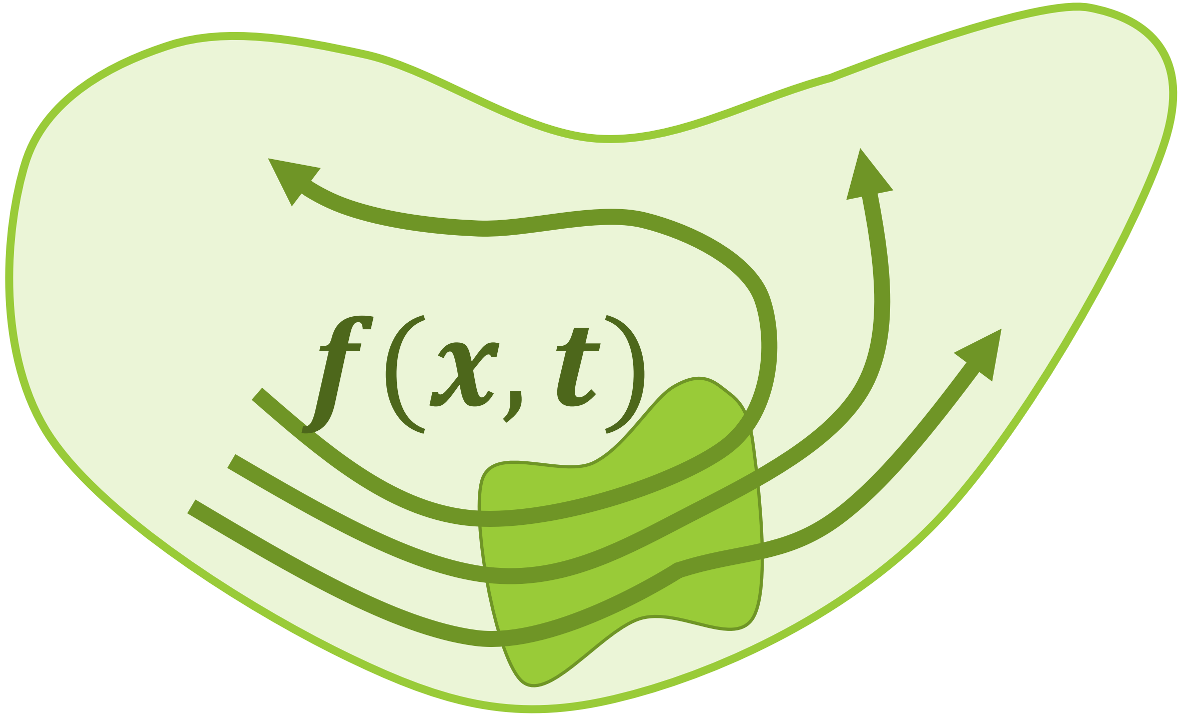
The Particle-In-Cell Method For Simulating Incompressible Fluids

CS 888 Final Project

Jonathan Panuelos

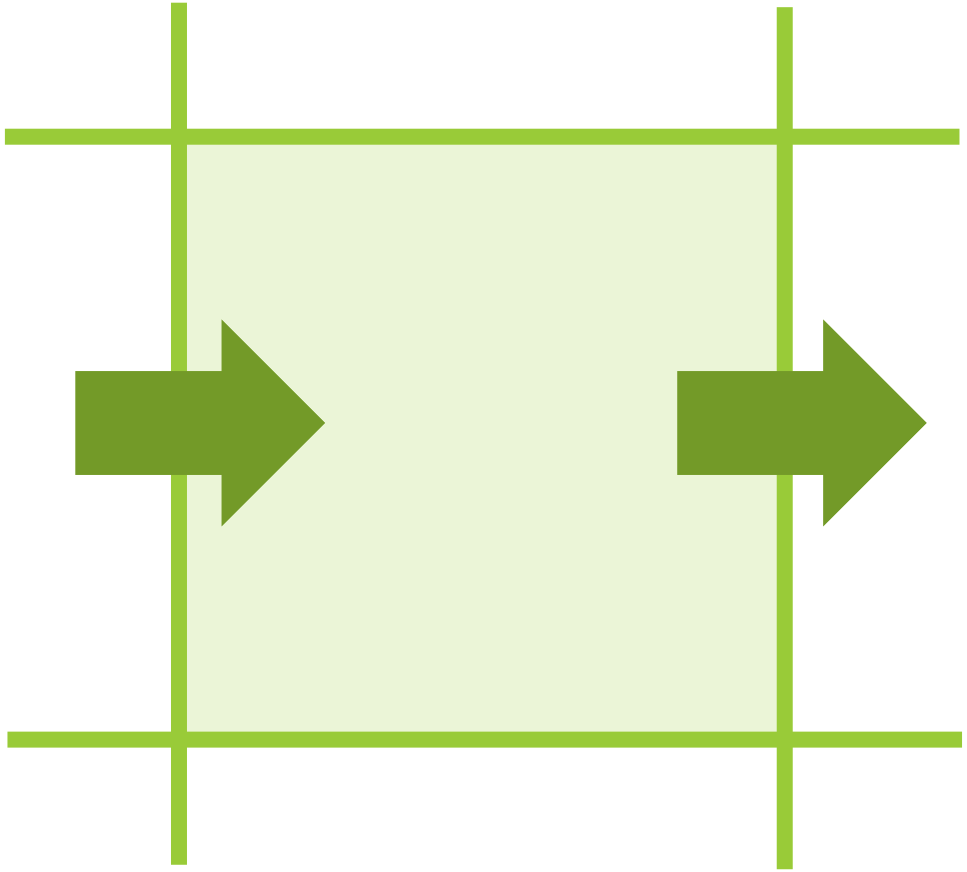
Motivation

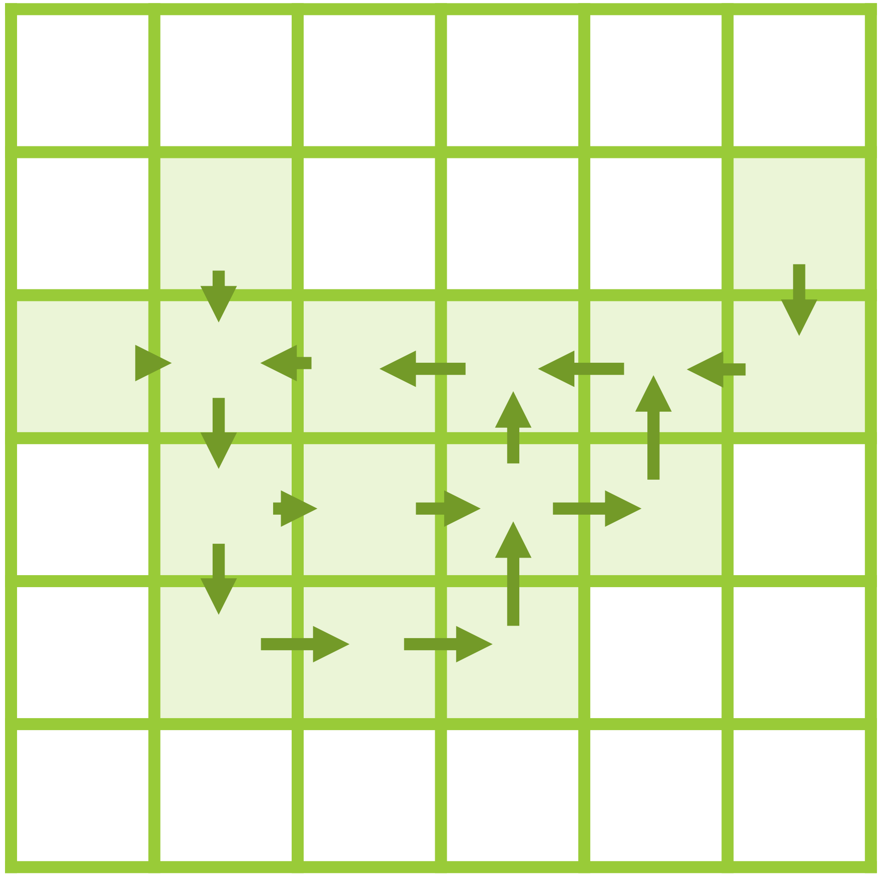


We want to simulate natural phenomena realistically in graphics

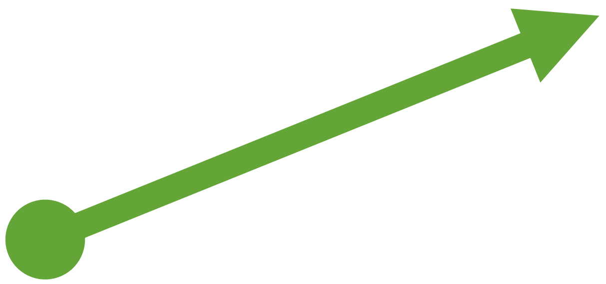
Can use conservation laws and enforce constraints such as incompressibility

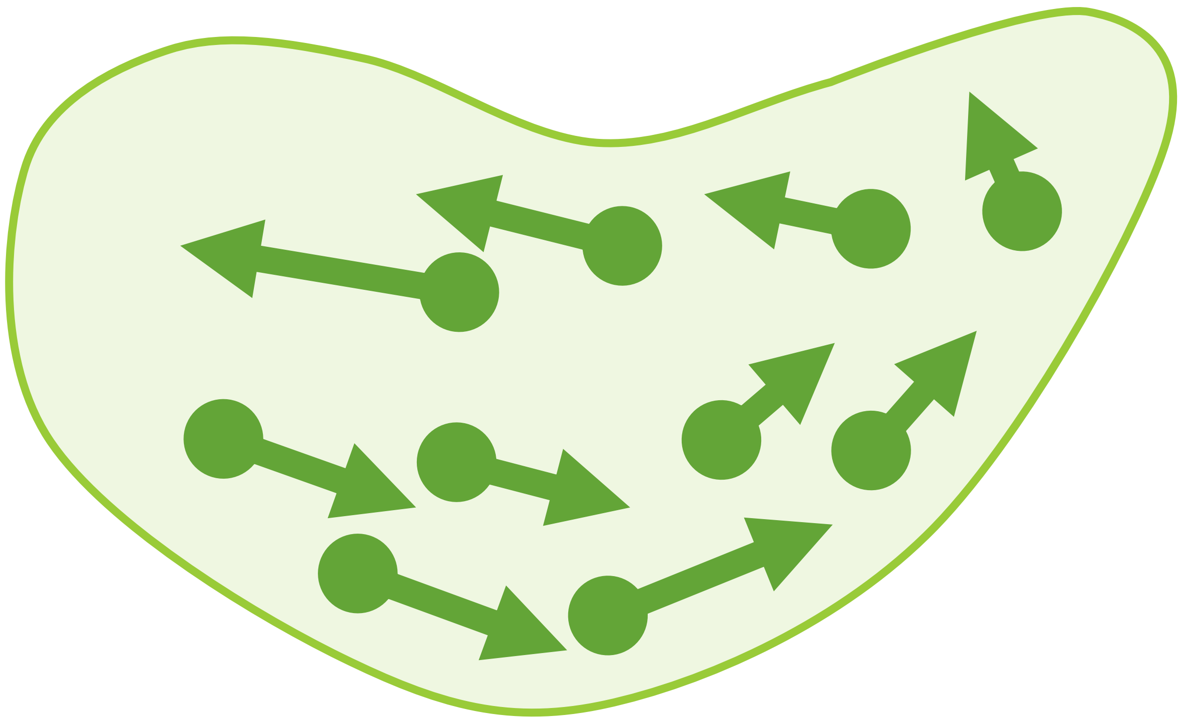
Eulerian Simulation





Lagrangian Simulation





Eulerian vs Lagrangian

Eulerian

Speed

Definite Neighbours

Easy FD Approximations

Lagrangian

Easy Advection

Free Momentum Conservation

The Particle-In-Cell Method

Advect fluid using particles

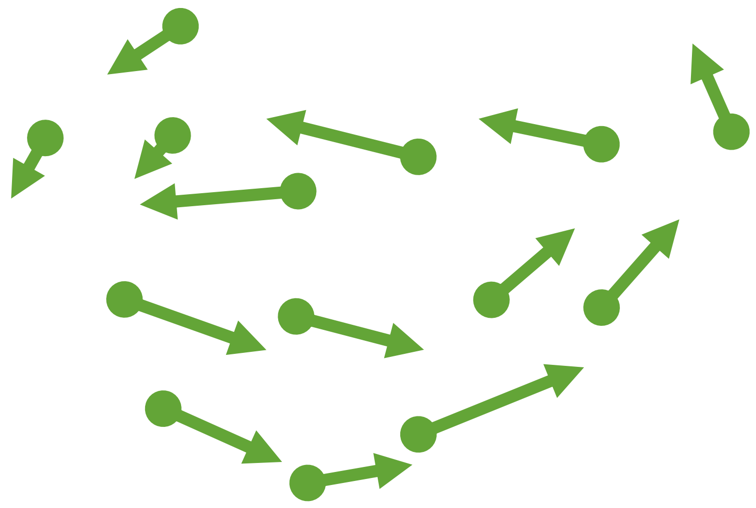
Use grid to evolve velocity

- Enforcing divergence-free velocities

My implementation largely based on *Animating Sand as a Fluid* (Zhu & Bridson 2005), *Fluid Simulation for Computer Graphics* (Bridson Textbook)

The Particle-In-Cell Method

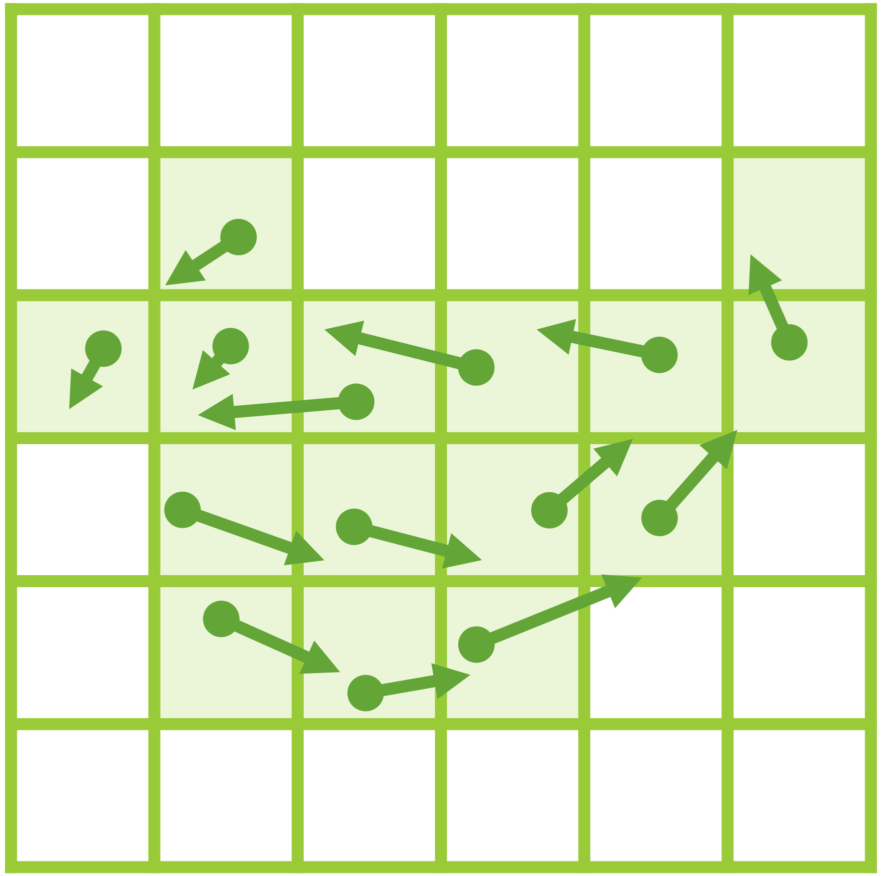
Start with particles holding velocities



The Particle-In-Cell Method

Start with particles holding velocities

Overlay a background grid

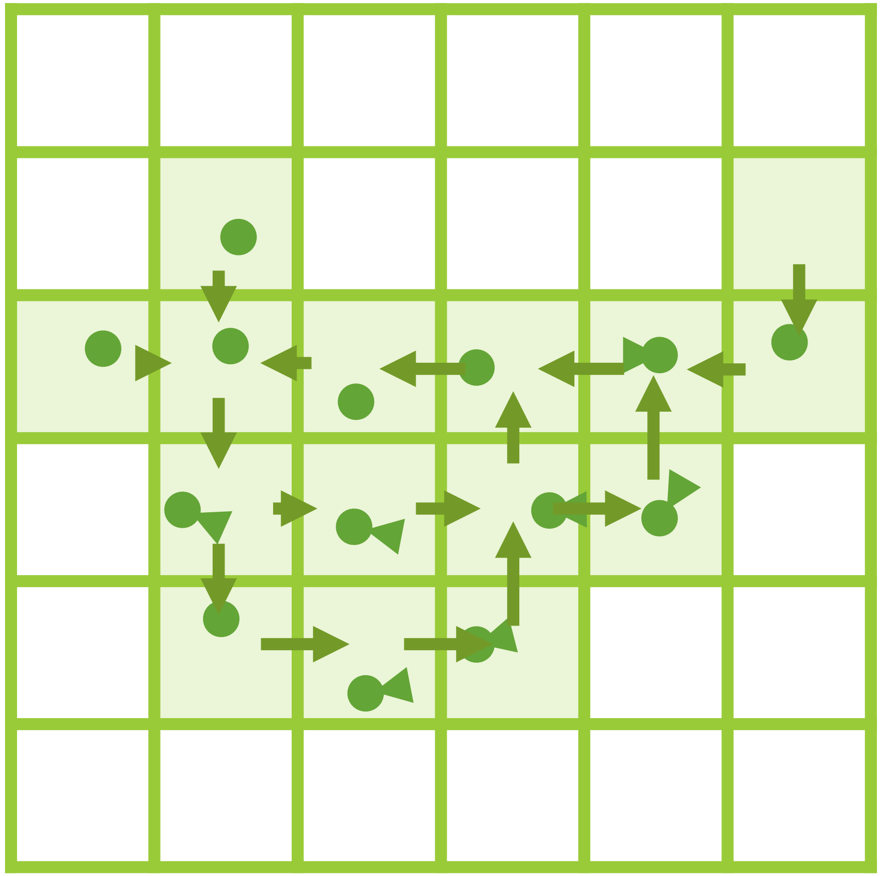


The Particle-In-Cell Method

Start with particles holding velocities

Overlay a background grid

Transfer velocities to background grid



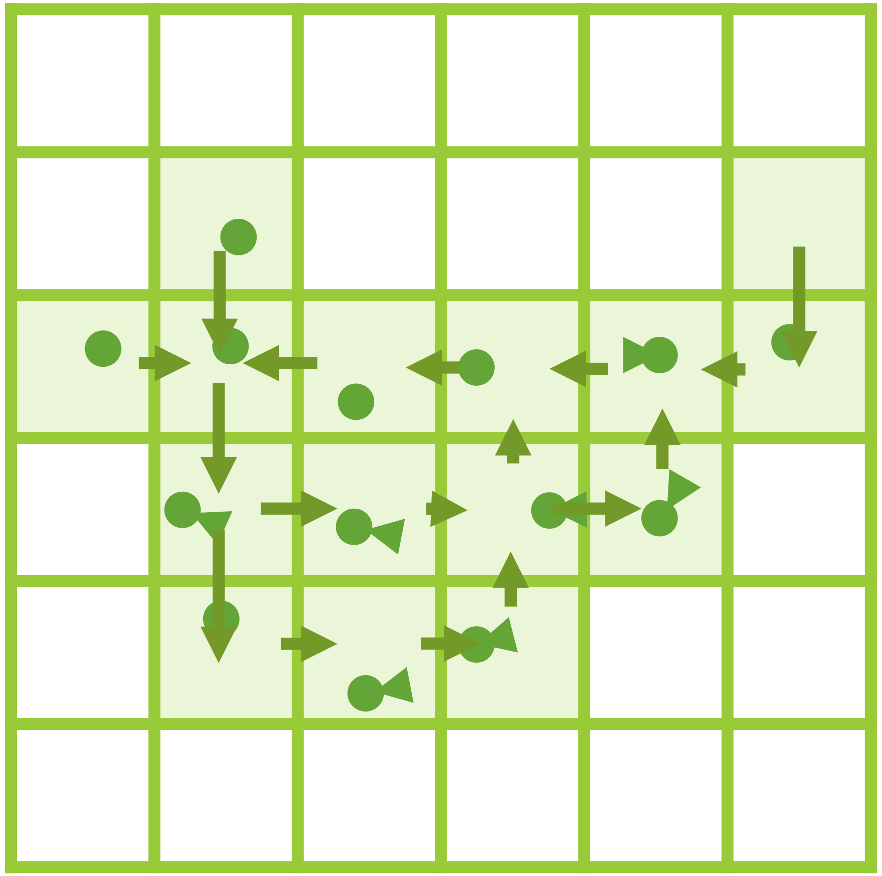
The Particle-In-Cell Method

Start with particles holding velocities

Overlay a background grid

Transfer velocities to background grid

Evolve velocities on the grid



The Particle-In-Cell Method

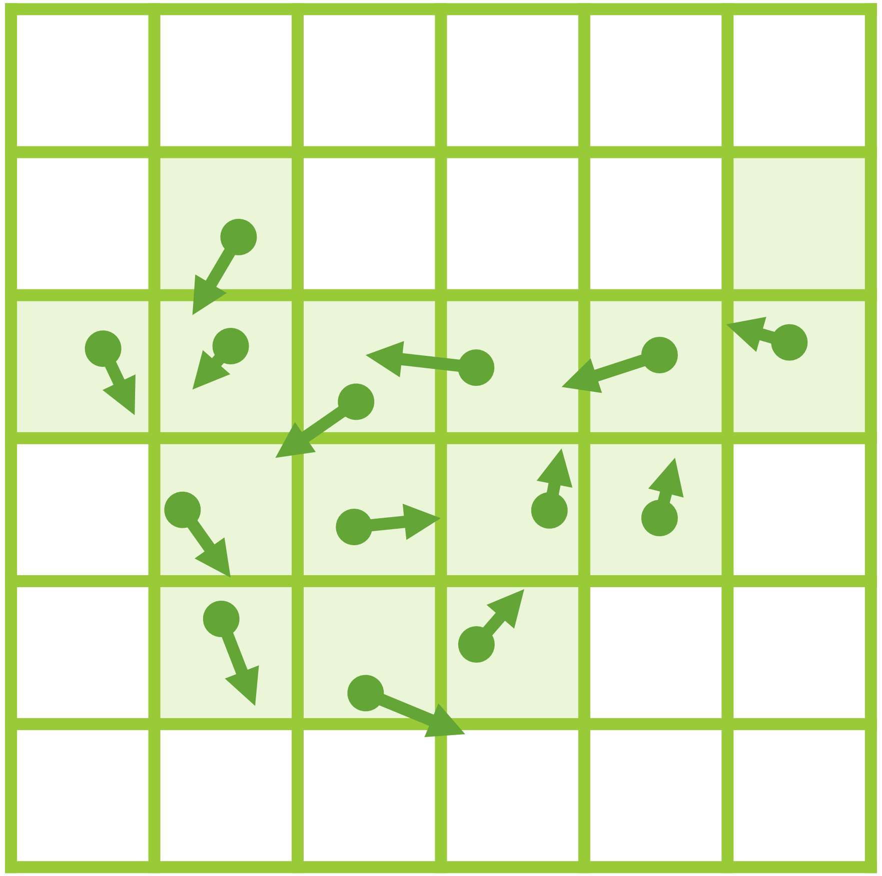
Start with particles holding velocities

Overlay a background grid

Transfer velocities to background grid

Evolve velocities on the grid

Transfer velocities to particles



The Particle-In-Cell Method

Start with particles holding velocities

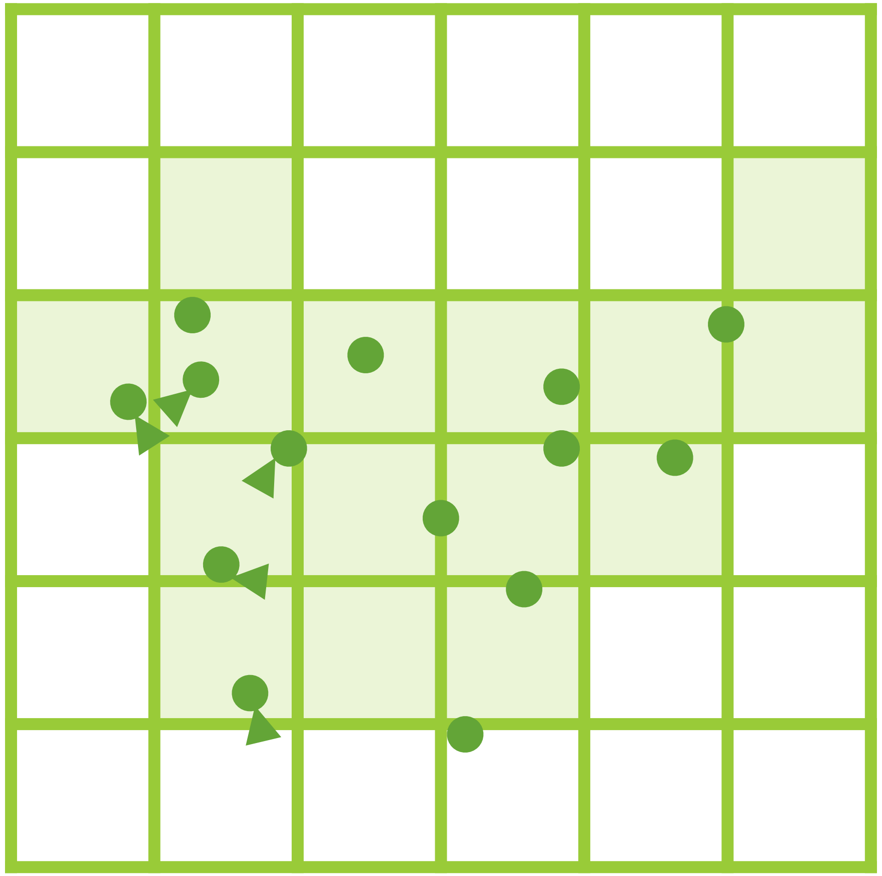
Overlay a background grid

Transfer velocities to background grid

Evolve velocities on the grid

Transfer velocities to particles

Advect particles



The Particle-In-Cell Method

Start with particles holding velocities

Overlay a background grid

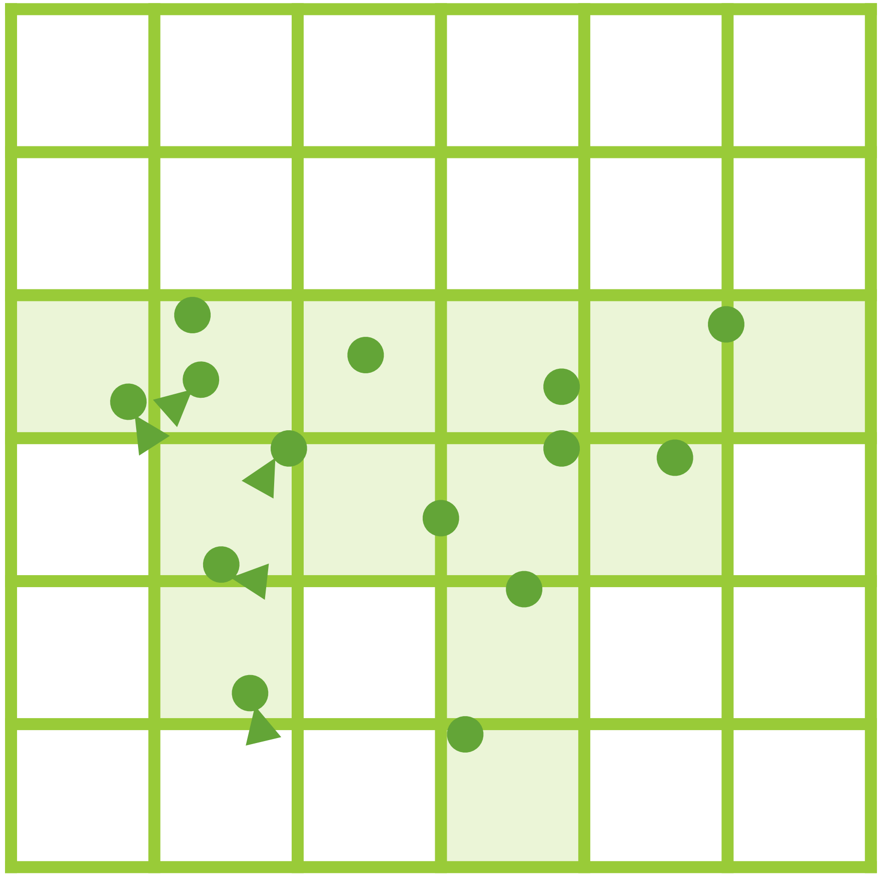
Transfer velocities to background grid

Evolve velocities on the grid

Transfer velocities to particles

Advect particles

Adjust fluid surface representation



The Particle-In-Cell Method

Start with particles holding velocities

Overlay a background grid

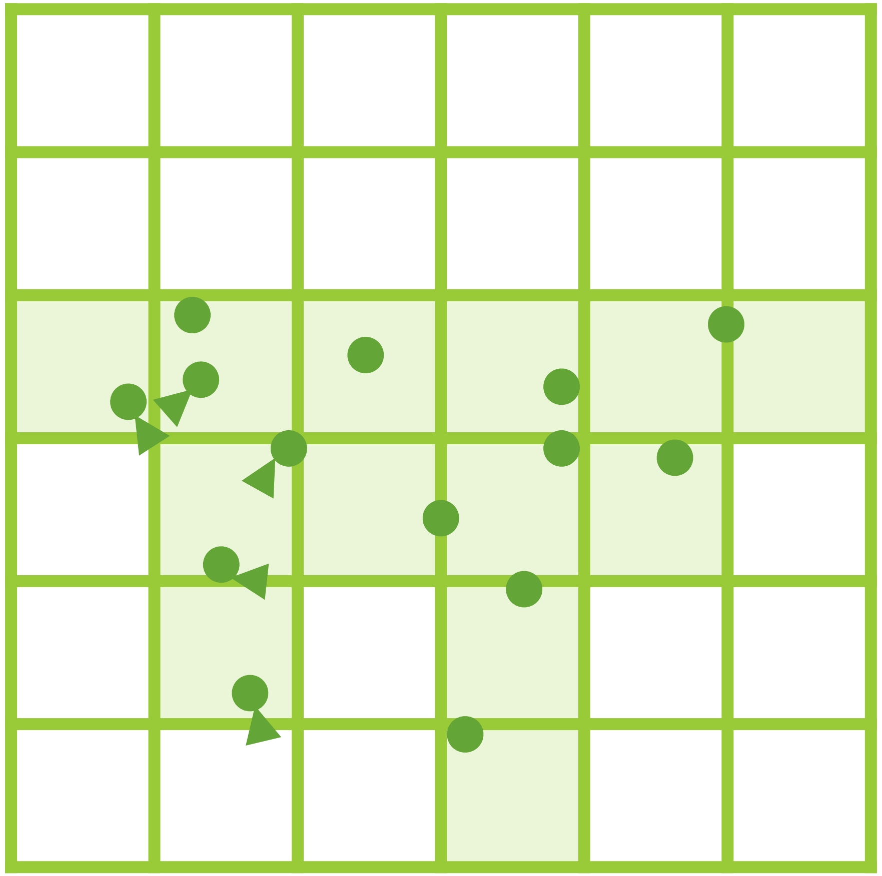
Transfer velocities to background grid

Evolve velocities on the grid

Transfer velocities to particles

Advect particles

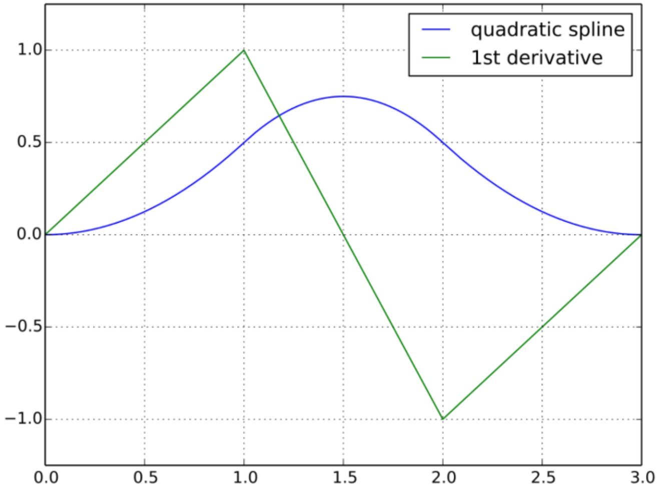
Adjust fluid surface representation

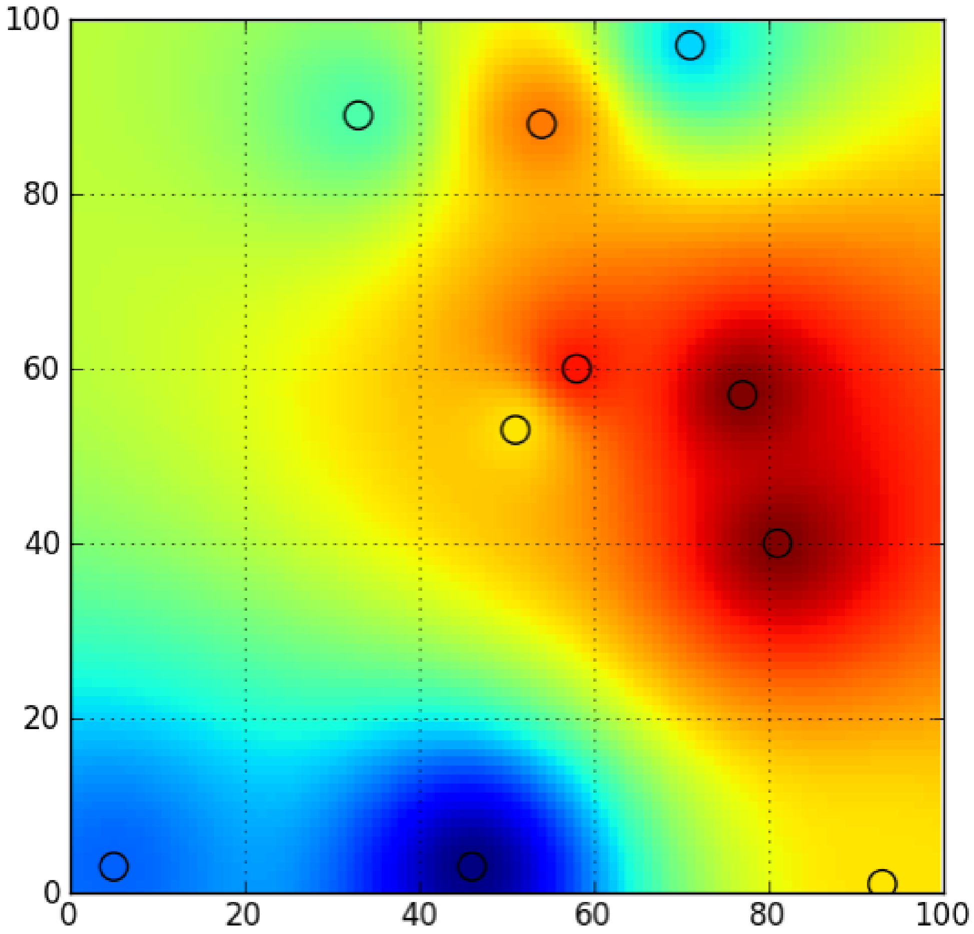


Particle to Grid Transfer

Take a weighted sum of particle velocities at each grid point

- Weight depends on distance
- Weighing function depends on kernel
- Use a quadratic B-spline kernel





https://sourceforge.net/p/geoms2/wiki/Inverse%20weighted%20distance/
https://en.wikipedia.org/wiki/B-spline#/media/File:Cardinal_quadratic_B_spline.svg

Fluid Solve

Apply any body and external forces

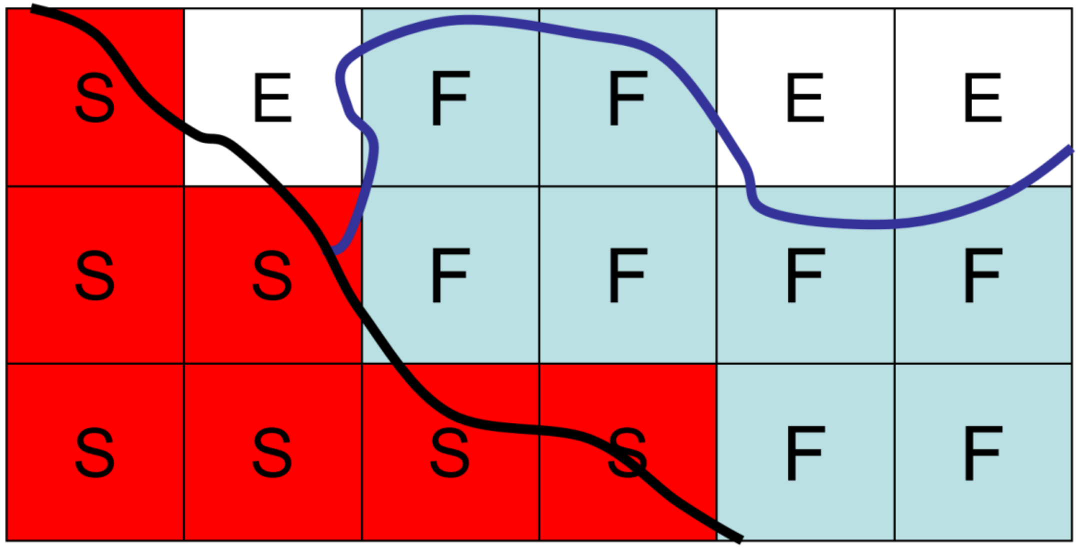
- Eg: gravity

Enforce incompressibility using standard grid-based methods

- Matrix solve on the fluid cells

Careful at boundary conditions

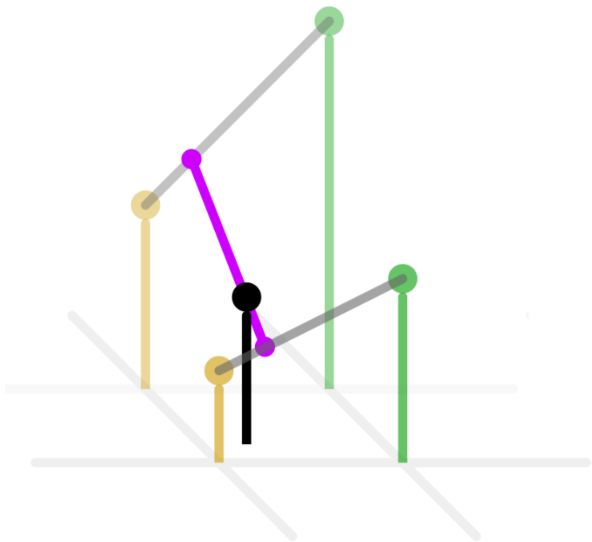
- “No stick” for solid
- Pressure is constant 0 for empty cells
- Voxellized boundaries

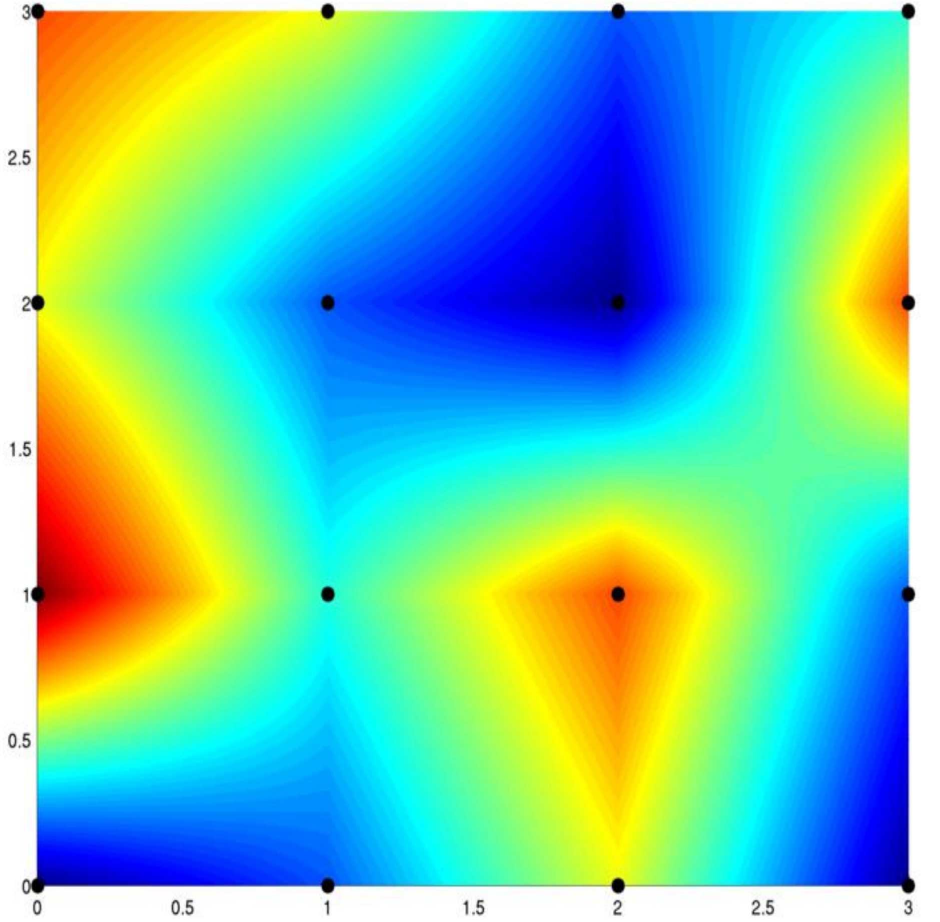


Grid to Particle Transfer

Simplest interpolation is bilinear

- Take a linear interpolation in one axis, then a linear interpolation across the other





Advection

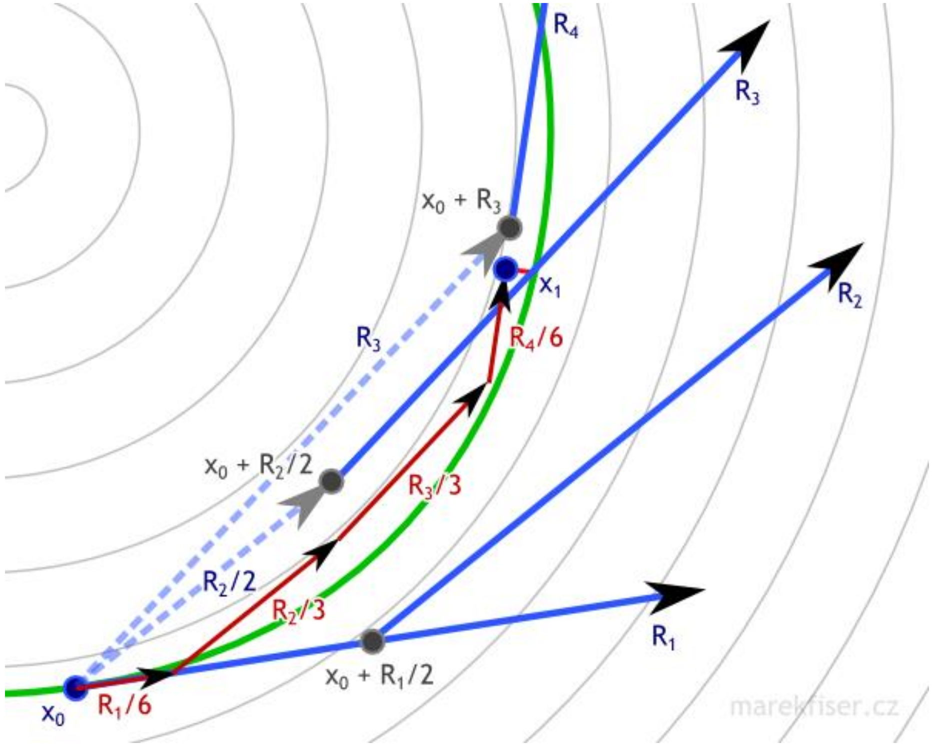
Advect using your favourite timestepping scheme

We use Ralson's 3rd Order Runge-Kutta

In practice, advection and grid-particle transfer must be done at the same time

- Intermediate steps of RK needs evaluations of the velocity fields

We also need to handle solid collisions



PIC vs FLIP

The original Particle-in-Cell method **transferred velocity** from grid to particle

Repeated transfer from particle to grid to particle causes a lot of numerical diffusion

- Due to repeated averaging steps

Alternative is called FLuid-Implicit-Particle method, which **transfers velocity change** from grid to particle

- Note that now you need to save the old velocities

In practice, use a linear combination of PIC/FLIP

Implementation Details

Initialize fluid with four "jittered" particles per cell

Use Eigen Linear Algebra library

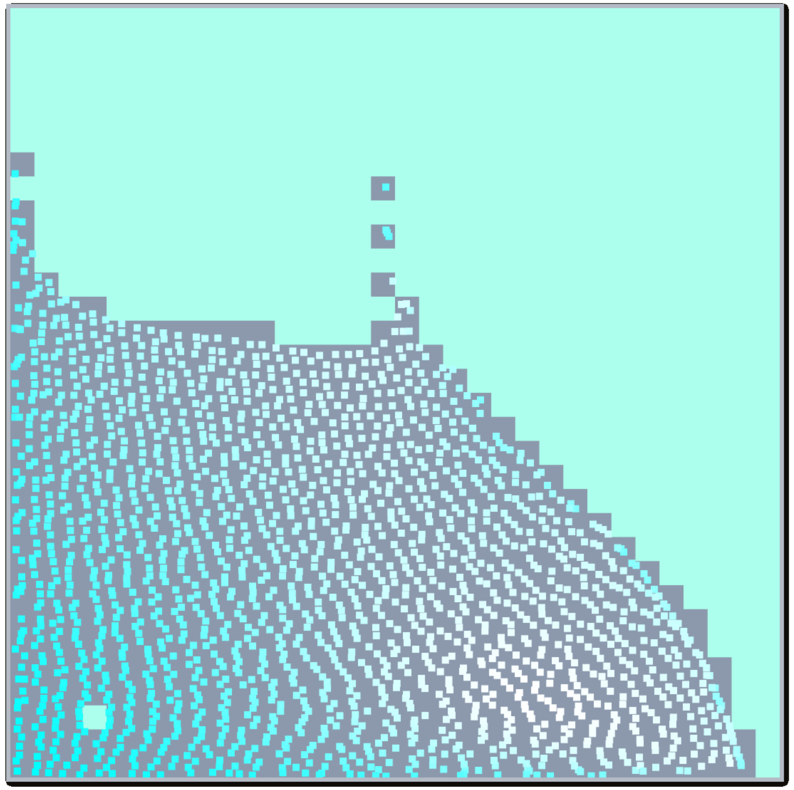
- Incomplete Cholesky sparse matrix solver

Use substepping in the advection step to satisfy CFL condition

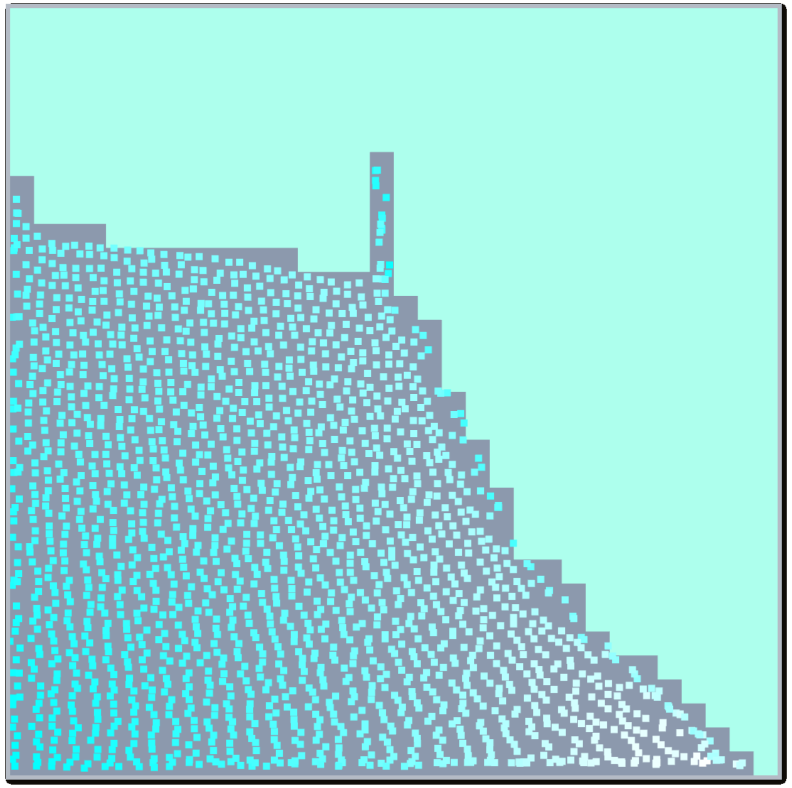
Solid collisions simply re-project backwards to where it initially collided

Use OpenGL for visualization

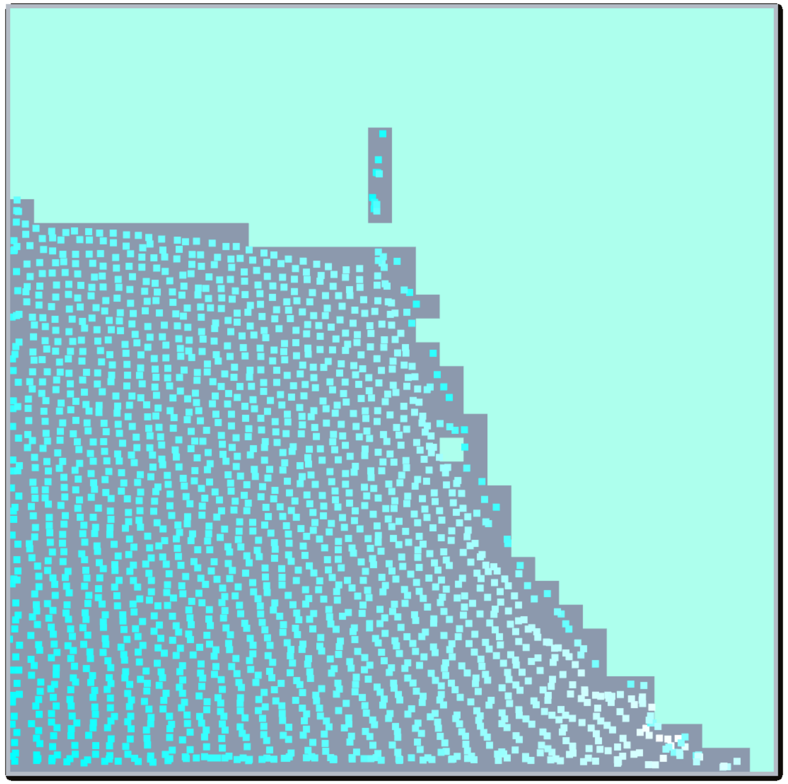
Results – Dam Break t=0.3



PIC

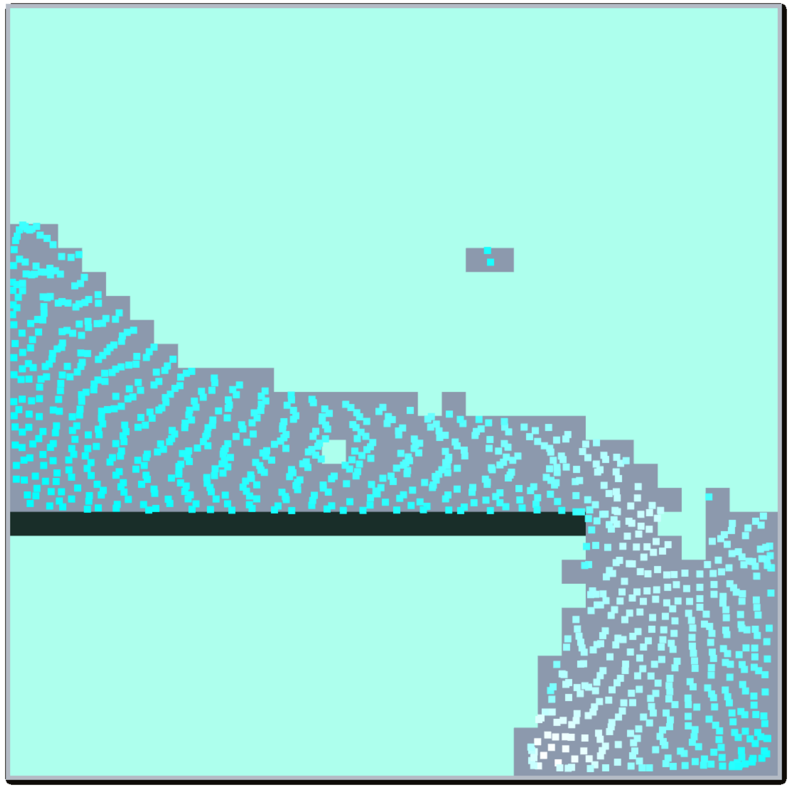


0.9 FLIP

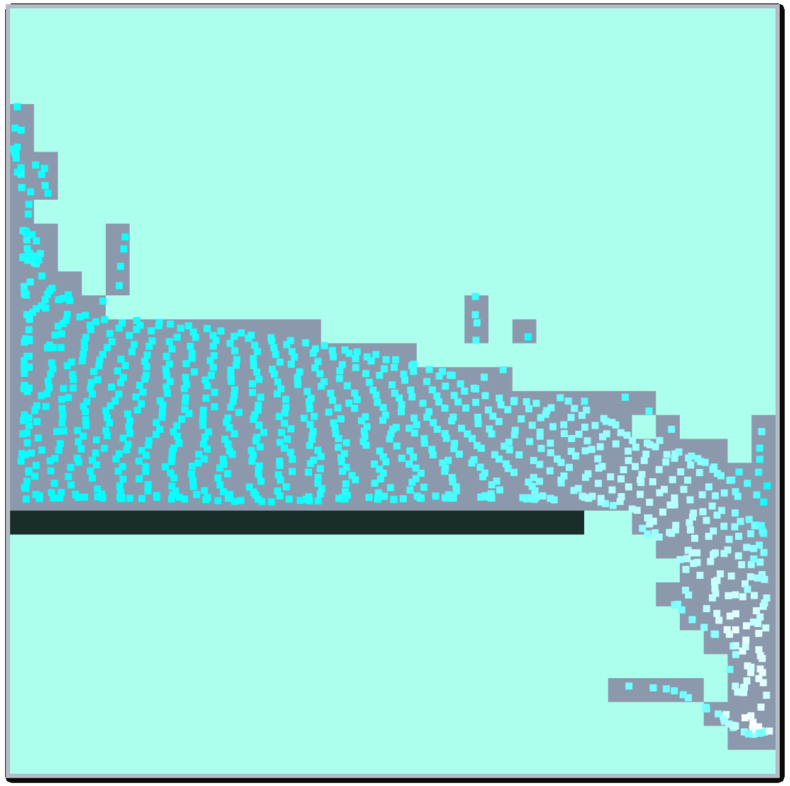


FLIP

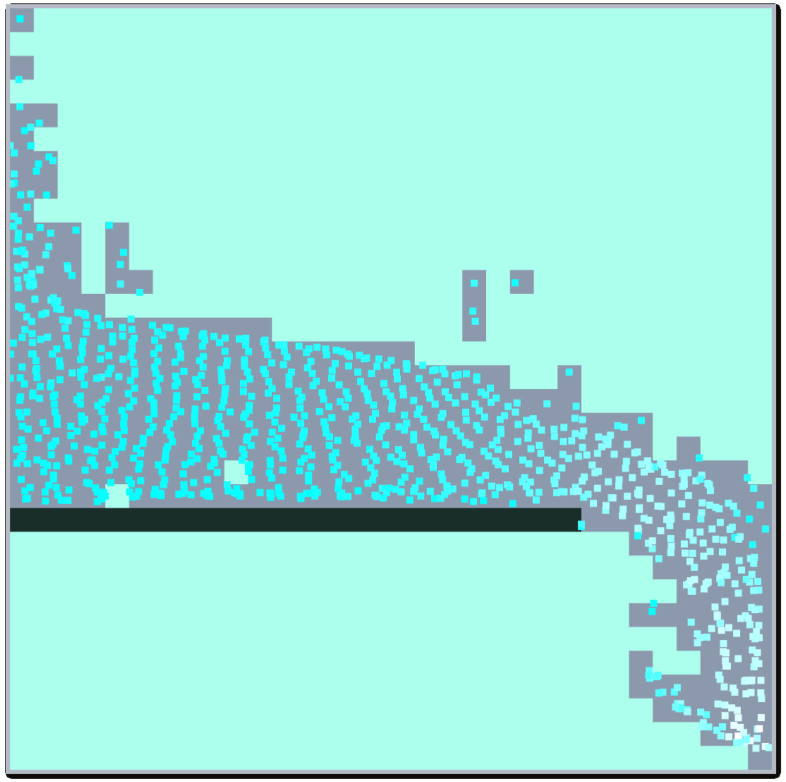
Results – Plank t=0.5



PIC

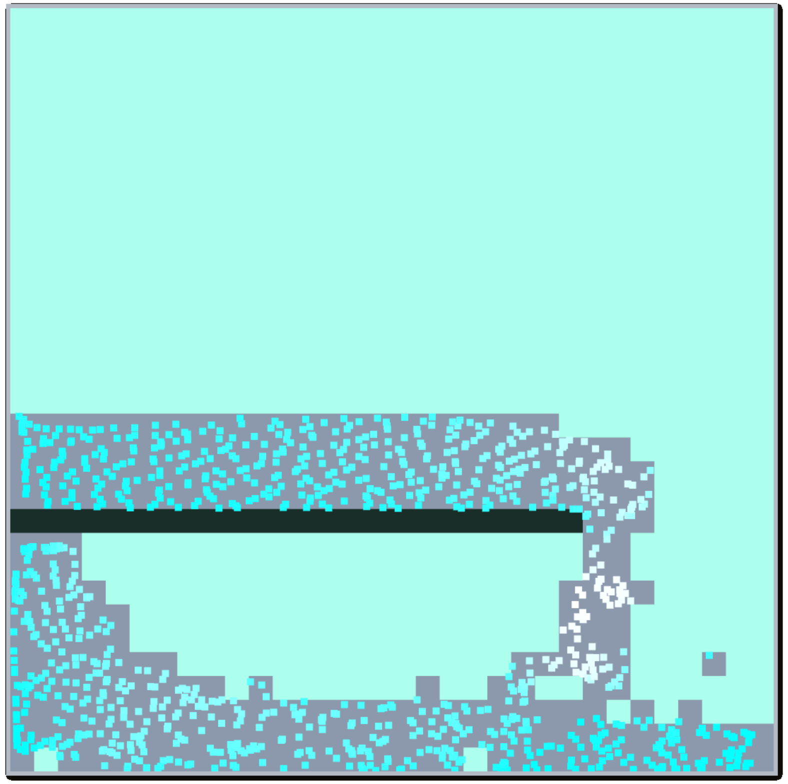


0.9 FLIP

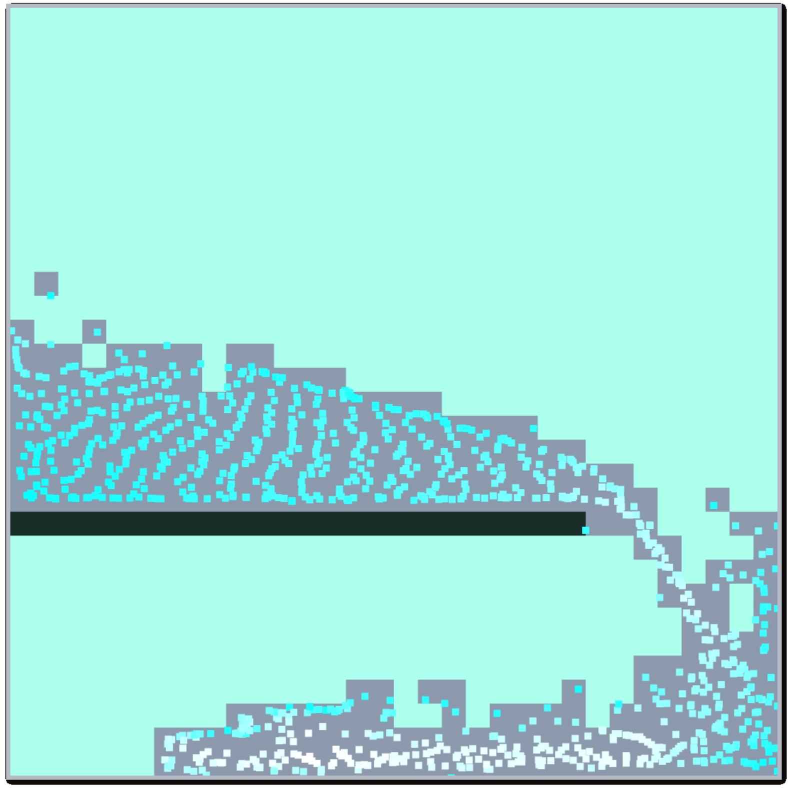


FLIP

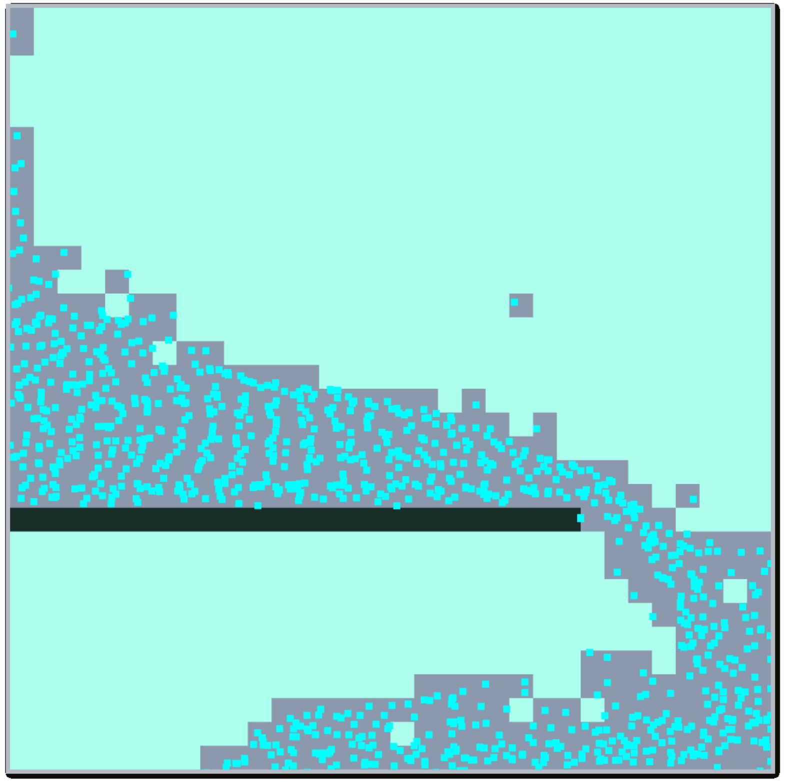
Results – Plank t=1.0



PIC

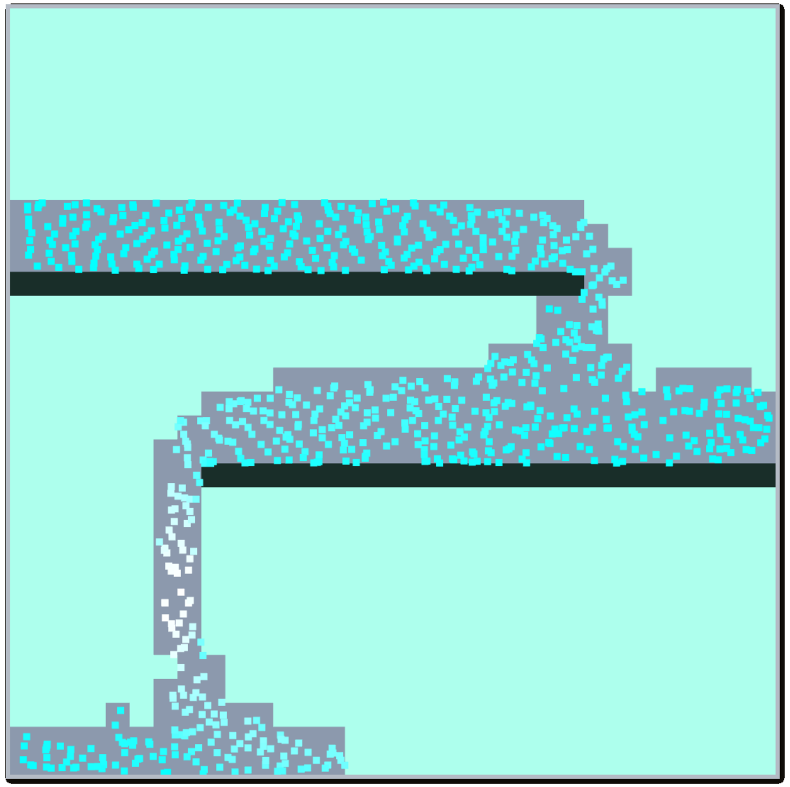


0.9 FLIP

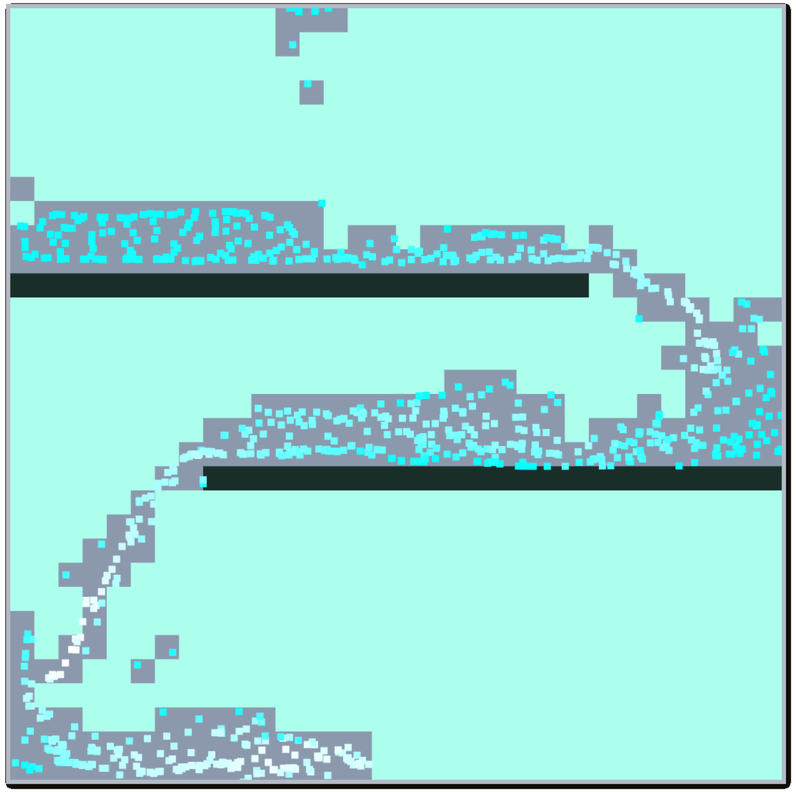


FLIP

Results – Maze t=1.5



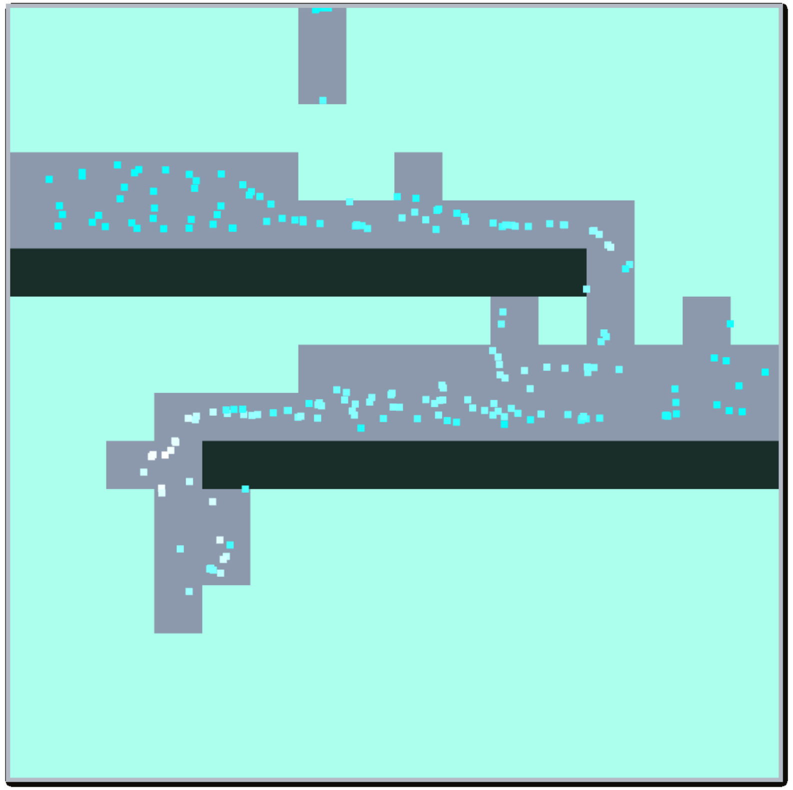
PIC



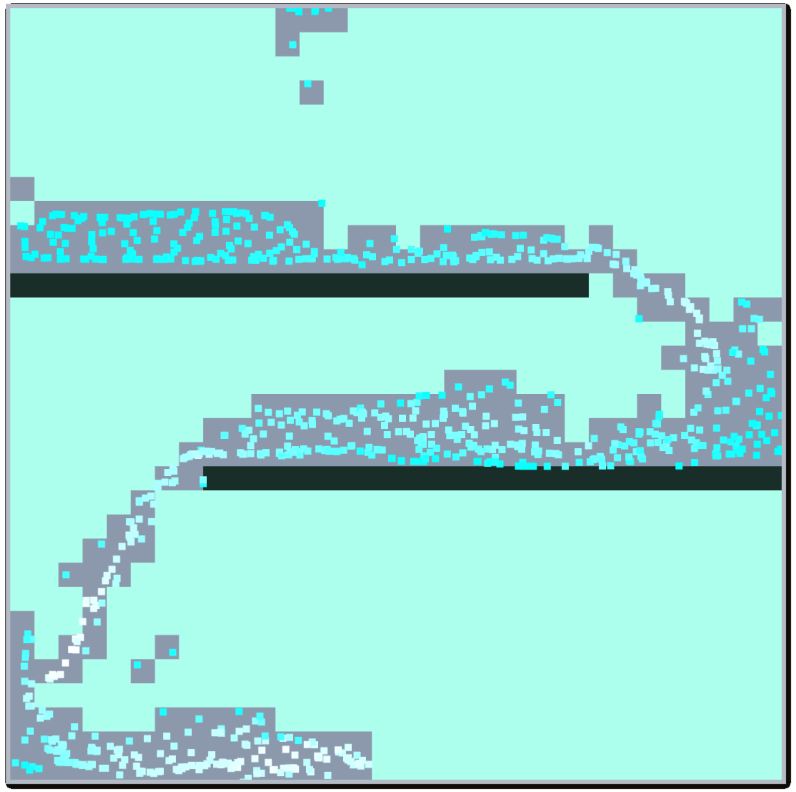
0.9 FLIP

FLIP

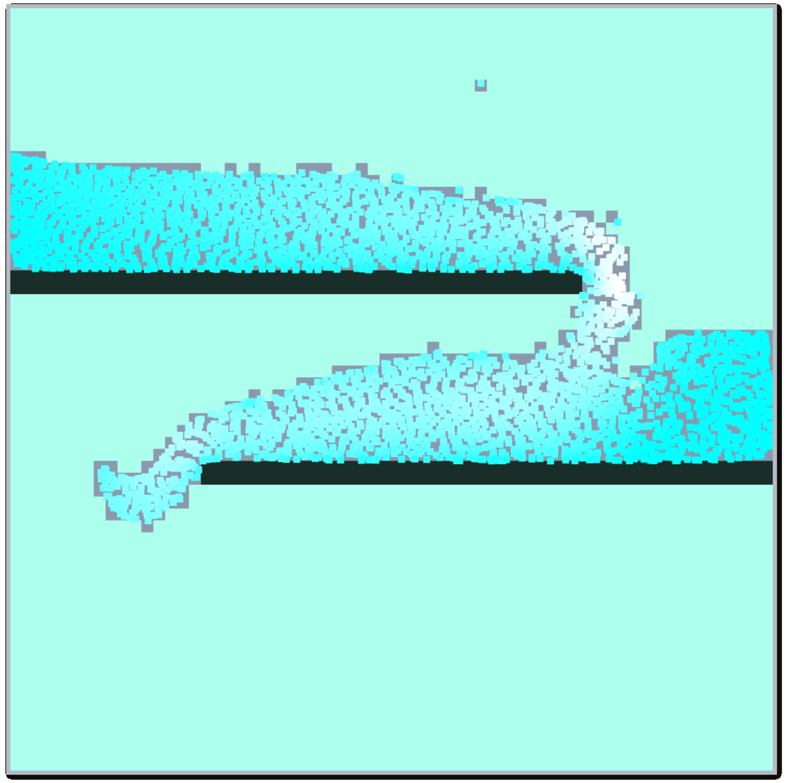
Results –Resolution Convergence?



16x16

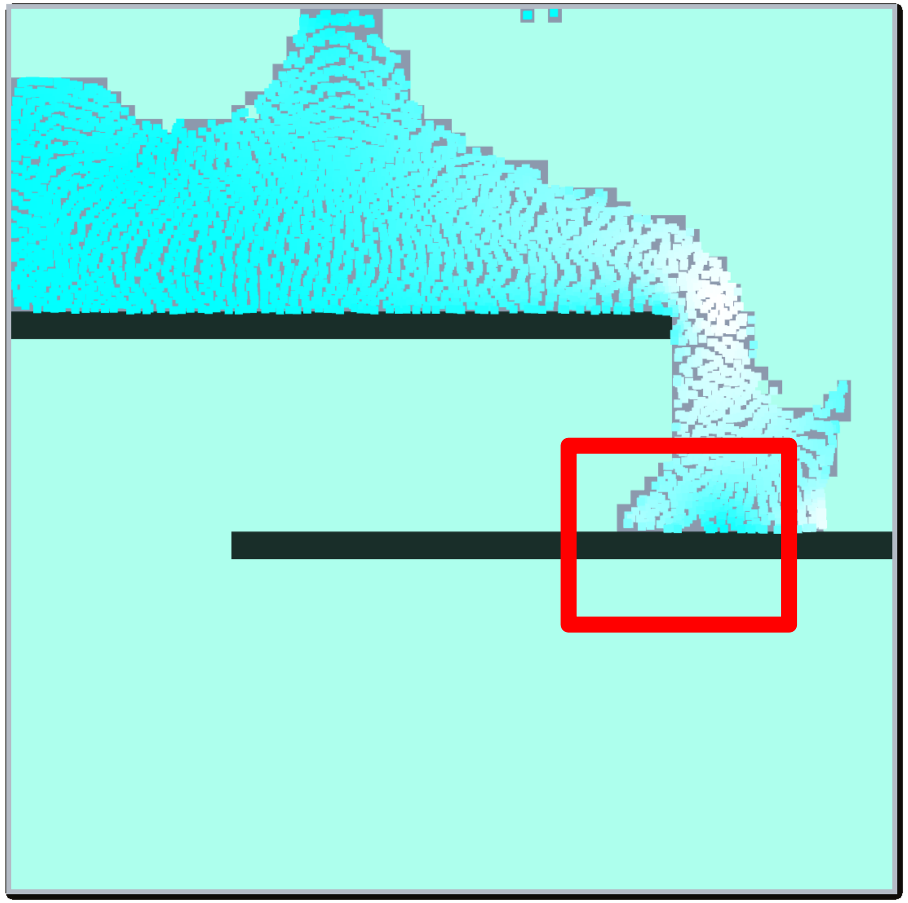


32x32



64x64

Results – Grid Artifacting



Future Work

Hybrid particle-grid methods are a rich area of current research

Many more features need to be implemented

- Level sets for more accurate surface tracking

More advanced particle-grid interpolations

- “Affine PIC” – regain rigid body rotation
- “Poly PIC” – include higher order modes