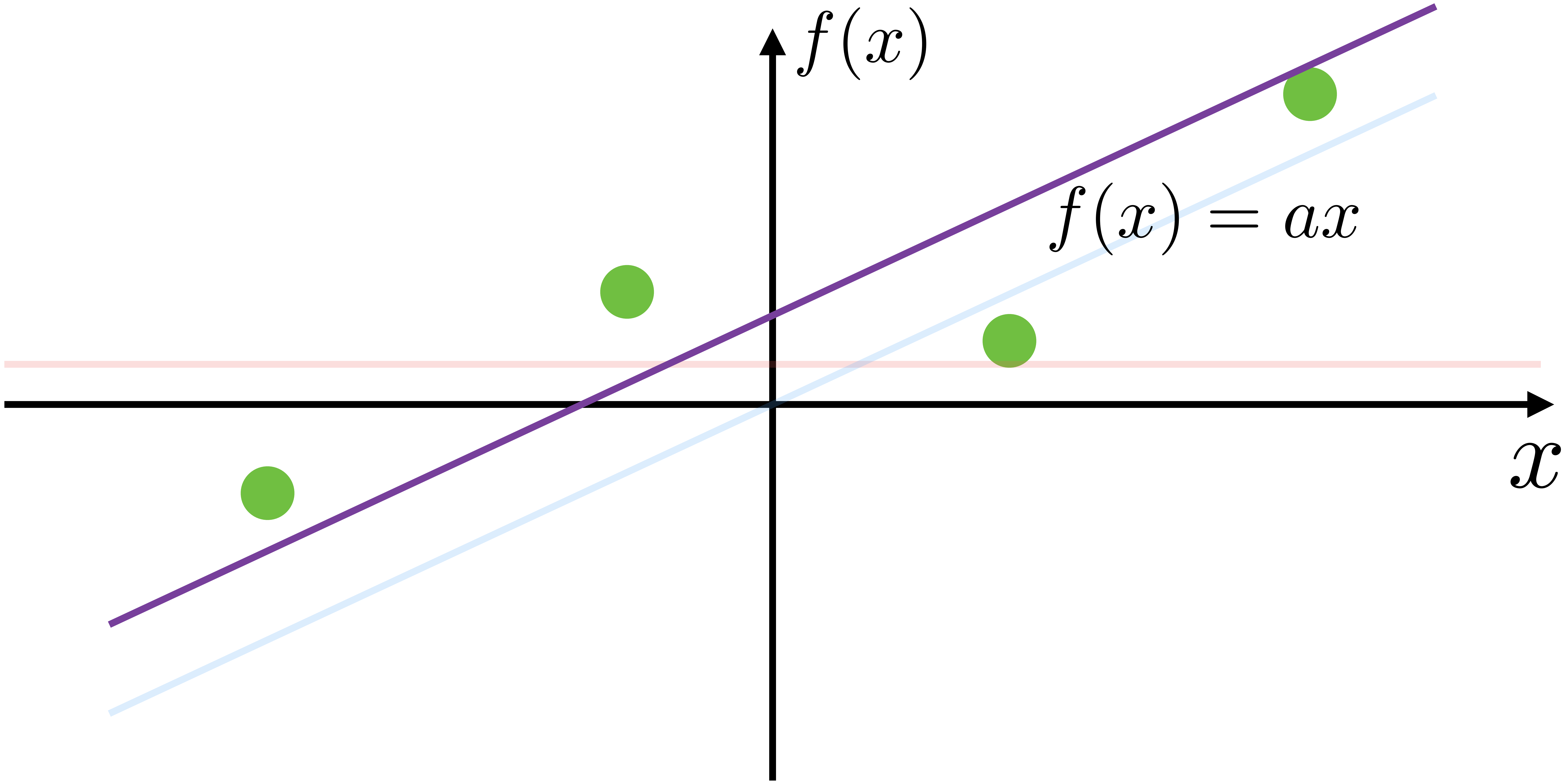
1D Curve Fitting



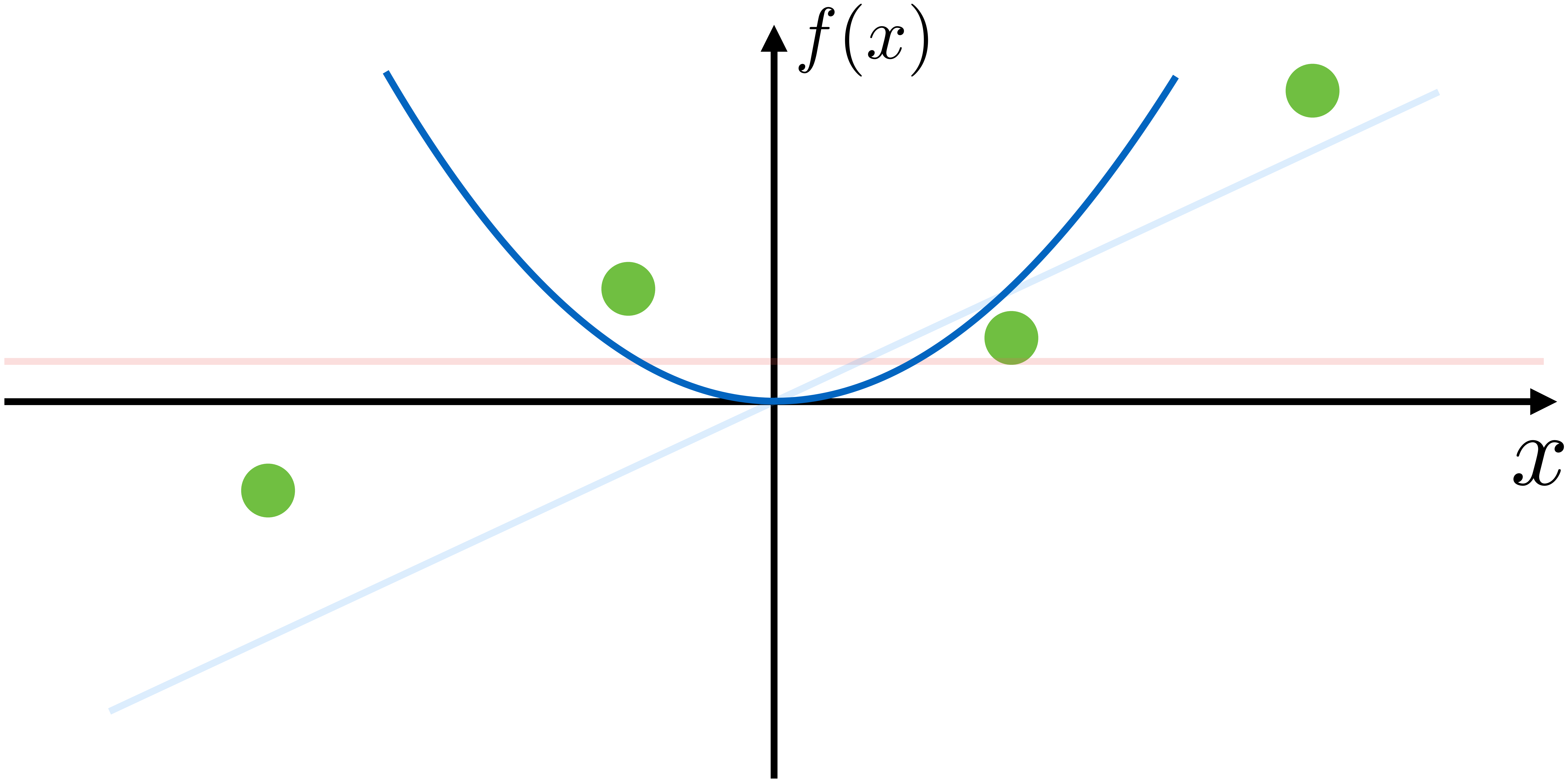
1D Curve Fitting



1D Curve Fitting

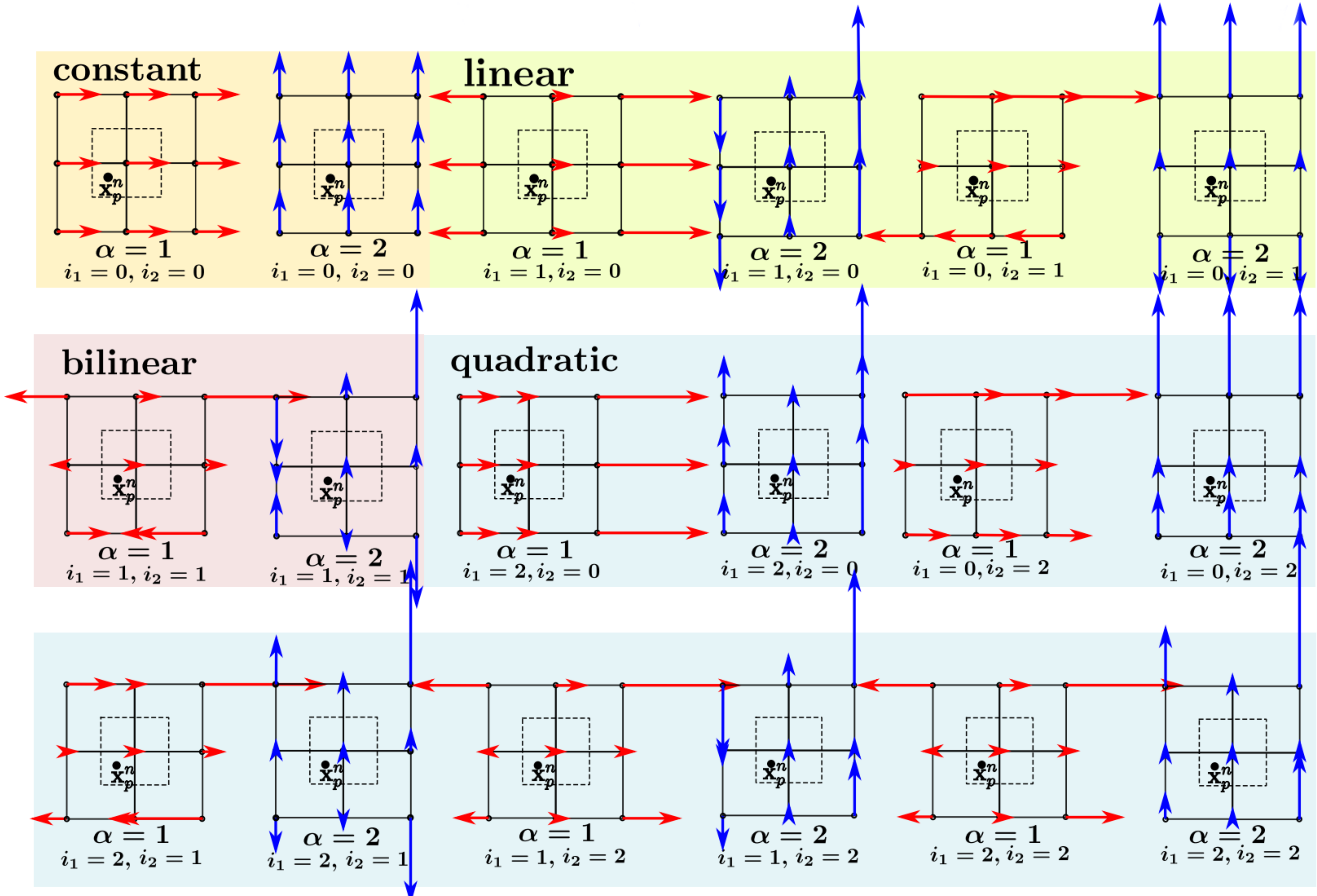


1D Curve Fitting



Least-Squares Transfers in 2D

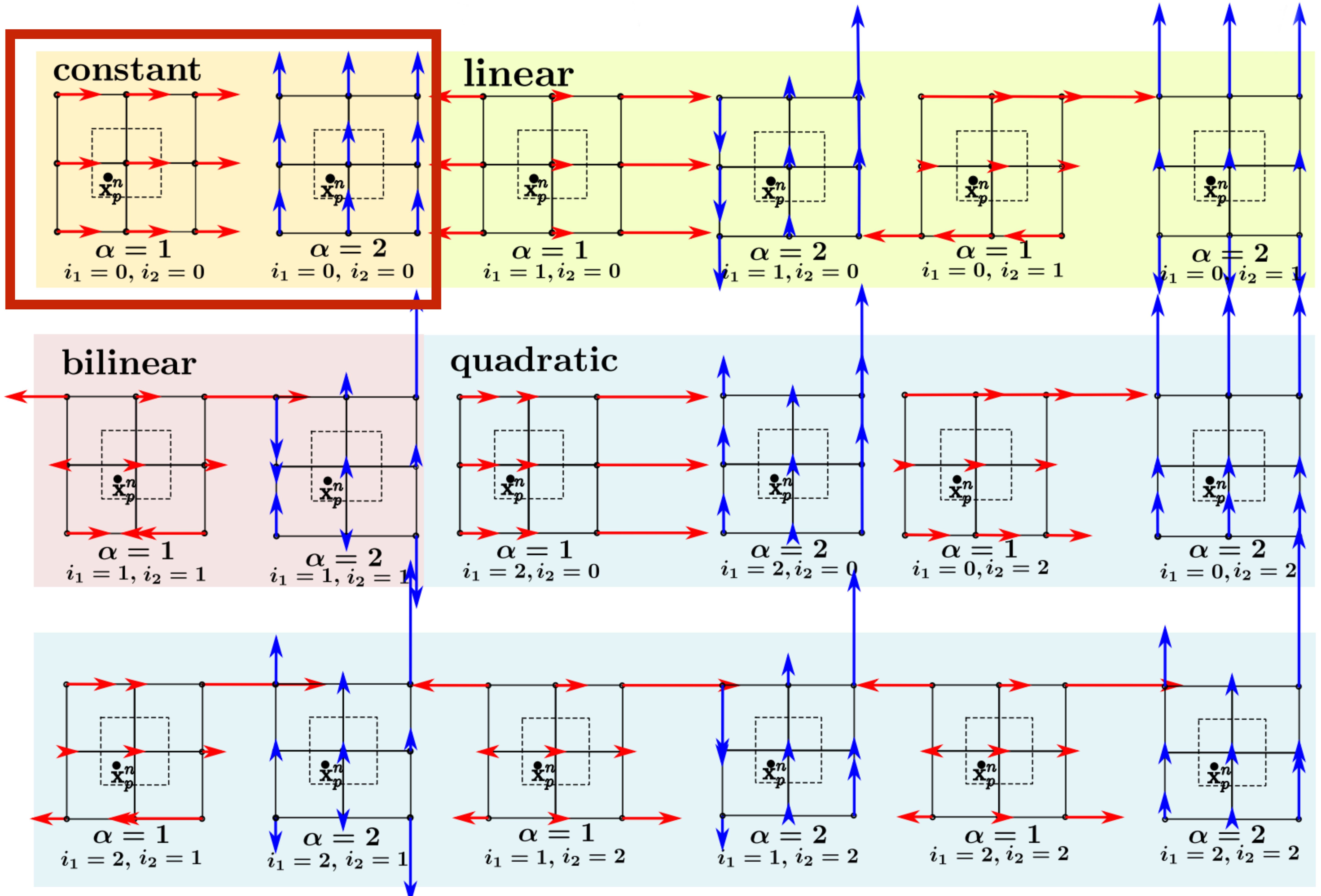
Figure from A Polynomial Particle-In-Cell Method, Fu et al. 2017



Least-Squares Transfers in 2D

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

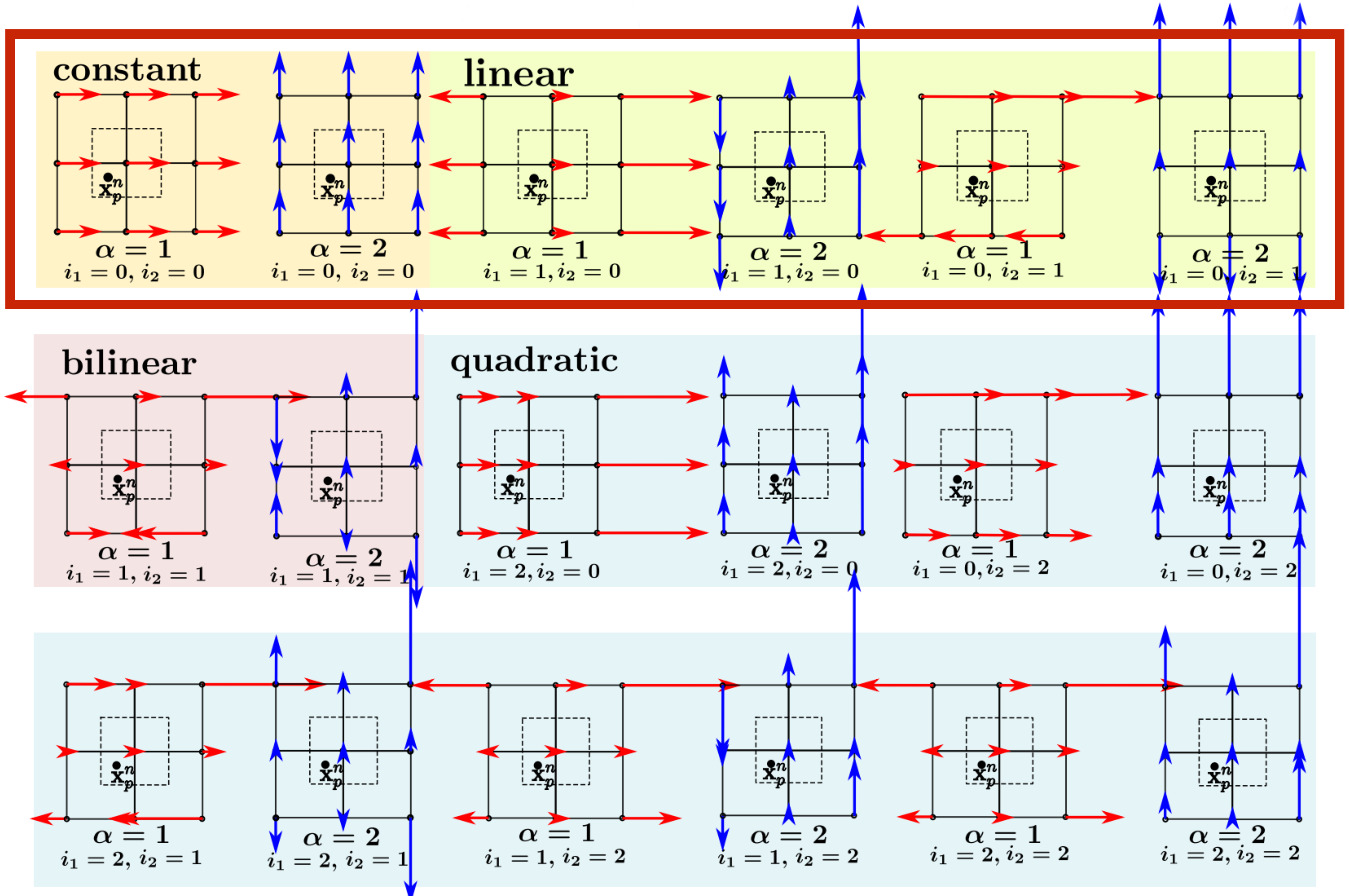
PIC



Least-Squares Transfers in 2D

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

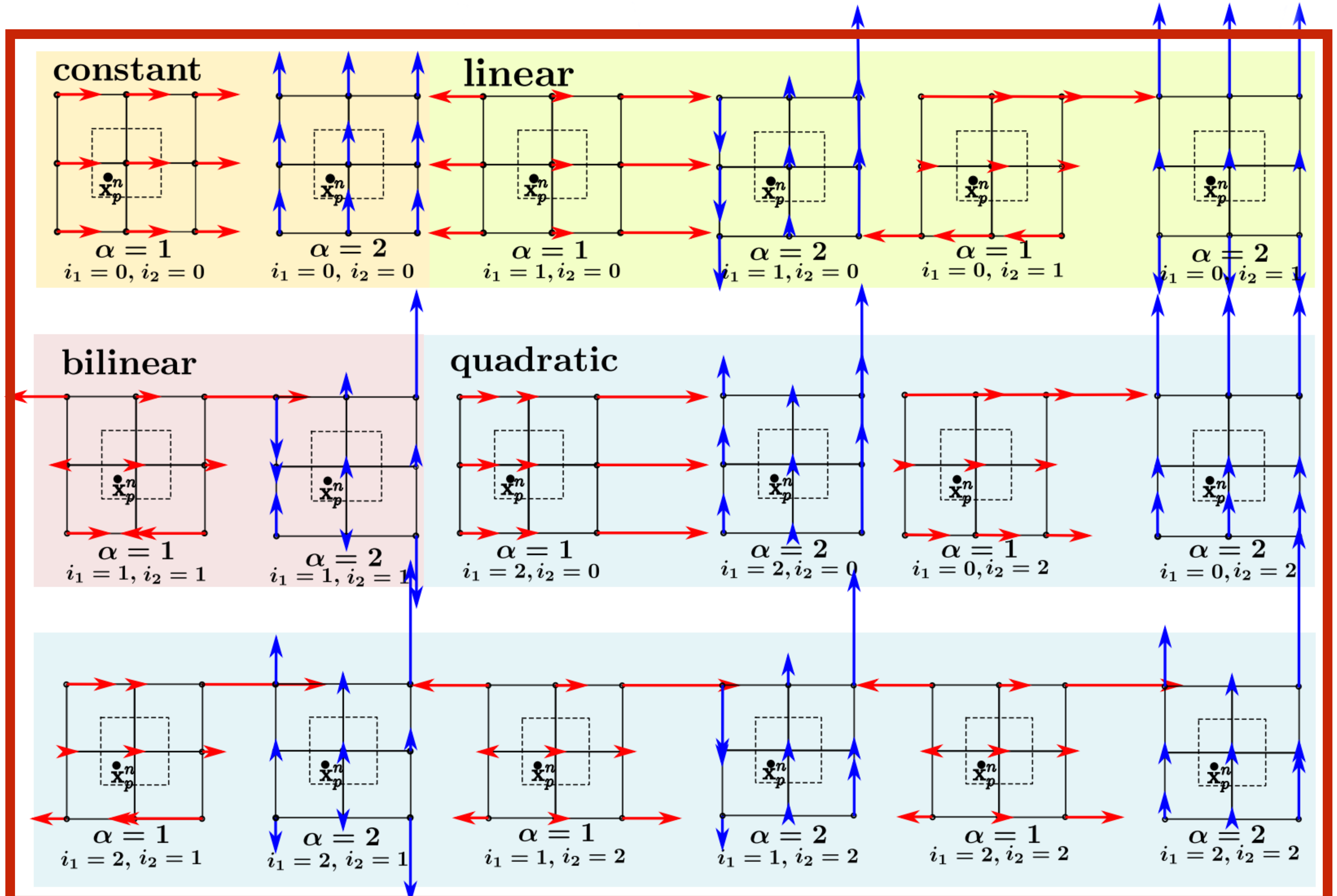
APIC



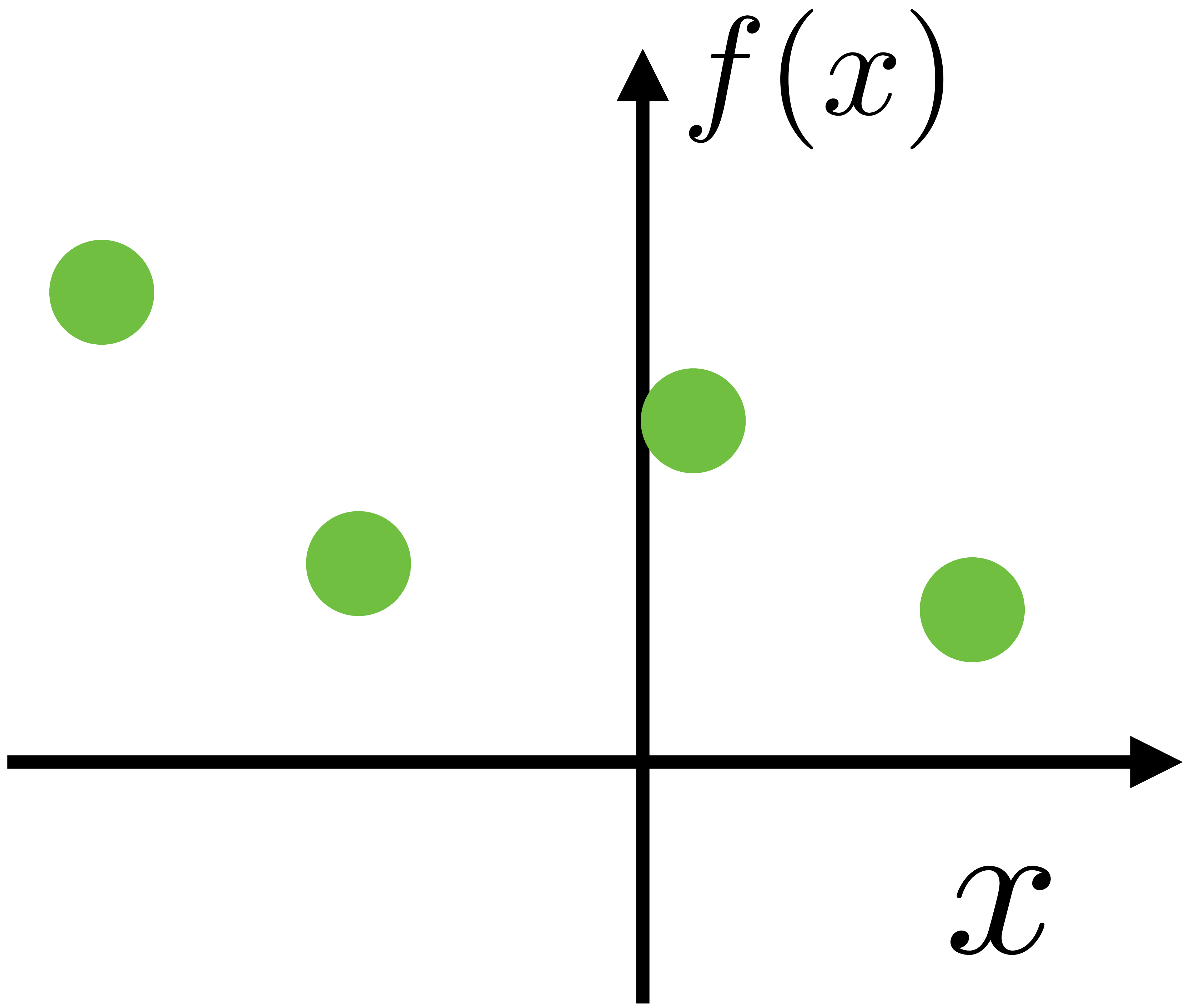
Least-Squares Transfers in 2D

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

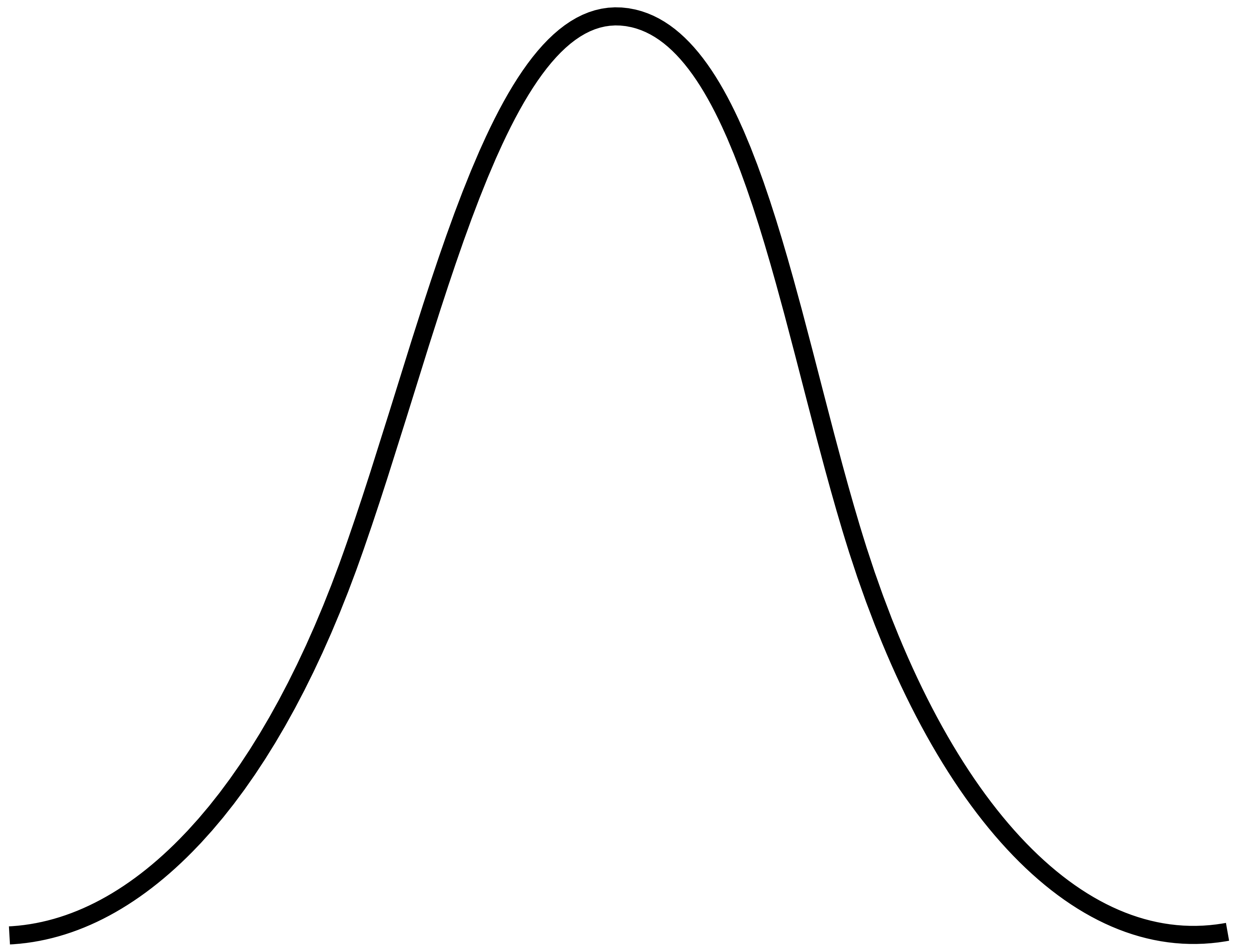
PolyPIC



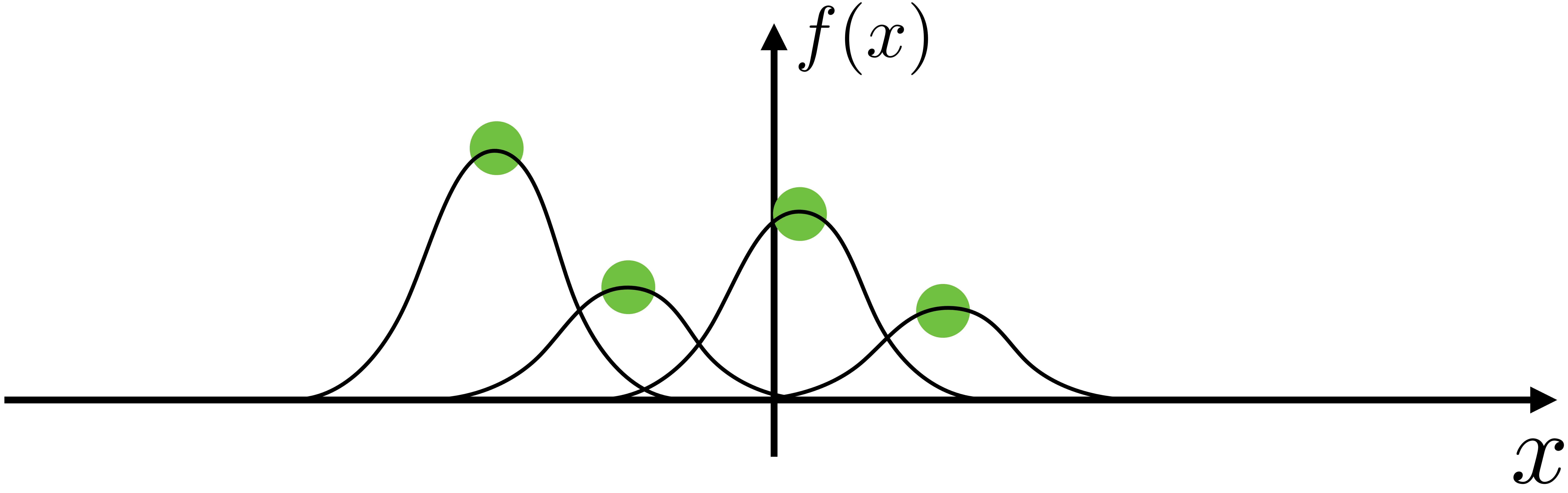
1D Curve Fitting: Spline Interpolation



+

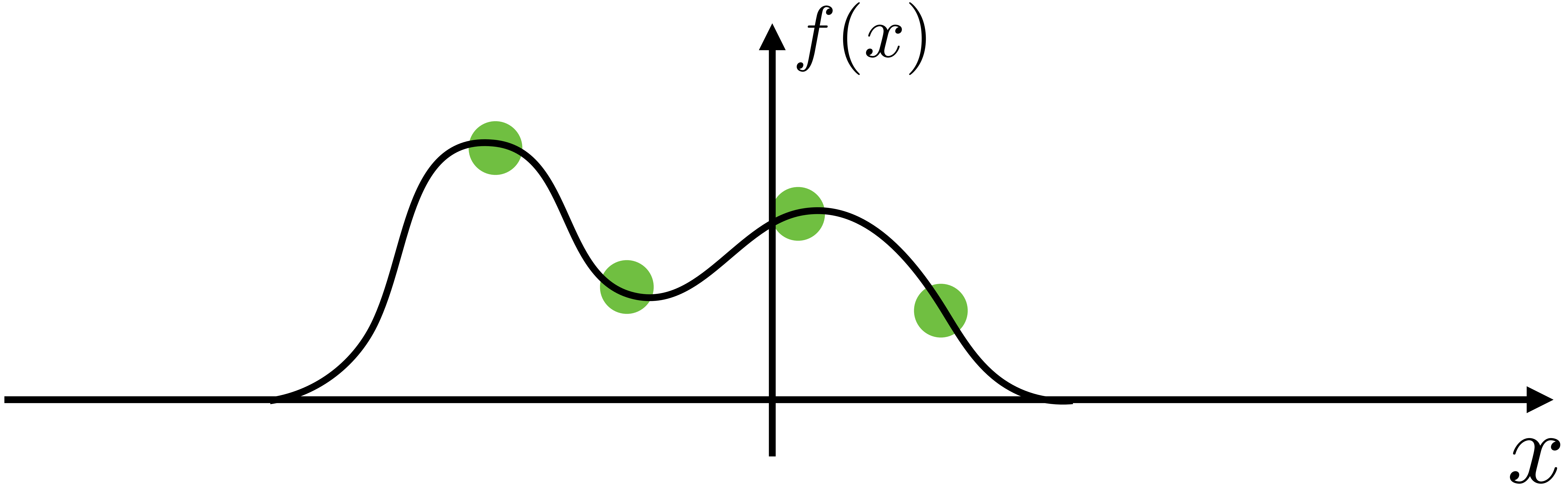


1D Curve Fitting: Spline Interpolation



"Shape functions" in FEM and MPM

1D Curve Fitting: Spline Interpolation



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

Which one to use?

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

Which one to use?

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

Which one to use?

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

Which one to use?

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

Which one to use?

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

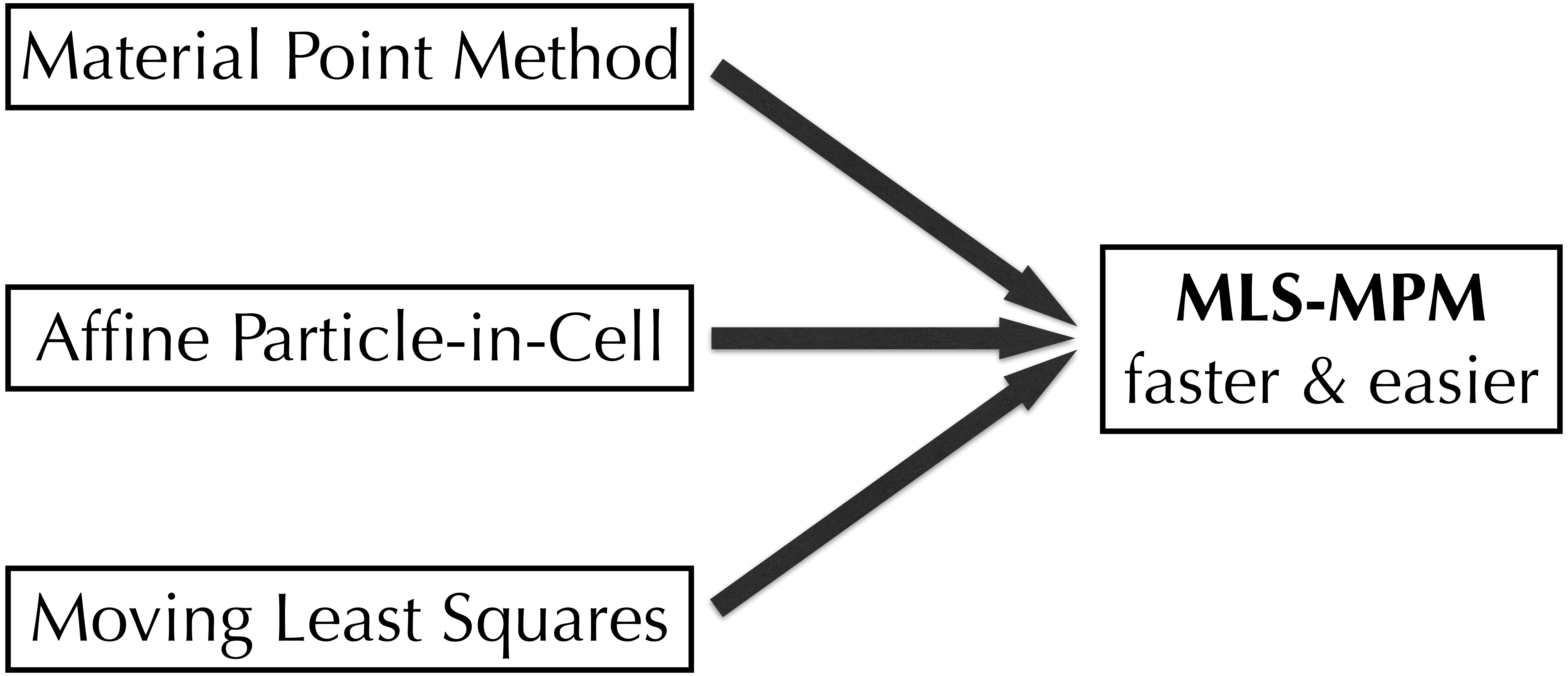
Material Point Method

Affine Particle-in-Cell

Material Point Method

Affine Particle-in-Cell

Moving Least Squares


Material Point Method
Affine Particle-in-Cell
Moving Least Squares
MLS-MPM
faster & easier

#include “taichi.h”

Material Point Method

Affine Particle-in-Cell

Moving Least Squares

MLS-MPM
faster & easier

```cpp
// The Moving Least Squares Material Point Method in 88 LoC (with comments)
//   To compile:   g++ mpm.cpp -std=c++14 -g -lX11 -lpthread -O2 -o mpm
#include "taichi.h" // Single header version of (a small part of) taichi
using namespace taichi;
const int n = 64 /*grid resolution (cells)*/, window_size = 500;
const real dt = 1e-4_f, frame_dt = 1e-3_f, dx = 1.0_f / n, inv_dx = 1.0_f / dx;
real mass = 1.0_f, vol = 1.0_f; // Particle mass and volume
real hardening = 10, E = 1e4 /* Young's Modulus*/, nu = 0.2 /*Poisson's Ratio*/;
real mu_0 = E/(2*(1+nu)), lambda_0=E*nu/((1+nu)*(1-2*nu));    // Lame parameters
using Vec = Vector2; using Mat = Matrix2; //Handy abbriviations for lin. algebra
struct Particle {Vec x/*position*/, v/*velocity*/; Mat B/*affine momentum*/;
  Mat F/*elastic deformation grad.*/;   real Jp /*det(plastic def. grad.)*/;
  Particle(Vec x, Vec v=Vec(0)) : x(x), v(v), B(0), F(1), Jp(1) {} };
std::vector<Particle> particles; // Particle states
Vector3 grid[n + 1][n + 1];// velocity with mass, note that node res=cell res+1

void advance(real dt) {                       // Simulation
  std::memset(grid, 0, sizeof(grid));  // Reset grid
  for (auto &p : particles) {             // P2G
    Vector2i base_coord = (p.x*inv_dx-Vec(0.5_f)).cast<int>();
    Vec fx = p.x * inv_dx - base_coord.cast<real>();
    // Quadratic kernels, see http://mpm.graphics Formula (123)
    Vec w[3]{Vec(0.5) * sqr(Vec(1.5) - fx), Vec(0.75) - sqr(fx - Vec(1.0)),
             Vec(0.5) * sqr(fx - Vec(0.5))};
    auto e = std::exp(hardening * (1.0_f - p.Jp)), mu=mu_0*e, lambda=lambda_0*e;
    real J = determinant(p.F);            //Current volume
    Mat r, s; polar_decomp(p.F, r, s); //Polor decomp. for Fixed Corotated Model
    auto force =                       // Negative Cauchy stress times dt and inv_dx
        inv_dx*dt*vol*(2*mu * (p.F-r) * transposed(p.F) + lambda * (J-1) * J);
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) { // Scatter to grid
      auto dpos = fx - Vec(i, j);
      Vector3 contrib(p.v * mass, mass);
      grid[base_coord.x + i][base_coord.y + j] +=
          w[i].x*w[j].y*(contrib+Vector3(4.0_f*(force+p.B*mass)*dpos));
    }
  }
  for(int i = 0; i <= n; i++) for(int j = 0; j <= n; j++) { //For all grid nodes
    auto &g = grid[i][j];
    if (g[2] > 0) {                                  // No need for epsilon here
      g /= g[2];                                     // Normalize by mass
      g += dt * Vector3(0, -100, 0);                 // Apply gravity
      real boundary=0.05,x=(real)i/n,y=real(j)/n;//boundary thickness,node coord
      if (x < boundary||x > 1-boundary||y > 1-boundary) g=Vector3(0);//Sticky BC
      if (y < boundary) g[1]=std::max(0.0_f, g[1]);              //"Separate" BC
    }   // "BC" stands for "boundary condition", which is applied to grid nodes
  }
  for (auto &p : particles) { // Grid to particle
    Vector2i base_coord = (p.x * inv_dx - Vec(0.5_f)).cast<int>();
    Vec fx = p.x * inv_dx - base_coord.cast<real>();
    Vec w[3]{Vec(0.5) * sqr(Vec(1.5) - fx), Vec(0.75) - sqr(fx - Vec(1.0)),
             Vec(0.5) * sqr(fx - Vec(0.5))};
    p.B = Mat(0); p.v = Vec(0);
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) {
      auto dpos = fx - Vec(i, j),
           grid_v = Vec(grid[base_coord.x + i][base_coord.y + j]);
      auto weight = w[i].x * w[j].y;
      p.v += weight * grid_v;                                   //  Velocity
      p.B += Mat::outer_product(weight * grid_v, dpos);         //    APIC B
    }
    p.x += dt * p.v;                                            // Advection
    auto F = (Mat(1) - (4 * inv_dx * dt) * p.B) * p.F;      // MLS-MPM F-update
    Mat svd_u, sig, svd_v; svd(F, svd_u, sig, svd_v); // SVD for snow Plasticity
    for (int i = 0; i < 2; i++)    // See SIGGRAPH 2013: MPM for Snow Simulation
      sig[i][i] = clamp(sig[i][i], 1.0_f - 2.5e-2_f, 1.0_f + 7.5e-3_f);
    real oldJ = determinant(F); F = svd_u * sig * transposed(svd_v);
    real Jp_new = clamp(p.Jp * oldJ / determinant(F), 0.6_f, 20.0_f);
    p.Jp = Jp_new; p.F = F;
  }
}

void add_object(Vec center) {    // Seed particles
  for (int i = 0; i < 1000; i++) // Randomly sample 1000 particles in the square
    particles.push_back(Particle((Vec::rand()*2.0_f-Vec(1))*0.08_f+center));  }

int main() {
  GUI gui("Taichi Demo: Real-time MLS-MPM 2D ", window_size, window_size);
  add_object(Vec(0.5,0.4));add_object(Vec(0.45,0.6));add_object(Vec(0.55,0.8));
  for (int i = 0;; i++) {                              // Main Loop
    advance(dt);                                       // Advance simulation
    if (i % int(frame_dt / dt) == 0) {                 // Redraw frame
      gui.canvas->clear(Vector4(0.7, 0.4, 0.2, 1.0_f)); // Clear background
      for (auto p : particles)                         // Draw particles
        gui.buffer[(p.x * (inv_dx*window_size/n)).cast<int>()] = Vector4(0.8);
      gui.update();                                    // Update GUI
    }//Reference: A Moving Least Squares Material Point Method with Displacement
  } //              Discontinuity and Two-Way Rigid Body Coupling (SIGGRAPH 2018)
}  //  By Yuanming Hu (who also wrote this 88-line version), Yu Fang, Ziheng Ge,
   //                           Ziyin Qu, Yixin Zhu, Andre Pradhana, Chenfanfu Jiang
```

#include “taichi.h”

Material Point Method

Affine Particle-in-Cell

Moving Least Squares

MLS-MPM
faster & easier

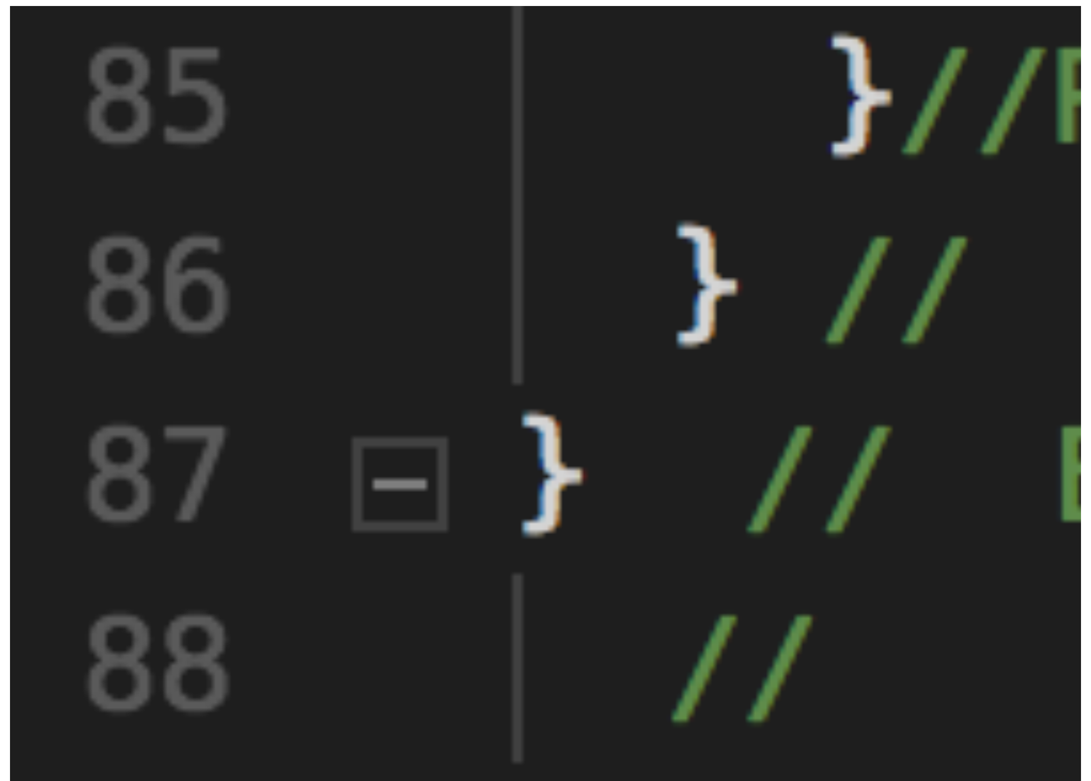
```cpp
// The Moving Least Squares Material Point Method in 88 LoC (with comments)
//   To compile:   g++ mpm.cpp -std=c++14 -g -lX11 -lpthread -O2 -o mpm
#include "taichi.h" // Single header version of (a small part of) taichi
using namespace taichi;
const int n = 64 /*grid resolution (cells)*/, window_size = 500;
const real dt = 1e-4_f, frame_dt = 1e-3_f, dx = 1.0_f / n, inv_dx = 1.0_f / dx;
real mass = 1.0_f, vol = 1.0_f; // Particle mass and volume
real hardening = 10, E = 1e4 /* Young's Modulus*/, nu = 0.2 /*Poisson's Ratio*/;
real mu_0 = E/(2*(1+nu)), lambda_0=E*nu/((1+nu)*(1-2*nu));    // Lame parameters
using Vec = Vector2; using Mat = Matrix2; //Handy abbriviations for lin. algebra
struct Particle {Vec x/*position*/, v/*velocity*/; Mat B/*affine momentum*/;
  Mat F/*elastic deformation grad.*/;   real Jp /*det(plastic def. grad.)*/;
  Particle(Vec x, Vec v=Vec(0)) : x(x), v(v), B(0), F(1), Jp(1) {} };
std::vector<Particle> particles; // Particle states
Vector3 grid[n + 1][n + 1];// velocity with mass, note that node res=cell res+1

void advance(real dt) {                // Simulation
  std::memset(grid, 0, sizeof(grid));  // Reset grid
  for (auto &p : particles) {          // P2G
    Vector2i base_coord = (p.x*inv_dx-Vec(0.5_f)).cast<int>();
    Vec fx = p.x * inv_dx - base_coord.cast<real>();
    // Quadratic kernels, see http://mpm.graphics Formula (123)
    Vec w[3]{Vec(0.5) * sqr(Vec(1.5) - fx), Vec(0.75) - sqr(fx - Vec(1.0)),
             Vec(0.5) * sqr(fx - Vec(0.5))};
    auto e = std::exp(hardening * (1.0_f - p.Jp)), mu=mu_0*e, lambda=lambda_0*e;
    real J = determinant(p.F);         //Current volume
    Mat r, s; polar_decomp(p.F, r, s); //Polor decomp. for Fixed Corotated Model
    auto force =                       // Negative Cauchy stress times dt and inv_dx
        inv_dx*dt*vol*(2*mu * (p.F-r) * transposed(p.F) + lambda * (J-1) * J);
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) { // Scatter to grid
      auto dpos = fx - Vec(i, j);
      Vector3 contrib(p.v * mass, mass);
      grid[base_coord.x + i][base_coord.y + j] +=
          w[i].x*w[j].y*(contrib+Vector3(4.0_f*(force+p.B*mass)*dpos));
    }
  }
  for(int i = 0; i <= n; i++) for(int j = 0; j <= n; j++) { //For all grid nodes
    auto &g = grid[i][j];
    if (g[2] > 0) {                                // No need for epsilon here
      g /= g[2];                                   // Normalize by mass
      g += dt * Vector3(0, -100, 0);               // Apply gravity
      real boundary=0.05,x=(real)i/n,y=real(j)/n;//boundary thickness,node coord
      if (x < boundary||x > 1-boundary||y > 1-boundary) g=Vector3(0);//Sticky BC
      if (y < boundary) g[1]=std::max(0.0_f, g[1]);             //"Separate" BC
    }    // "BC" stands for "boundary condition", which is applied to grid nodes
  }
  for (auto &p : particles) { // Grid to particle
    Vector2i base_coord = (p.x * inv_dx - Vec(0.5_f)).cast<int>();
    Vec fx = p.x * inv_dx - base_coord.cast<real>();
    Vec w[3]{Vec(0.5) * sqr(Vec(1.5) - fx), Vec(0.75) - sqr(fx - Vec(1.0)),
             Vec(0.5) * sqr(fx - Vec(0.5))};
    p.B = Mat(0); p.v = Vec(0);
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) {
      auto dpos = fx - Vec(i, j),
           grid_v = Vec(grid[base_coord.x + i][base_coord.y + j]);
      auto weight = w[i].x * w[j].y;
      p.v += weight * grid_v;                                  //  Velocity
      p.B += Mat::outer_product(weight * grid_v, dpos);        //    APIC B
    }
    p.x += dt * p.v;                                           // Advection
    auto F = (Mat(1) - (4 * inv_dx * dt) * p.B) * p.F;     // MLS-MPM F-update
    Mat svd_u, sig, svd_v; svd(F, svd_u, sig, svd_v); // SVD for snow Plasticity
    for (int i = 0; i < 2; i++)    // See SIGGRAPH 2013: MPM for Snow Simulation
      sig[i][i] = clamp(sig[i][i], 1.0_f - 2.5e-2_f, 1.0_f + 7.5e-3_f);
    real oldJ = determinant(F); F = svd_u * sig * transposed(svd_v);
    real Jp_new = clamp(p.Jp * oldJ / determinant(F), 0.6_f, 20.0_f);
    p.Jp = Jp_new; p.F = F;
  }
}

void add_object(Vec center) {    // Seed particles
  for (int i = 0; i < 1000; i++) // Randomly sample 1000 particles in the square
    particles.push_back(Particle((Vec::rand()*2.0_f-Vec(1))*0.08_f+center));  }

int main() {
  GUI gui("Taichi Demo: Real-time MLS-MPM 2D ", window_size, window_size);
  add_object(Vec(0.5,0.4));add_object(Vec(0.45,0.6));add_object(Vec(0.55,0.8));
  for (int i = 0;; i++) {                              // Main Loop
    advance(dt);                                       // Advance simulation
    if (i % int(frame_dt / dt) == 0) {                 // Redraw frame
      gui.canvas->clear(Vector4(0.7, 0.4, 0.2, 1.0_f)); // Clear background
      for (auto p : particles)                         // Draw particles
        gui.buffer[(p.x * (inv_dx*window_size/n)).cast<int>()] = Vector4(0.8);
      gui.update();                                    // Update GUI
    }//Reference: A Moving Least Squares Material Point Method with Displacement
  } //               Discontinuity and Two-Way Rigid Body Coupling (SIGGRAPH 2018)
}  //  By Yuanming Hu (who also wrote this 88-line version), Yu Fang, Ziheng Ge,
  //                          Ziyin Qu, Yixin Zhu, Andre Pradhana, Chenfanfu Jiang
```

Implement Interactive MLS-MPM within
88 lines of code (comments included)!



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	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

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	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

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	$\nabla \boldsymbol{v}_p^{n+1} = \sum_{\boldsymbol{i}} \boldsymbol{v}_{\boldsymbol{i}}^{n+1} (\nabla w_{\boldsymbol{ip}}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \boxed{\mathbf{C}_{\boldsymbol{p}}^{\boldsymbol{n}+1}}$
Deformation Gradient Update	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}}) \right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}}) \right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

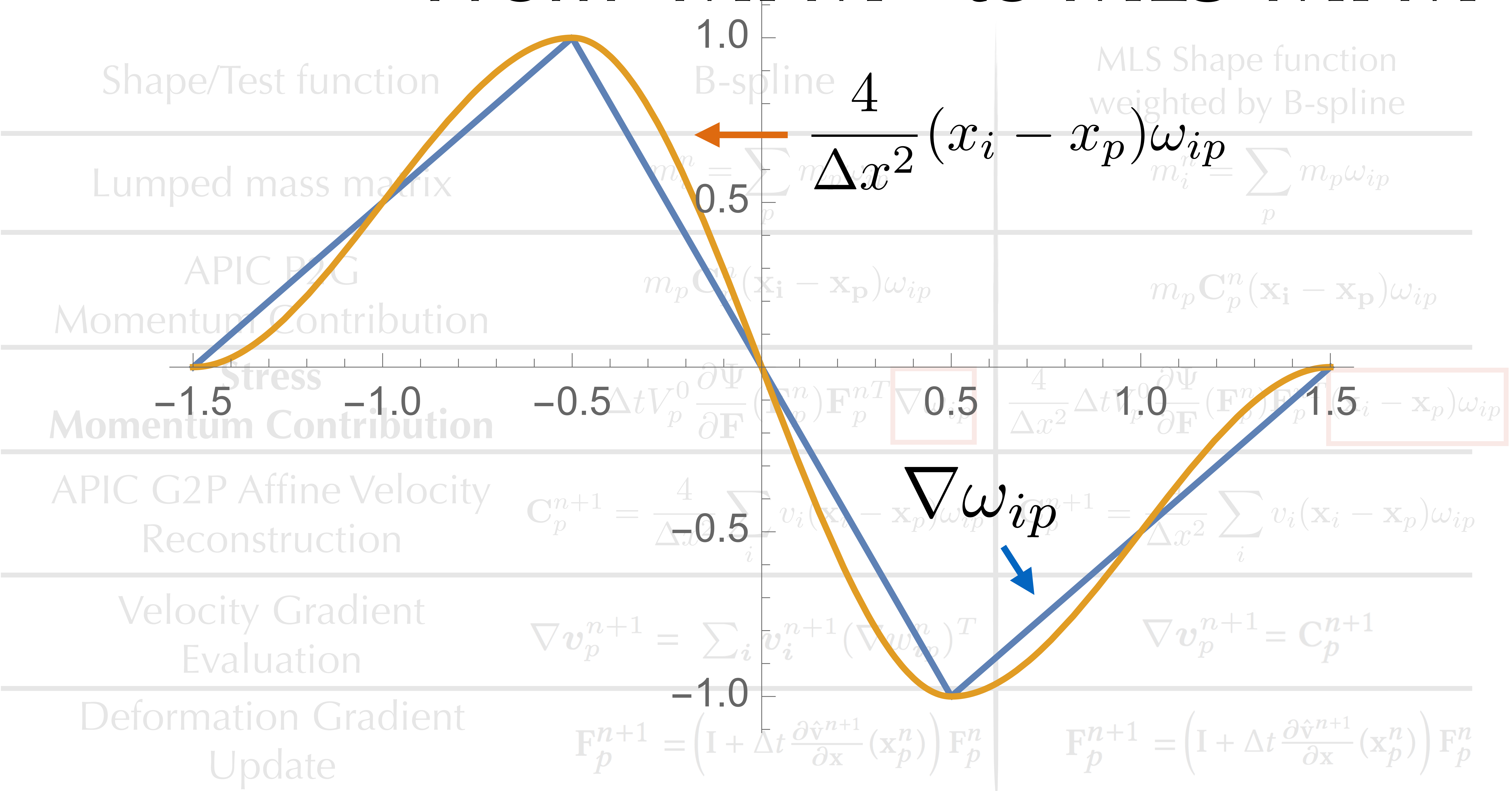
From **MPM** to **MLS-MPM**

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	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$	$\mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_{\boldsymbol{p}}^{\boldsymbol{n}})\right) \mathbf{F}_{\boldsymbol{p}}^{\boldsymbol{n}}$

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	$\nabla \boldsymbol{v}_p^{n+1} = \sum_i \boldsymbol{v}_i^{n+1} (\nabla w_{ip}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_p^{n+1}$
Deformation Gradient Update	$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_p^n)\right) \mathbf{F}_p^n$	$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}} (\mathbf{x}_p^n)\right) \mathbf{F}_p^n$

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



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	$\nabla \boldsymbol{v}_p^{n+1} = \sum_i \boldsymbol{v}_i^{n+1} (\nabla w_{ip}^n)^T$	$\nabla \boldsymbol{v}_p^{n+1} = \mathbf{C}_p^{n+1}$
Deformation Gradient Update	$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}}(\mathbf{x}_p^n)\right) \mathbf{F}_p^n$	$\mathbf{F}_p^{n+1} = \left(\mathbf{I} + \Delta t \frac{\partial \hat{\mathbf{v}}^{n+1}}{\partial \mathbf{x}}(\mathbf{x}_p^n)\right) \mathbf{F}_p^n$

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

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

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)

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)

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

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

Contributions

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

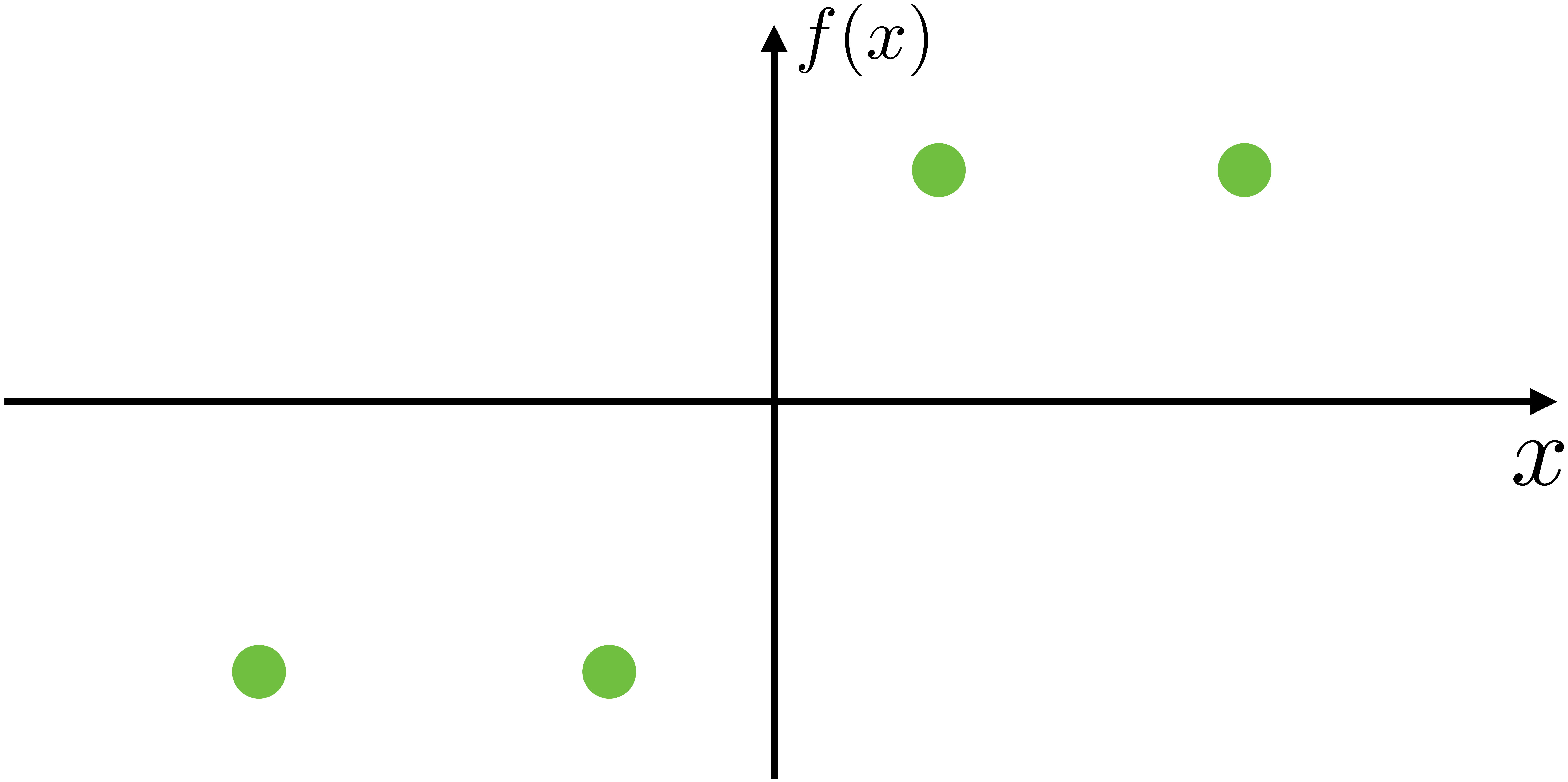
-100 lines of code!

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

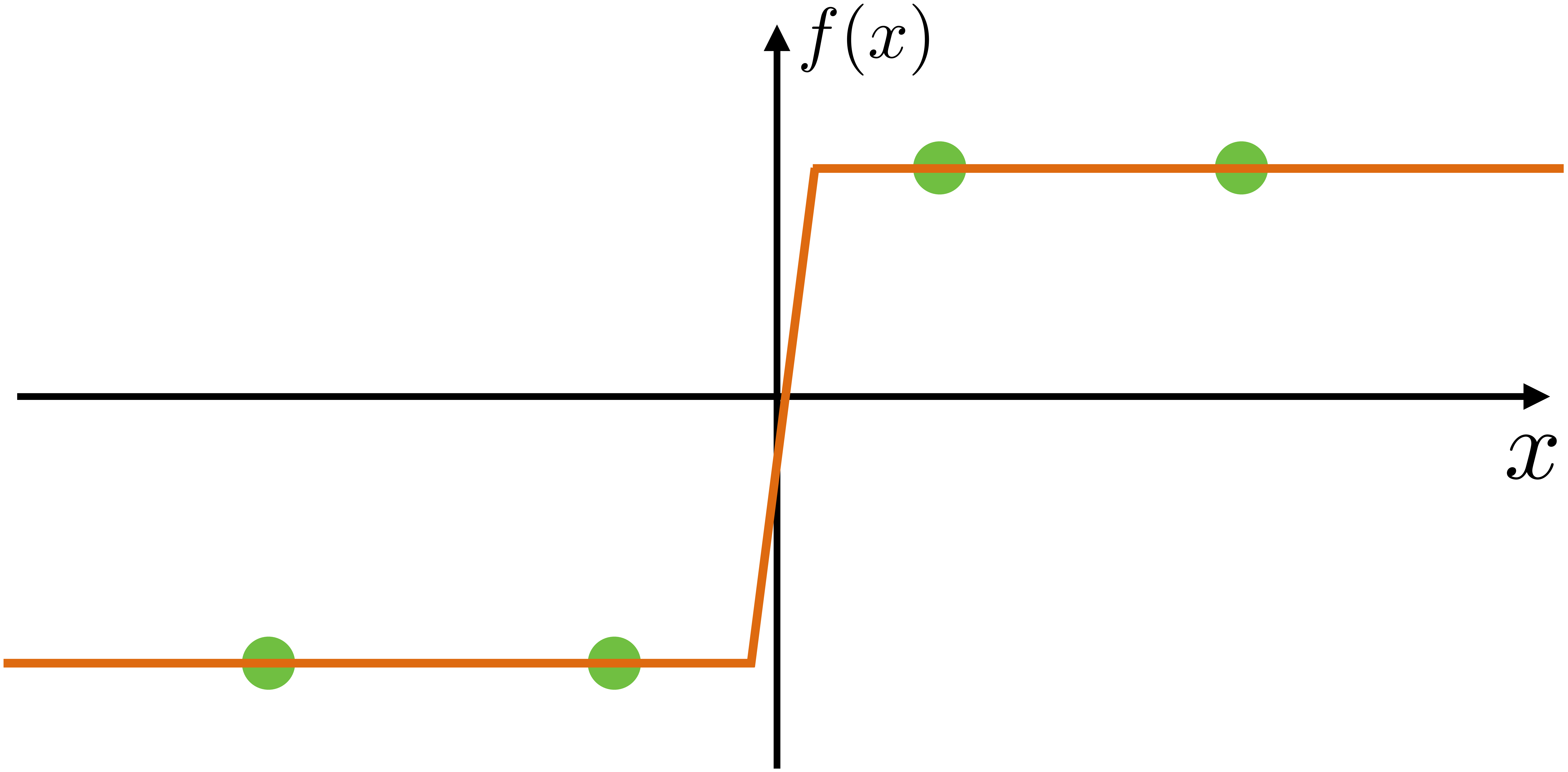
Contributions

- **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

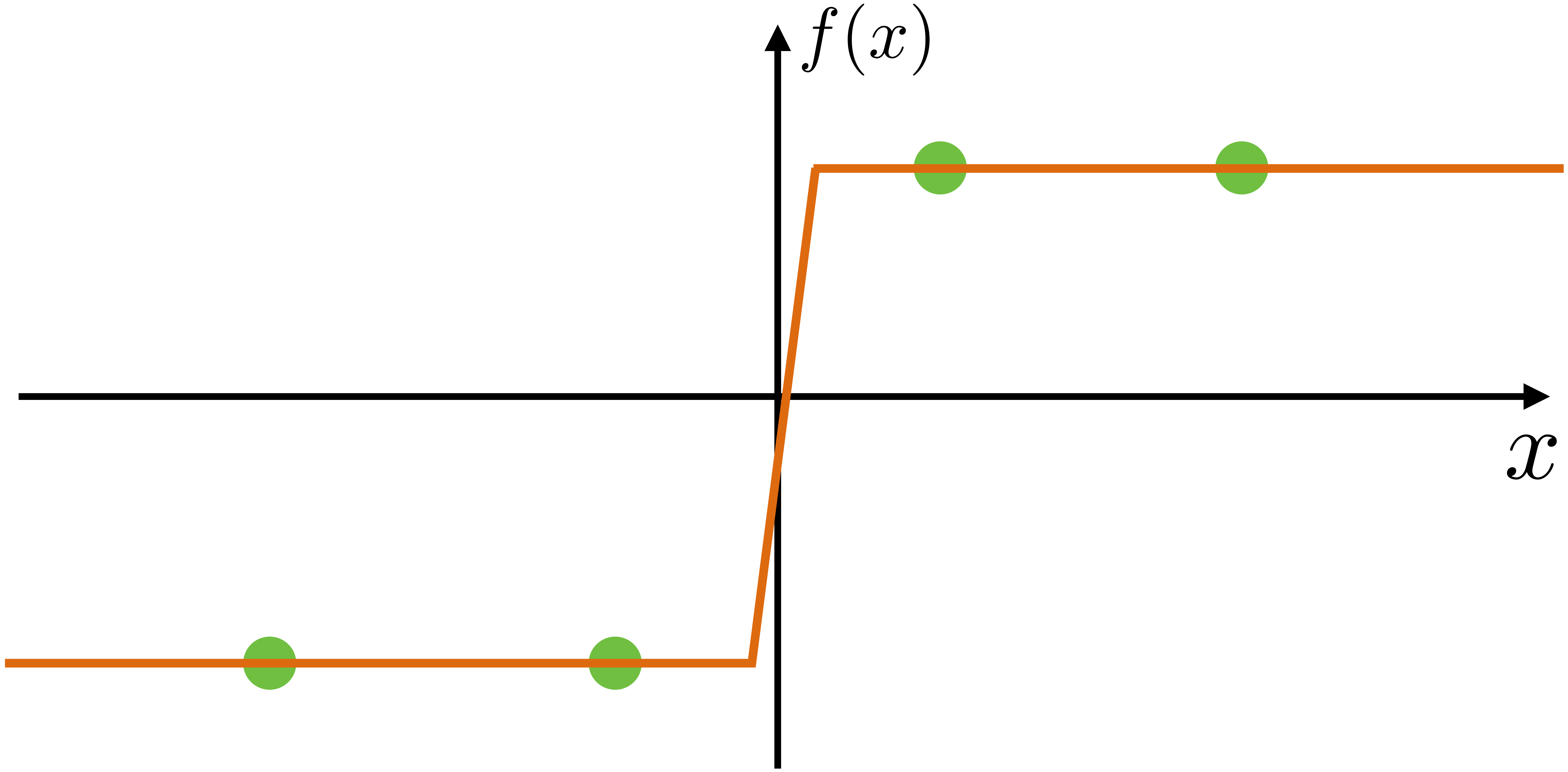
1D Curve Fitting



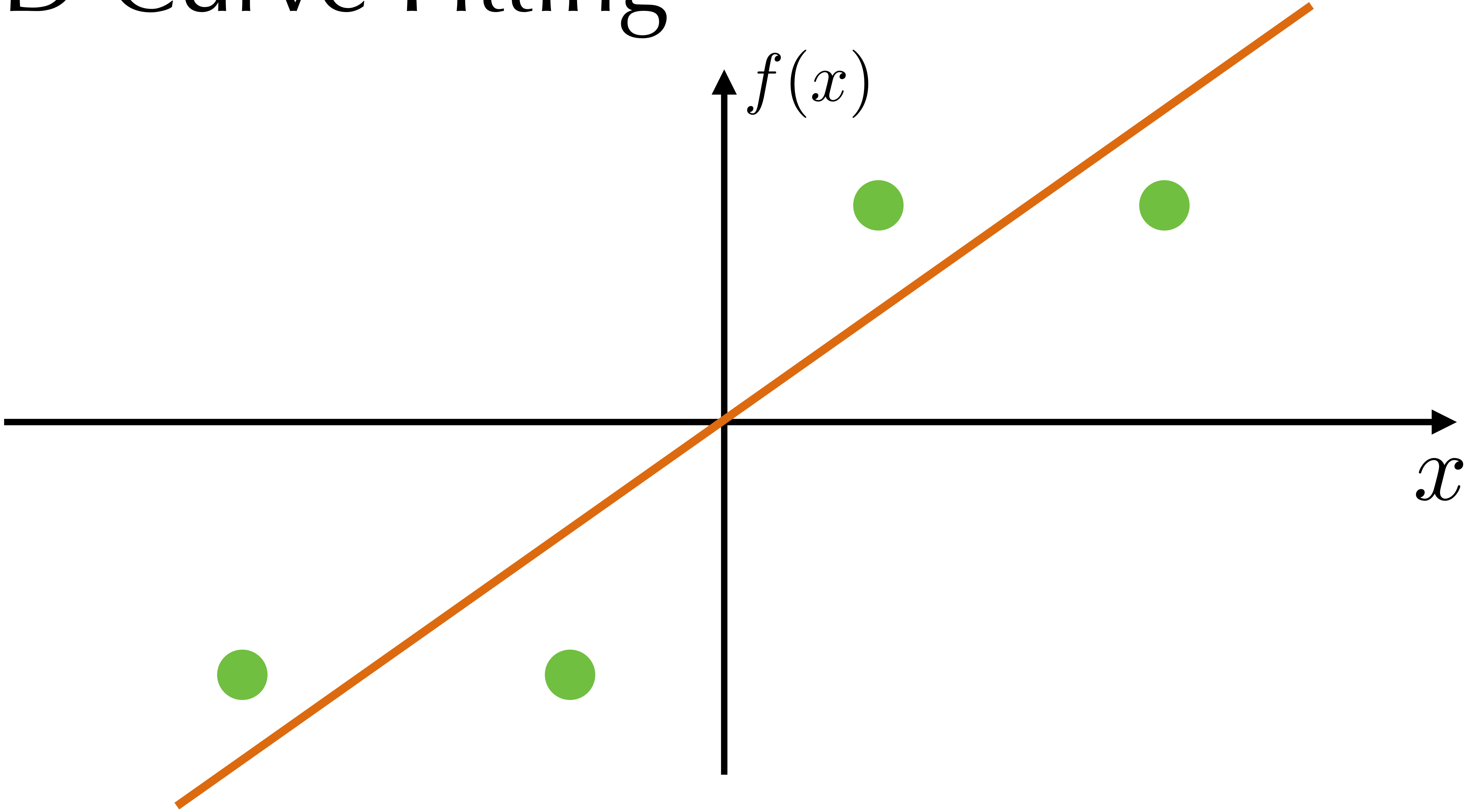
1D Curve Fitting

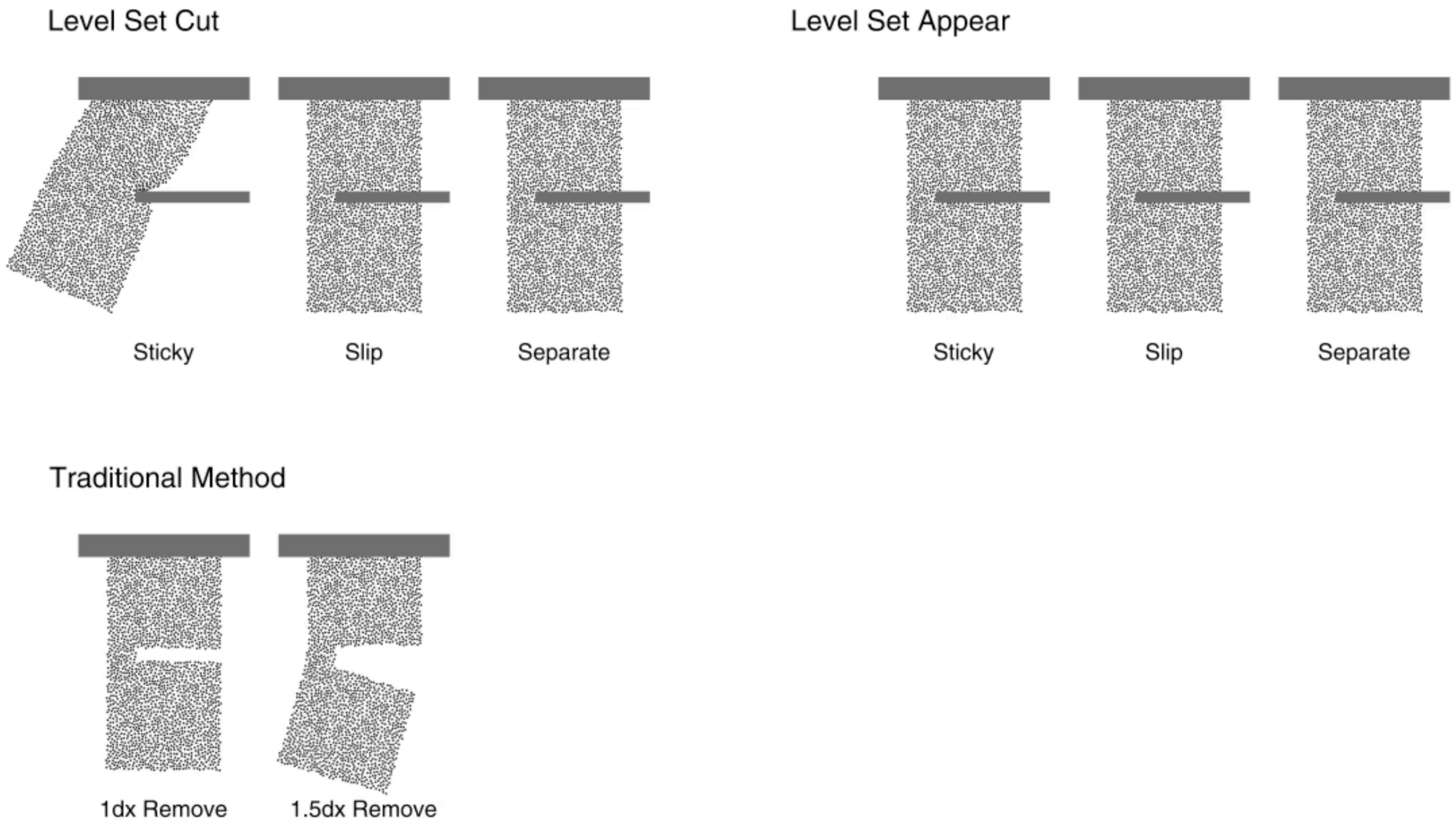


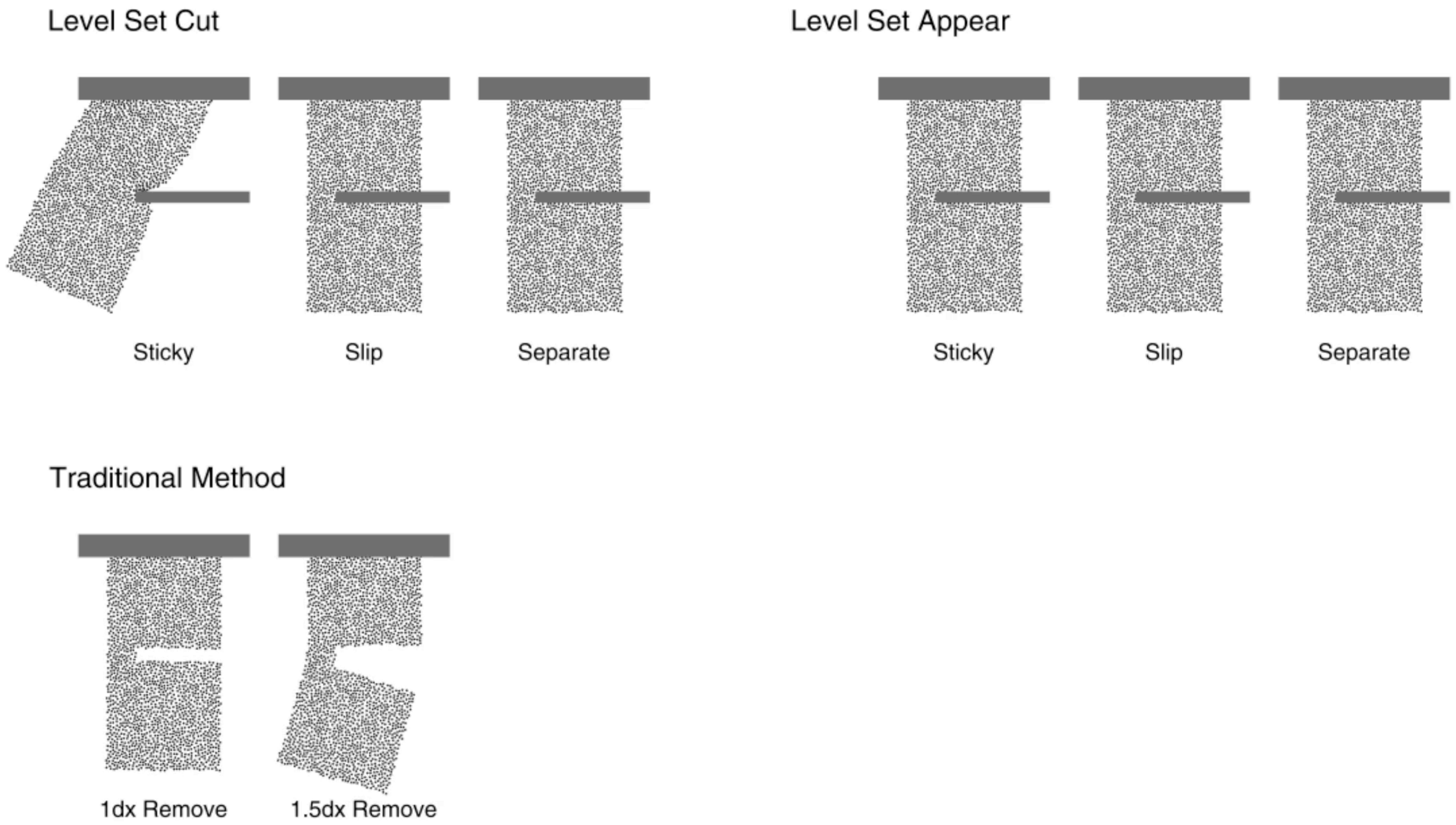
1D Curve Fitting

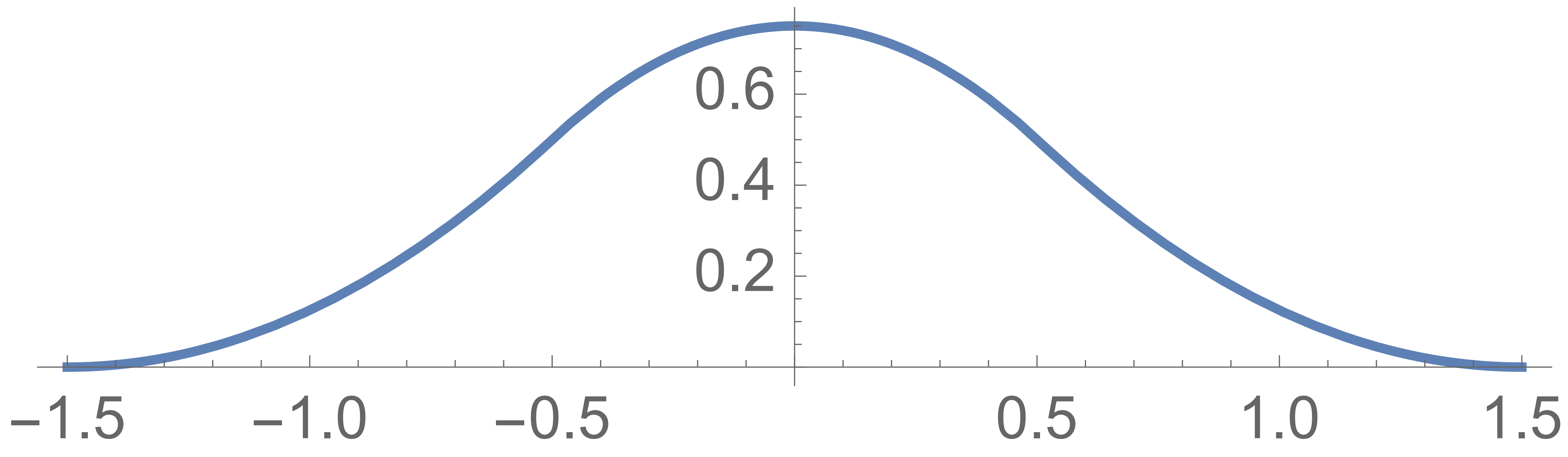


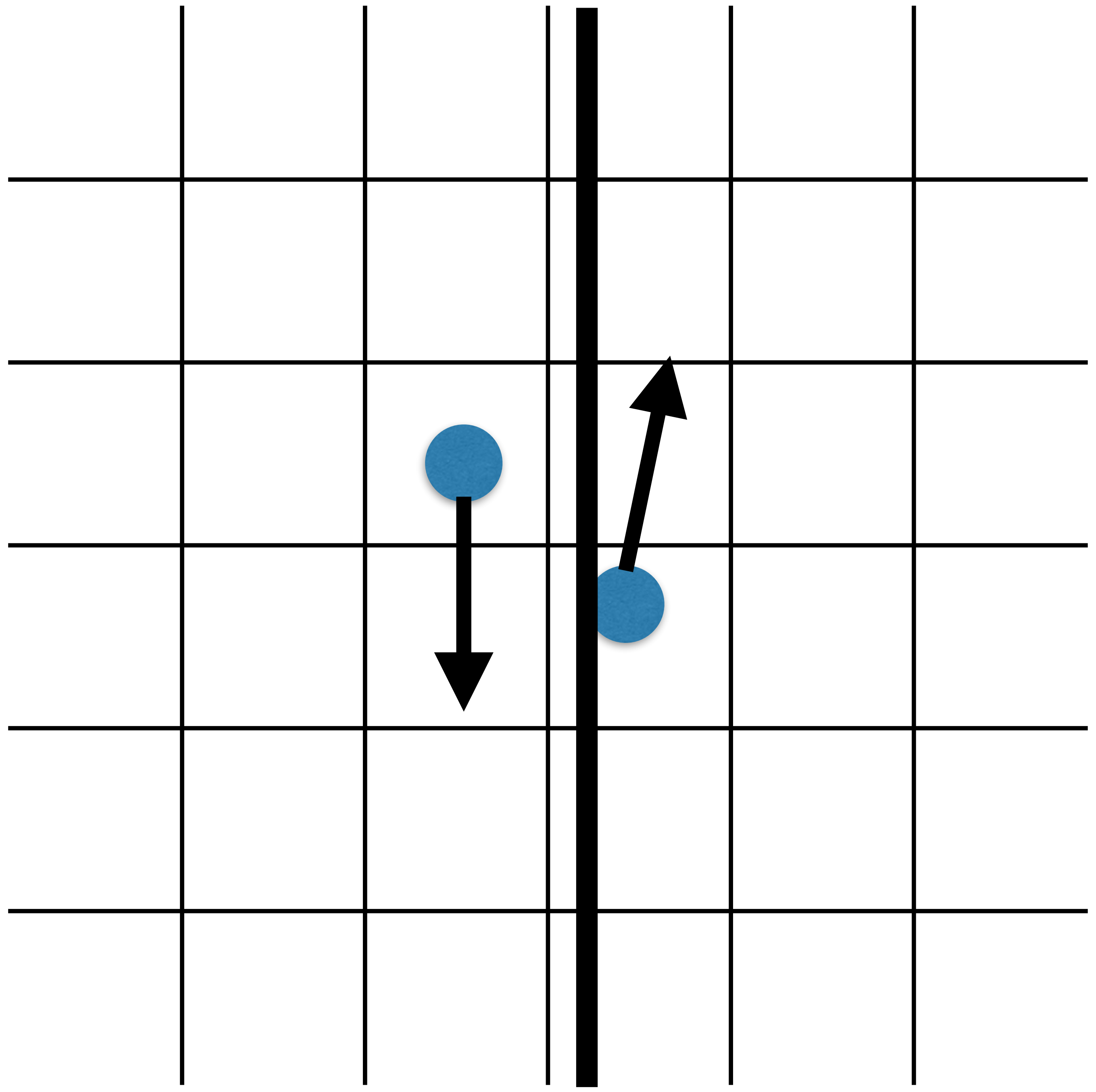
1D Curve Fitting



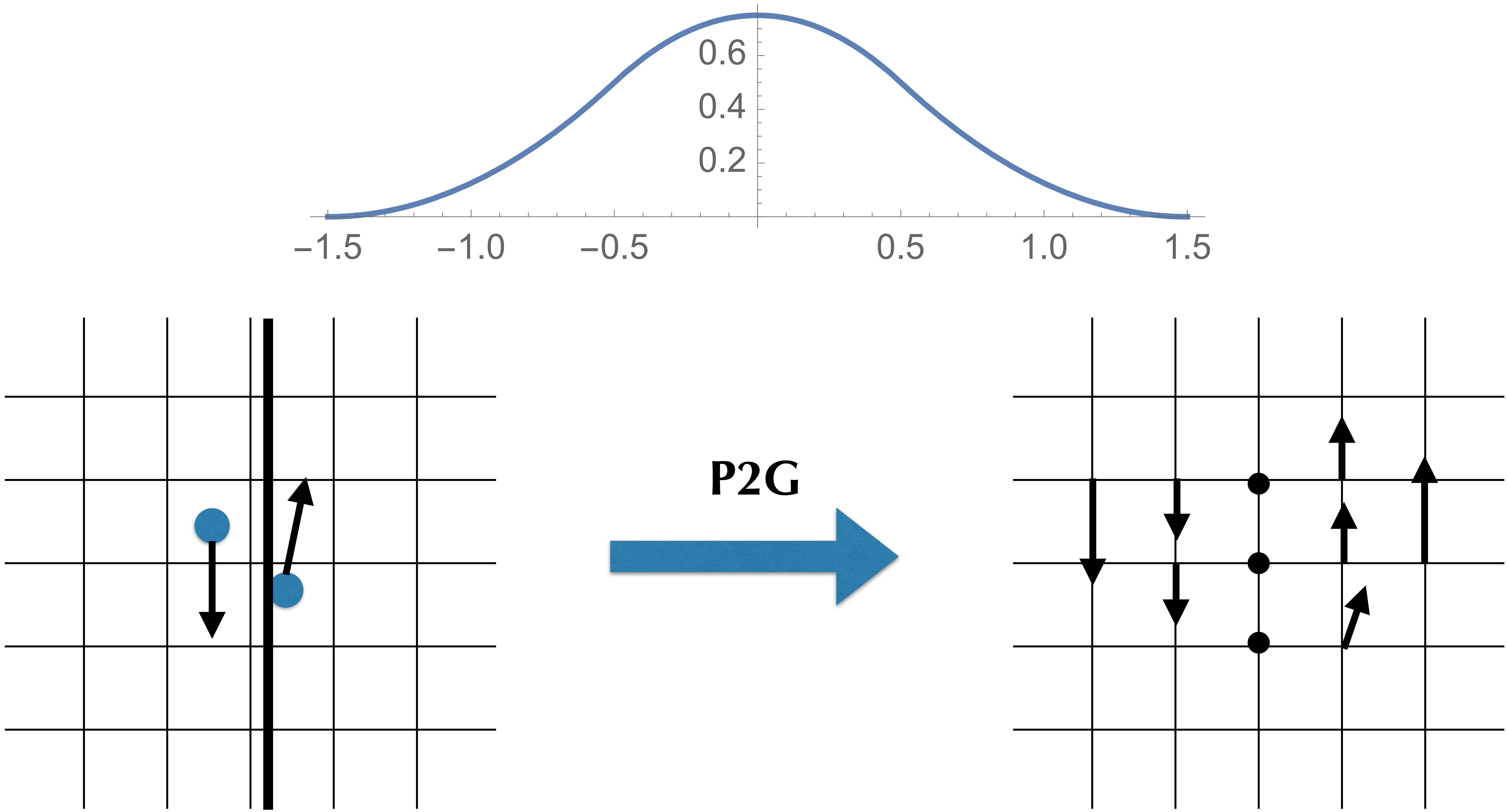

Level Set Cut
Sticky
Slip
Separate
Level Set Appear
Sticky
Slip
Separate
Traditional Method
1dx Remove
1.5dx Remove


Level Set Cut
Sticky
Slip
Separate
Level Set Appear
Sticky
Slip
Separate
Traditional Method
1dx Remove
1.5dx Remove

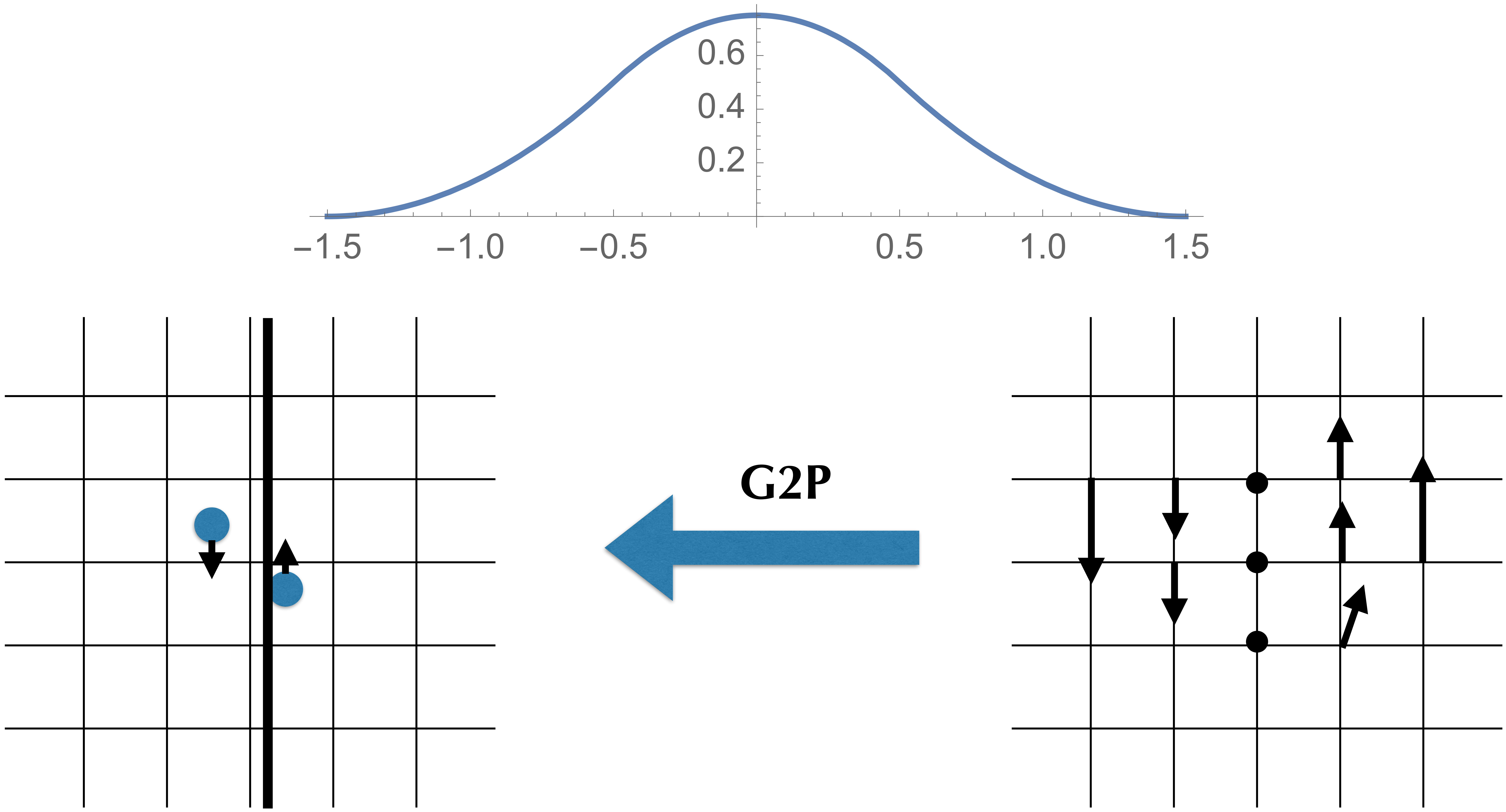




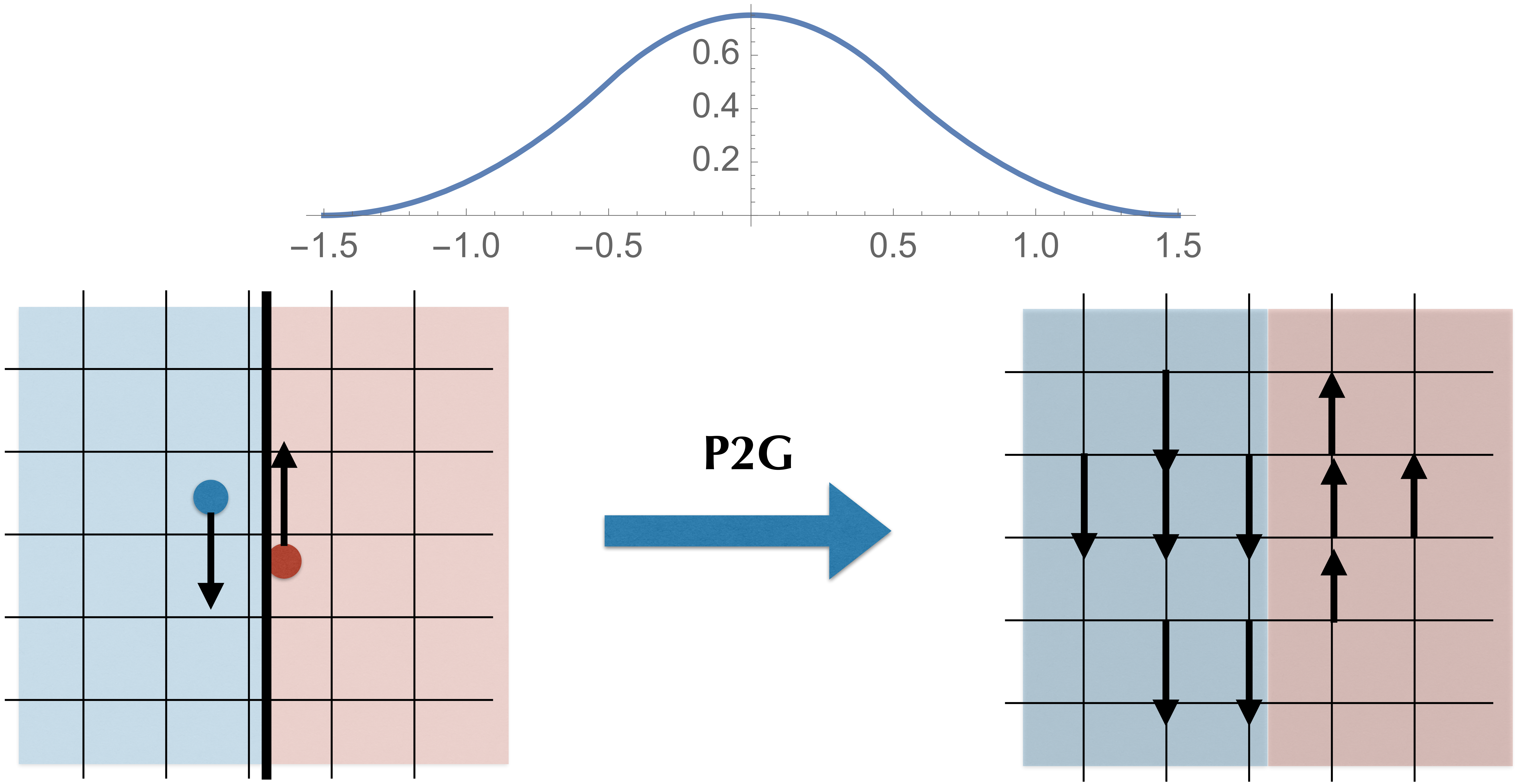
P2G



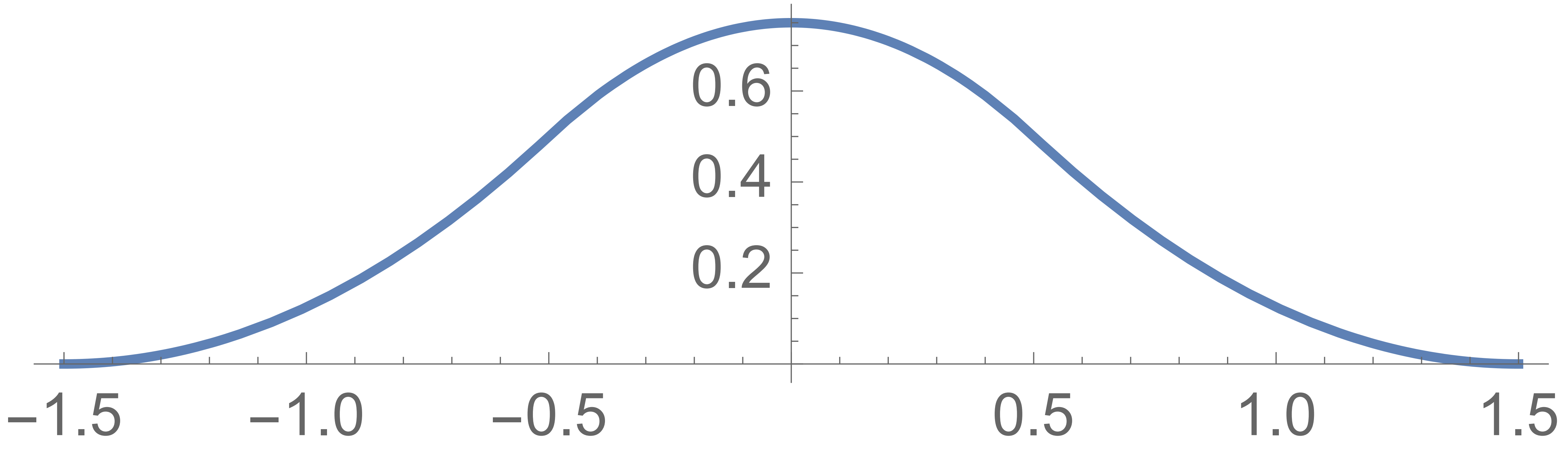
0.6
0.4
0.2
−1.5
−1.0
−0.5
0.5
1.0
1.5
P2G

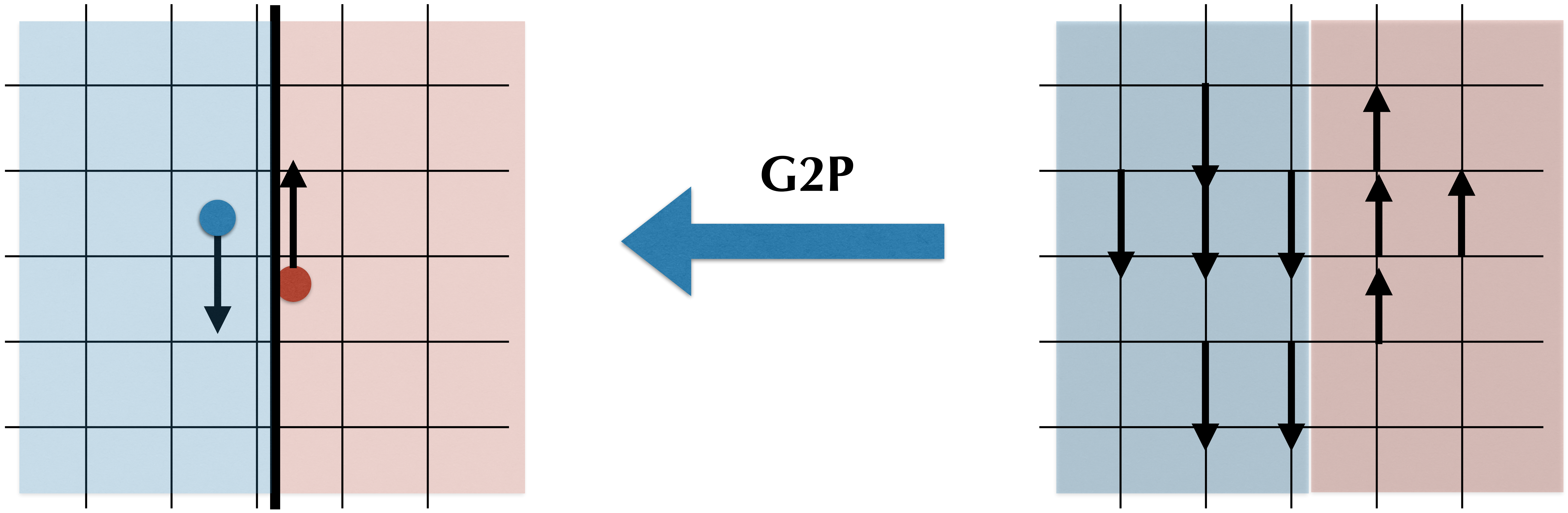

0.6
0.4
0.2
−1.5
−1.0
−0.5
0.5
1.0
1.5
G2P

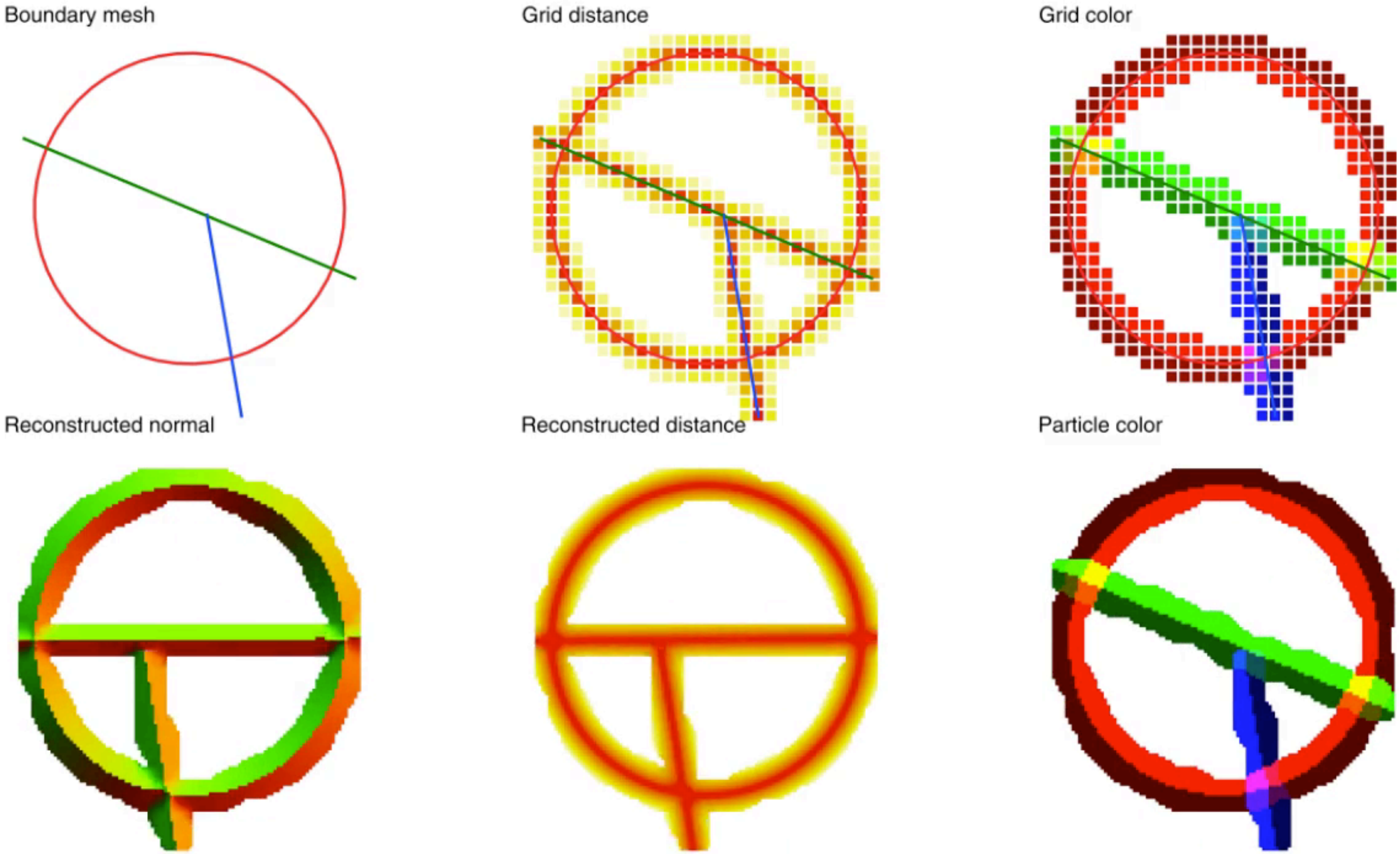
Velocity Discontinuity (Compatible Particle-in-Cell, CPIC)

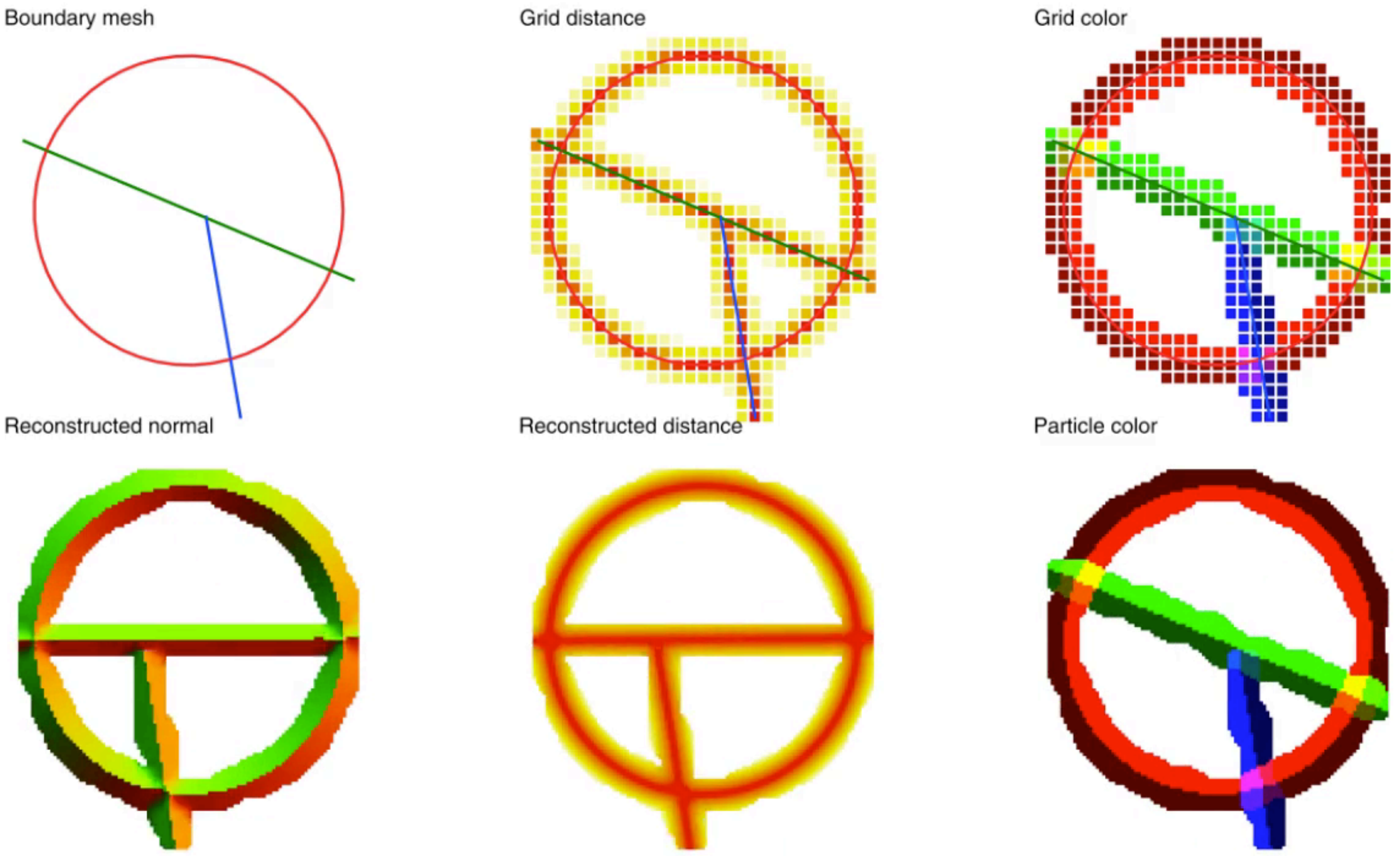


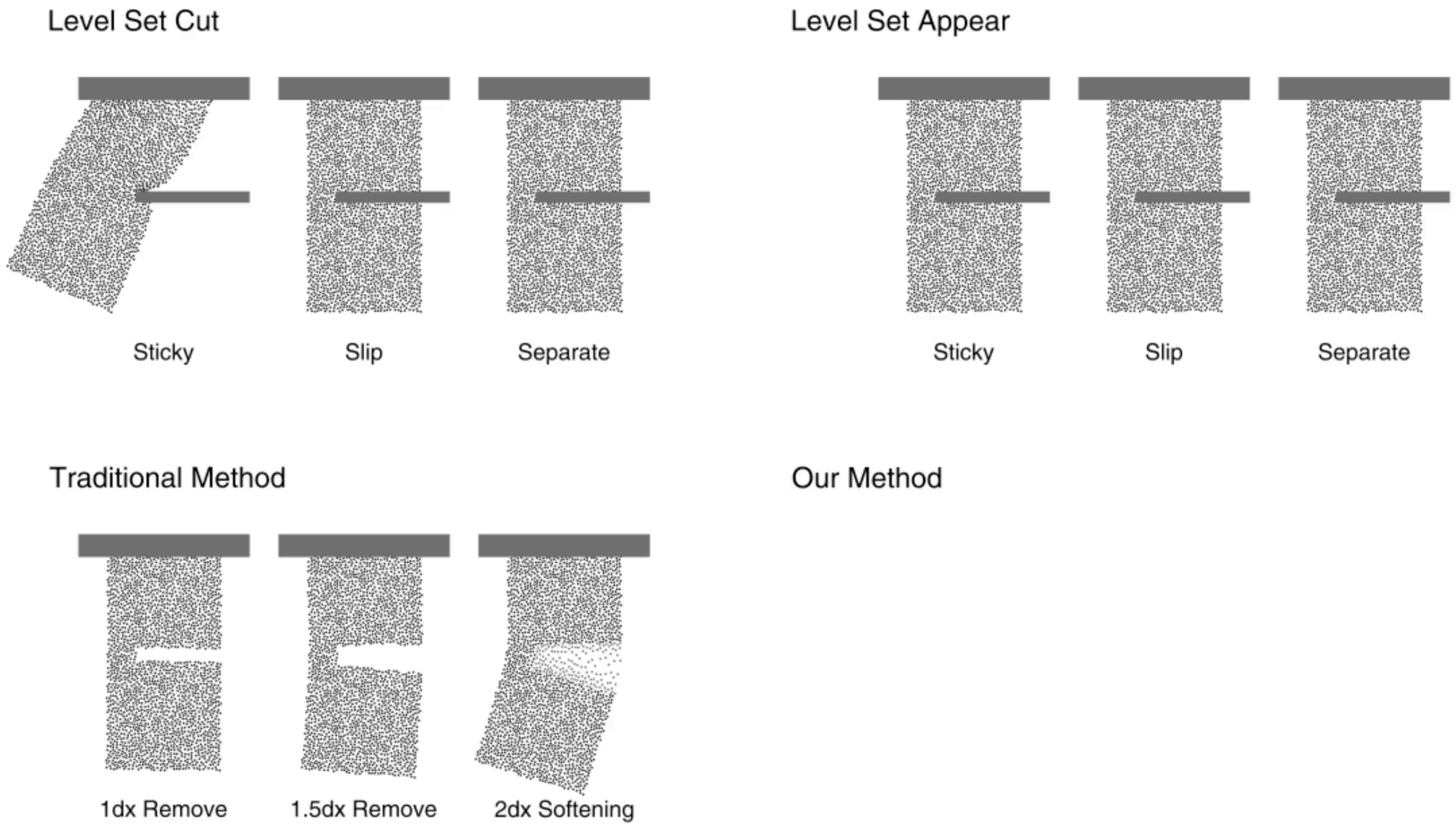
Velocity Discontinuity (Compatible Particle-in-Cell, CPIC)

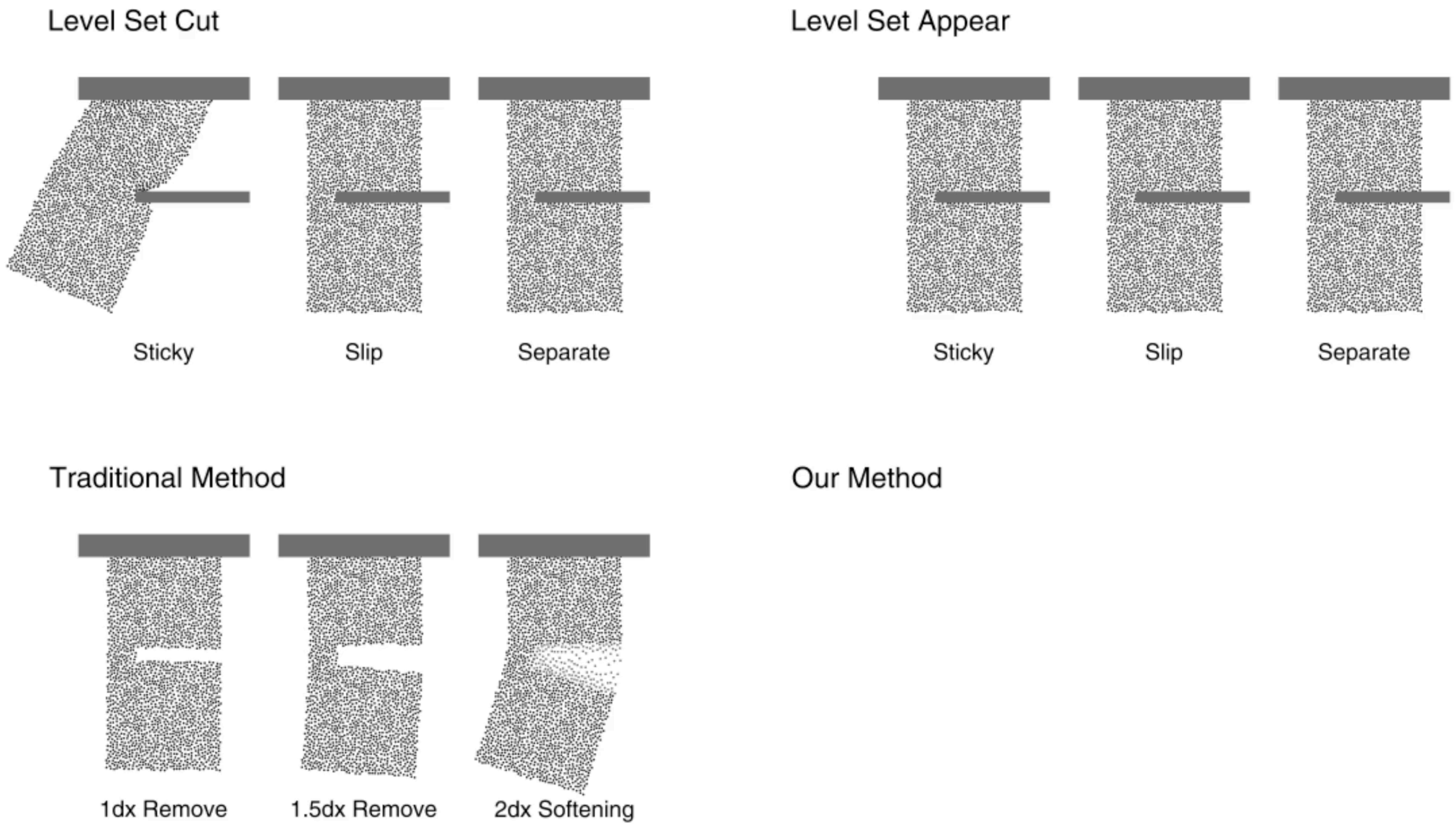


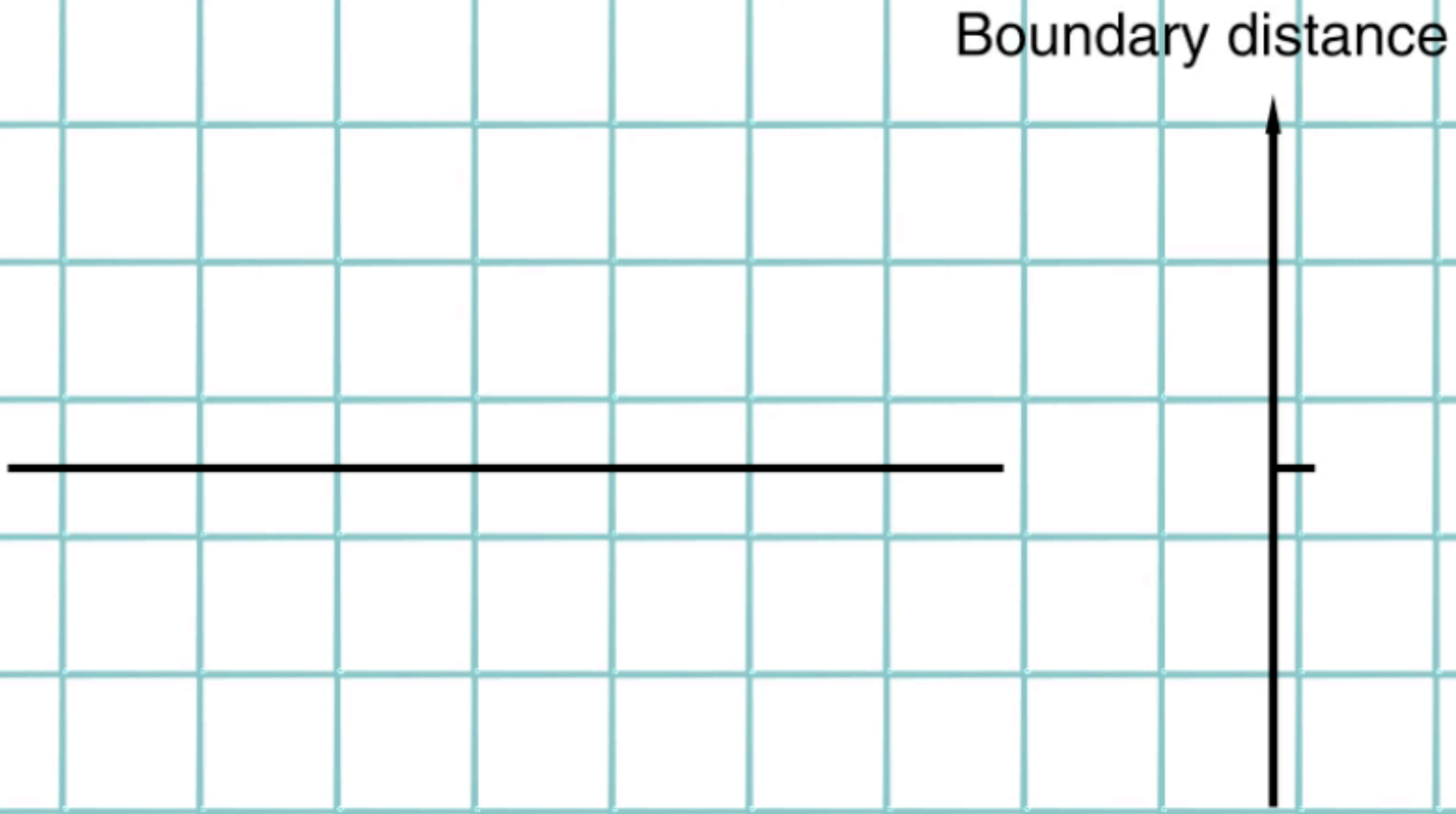


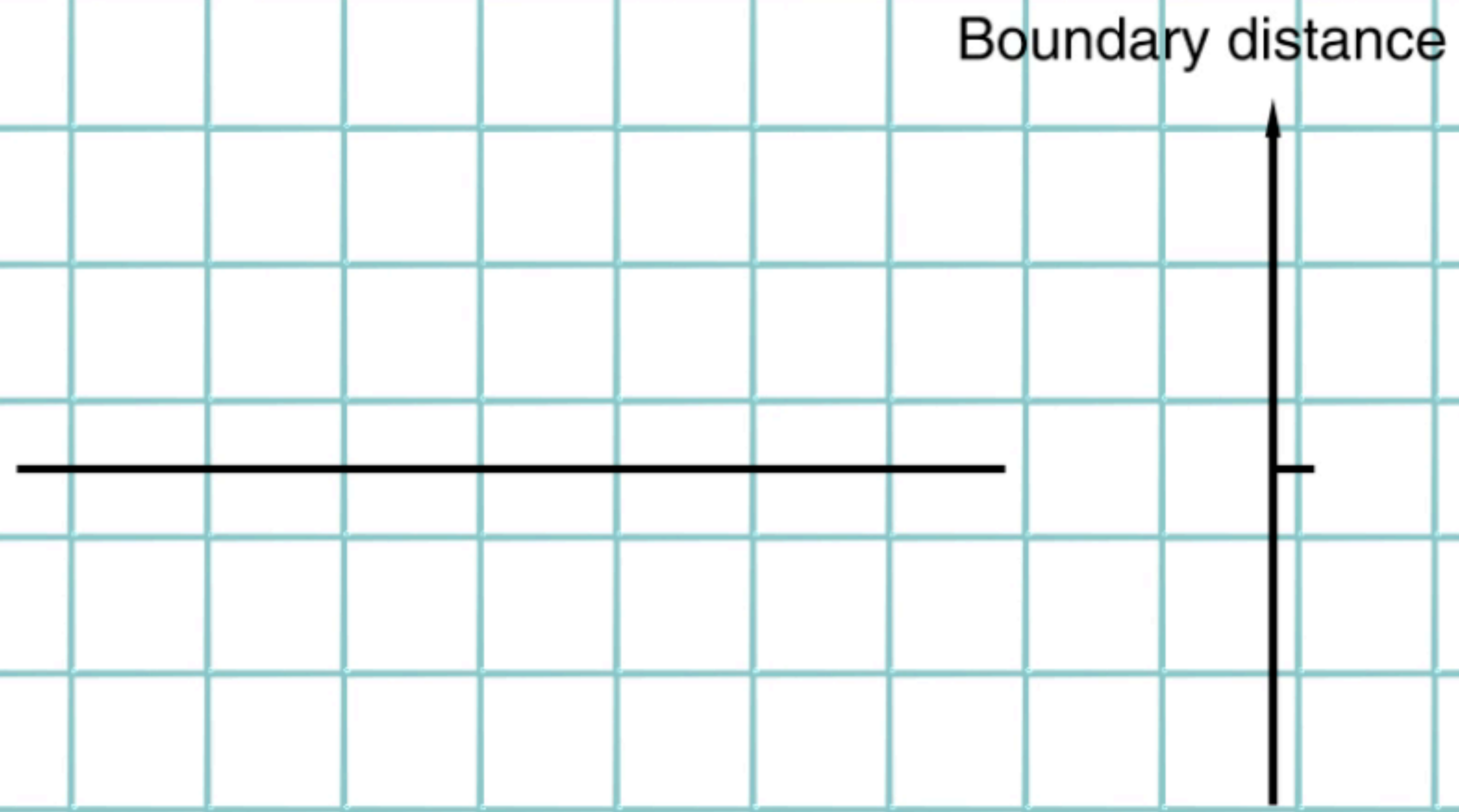

Boundary mesh
Grid distance
Grid color
Reconstructed normal
Reconstructed distance
Particle color


Boundary mesh
Grid distance
Grid color
Reconstructed normal
Reconstructed distance
Particle color


Level Set Cut
Sticky
Slip
Separate
Level Set Appear
Sticky
Slip
Separate
Traditional Method
1dx Remove
1.5dx Remove
2dx Softening
Our Method

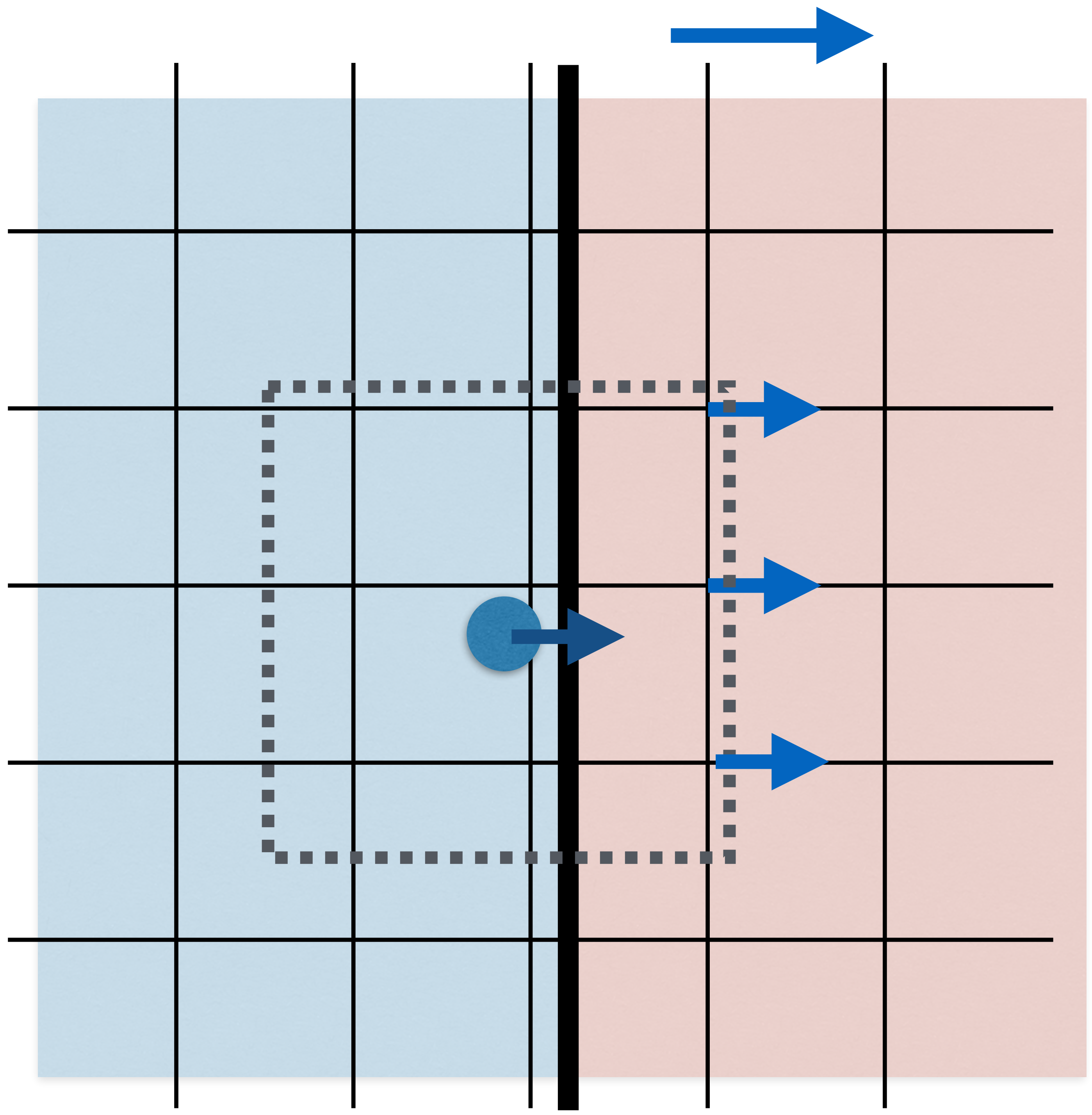

Level Set Cut
Sticky
Slip
Separate
Level Set Appear
Sticky
Slip
Separate
Traditional Method
1dx Remove
1.5dx Remove
2dx Softening
Our Method


Boundary distance

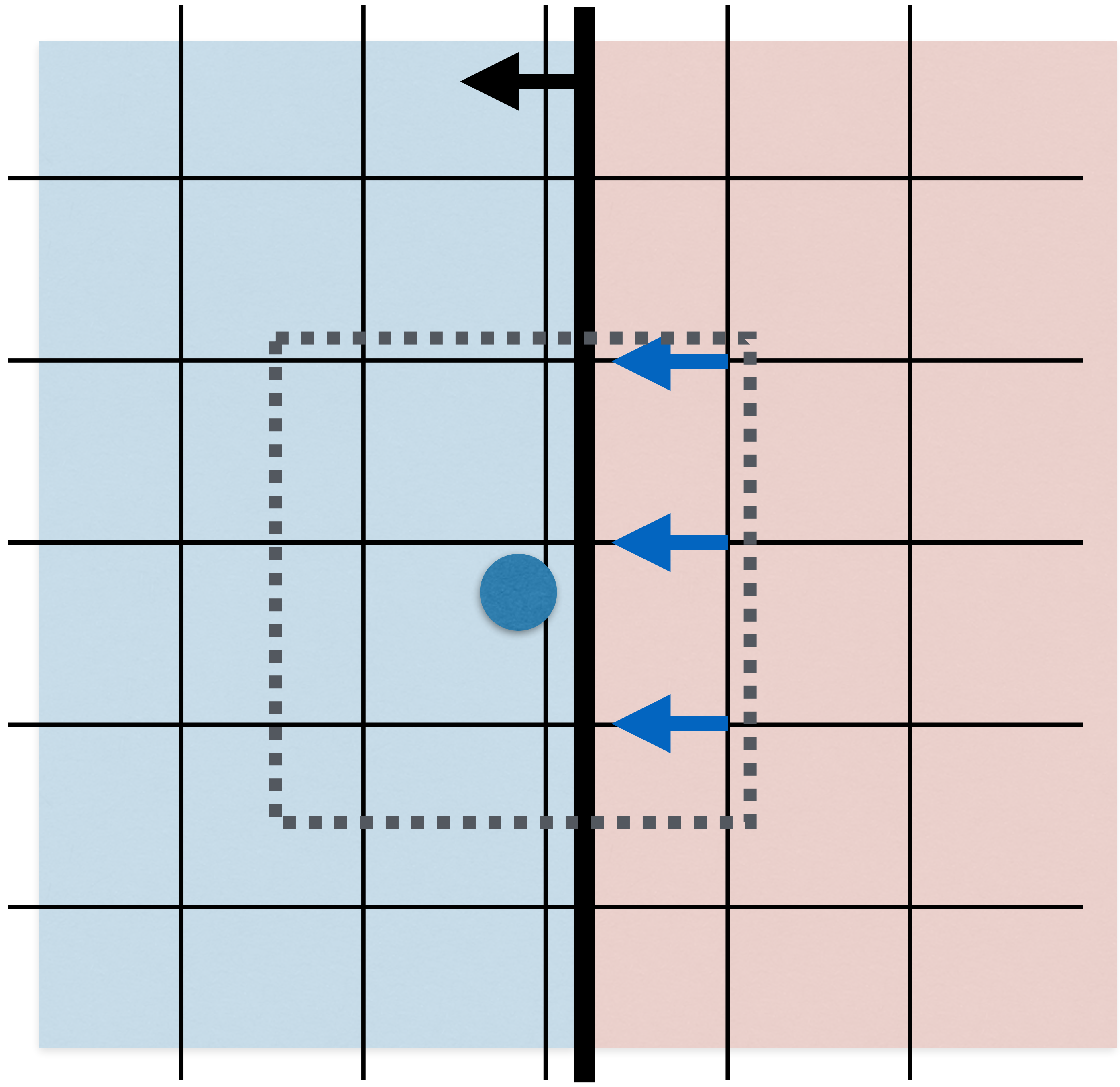

Boundary distance

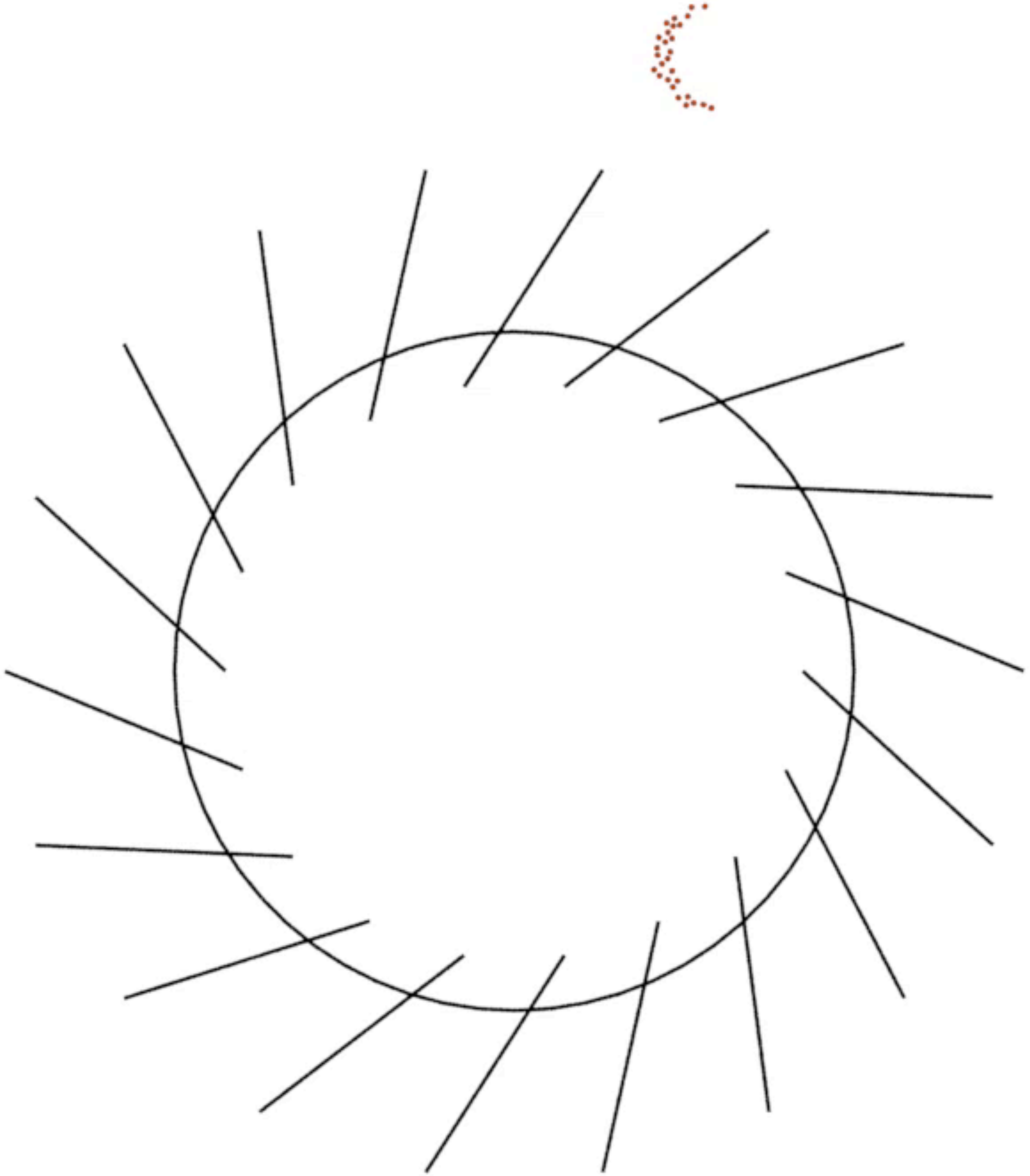
Two-way Rigid Body Coupling

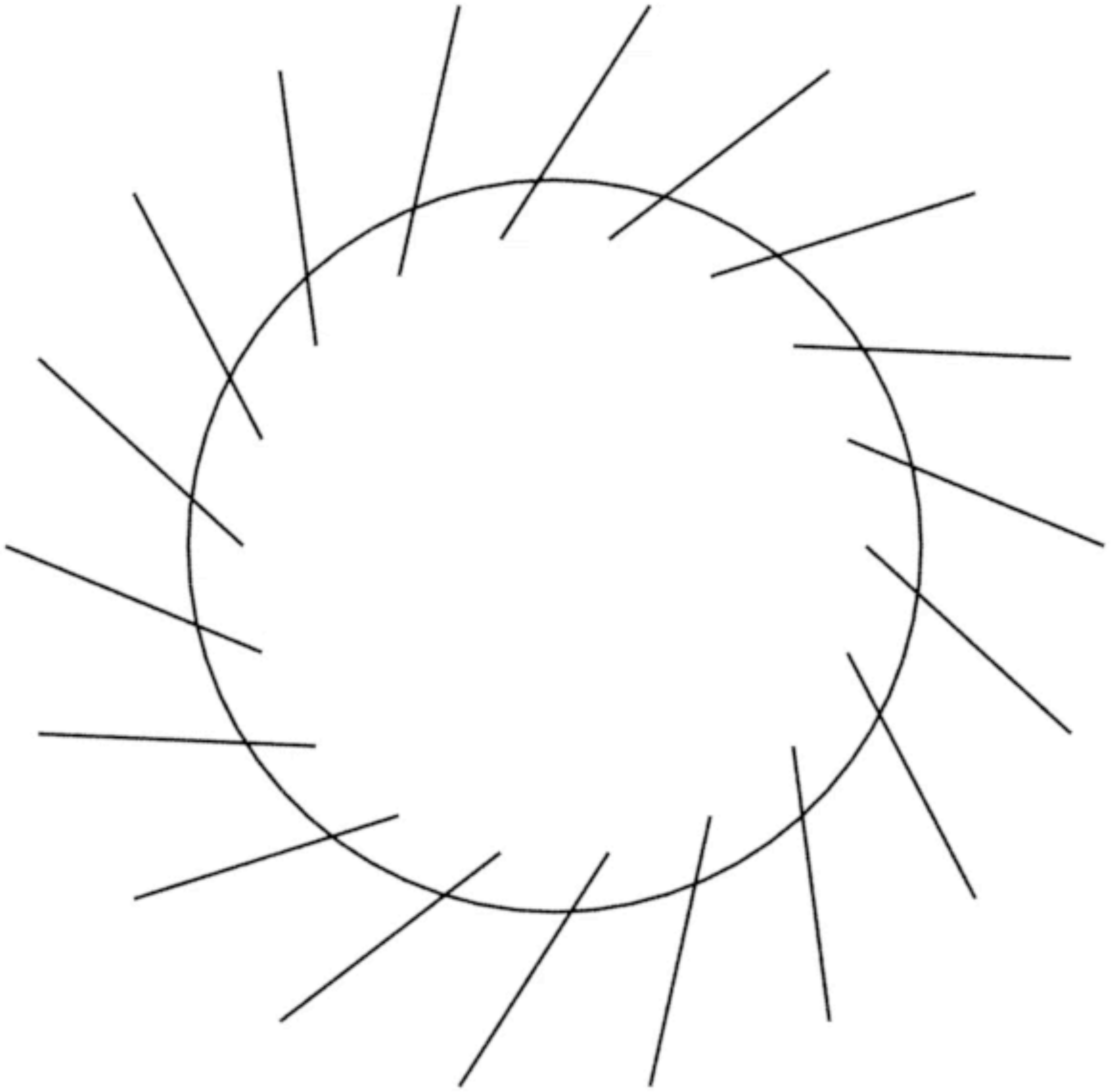
Particle to rigid body (P2G)

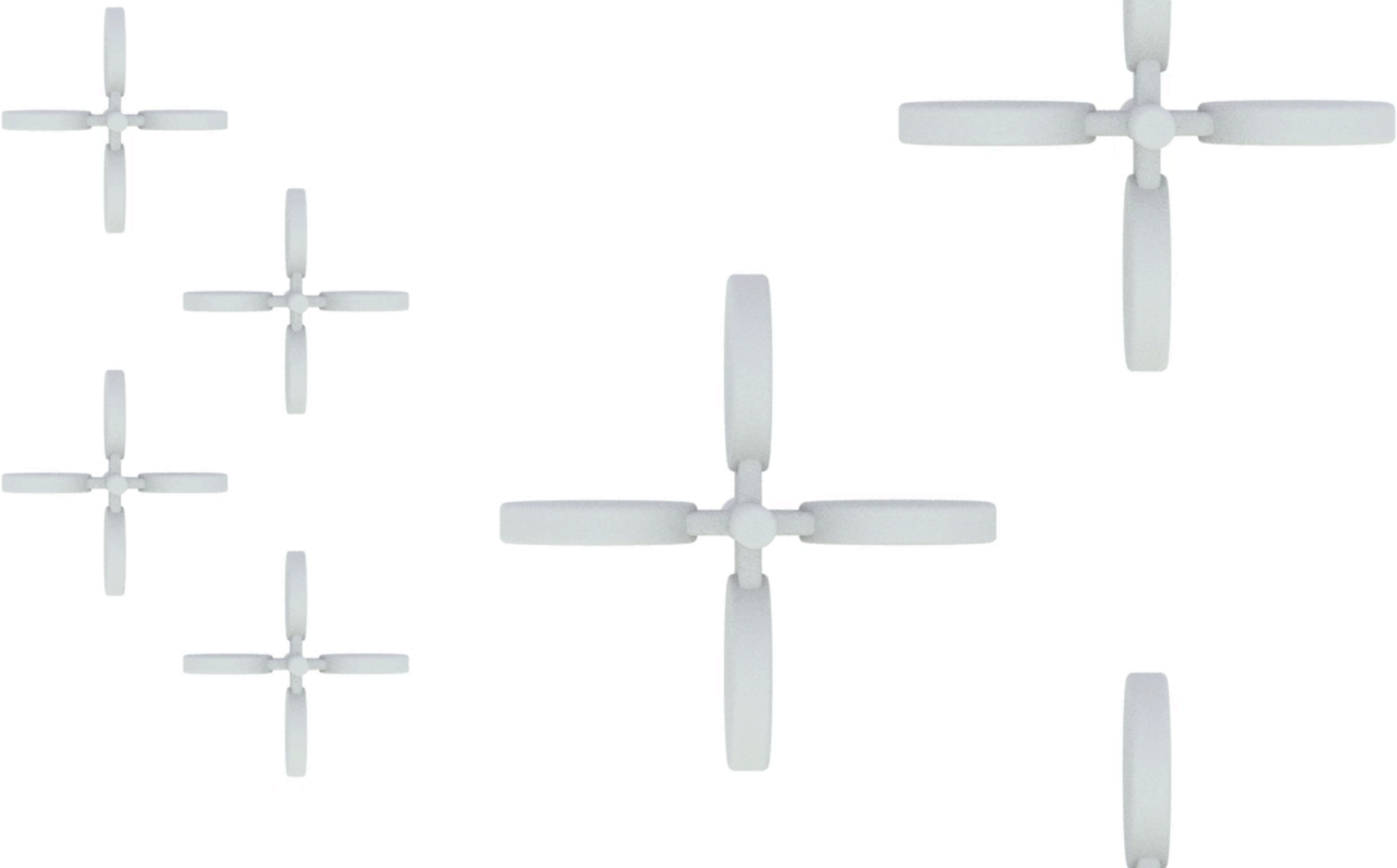


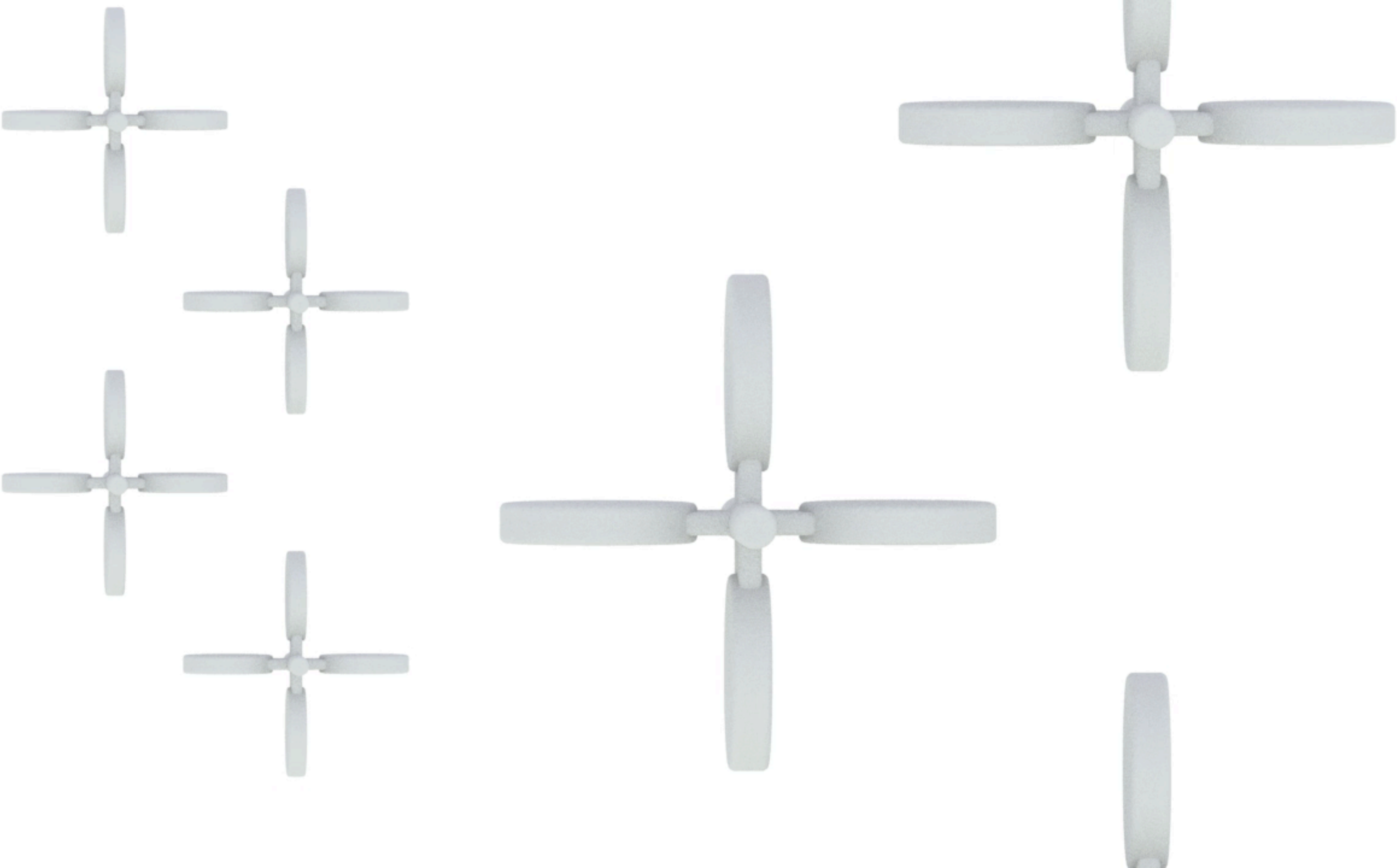
Rigid body to particle (G2P)







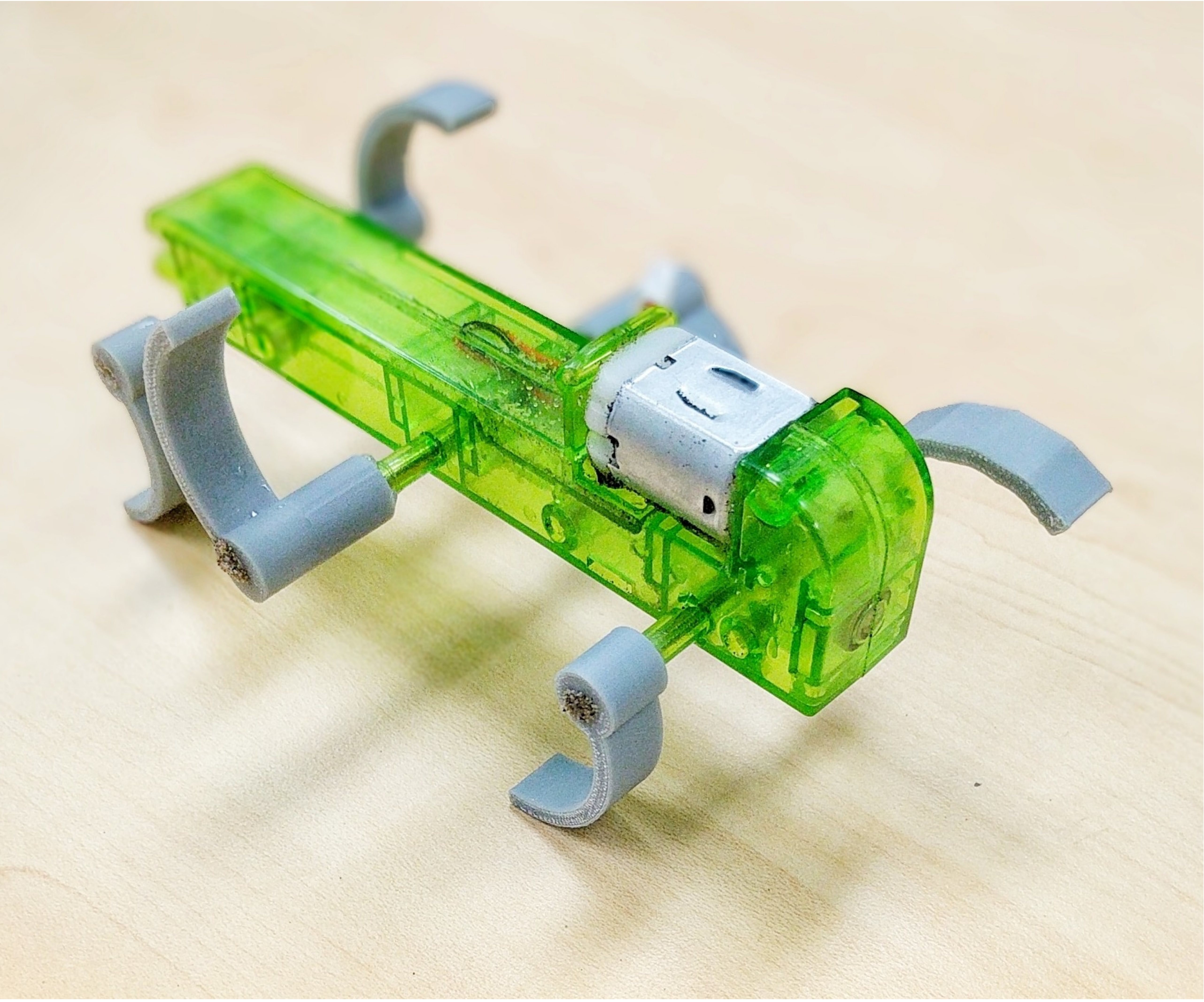


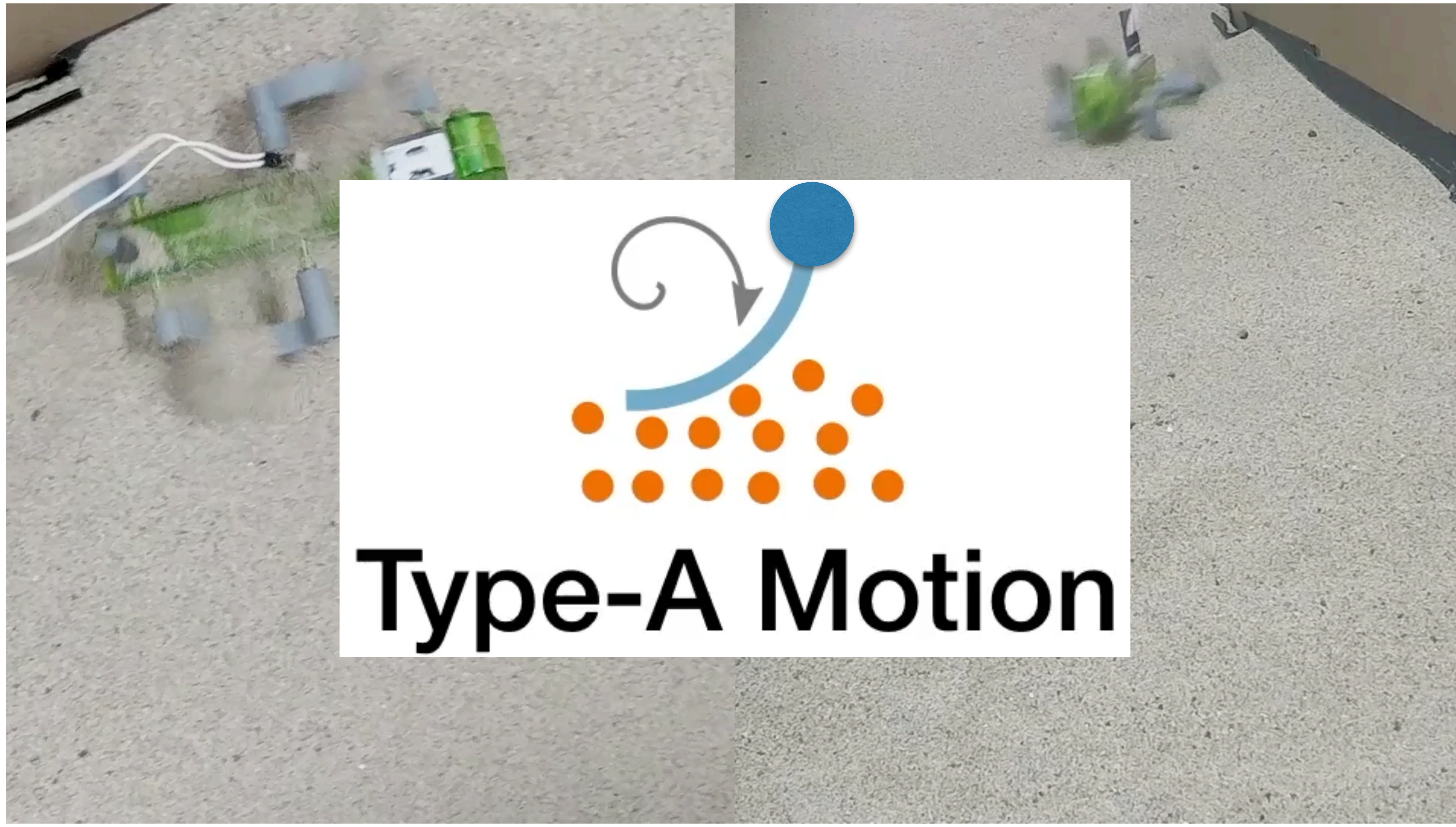


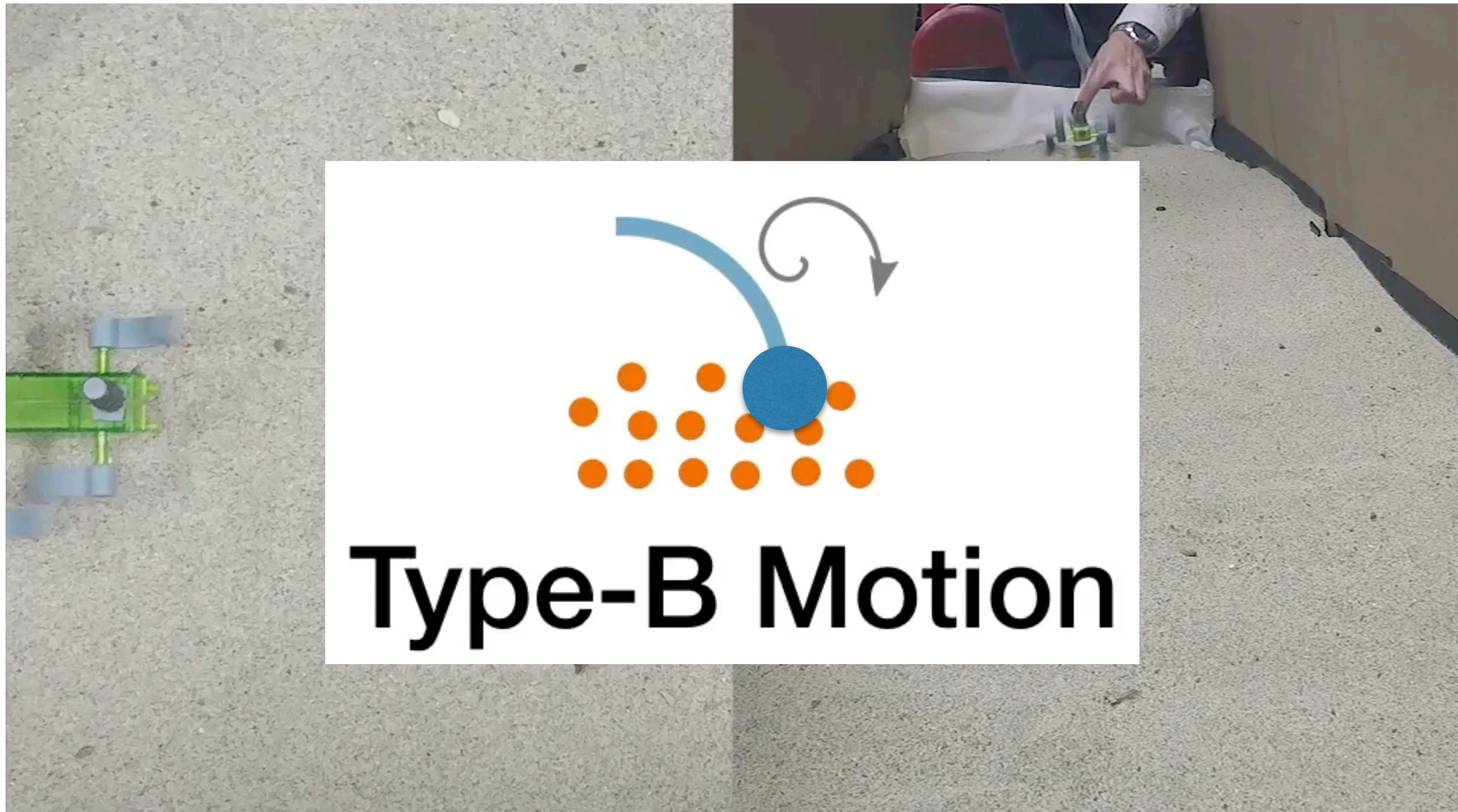

Inflow speed: 0.5
Wheel density: 1.0
Inflow speed: 1.0
Wheel density: 4.0
Inflow speed: 0.5
Wheel density: 4.0


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

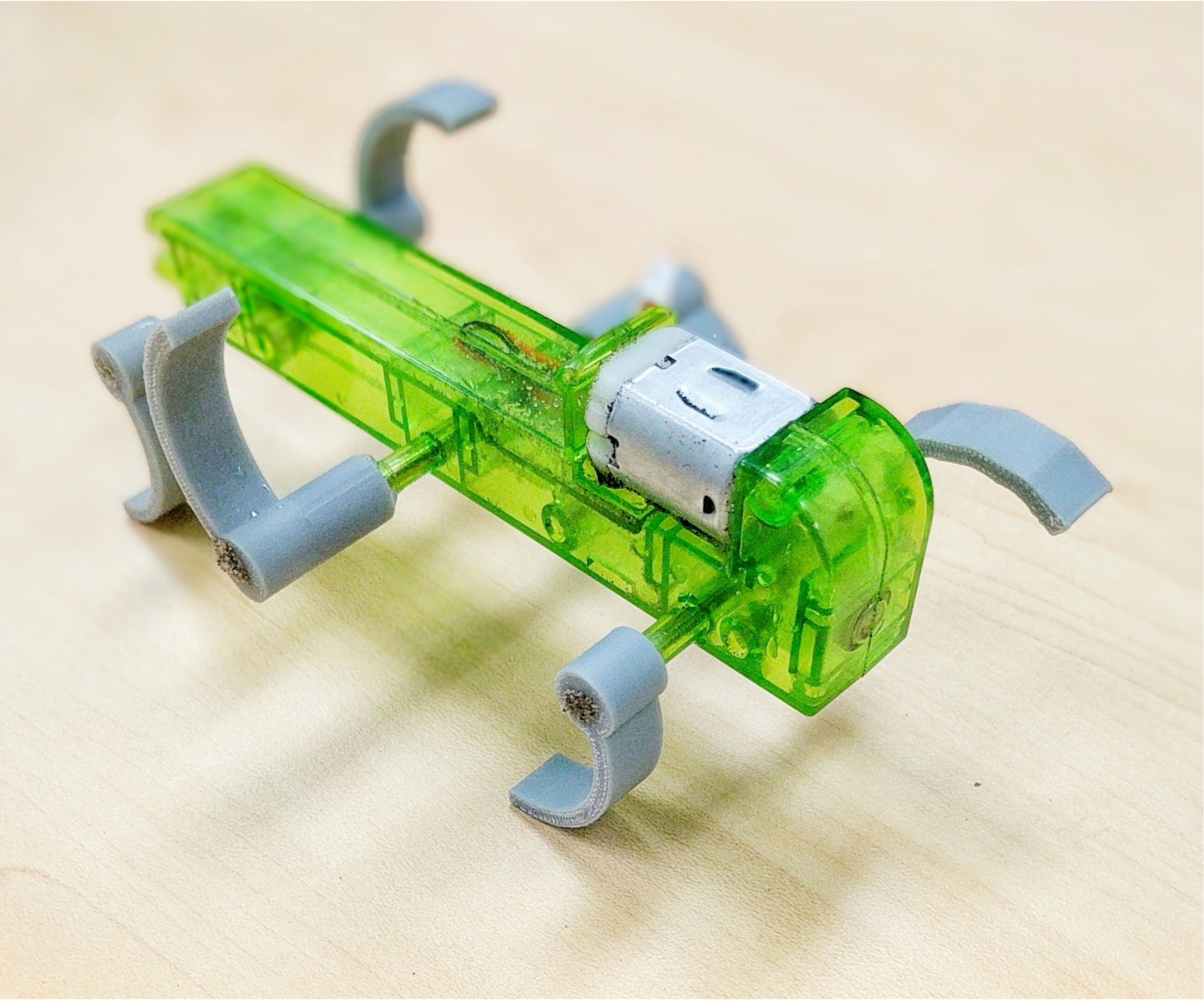
Terradynamics:
Robot and granular media



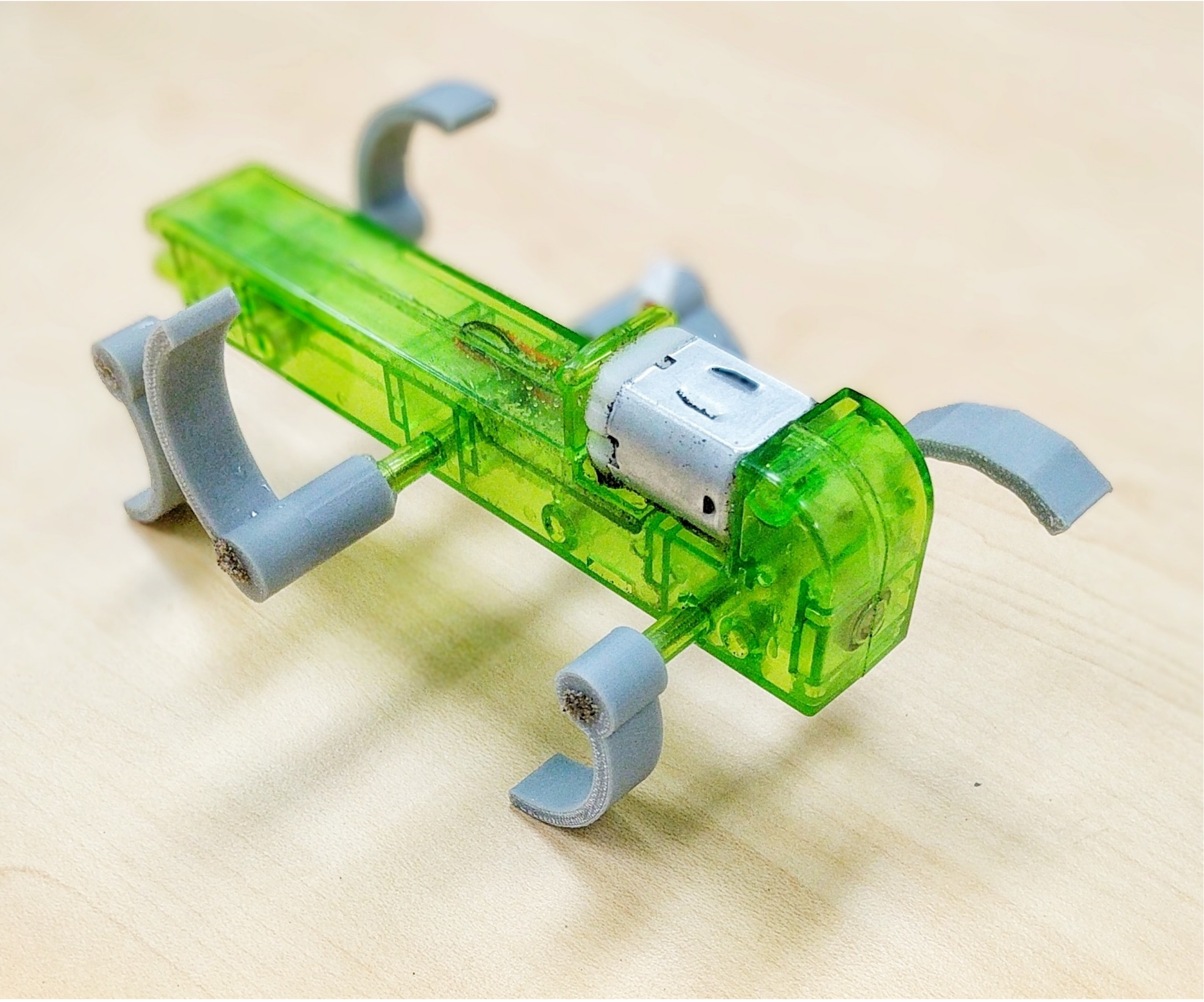


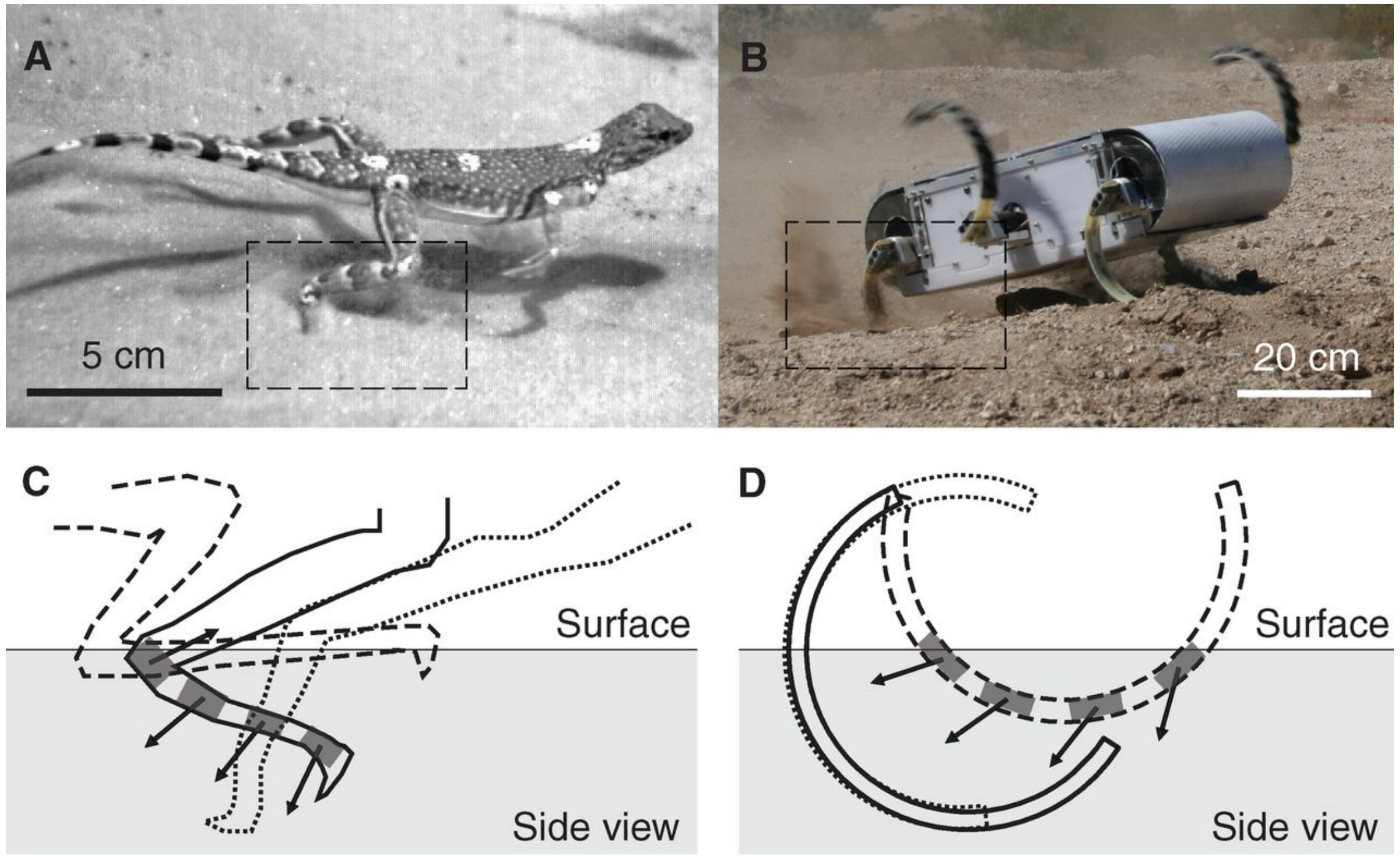


Terradynamics:
Robot and granular media



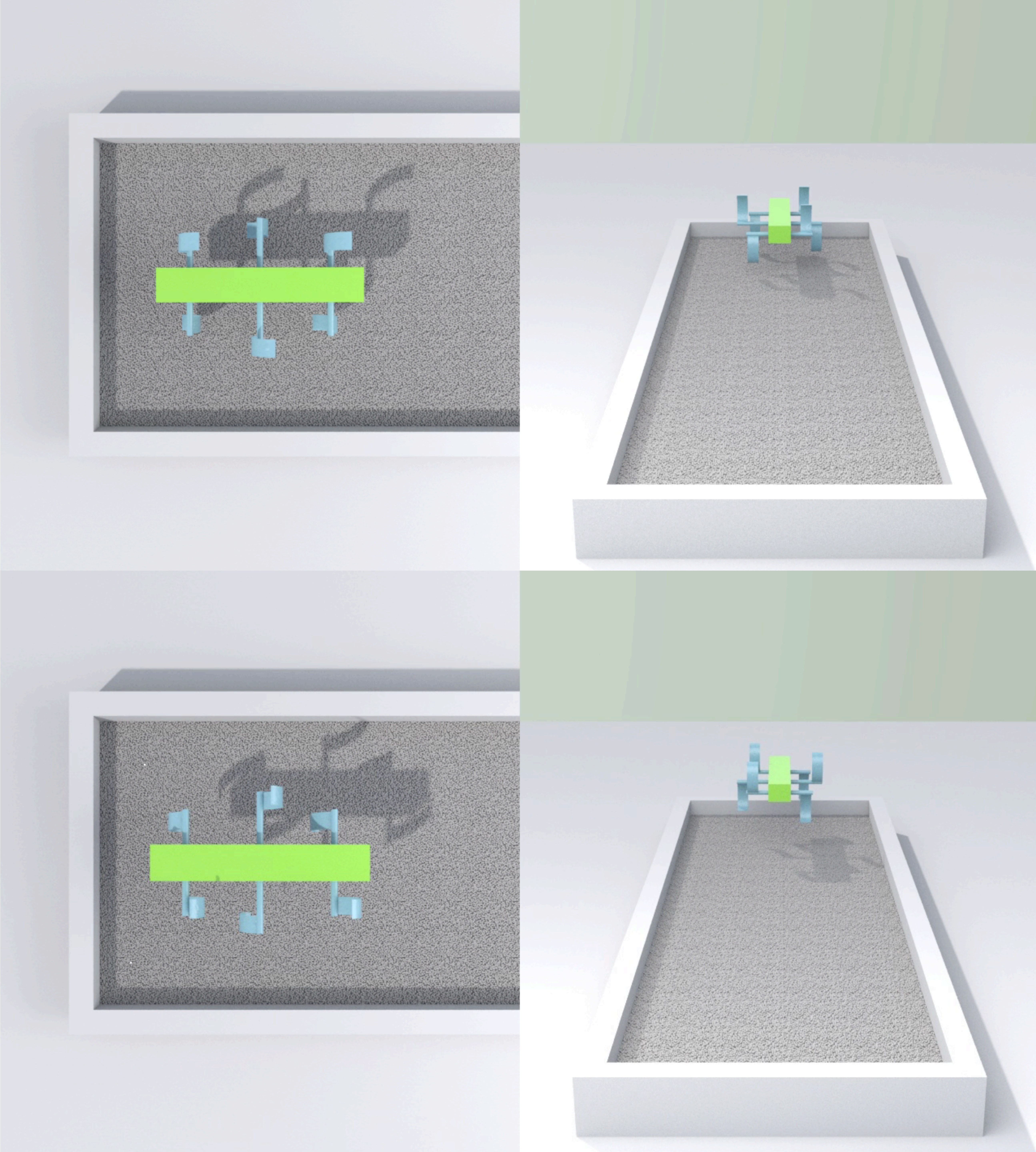
Terradynamics:
Robot and granular media

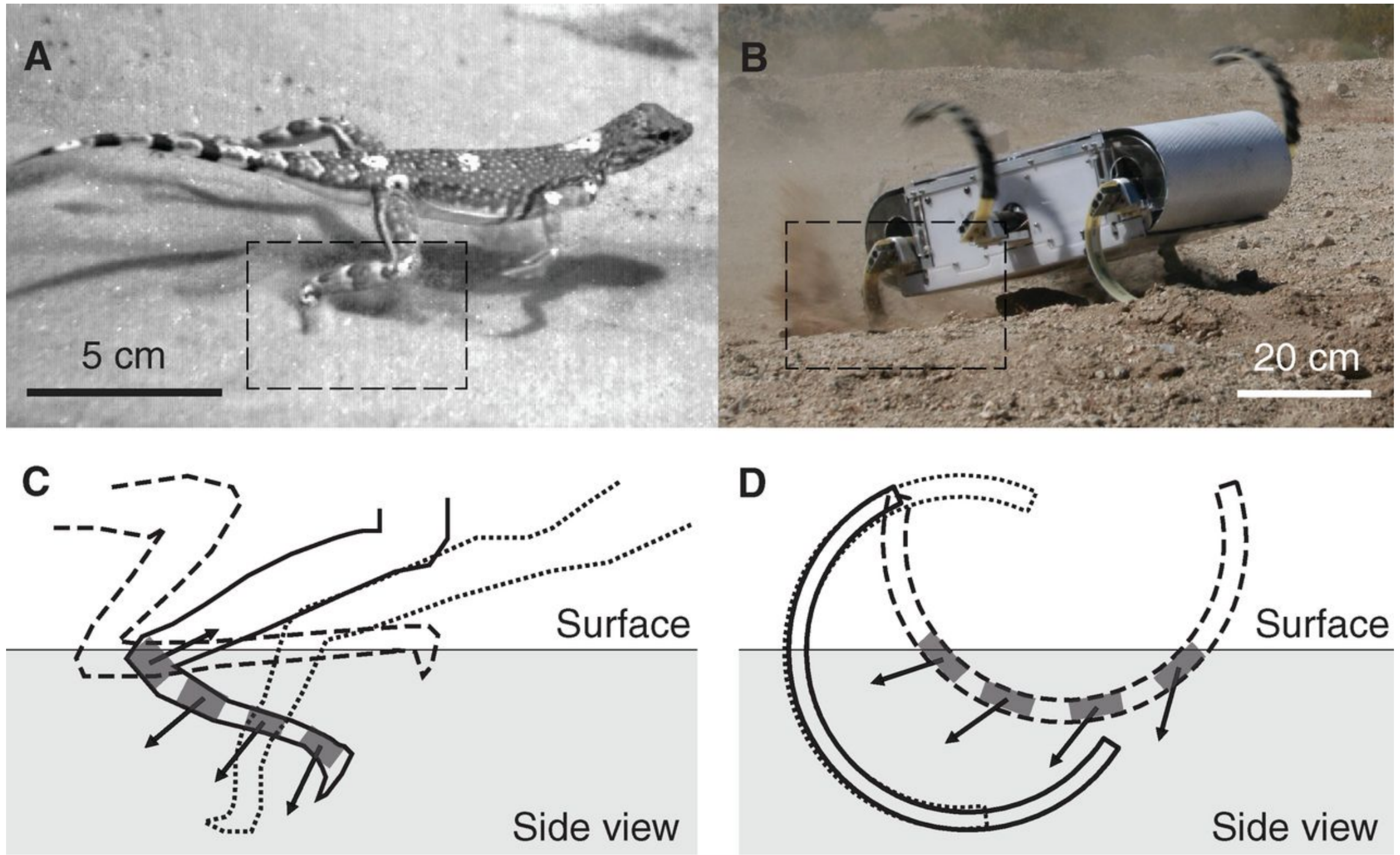




[Li et al., A terradynamics of legged locomotion on granular media. **Science 2013**]

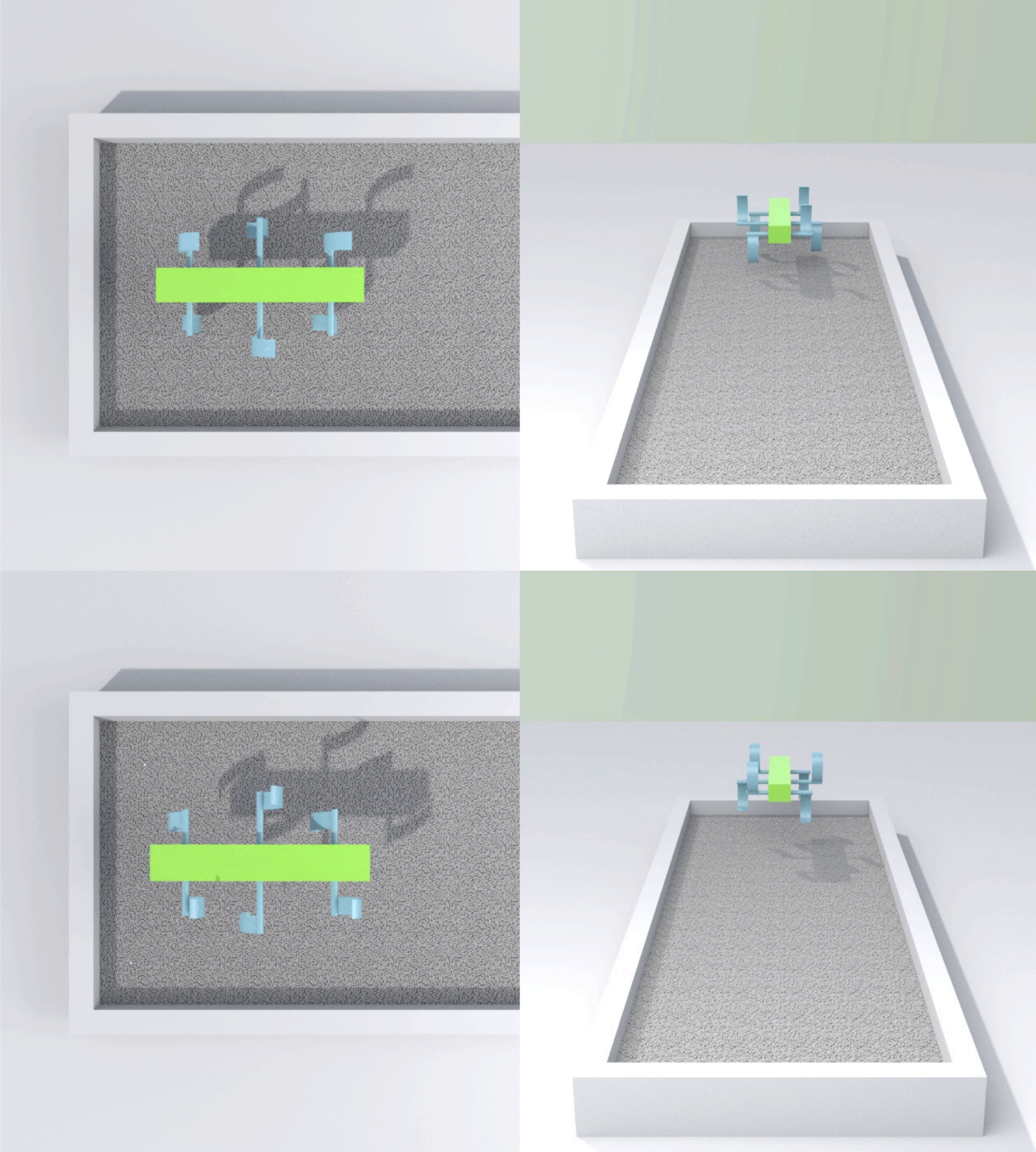
This direction should be faster →





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

This direction should 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

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

Reproducible every demo with a python script:

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

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***

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!

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