Lecture 17:

Introduction to Physical Simulation

Computer Graphics and Imaging
UC Berkeley CS184/284A

The majority of these slides courtesy of James O'Brien and Keenan Crane.

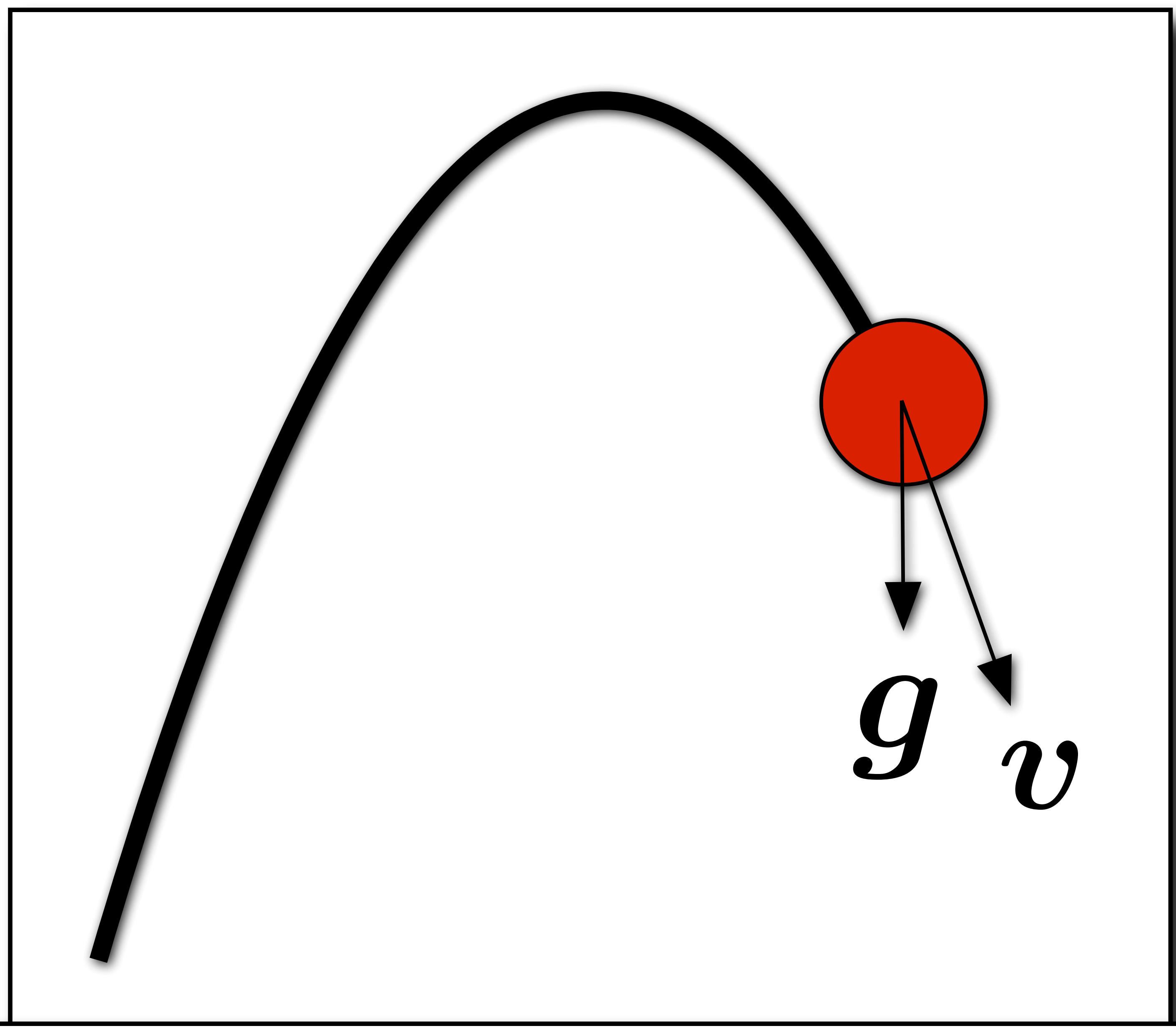
Newton's Law

$$F = ma$$

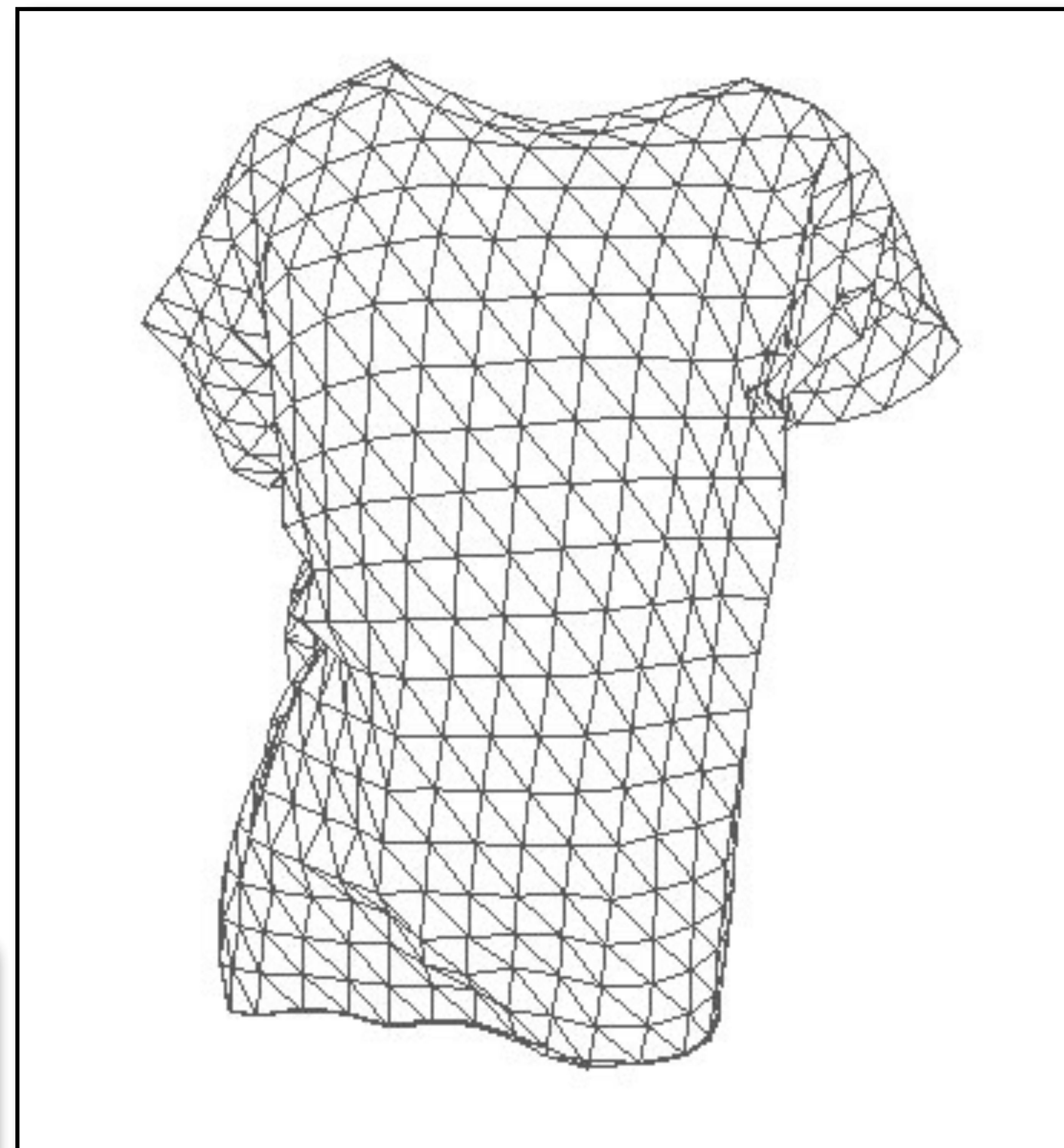
Force — Mass — Acceleration

Physically Based Animation

Generate motion of objects using numerical simulation



$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t \boldsymbol{v}^t + \frac{1}{2}(\Delta t)^2 \boldsymbol{a}^t$$



Example: Cloth Simulation

Example: Fluids

Macklin and Müller, Position Based Fluids TOG 2013

Particle Systems

Single particles are very simple

Large groups can produce interesting effects

Supplement basic ballistic rules

- Gravity
- Friction, drag
- Collisions
- Force fields
- Springs
- Interactions
- Others…

Karl Sims, SIGGRAPH 1990

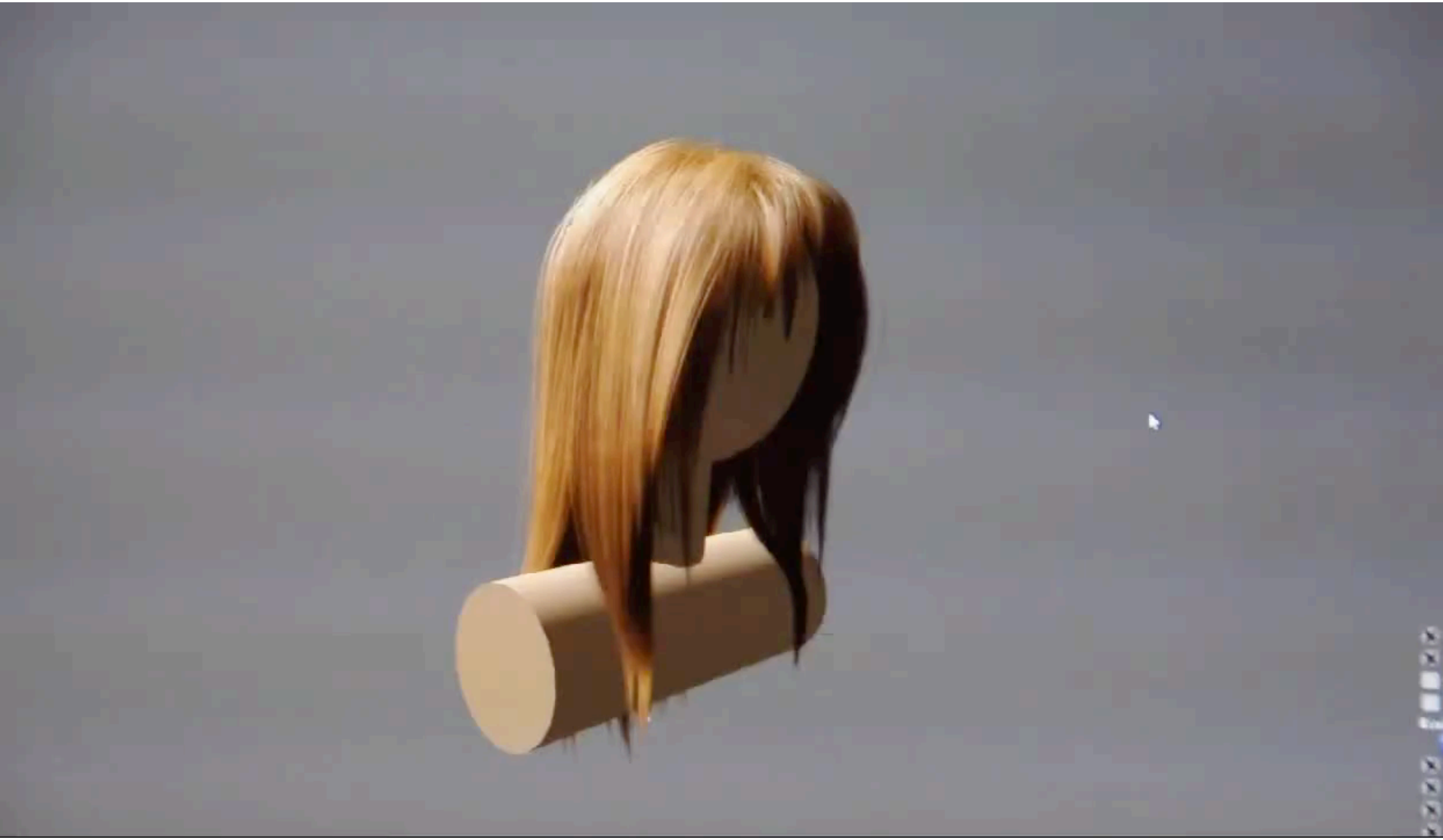
Mass + Spring Systems:

Example of Modeling a Dynamical System

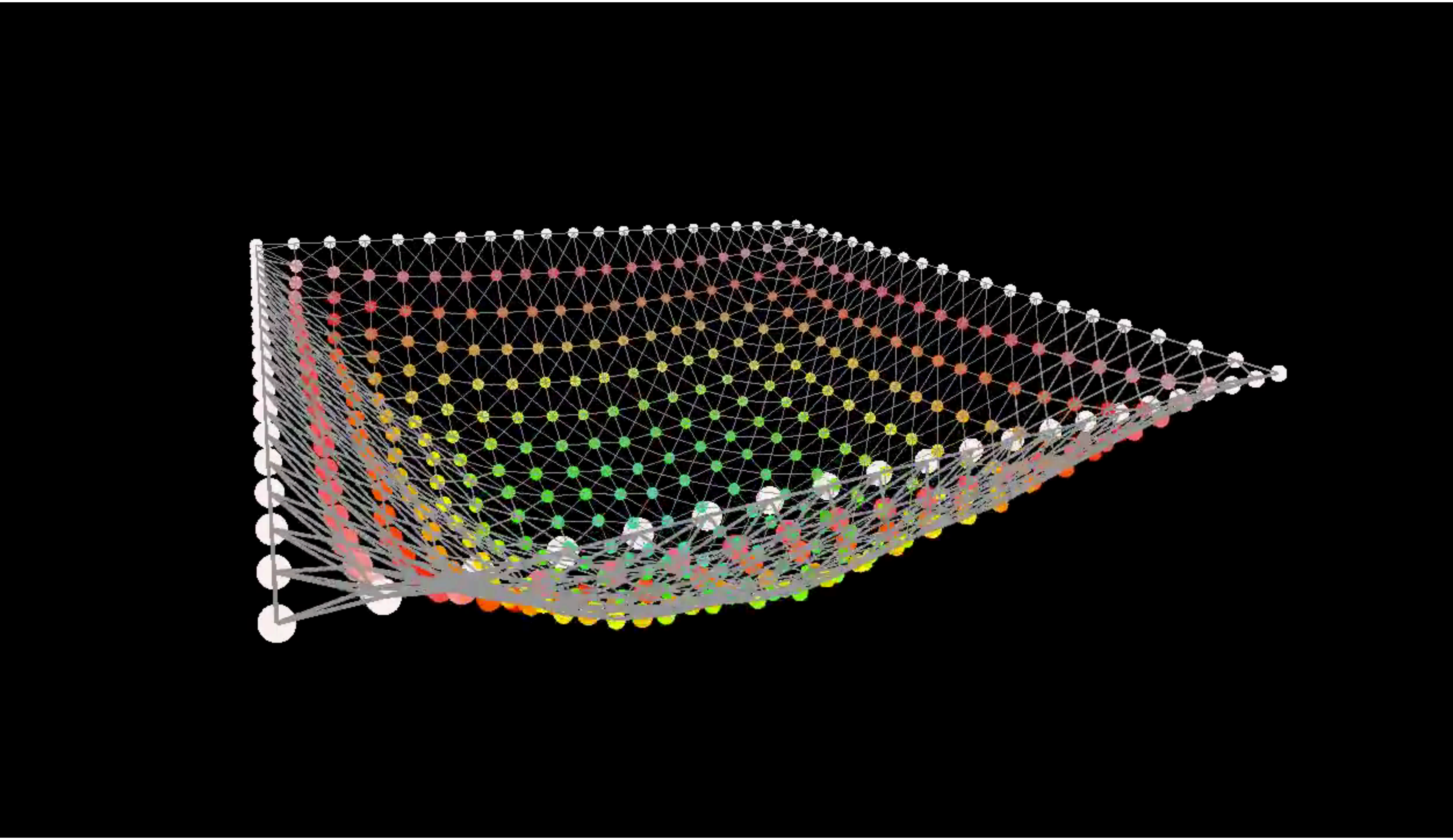
Example: Mass Spring Rope

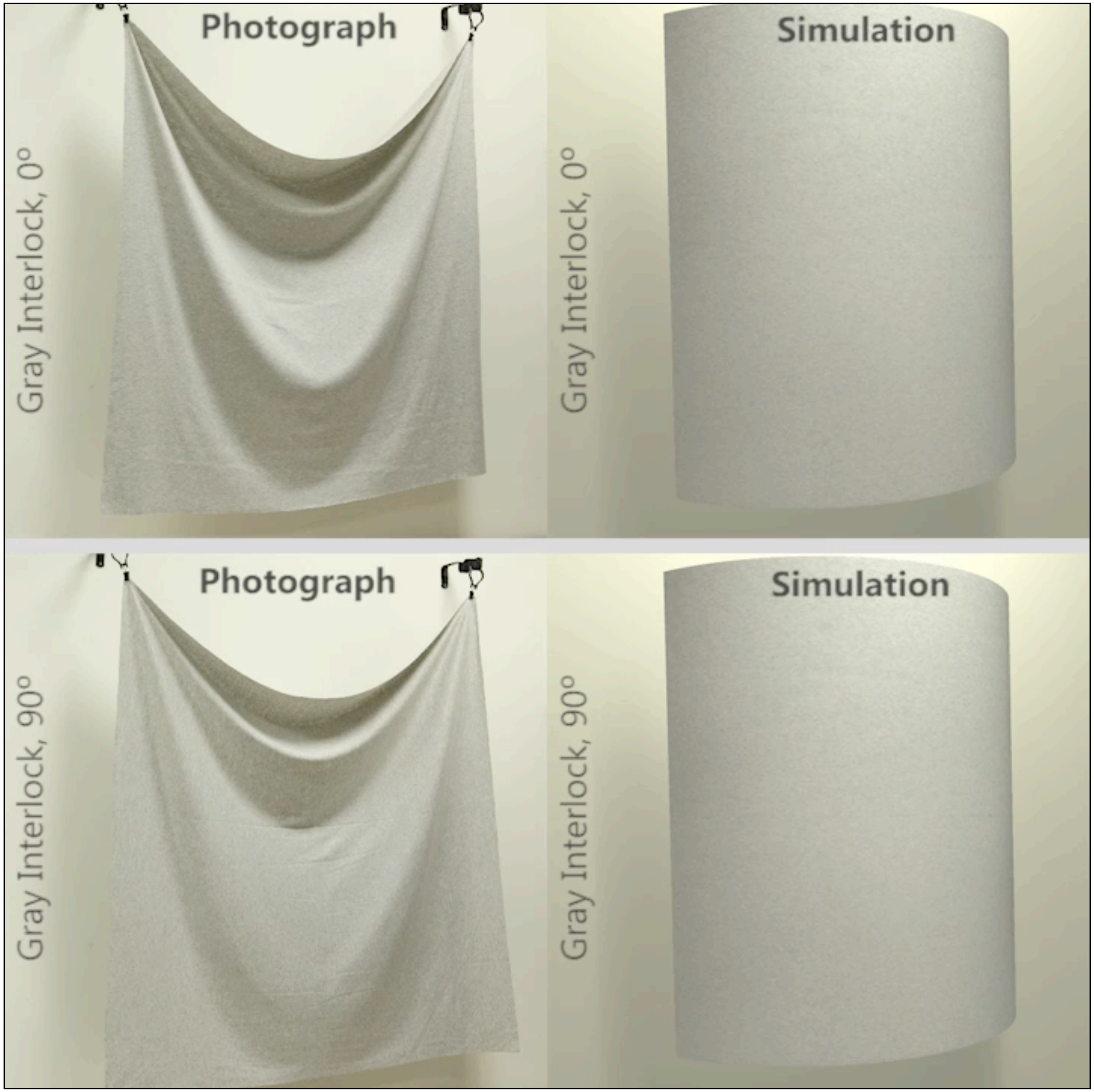
Credit: Elizabeth Labelle, https://youtu.be/Co8enp8CH34

Example: Hair



Example: Mass Spring Mesh

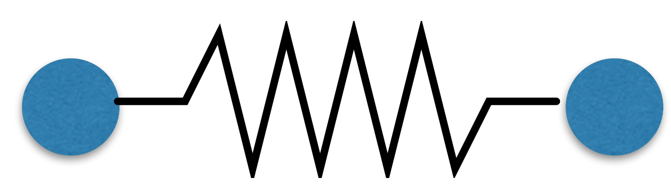




Huamin Wang, Ravi Ramamoorthi, and James F. O'Brien. "Data-Driven Elastic Models for Cloth: Modeling and Measurement". *ACM Transactions on Graphics*, 30(4):71:1–11, July 2011. Proceedings of ACM SIGGRAPH 2011, Vancouver, BC Canada.

A Simple Spring

Idealized spring



$$\boldsymbol{f}_{a \to b} = k_s (\boldsymbol{b} - \boldsymbol{a})$$

$$\boldsymbol{f}_{b \to a} = -\boldsymbol{f}_{a \to b}$$

Force pulls points together

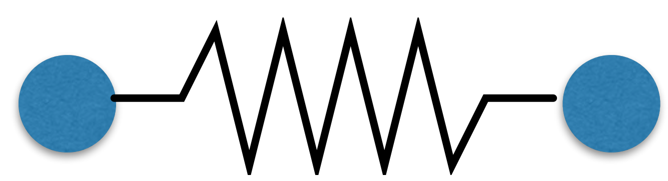
Strength proportional to displacement (Hooke's Law)

k_s is a spring coefficient: stiffness

Problem: this spring wants to have zero length

Non-Zero Length Spring

Spring with non-zero rest length



$$\boldsymbol{f}_{a \rightarrow b} = k_S \frac{\boldsymbol{b} - \boldsymbol{a}}{||\boldsymbol{b} - \boldsymbol{a}||} (||\boldsymbol{b} - \boldsymbol{a}|| - l)$$

Rest length

Problem: oscillates forever

Dot Notation for Derivatives

If x is a vector for the position of a point of interest, we will use dot notation for velocity and acceleration:

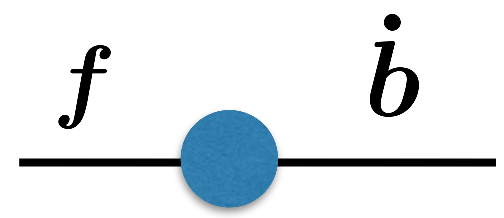
$$\boldsymbol{x}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{v}$$

$$\ddot{\boldsymbol{x}} = \boldsymbol{a}$$

Simple Motion Damping

Simple motion damping



$$\boldsymbol{f} = -k_d \dot{\boldsymbol{b}}$$

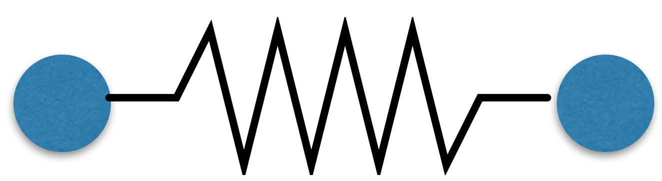
- Behaves like viscous drag on motion
- Slows down motion in the direction of motion
- k_d is a damping coefficient

Problem: slows down *all* motion

- Want a rusty spring's oscillations to slow down, but should it also fall to the ground more slowly?

Internal Damping for Spring

Damp only the internal, spring-driven motion

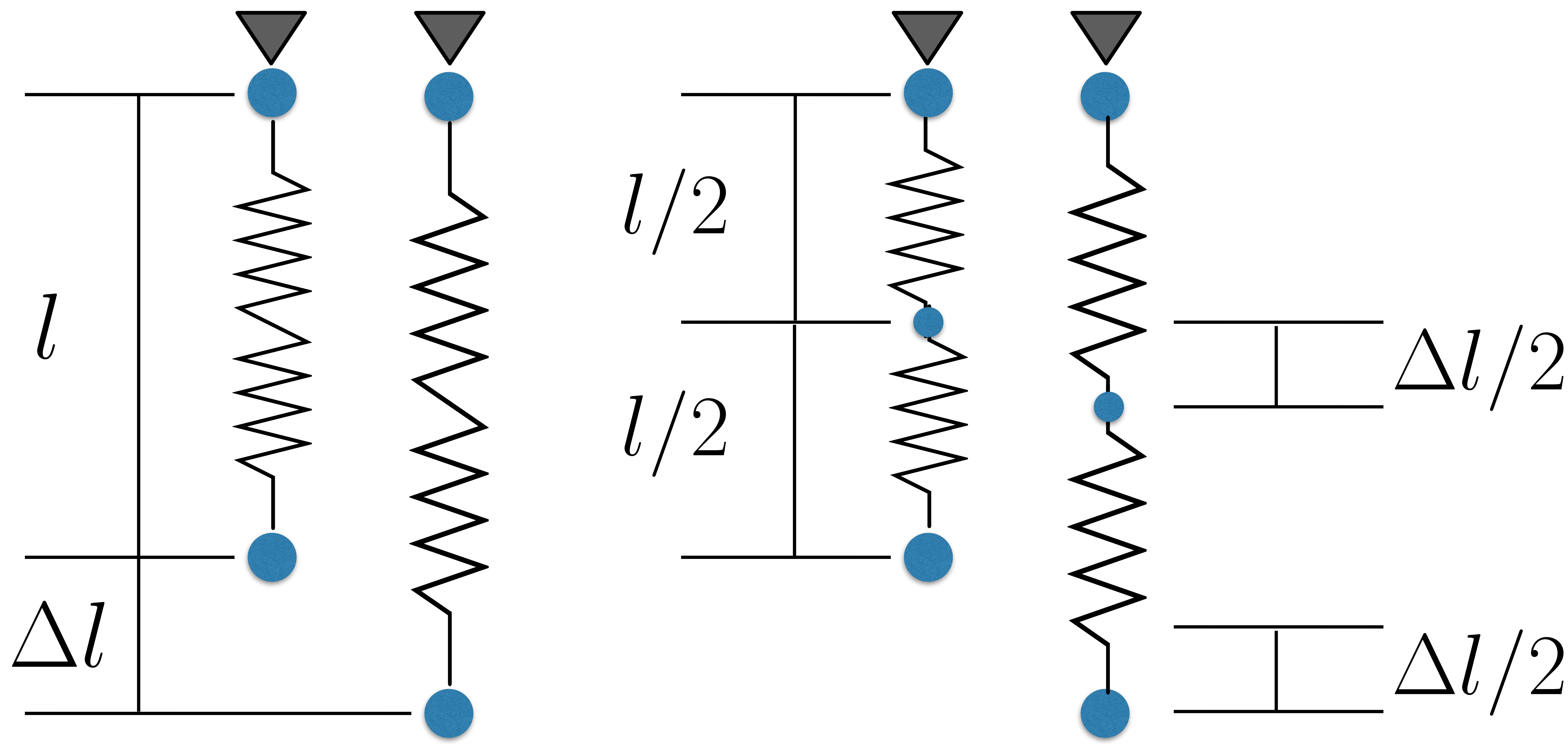


$$\boldsymbol{f}_a = -k_d \frac{\boldsymbol{b}-\boldsymbol{a}}{||\boldsymbol{b}-\boldsymbol{a}||}(\dot{\boldsymbol{b}}-\dot{\boldsymbol{a}}) \cdot \frac{\boldsymbol{b}-\boldsymbol{a}}{||\boldsymbol{b}-\boldsymbol{a}||}$$

- Viscous drag only on change in spring length
  - Won't slow group motion for the spring system (e.g. global translation or rotation of the group)

Spring Constants

Consider two "resolutions" to model a single spring



Problem: constant k_s produces different force on bottom spring for these two different discretizations

Spring Constants

Problem: constant k_s gives inconsistent results with different discretizations of our spring/mass structures

- E.g. 10x10 vs 20x20 mesh for cloth simulation would give different results, and we want them to be the same, just higher level of detail

Solution:

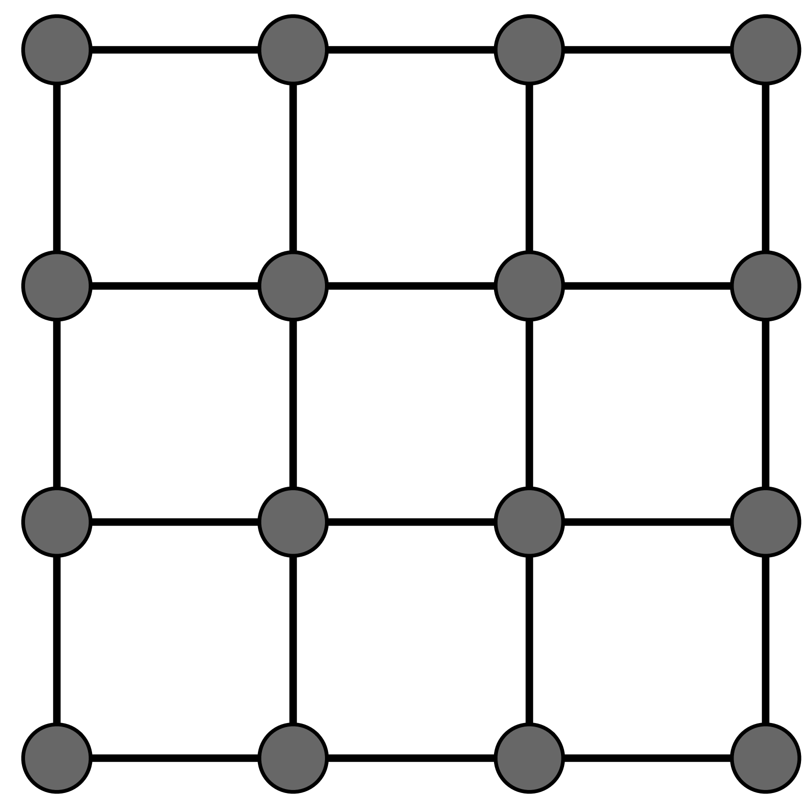
- Change in length is not what we want to measure
- We want to consider the strain = change in length as fraction of original length

$$\epsilon = \frac{\Delta l}{l_0}$$

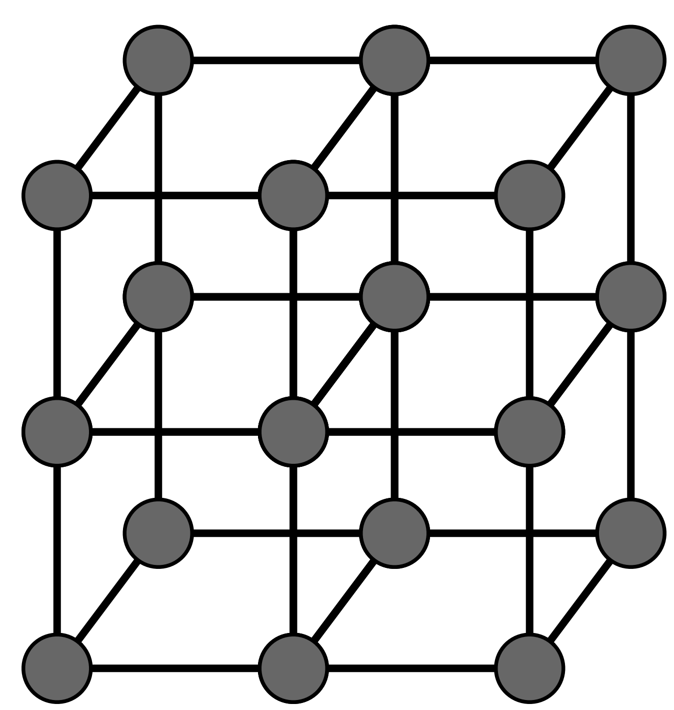
- Implementation 1: divide spring force by spring length
- Implementation 2: normalize k_s by spring length

Structures from Springs

Sheets



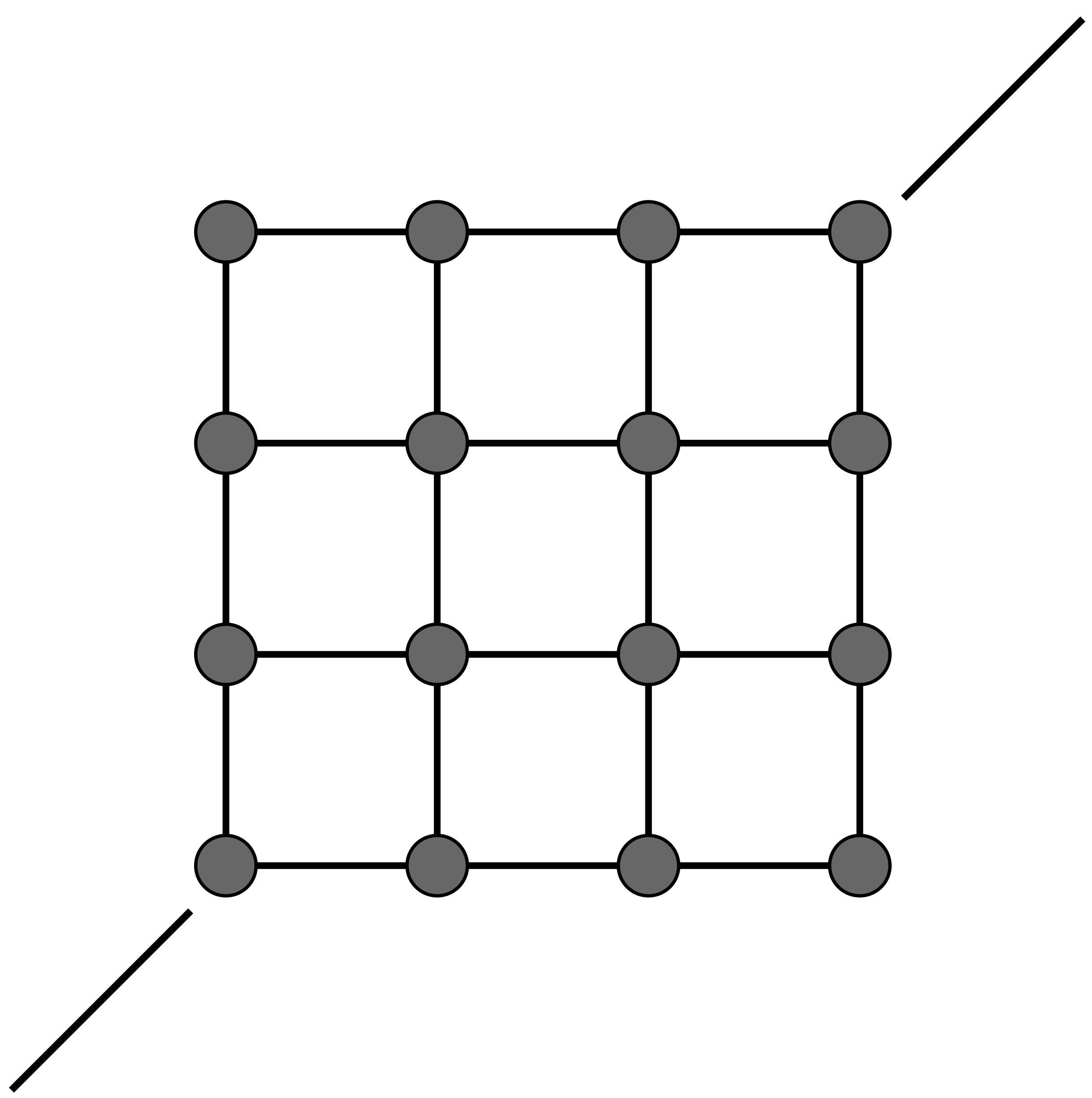
Blocks



Others

Structures from Springs

Behavior is determined by structure linkages

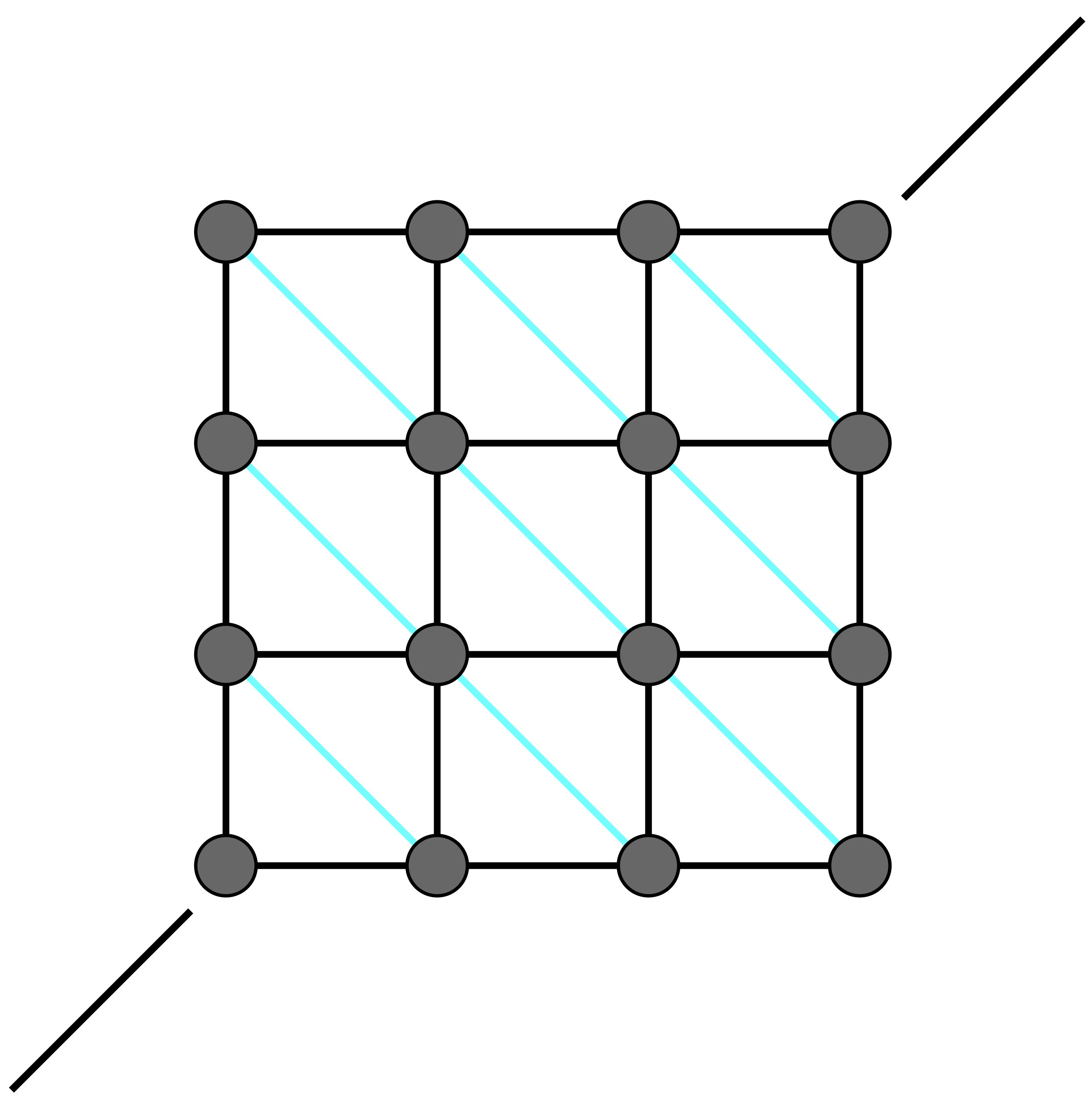


This structure will not resist shearing

This structure will not resist out-of-plane bending…

Structures from Springs

Behavior is determined by structure linkages

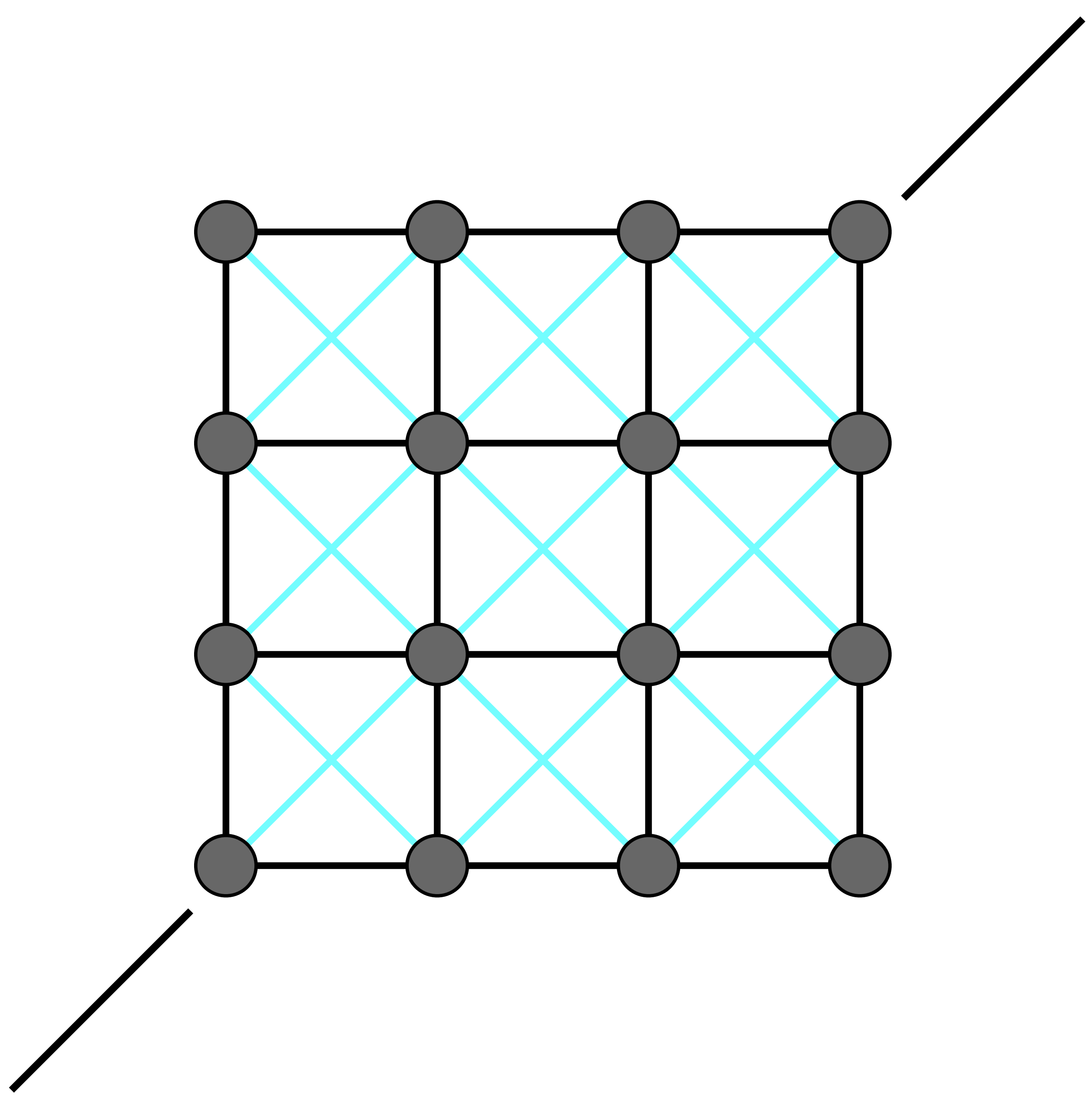


This structure will resist shearing but has anisotropic bias

This structure will not resist out-of-plane bending either…

Structures from Springs

Behavior is determined by structure linkages

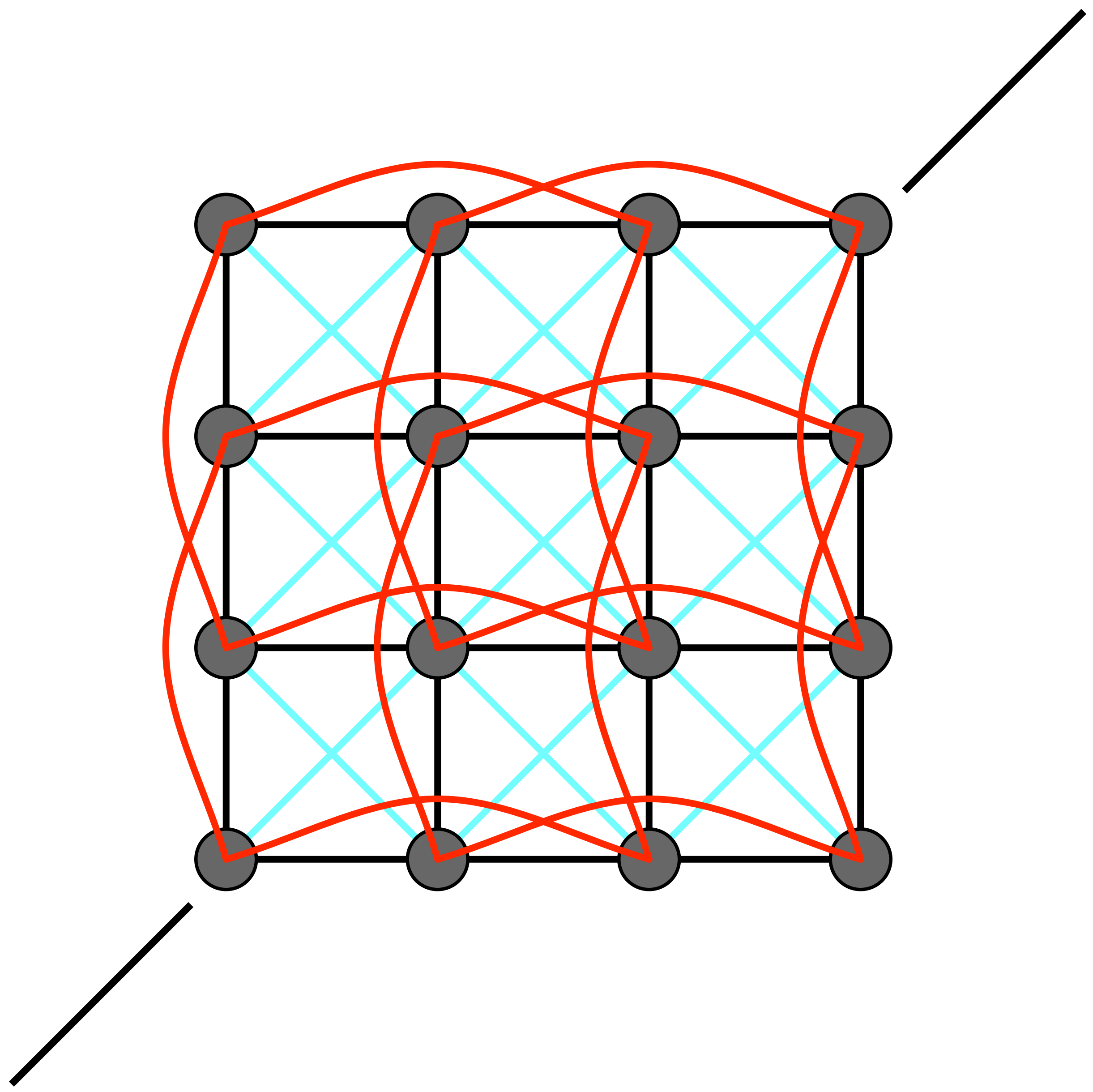


This structure will resist shearing.
Less directional bias.

This structure will not resist out-of-plane bending either…

Structures from Springs

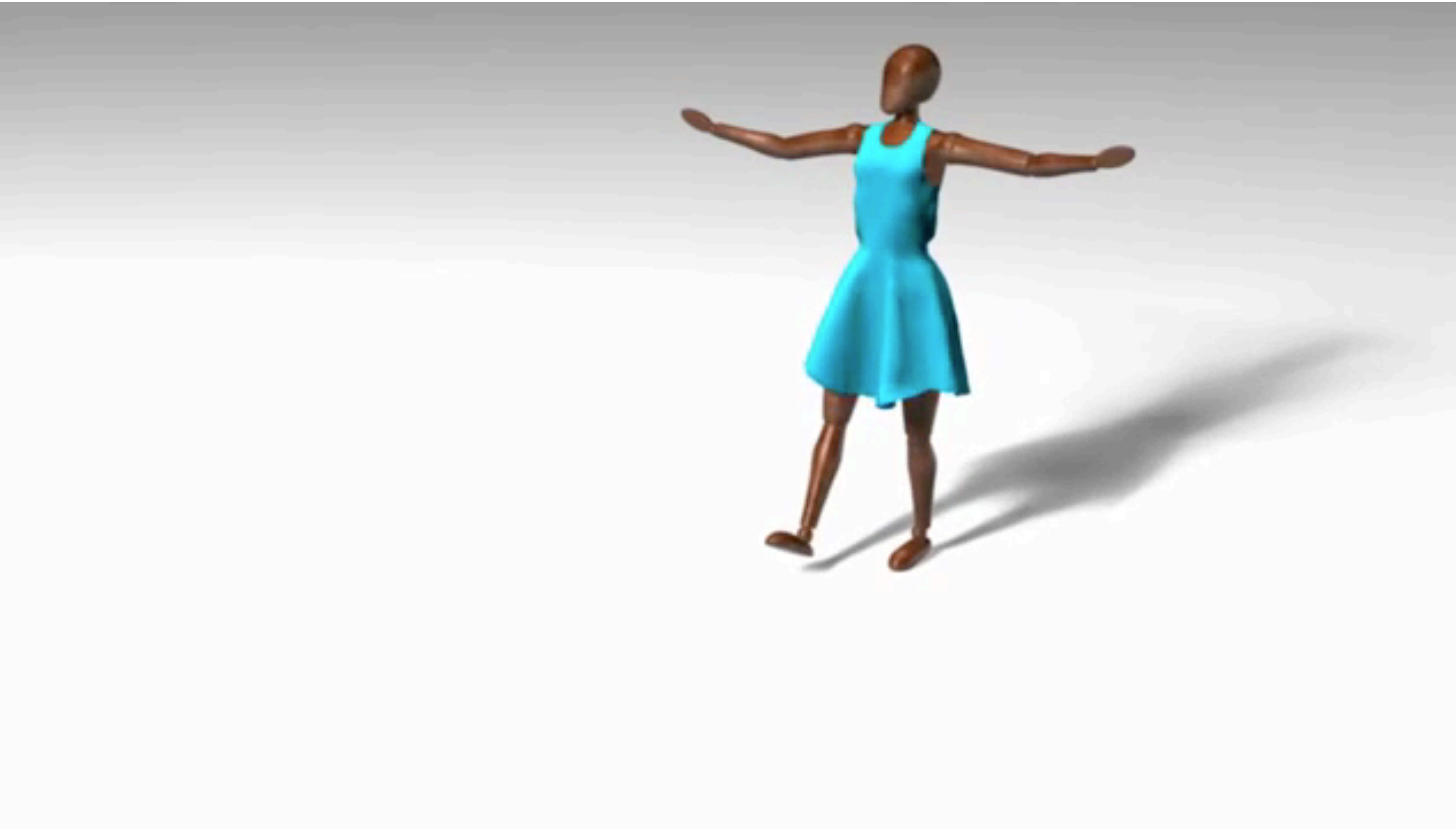
They behave like what they are (obviously!)



This structure will resist shearing.
Less directional bias.

This structure will resist out-of-plane bending
Red springs should be much weaker

Example: Mass Spring Dress + Character



Particle Simulation

Euler's Method

Euler's Method (a.k.a. Forward Euler, Explicit)

- Simple iterative method
- Commonly used
- Very inaccurate
- Most often goes unstable

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^{t} + \Delta t\, \dot{\boldsymbol{x}}^{t}$$

$$\dot{\boldsymbol{x}}^{t+\Delta t} = \dot{\boldsymbol{x}}^{t} + \Delta t\, \ddot{\boldsymbol{x}}^{t}$$

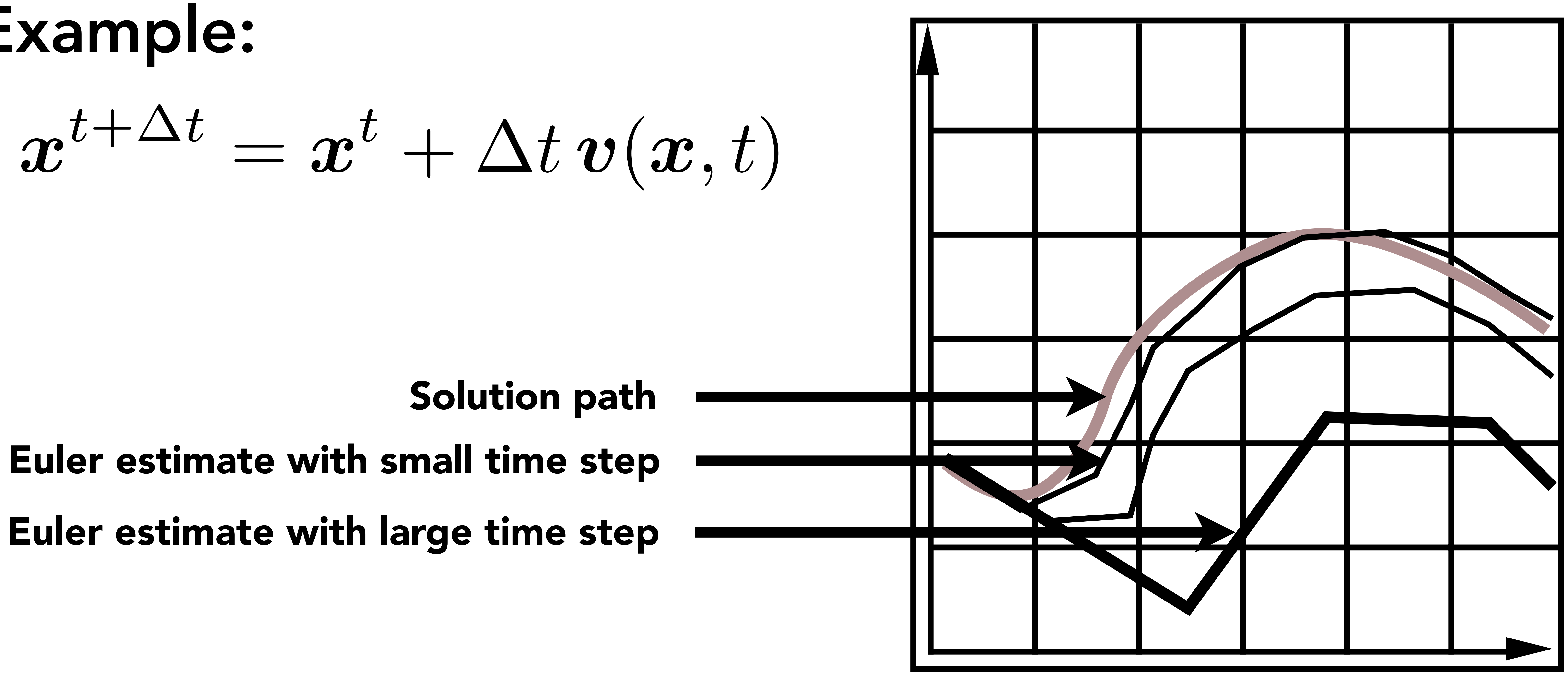
Euler's Method - Errors

With numerical integration, errors accumulate

Euler integration is particularly bad

Example:

$$x^{t+\Delta t} = x^t + \Delta t\, v(x, t)$$



Witkin and Baraff

Errors and Instability

Solving by numerical integration with finite differences leads to two problems

Errors

- Errors at each time step accumulate. Accuracy decreases as simulation proceeds
- Accuracy may not be critical in graphics applications

Instability

- Errors can compound, causing the simulation to diverge even when the underlying system does not
- Lack of stability is a fundamental problem in simulation, and cannot be ignored

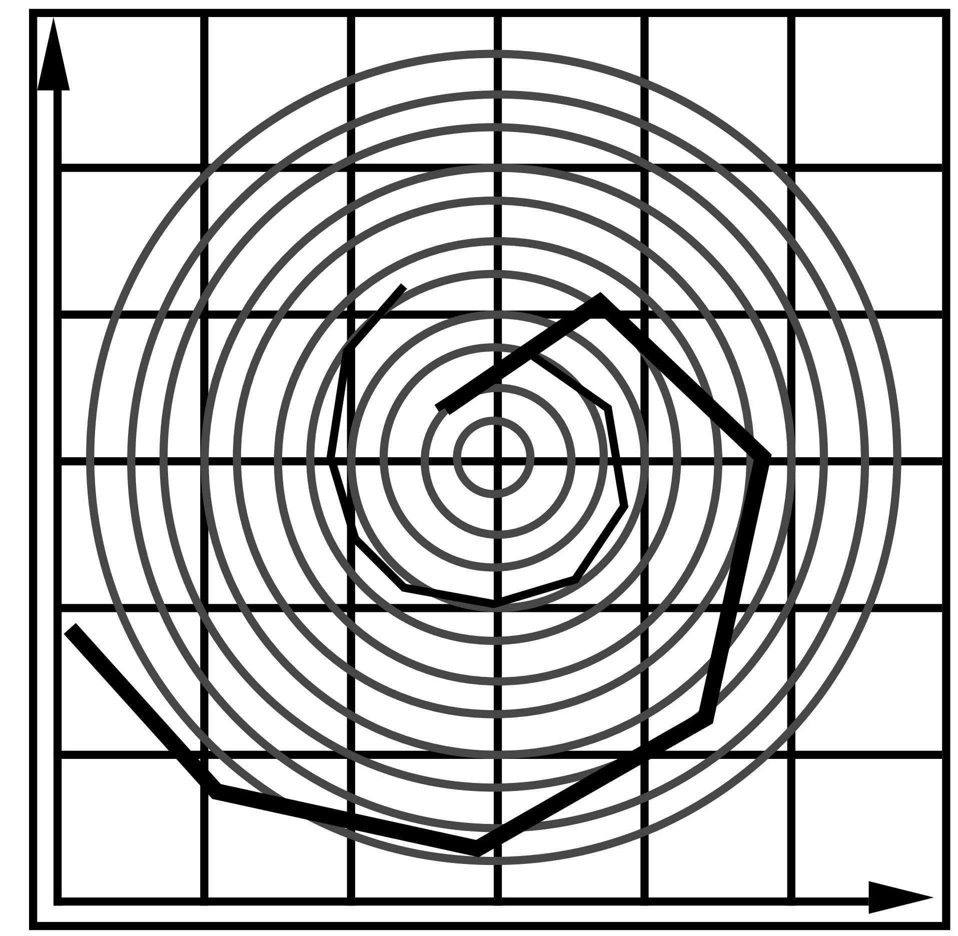
Instability of Forward Euler Method

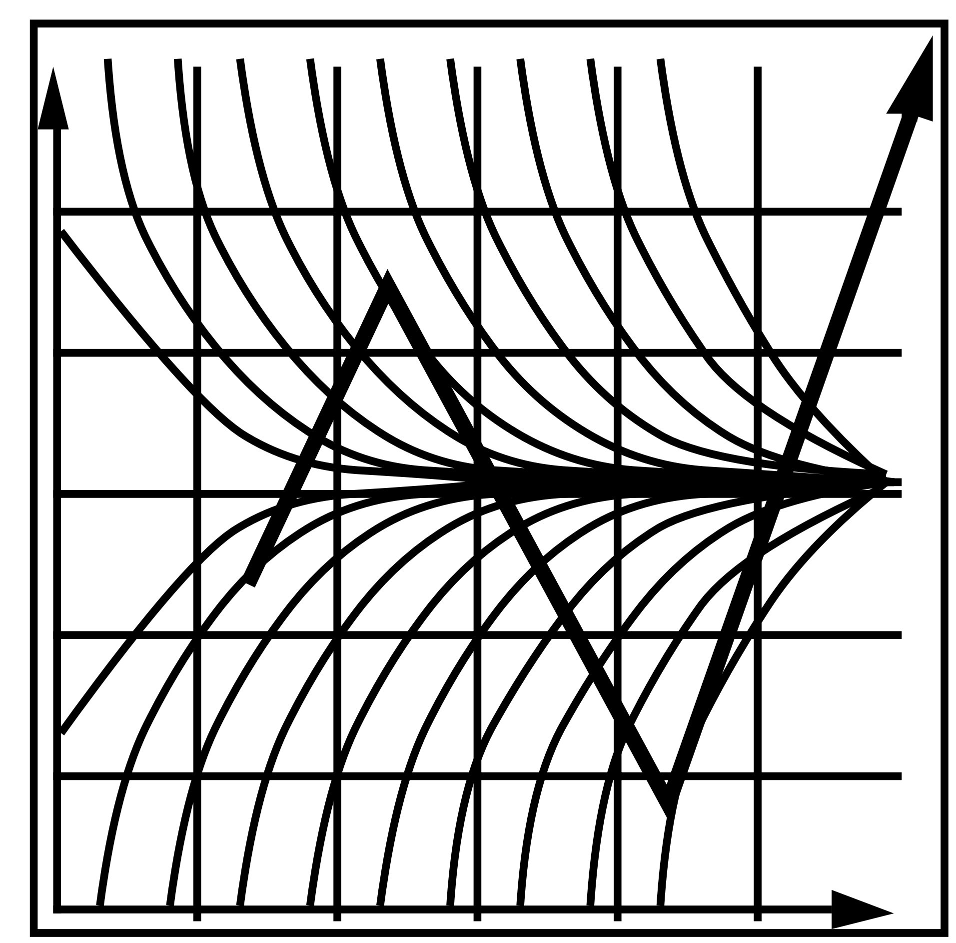
Forward Euler (explicit)

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t\, \boldsymbol{v}(\boldsymbol{x}, t)$$

Two key problems:

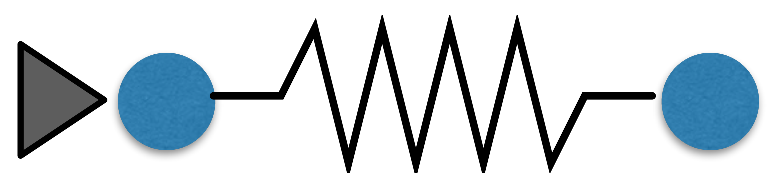
- Inaccuracies increase as time step Δt increases
- Instability is a common, serious problem that can cause simulation to diverge





Witkin and Baraff

Instability Example (Spring)



$$\boldsymbol{f}_{a \rightarrow b} = k_s(\boldsymbol{b} - \boldsymbol{a})$$

When mass is moving inward:

- Force is decreasing
- Each time-step overestimates the velocity change (increases energy)

When mass gets to origin

- Has velocity that is too high, now traveling outward

When mass is moving outward

- Force is increasing
- Each time-step underestimates the velocity change (increases energy)

At each motion cycle, mass gains energy exponentially

Combating Instability

Some Methods to Combat Instability

Modified Euler

- Average velocities at start and endpoint

Adaptive step size

- Compare one step and two half-steps, recursively, until error is acceptable

Implicit methods

- Use the velocity at the next time step (hard)

Position-based / Verlet integration

- Constrain positions and velocities of particles after time step

Modified Euler

Modified Euler

- Average velocity at start and end of step
- OK if system is not very stiff (k_s small enough)
- But, still unstable

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \frac{\Delta t}{2}\left(\dot{\boldsymbol{x}}^t + \dot{\boldsymbol{x}}^{t+\Delta t}\right)$$

$$\dot{\boldsymbol{x}}^{t+\Delta t} = \dot{\boldsymbol{x}}^t + \Delta t\, \ddot{\boldsymbol{x}}^t$$

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t\, \dot{\boldsymbol{x}}^t + \frac{(\Delta t)^2}{2}\, \ddot{\boldsymbol{x}}^t$$

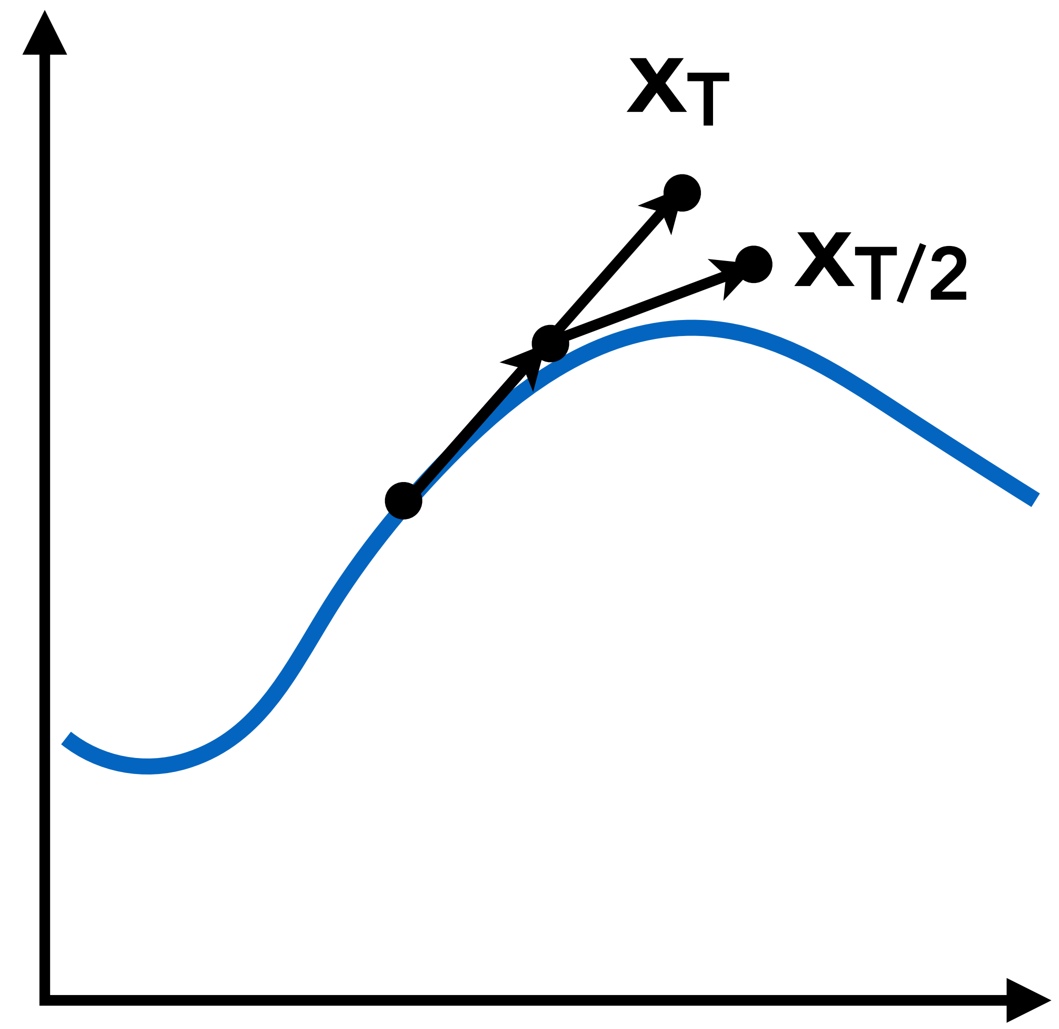
Adaptive Step Size

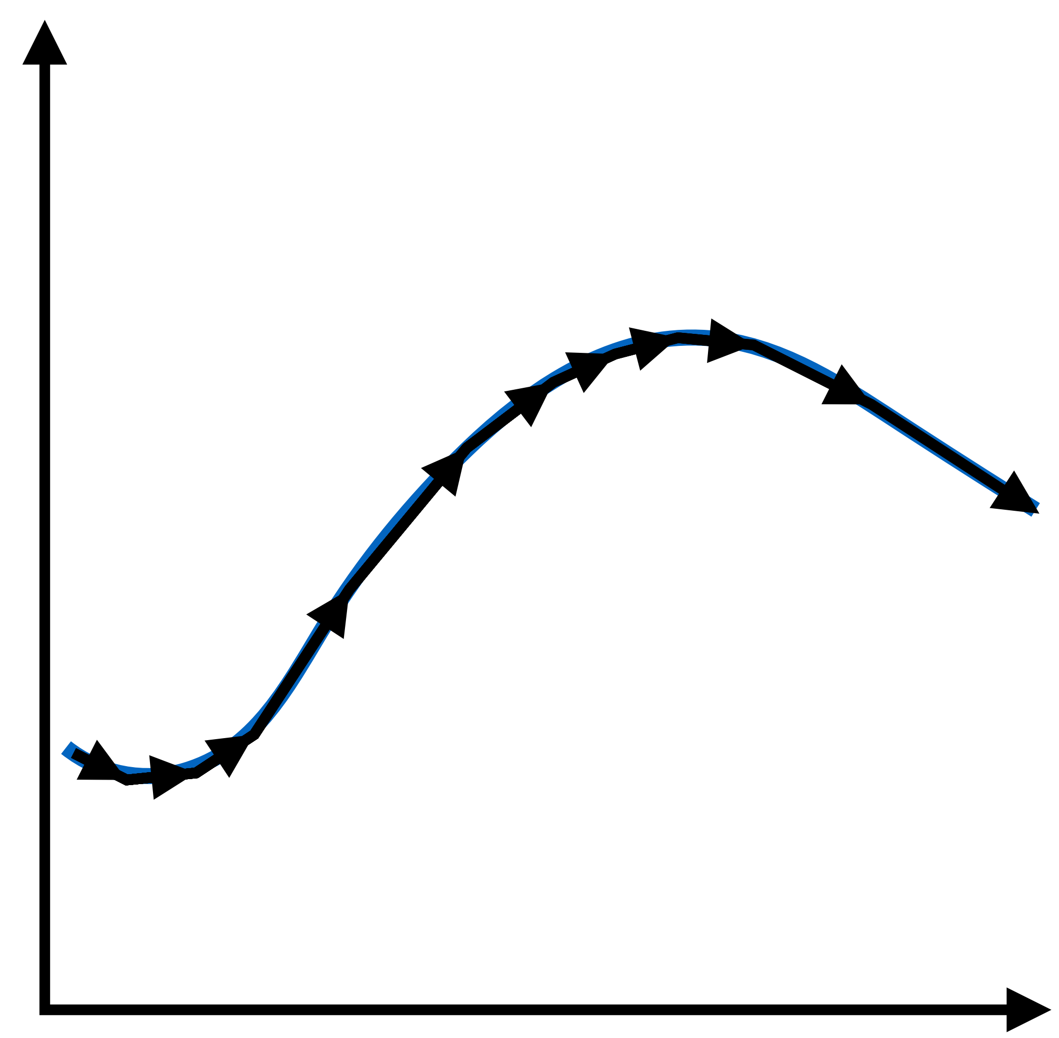
Adaptive step size

- Technique for choosing step size based on error estimate
- Highly recommended technique
- But may need very small steps!

Repeat until error is below threshold:

- Compute x_T an Euler step, size T
- Compute $x_{T/2}$ two Euler steps, size T/2
- Compute error $\| x_T - x_{T/2} \|$
- If (error > threshold) reduce step size and try again





Slide credit: Funkhouser

Implicit Euler Method

Implicit methods

- Informally called backward methods
- Use derivatives in the future, for the current step

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t \, \dot{\boldsymbol{x}}^{t+\Delta t}$$
$$\dot{\boldsymbol{x}}^{t+\Delta t} = \dot{\boldsymbol{x}}^t + \Delta t \, \ddot{\boldsymbol{x}}^{t+\Delta t}$$

$$\dot{\boldsymbol{x}}^{t+\Delta t} = \mathsf{V}(\boldsymbol{x}^{t+\Delta t}, \dot{\boldsymbol{x}}^{t+\Delta t}, t + \Delta t)$$
$$\ddot{\boldsymbol{x}}^{t+\Delta t} = \mathsf{A}(\boldsymbol{x}^{t+\Delta t}, \dot{\boldsymbol{x}}^{t+\Delta t}, t + \Delta t)$$

Implicit Euler Method

Implicit methods

- Informally called backward methods
- Use derivatives in the future, for the current step

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t \, \mathsf{V}(\boldsymbol{x}^{t+\Delta t}, \dot{\boldsymbol{x}}^{t+\Delta t}, t + \Delta t)$$

$$\dot{\boldsymbol{x}}^{t+\Delta t} = \dot{\boldsymbol{x}}^t + \Delta t \, \mathsf{A}(\boldsymbol{x}^{t+\Delta t}, \dot{\boldsymbol{x}}^{t+\Delta t}, t + \Delta t)$$

- Solve nonlinear problem for $x^{t+\Delta t}$ and $\dot{x}^{t+\Delta t}$
- Use root-finding algorithm, e.g. Newton's method
- Can be made unconditionally stable

Position-Based / Verlet Integration

Idea:

- After modified Euler forward-step, constrain positions of particles to prevent divergent, unstable behavior
- Use constrained positions to calculate velocity
- Both of these ideas will dissipate energy, stabilize

Pros / cons

- Fast and simple
- Not physically based, dissipates energy (error)
- Highly recommended (assignment)

Position-Based / Verlet Integration

Algorithm 1 Position-based dynamics

for all vertices i **do**
  initialize $\mathbf{x}_i = \mathbf{x}_i^0$, $\mathbf{v}_i = \mathbf{v}_i^0$, $w_i = 1/m_i$
end for
loop
  for all vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{\text{ext}}(\mathbf{x}_i)$
  for all vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  for all vertices i **do** genCollConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
  loop solverIteration **times**
    projectConstraints($C_1, \ldots, C_{M+M_{\text{Coll}}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  end loop
  for all vertices i **do**
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  end for
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
end loop

Position-Based Simulation Methods in Computer Graphics
Bender, Müller, Macklin, Eurographics 2015

Particle Systems

Particle Systems

Model dynamical systems as collections of large numbers of particles

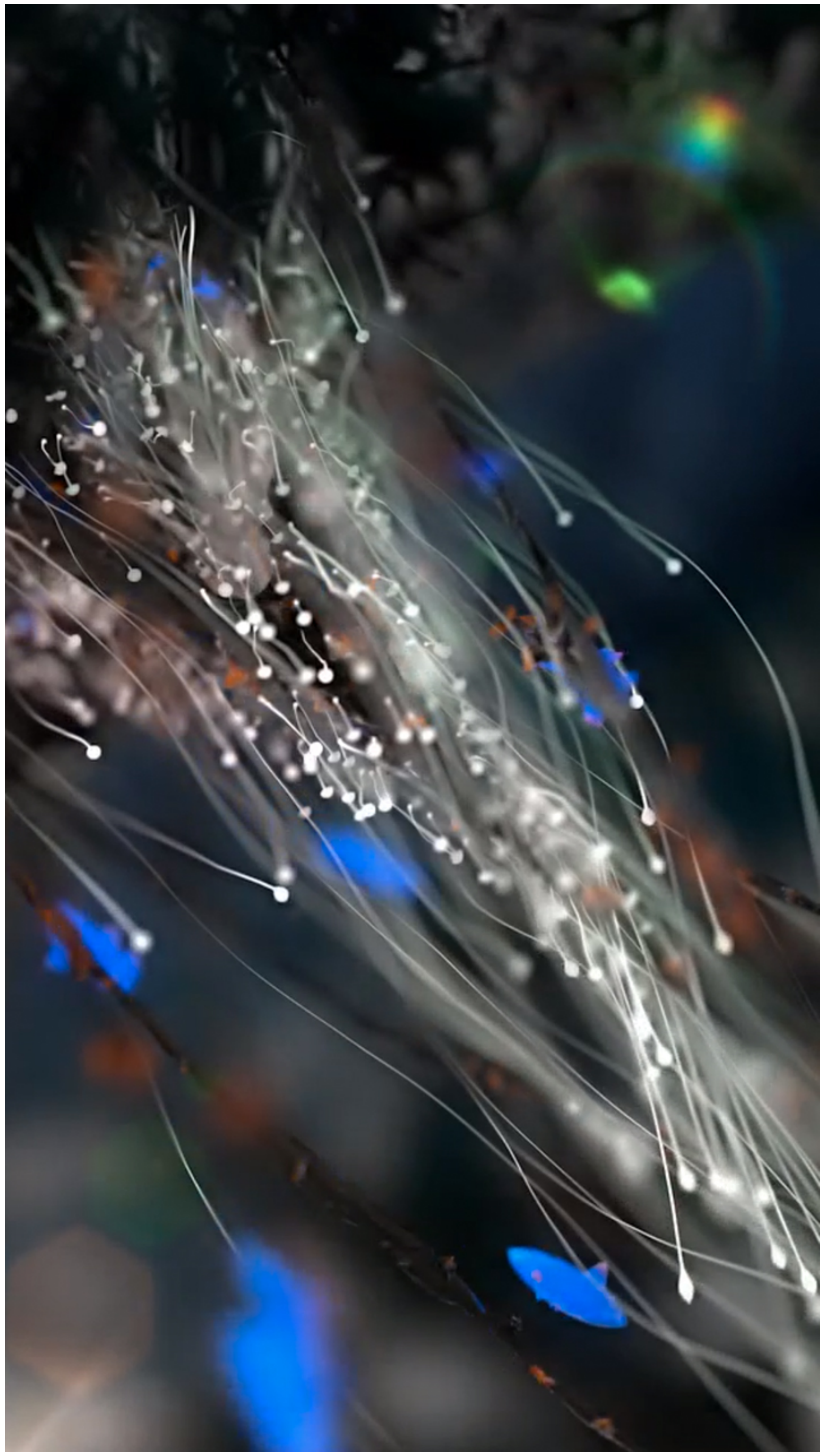
Each particle's motion is defined by a set of physical (or non-physical) forces

Popular technique in graphics and games

- Easy to understand, implement
- Scalable: fewer particles for speed, more for higher complexity

Challenges

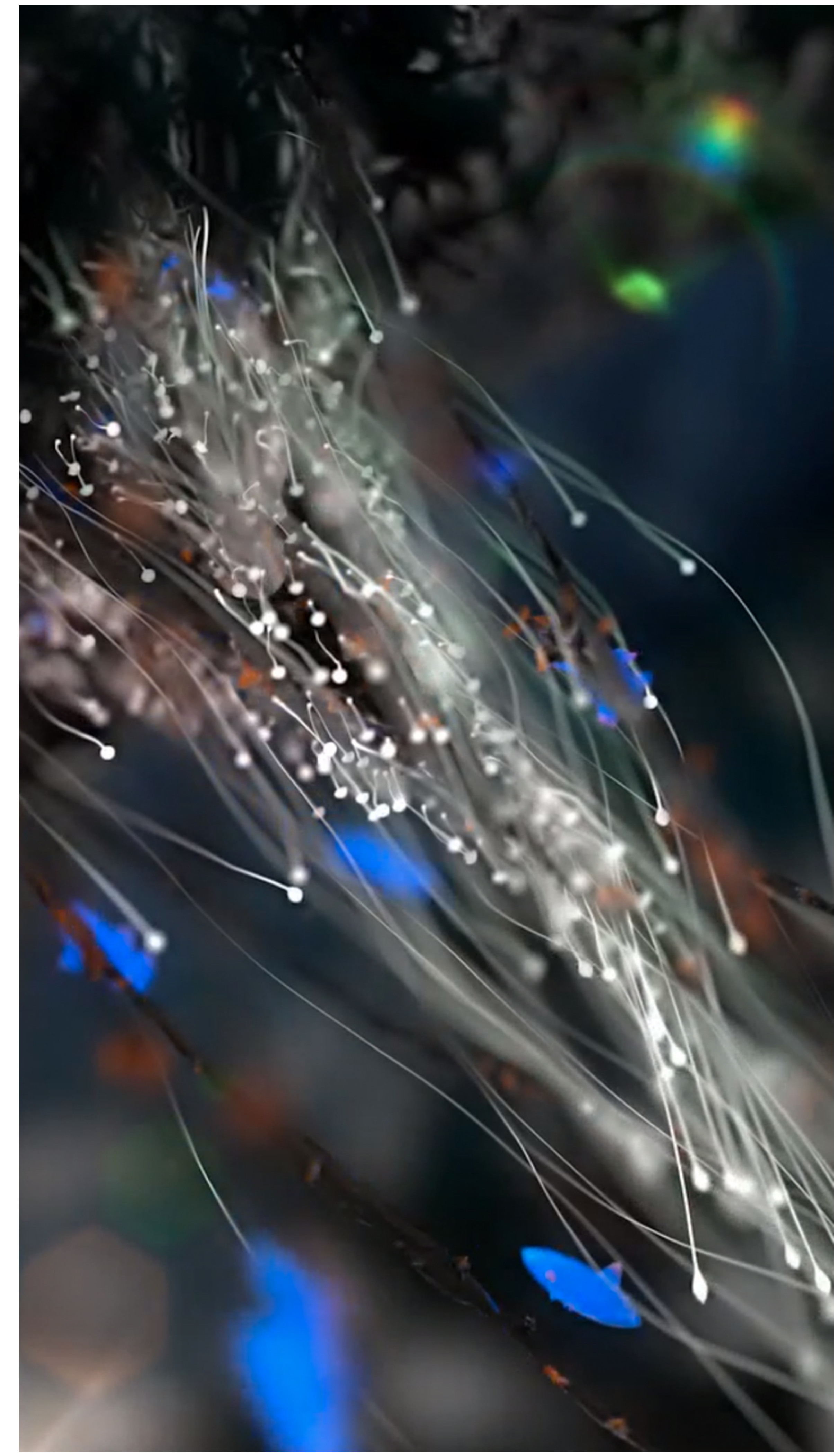
- May need *many* particles (e.g. fluids)
- May need acceleration structures (e.g. to find nearest particles for interactions)



Particle System Animations

For each frame in animation

- [If needed] Create new particles
- Calculate forces on each particle
- Update each particle's position and velocity
- [If needed] Remove dead particles
- Render particles



Particle System Forces

Attraction and repulsion forces

- Gravity, electromagnetism, …
- Springs, propulsion, …

Damping forces

- Friction, air drag, viscosity, …

Collisions

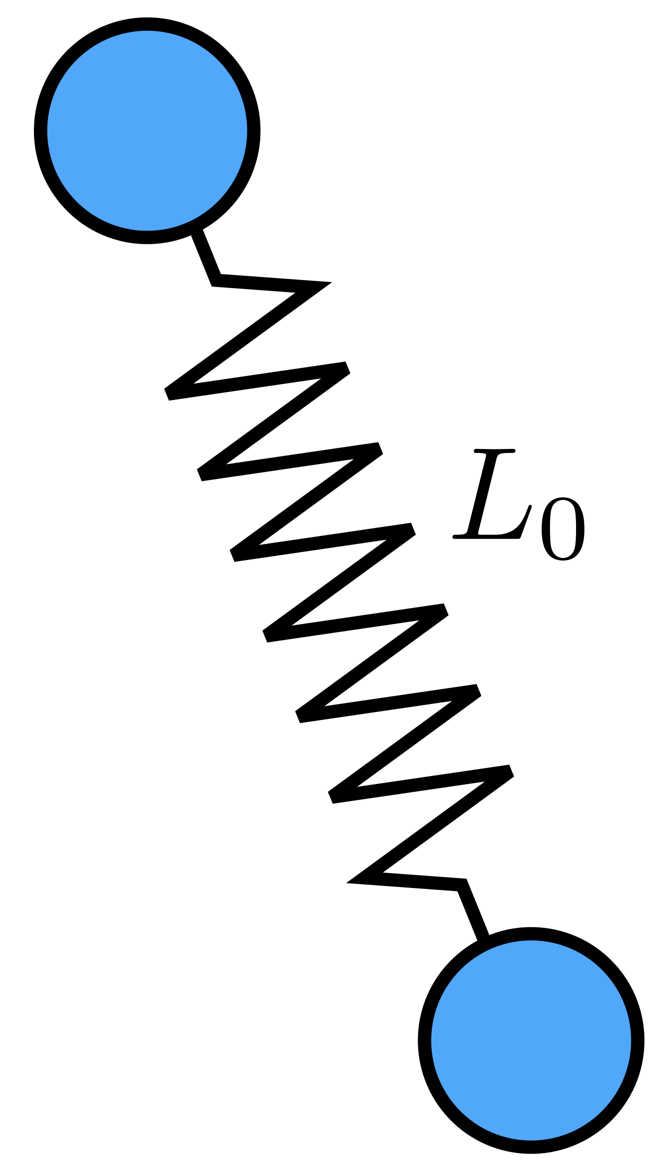
- Walls, containers, fixed objects, …
- Dynamic objects, character body parts, …

Already Discussed Springs

Internally-damped non-zero length spring

$$\boldsymbol{f}_{a \to b} = k_s \frac{\boldsymbol{b} - \boldsymbol{a}}{||\boldsymbol{b} - \boldsymbol{a}||} (||\boldsymbol{b} - \boldsymbol{a}|| - l)$$

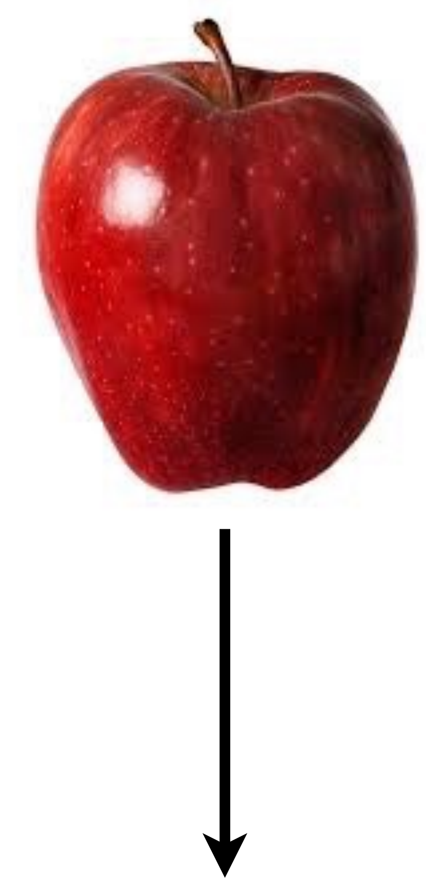
$$- k_d \frac{\boldsymbol{b} - \boldsymbol{a}}{||\boldsymbol{b} - \boldsymbol{a}||} (\dot{\boldsymbol{b}} - \dot{\boldsymbol{a}}) \cdot \frac{\boldsymbol{b} - \boldsymbol{a}}{||\boldsymbol{b} - \boldsymbol{a}||}$$



Simple Gravity

Gravity at earth's surface due to earth

- F = –mg
- m is mass of object
- g is gravitational acceleration,
  $g = -9.8 m/s^2$



$$F_g = -mg$$

$$g = (0, 0, -9.8)\,\mathrm{m/s^2}$$

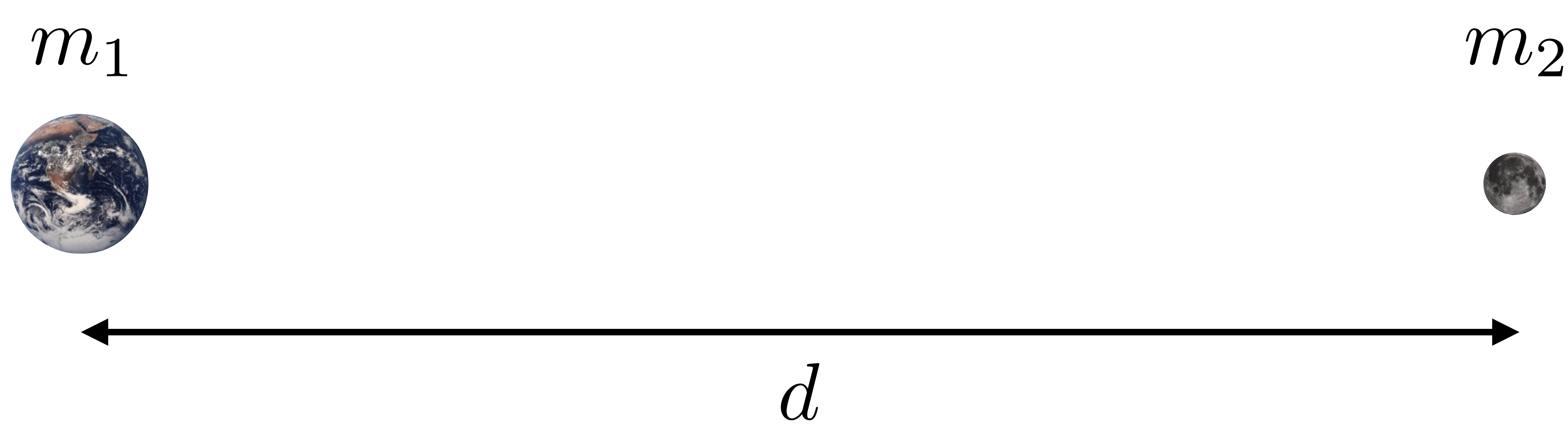
Gravitational Attraction

Newton's universal law of gravitation

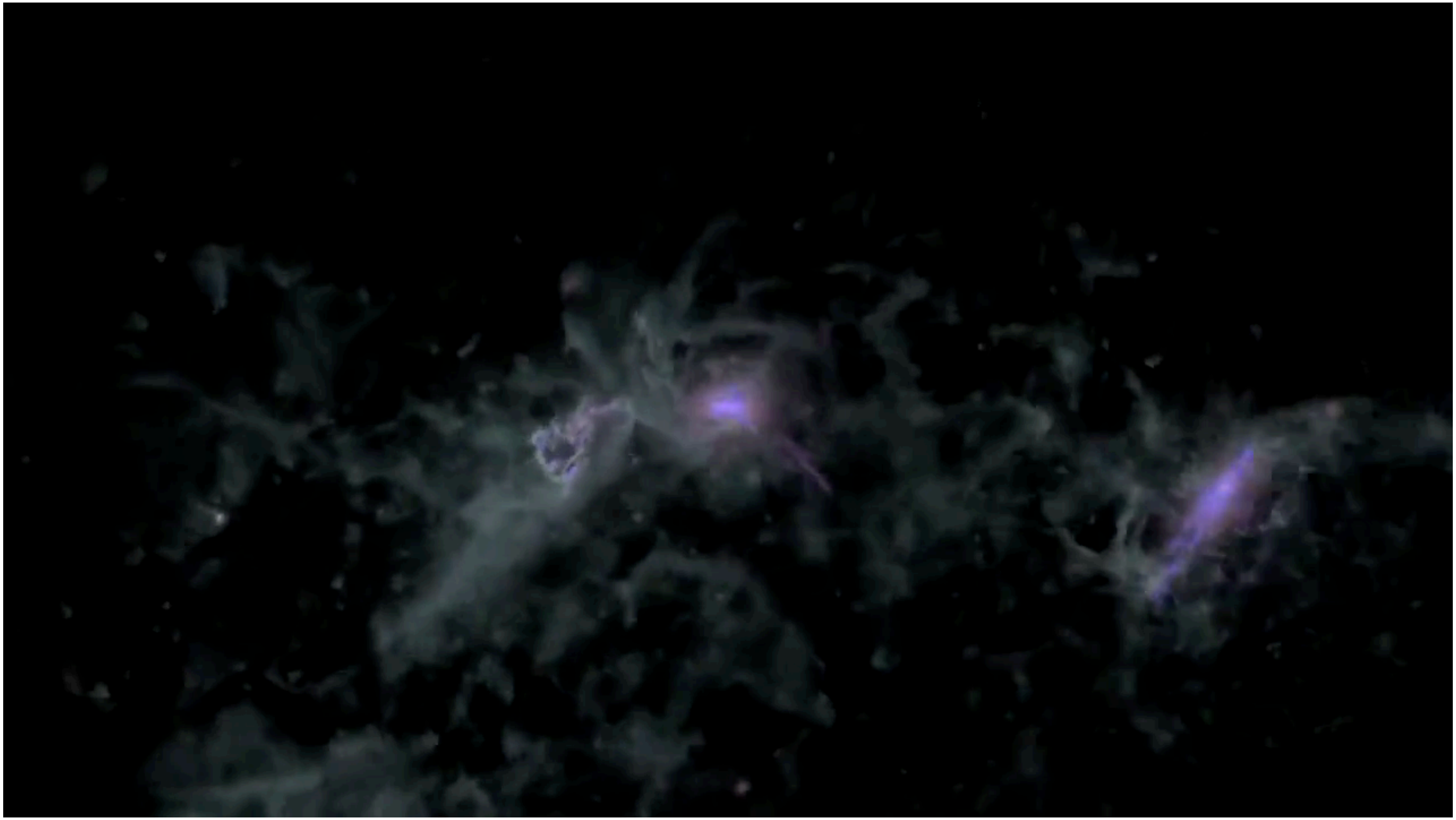
- Gravitational pull between particles

$$F_g = G\frac{m_1 m_2}{d^2}$$

$$G = 6.67428 \times 10^{-11}\,\mathrm{Nm^2kg^{-2}}$$

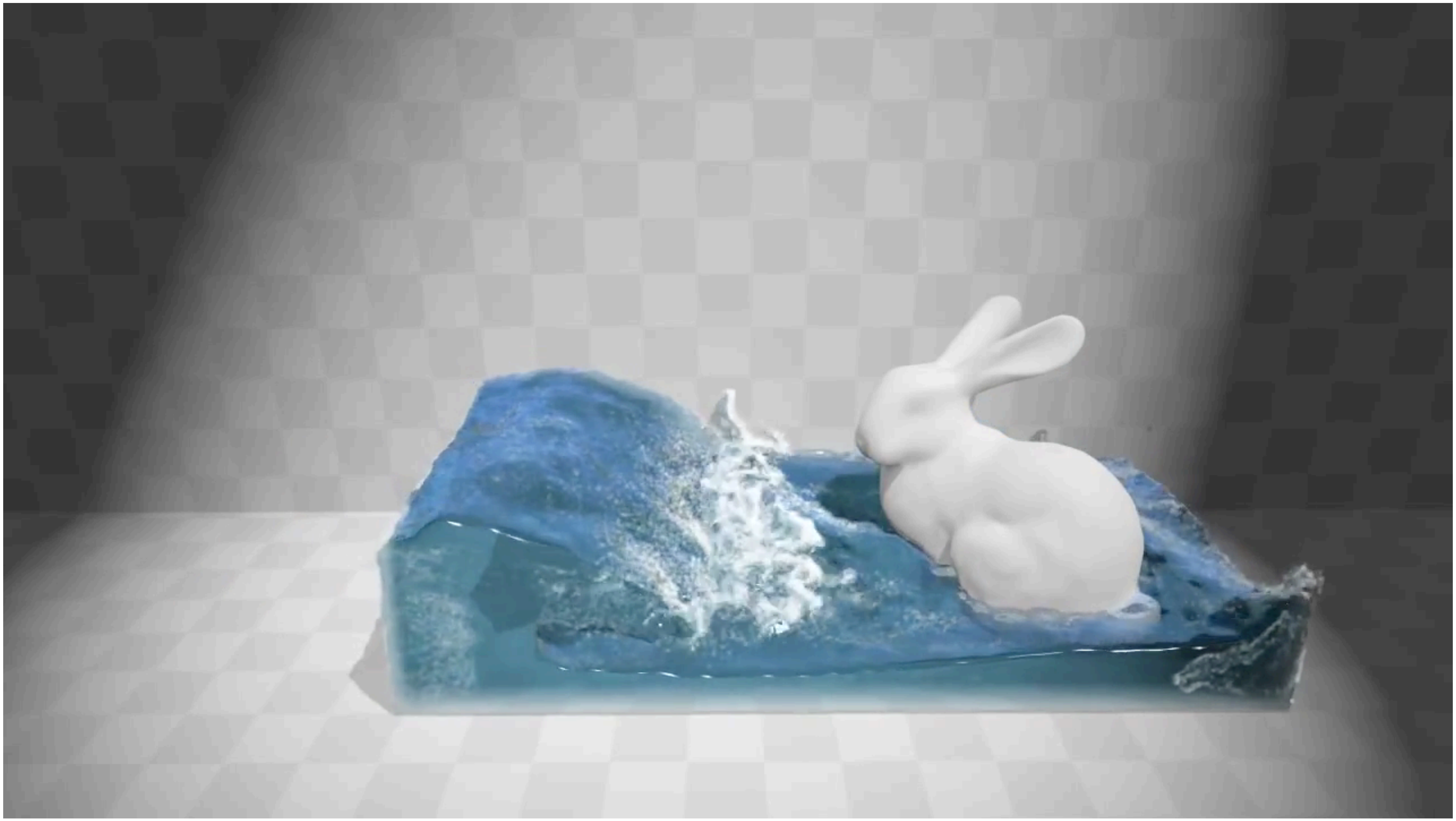


Example: Galaxy Simulation



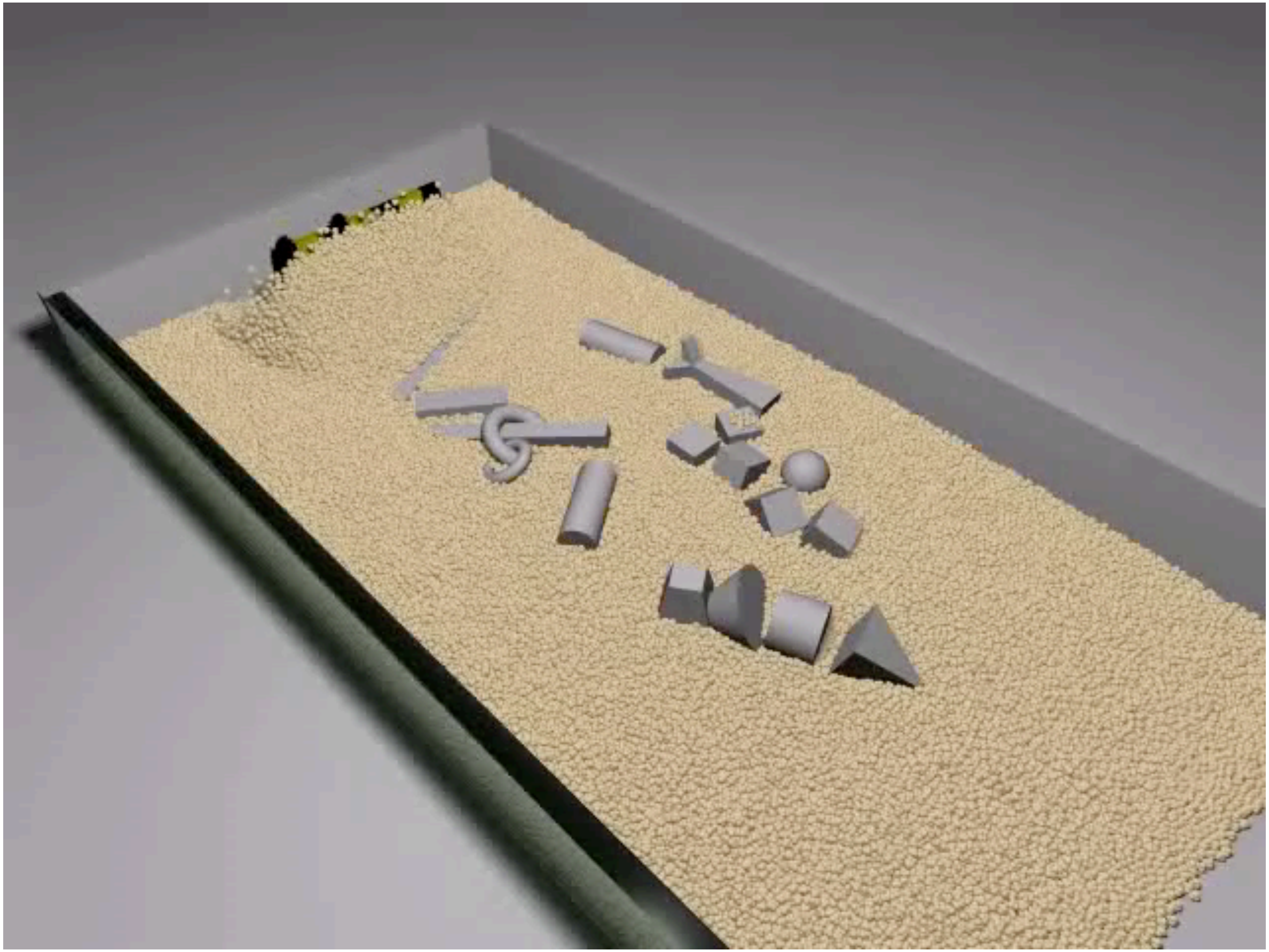
Disk galaxy simulation, NASA Goddard

Example: Particle-Based Fluids



Macklin and Müller, Position Based Fluids , TOG 2013

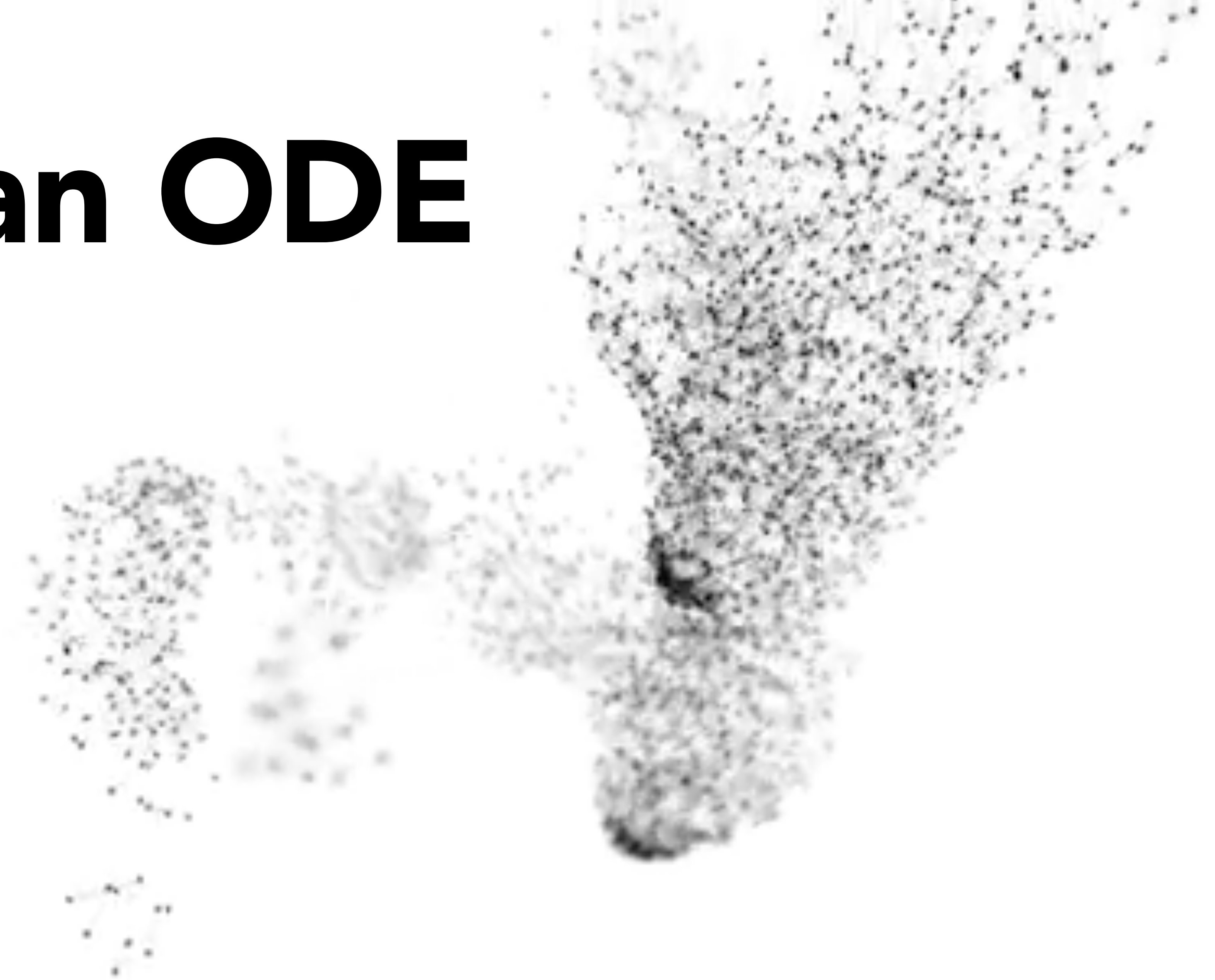
Example: Granular Materials



Bell et al, "Particle-Based Simulation of Granular Materials"

Example: Flocking Birds

Simulated Flocking as an ODE



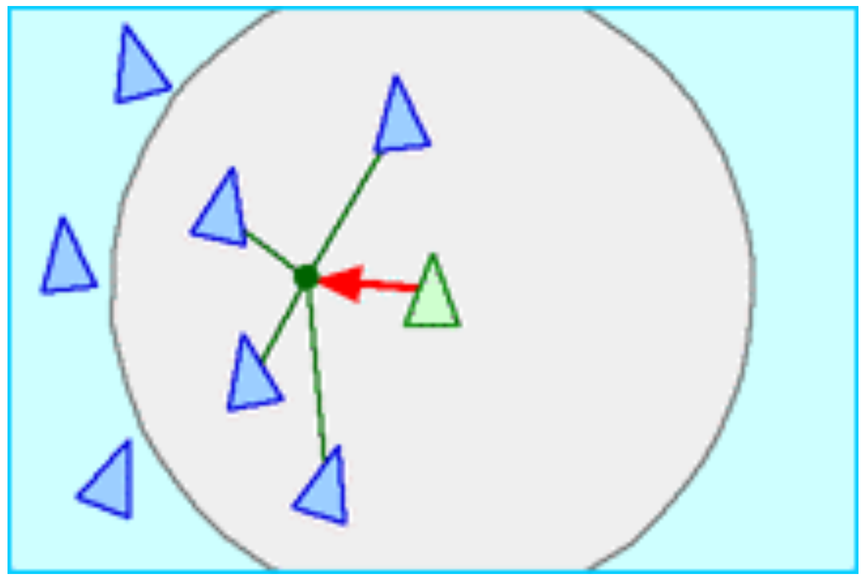
Model each bird as a particle

Subject to very simple forces:

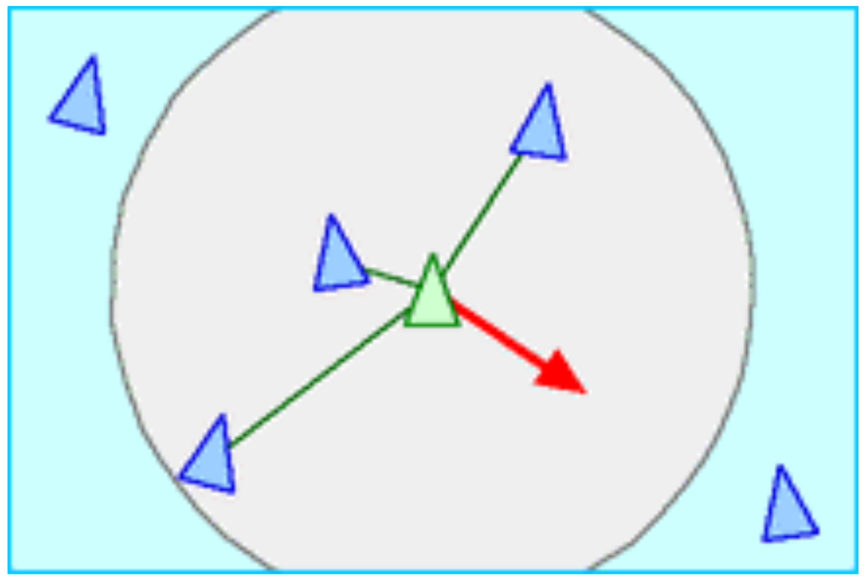
- attraction to center of neighbors
- repulsion from individual neighbors
- alignment toward average trajectory of neighbors

Simulate evolution of large particle system numerically

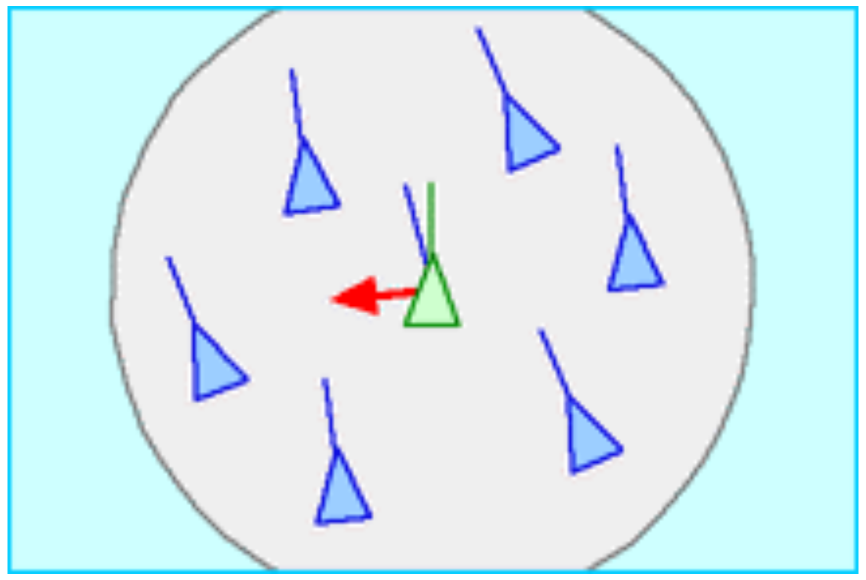
Emergent complex behavior (also seen in fish, bees, ...)



attraction



repulsion



alignment

Credit: Craig Reynolds (see http://www.red3d.com/cwr/boids/)

Slide credit: Keenan Crane

Example: Crowds

Where are the bottlenecks in a building plan?

Example: Crowds + "Rock" Dynamics

Suggested Reading

Physically Based Modeling: Principles and Practice

- Andy Witkin and David Baraff
- http://www-2.cs.cmu.edu/~baraff/sigcourse/index.html

Numerical Recipes in C++

- Chapter 16

Any good text on integrating ODE's

Just Scratching the Surface…

Physical simulation is a huge field in graphics, engineering, science

Today: intro to particle systems, solving ODEs

Partial differential equations

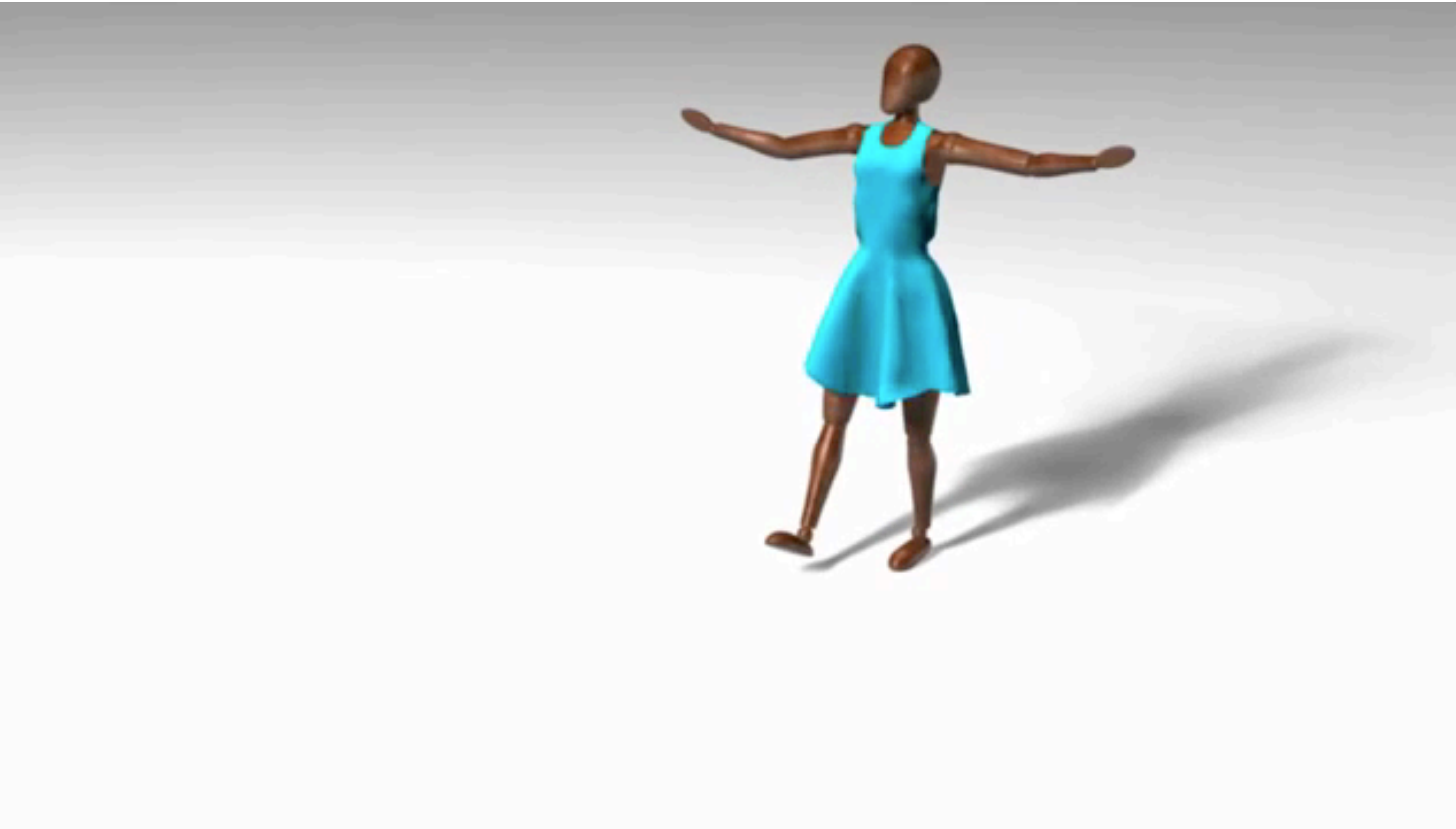
- Diffusion equation, heat equation, …
- Used in graphics for liquids, smoke, fire, etc.

Rigid body

Simulation of sound

…

Example: Mass Spring Dress + Character



FEM (Finite Element Method) Instead of Springs

Things to Remember

Physical simulation = mathematical modeling of dynamical systems & solution by numerical integration

Particle systems

- Flexible force modeling, e.g. spring-mass sytems, gravitational attraction, fluids, flocking behavior
- Newtonian equations of motion = ODEs
- Solution by numerical integration of ODEs: Explicit Euler, Implicit Euler, Adaptive, Position-Based / Verlet
- Error and instability, methods to combat instability

Acknowledgments

Many thanks to James O'Brien, Keenan Crane and Tom Funkhouser for lecture resources.

Example: Fluids

Macklin and Müller, Position Based Fluids TOG 2013

Problem Setup

Lagrangian Formulation

- Where in space did this material move to?
- Commonly used for solid materials

Eulerian Formulation

- What material is at this location in space?
- Commonly used for fluids
  - Why: Because fluids don't remember their shape

Problem Discretization

Grids

- Store quantities on a grid
- Fluid move "through" grid
- Scales reasonably well to large systems
- Surface tracking is challenging

Particles

- Fluid defined by locations of particles
- Inter-particle forces create fluid behavior
- Scaling to large systems not simple
- Surface tracking less difficult

Many popular methods combine grids and particles

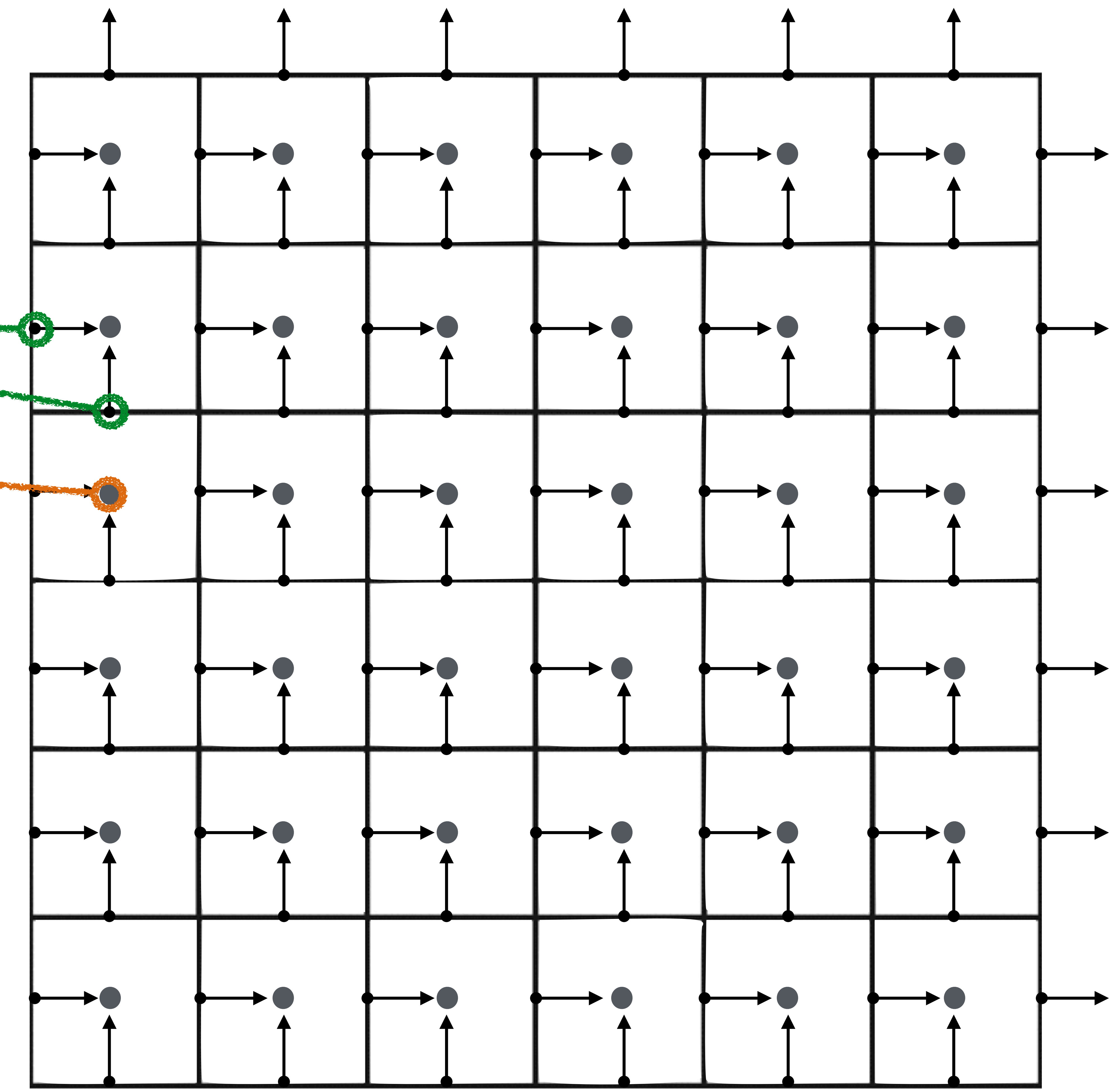
Fluid Grid

Store Fluid State On Grid

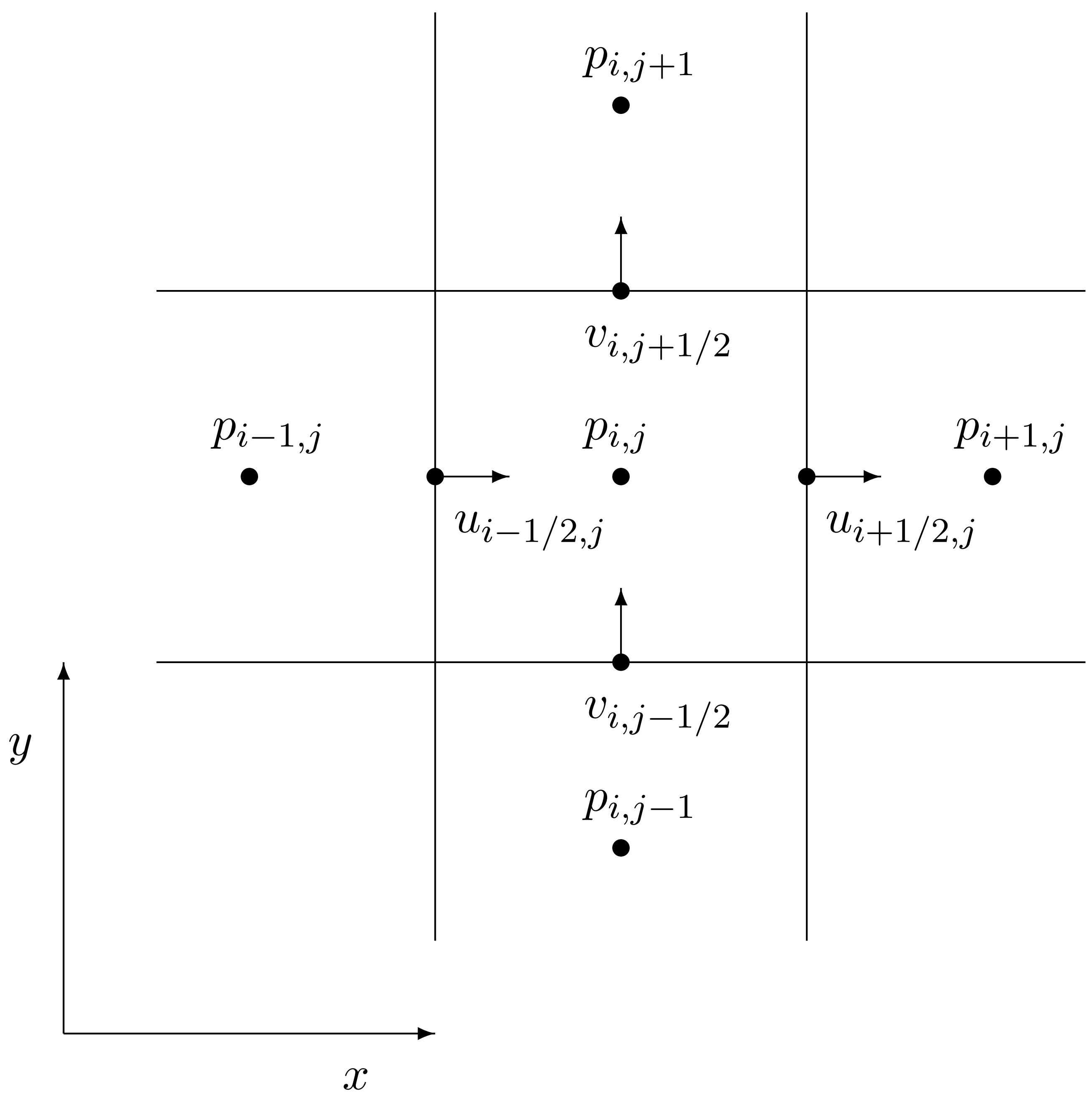
- Velocity
- Pressure
- Density

Staggered Grid

- Bilinear interpolation
- Seems odd at first
- Very useful
- Non-staggered produces unstable checkerboard

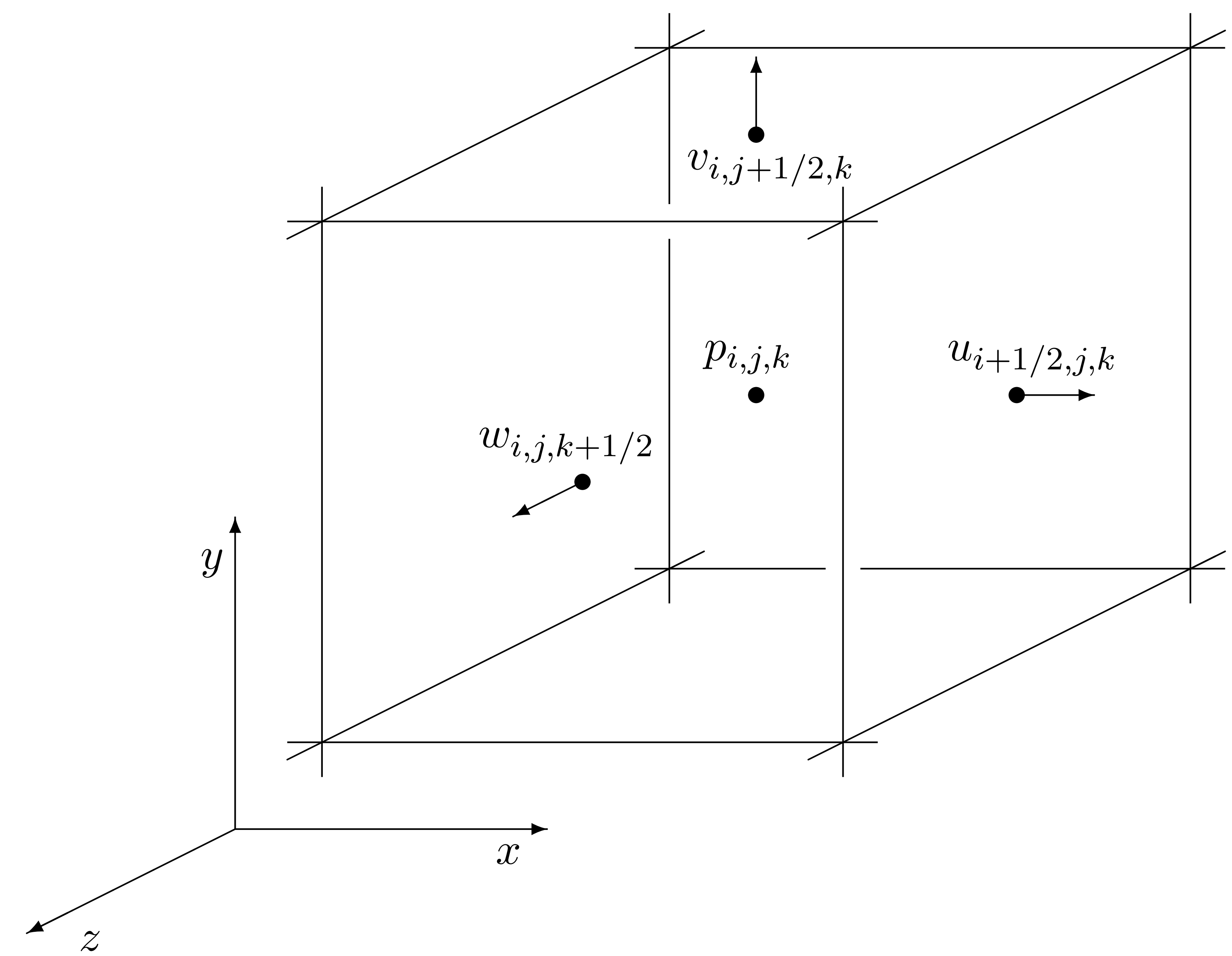


Fluid Grid



2D Staggered Grid

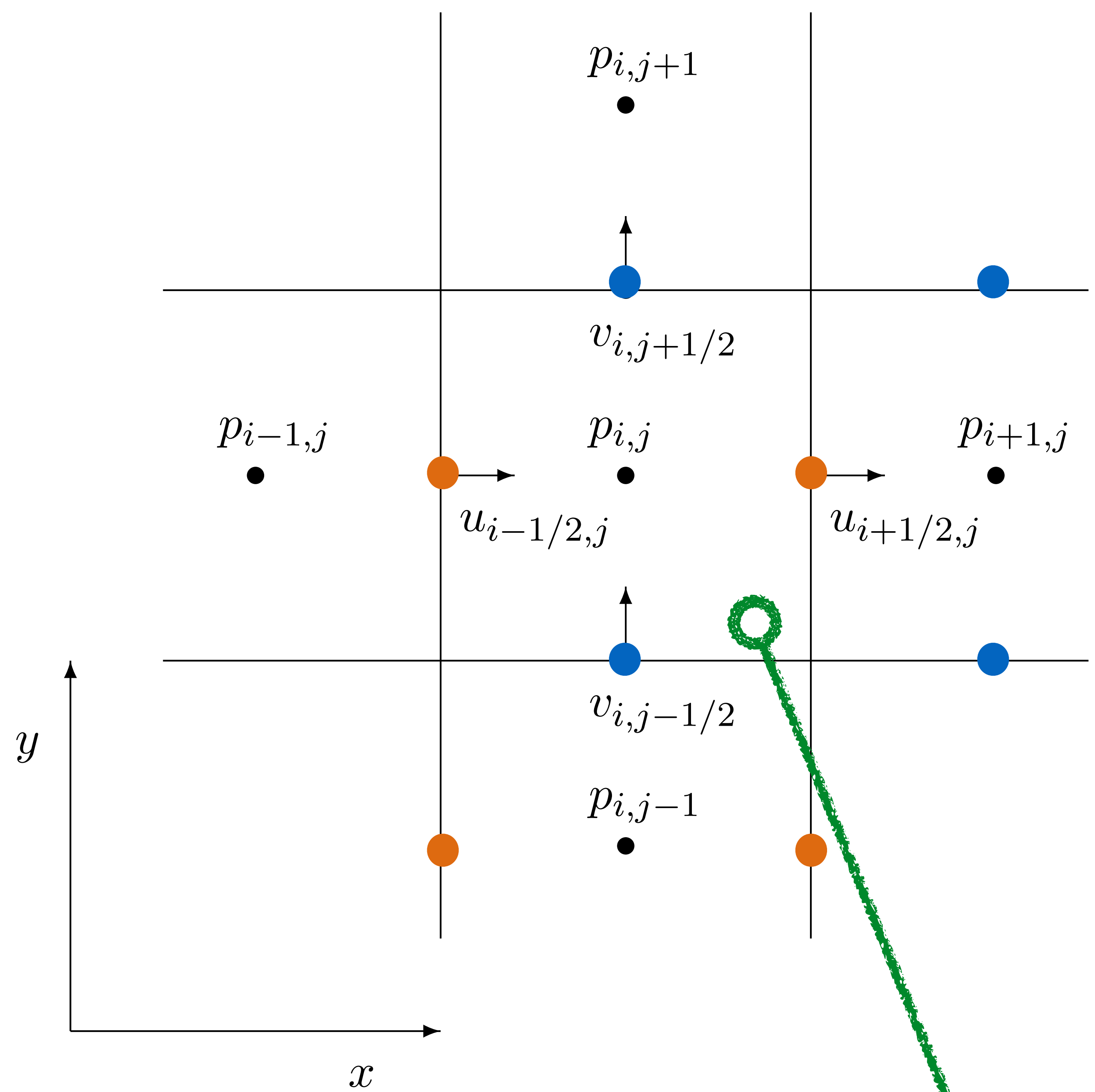
$$\mathbf{u} = \mathbf{u}(x, y)$$
$$p = p(x, y)$$



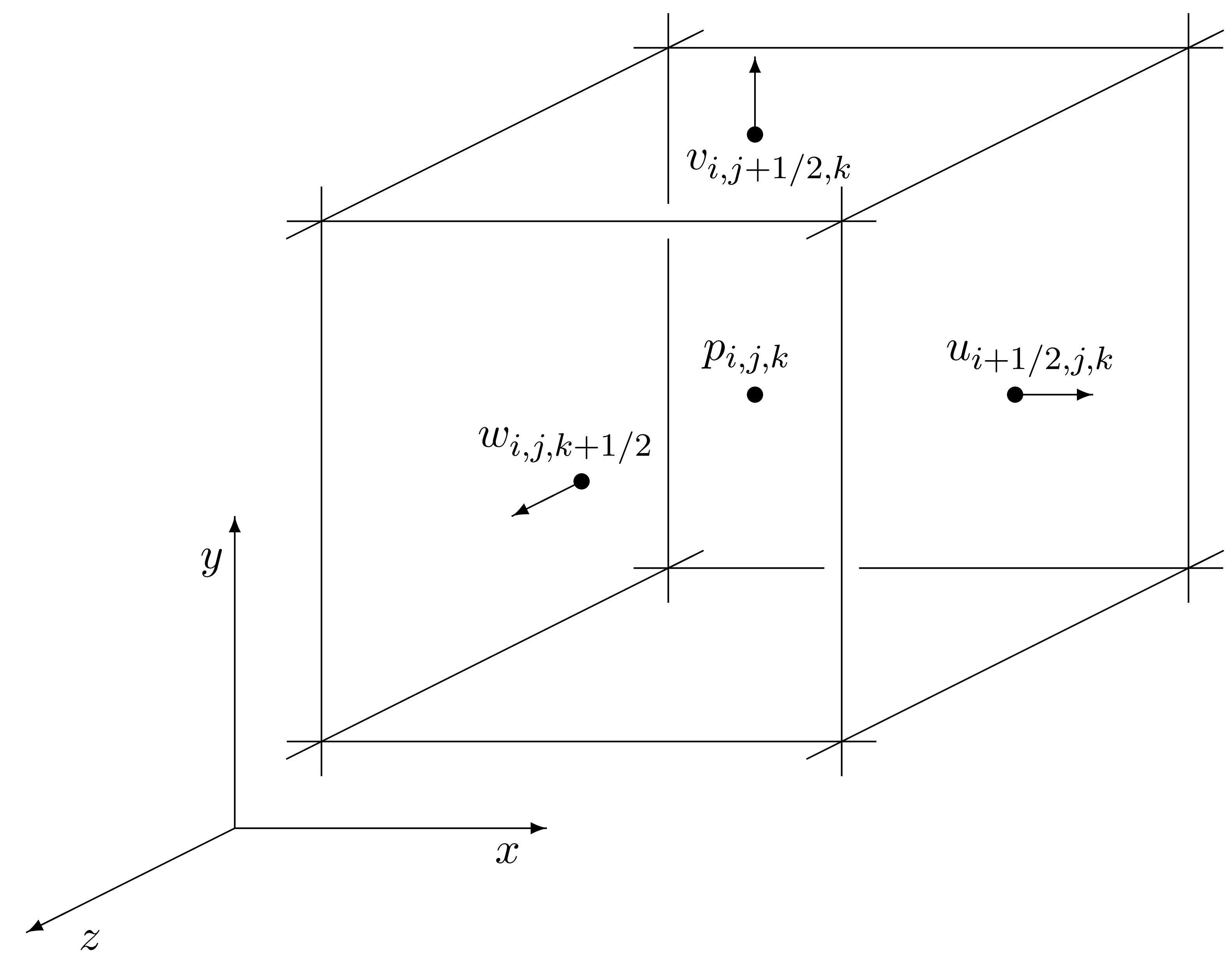
3D Staggered Grid

$$\mathbf{u} = \mathbf{u}(x, y, z)$$
$$p = p(x, y, z)$$

Fluid Grid



2D Staggered Grid



3D Staggered Grid

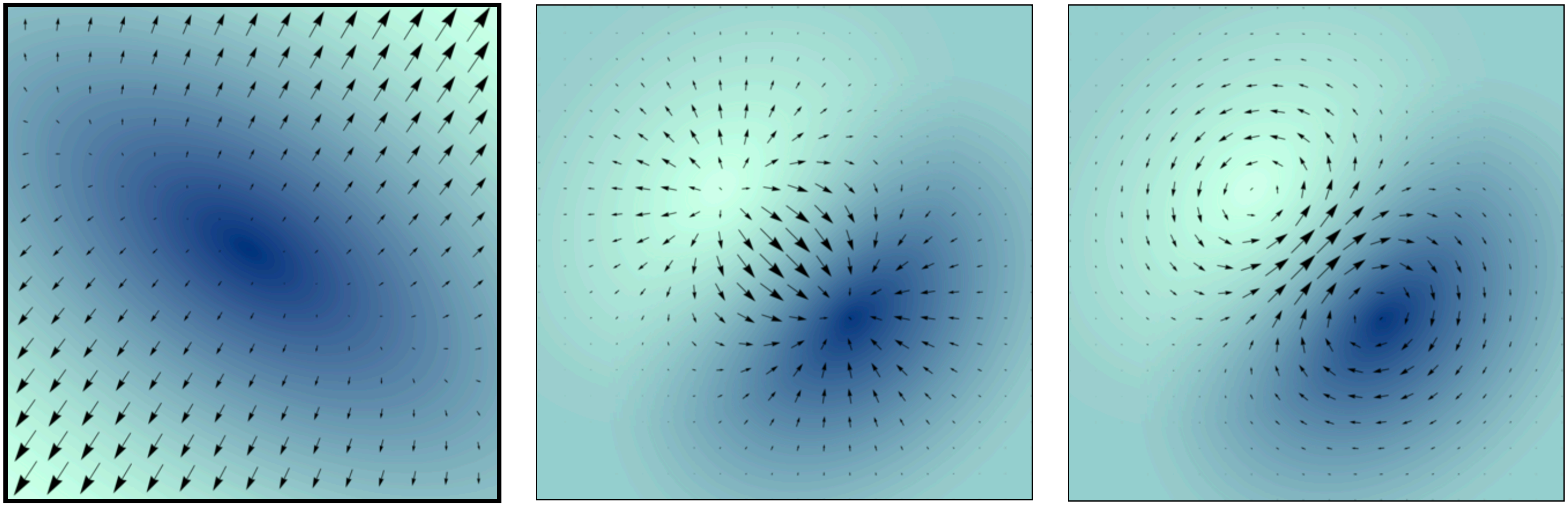
$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

$\mathbf{u}_x = u =$ **BiLinear(· , · , · , ·)**

$\mathbf{u}_y = v =$ **BiLinear(· , · , · , ·)**

Vector Fields

$$\mathbf{v} = \mathbf{v}(x, y)$$
$$p = p(x, y)$$



Gradient:

Direction of greatest change

$$\textbf{grad}(p(x,y)) = \nabla p\,|_{x,y} = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

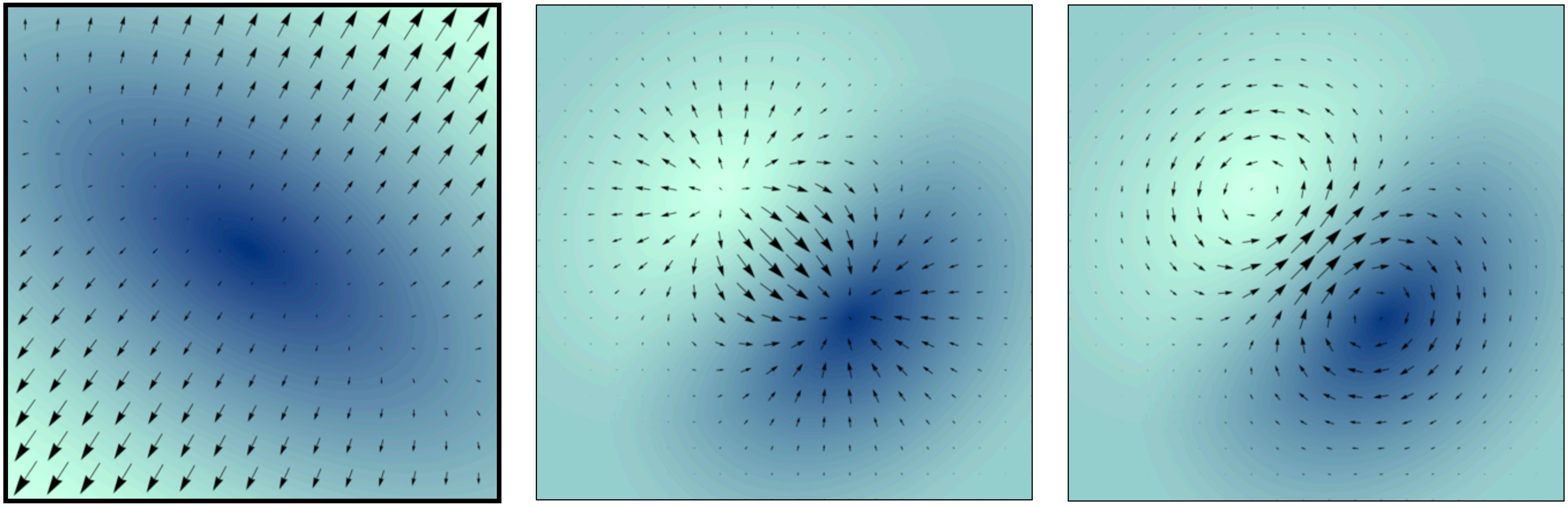
$$\textbf{grad}(p) = \nabla p$$

The ∇ is a differential operator, like $\frac{\partial}{\partial x}$, but a vector

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix}$$

Vector Fields

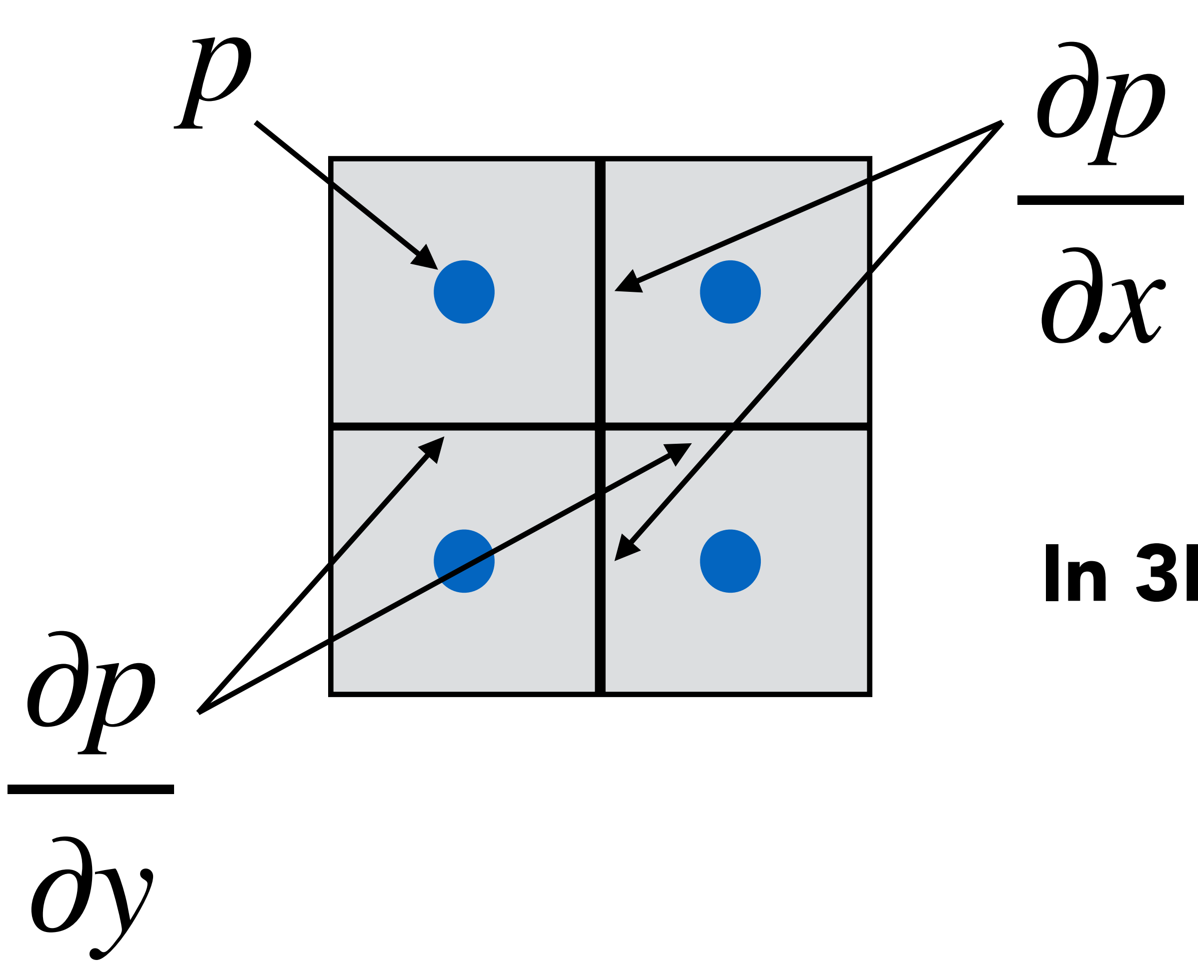
$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \qquad \mathbf{v} = \mathbf{v}(x, y) \qquad p = p(x, y)$$



Gradient:

Direction of greatest change

$$\textbf{grad}(p) = \nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

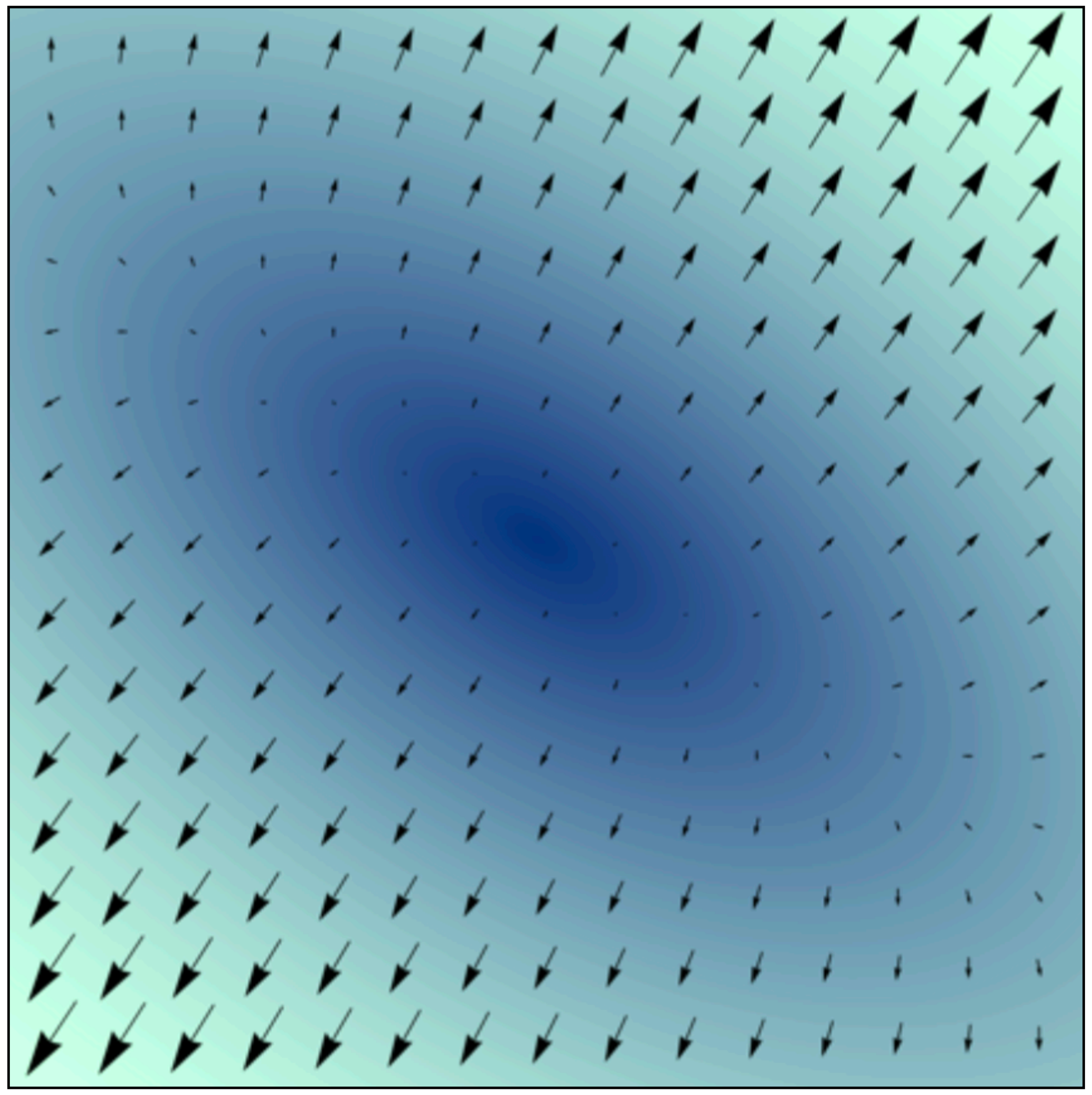


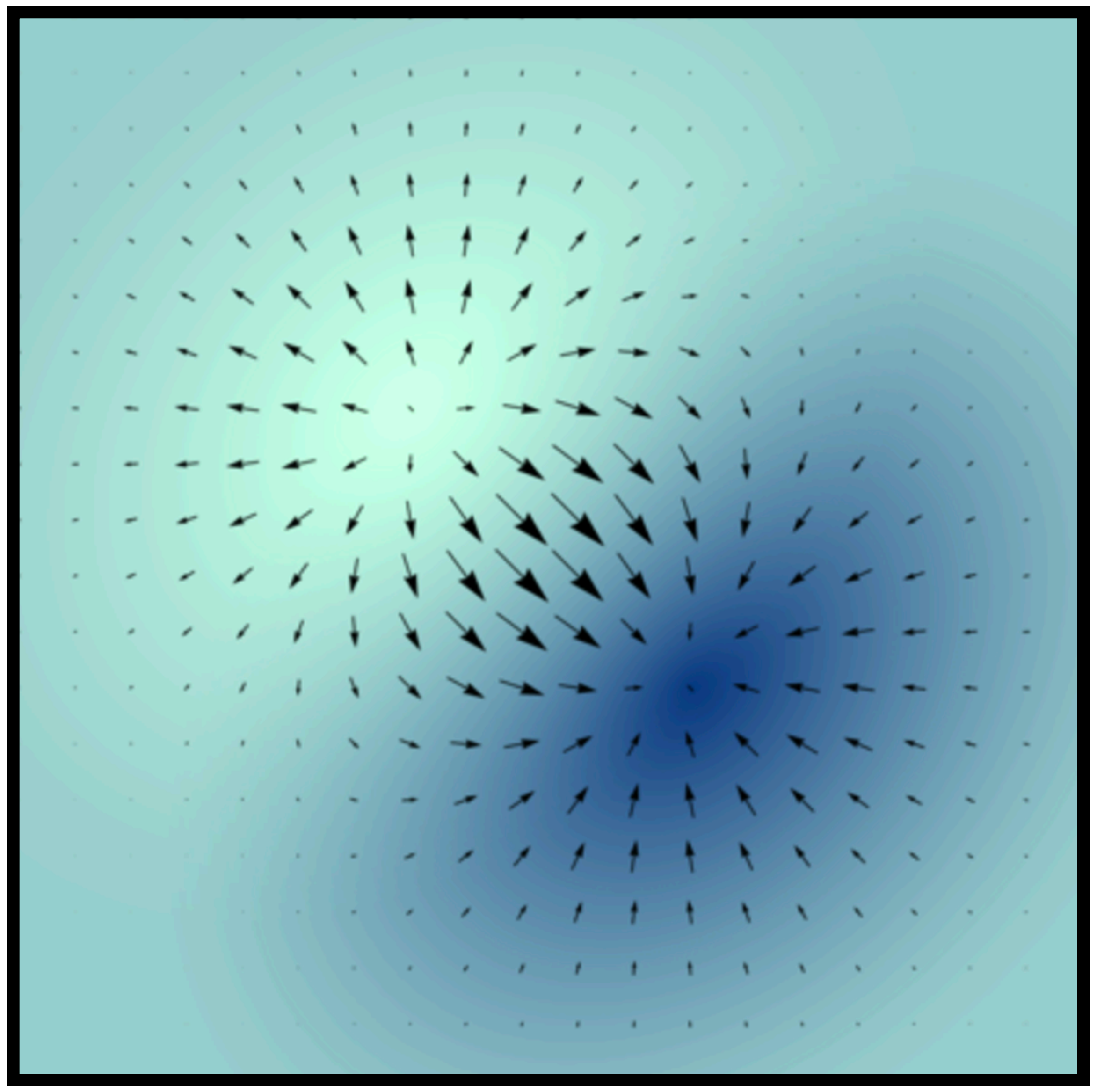
In 3D, cell centers and faces

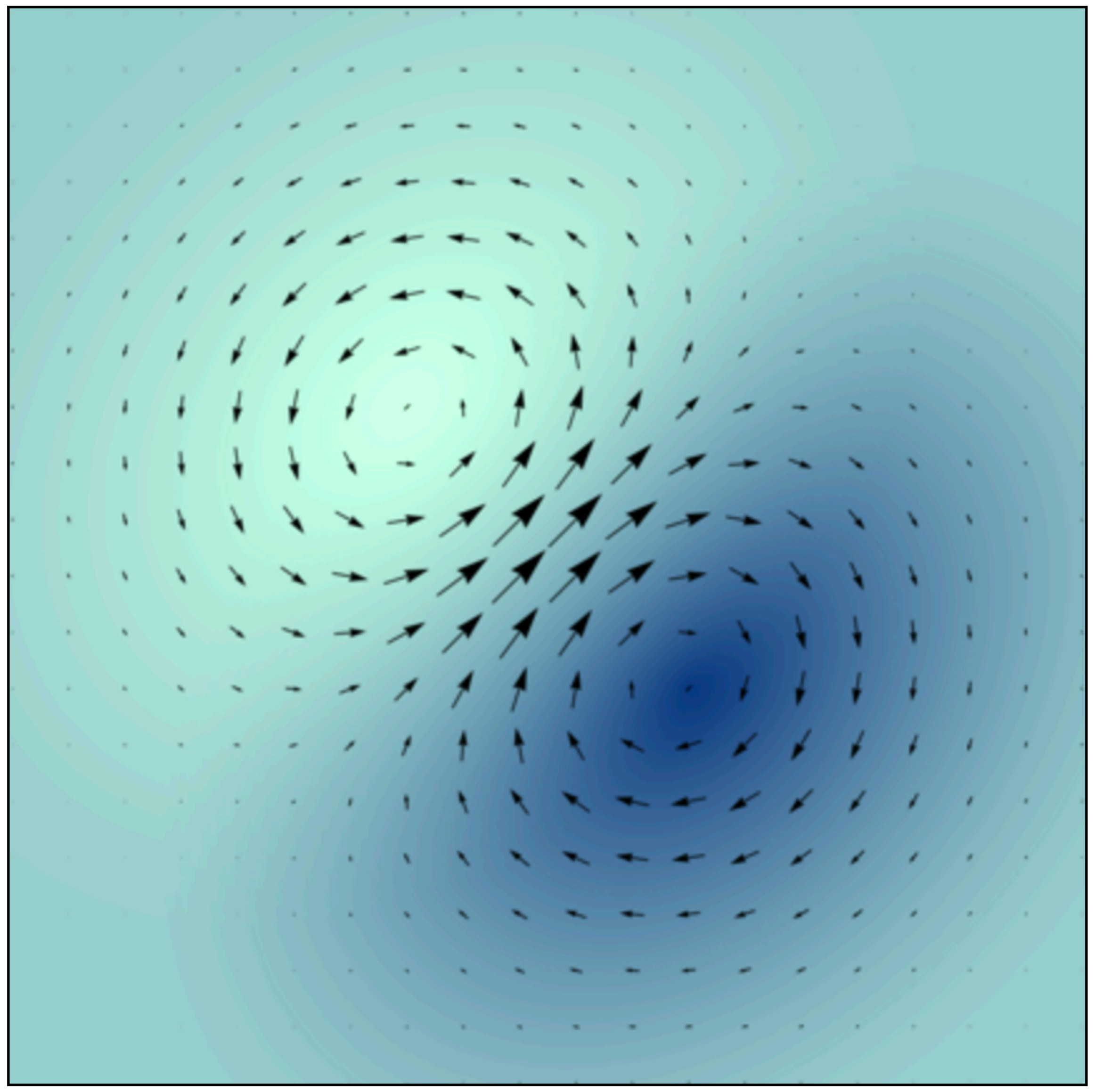
Vector Fields

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix}$$

$$\mathbf{v} = \mathbf{v}(x, y)$$
$$p = p(x, y)$$







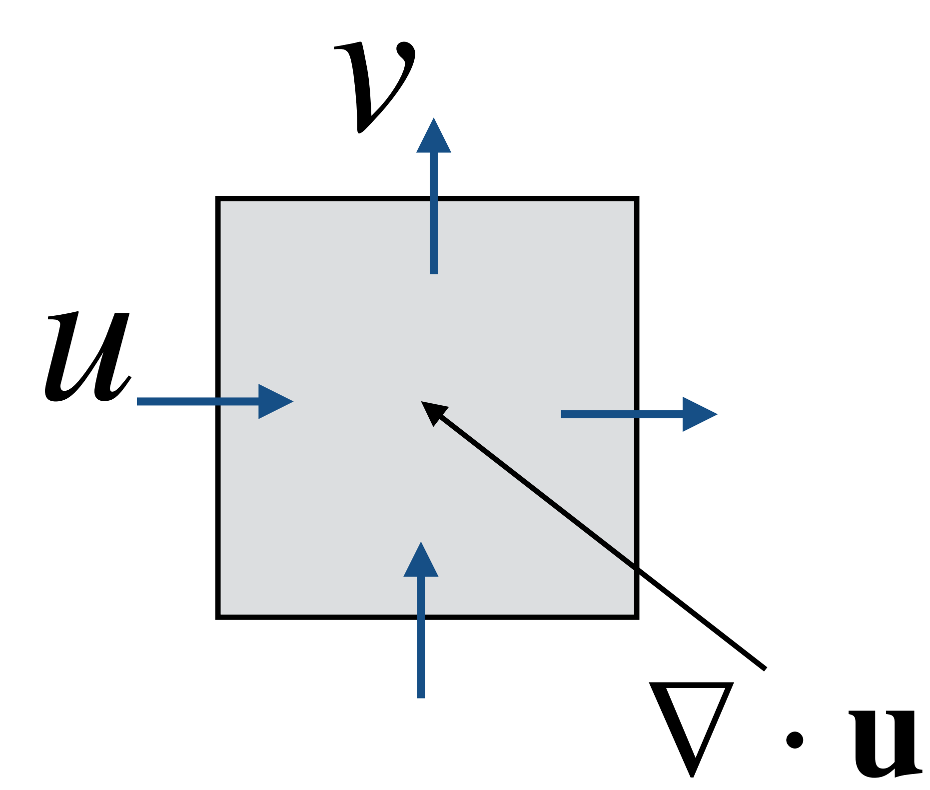
Gradient:
Direction of greatest change

$$\textbf{grad}(p) = \nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

Divergence:
Net flow in or out of region

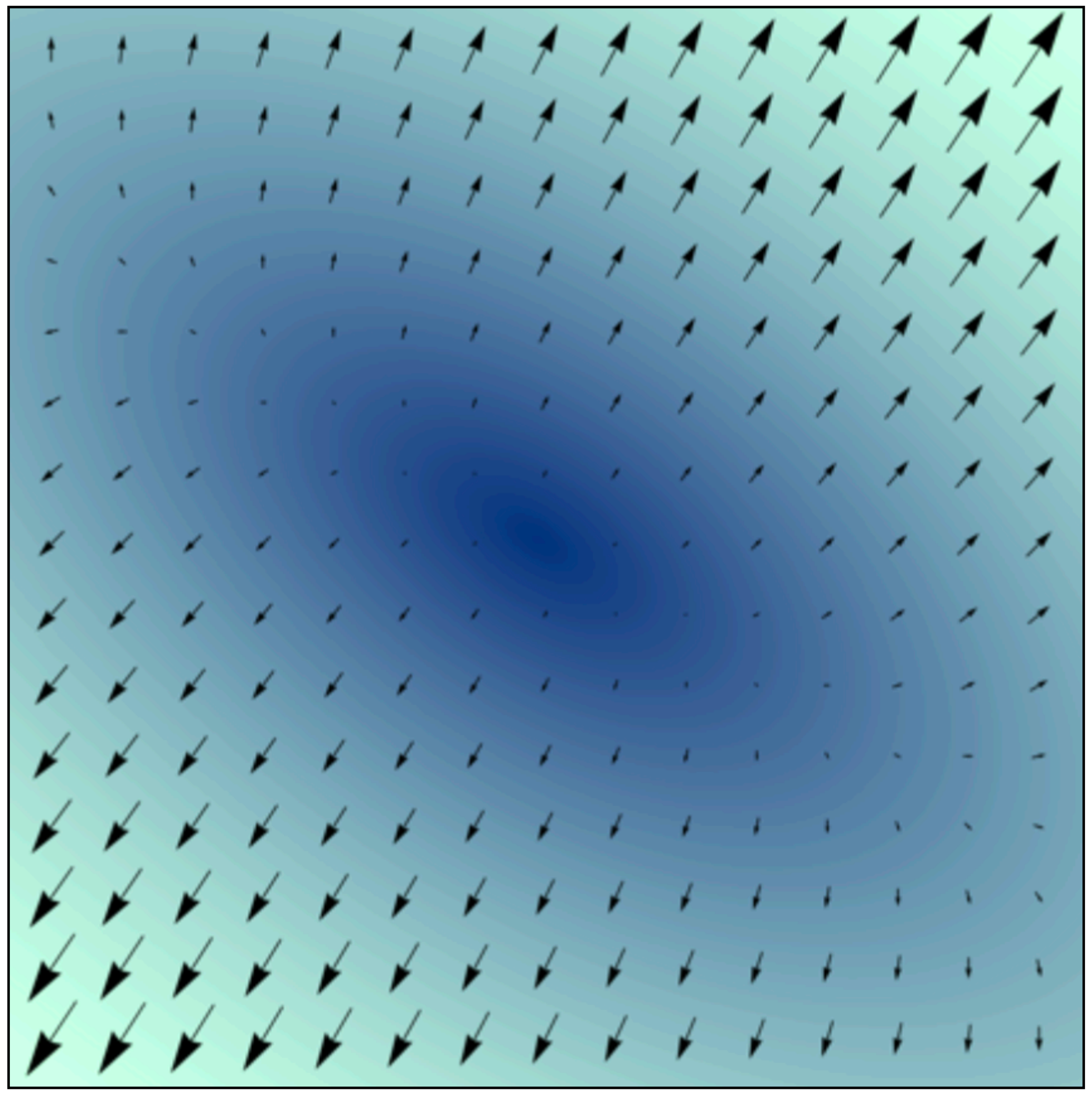
$$\textbf{div}(\mathbf{u}) = \nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

In 3D, cell centers and faces

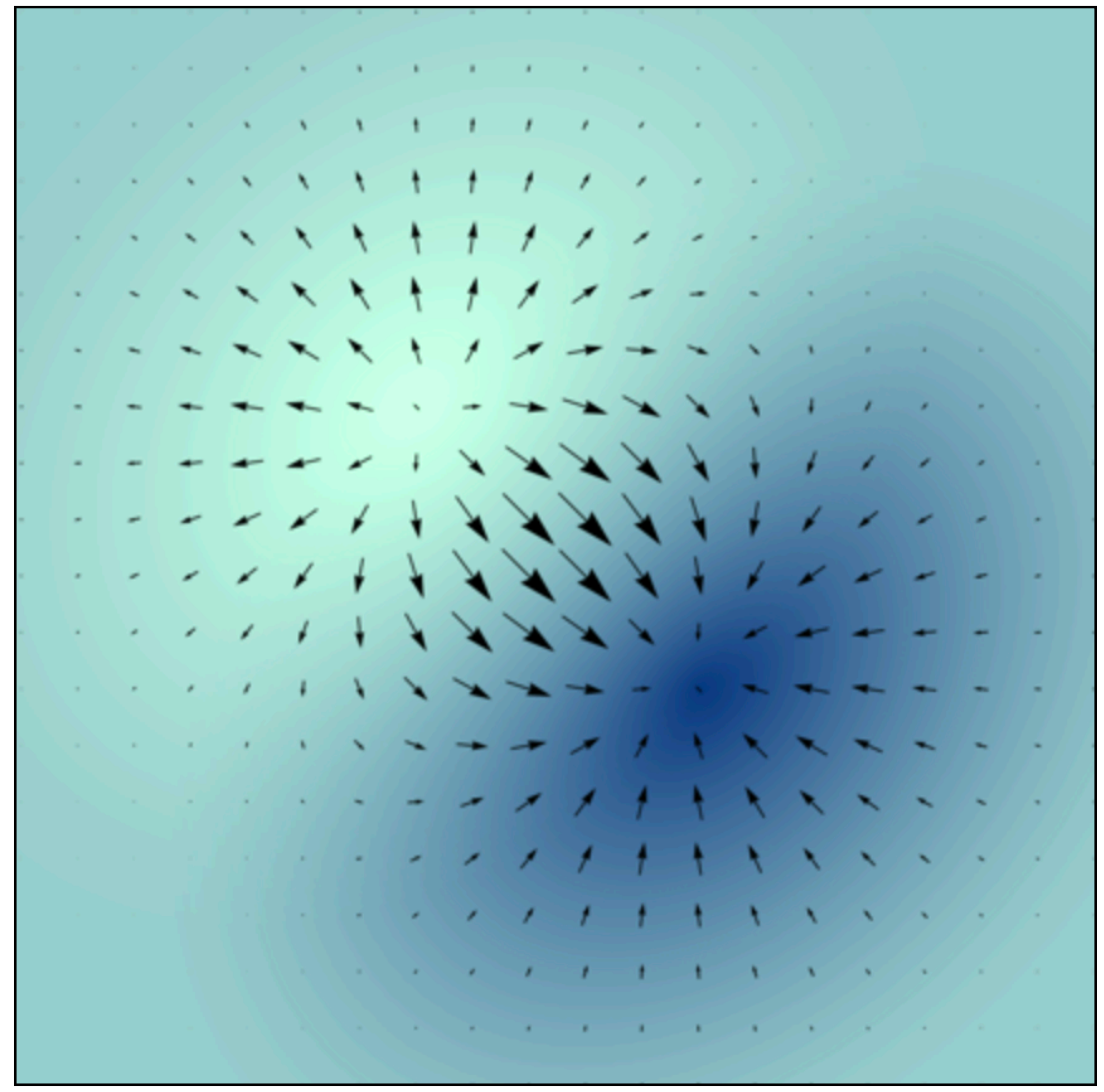


Vector Fields

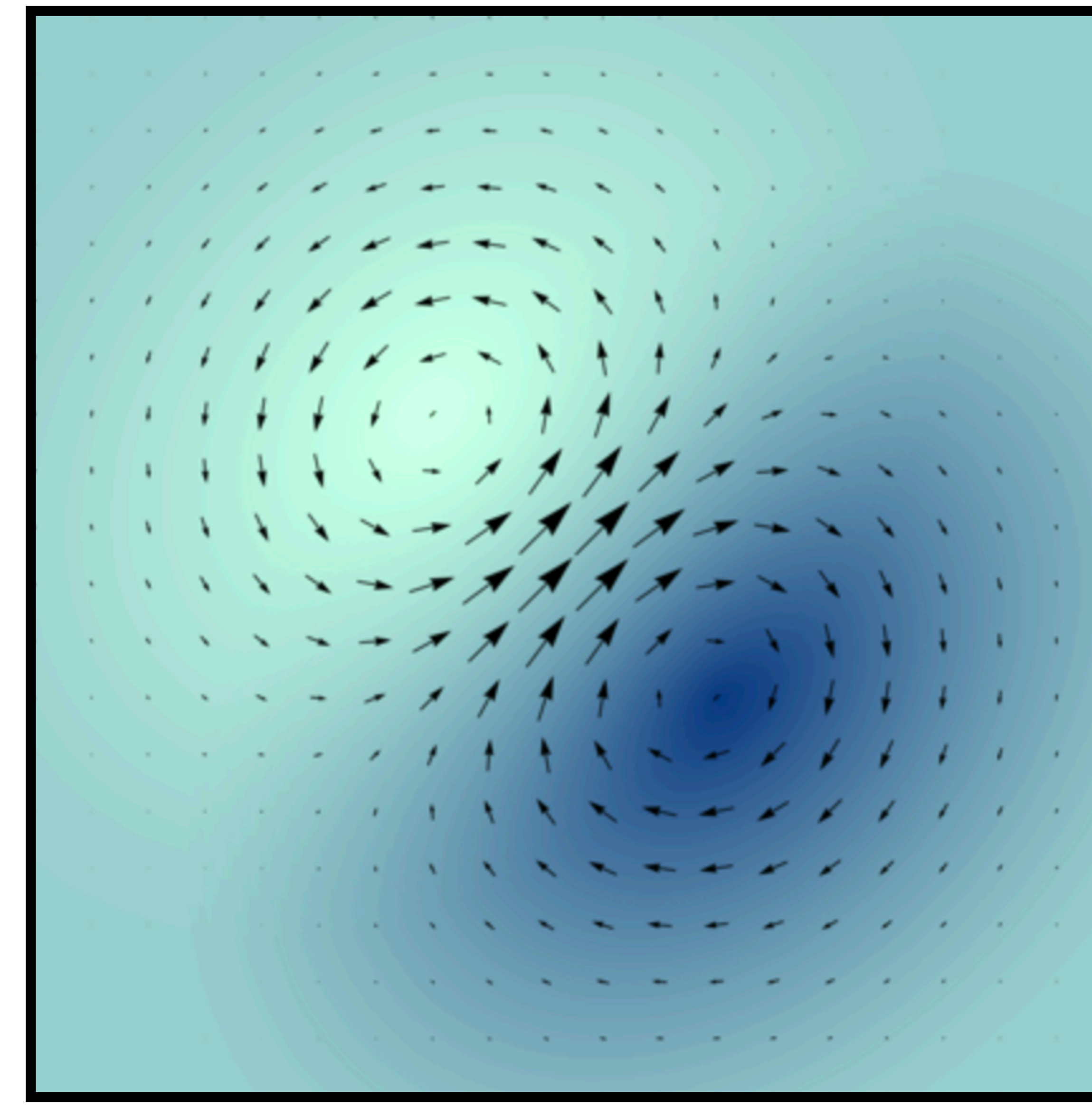
$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \qquad \mathbf{v} = \mathbf{v}(x, y) \quad p = p(x, y)$$



Gradient:
Direction of greatest change



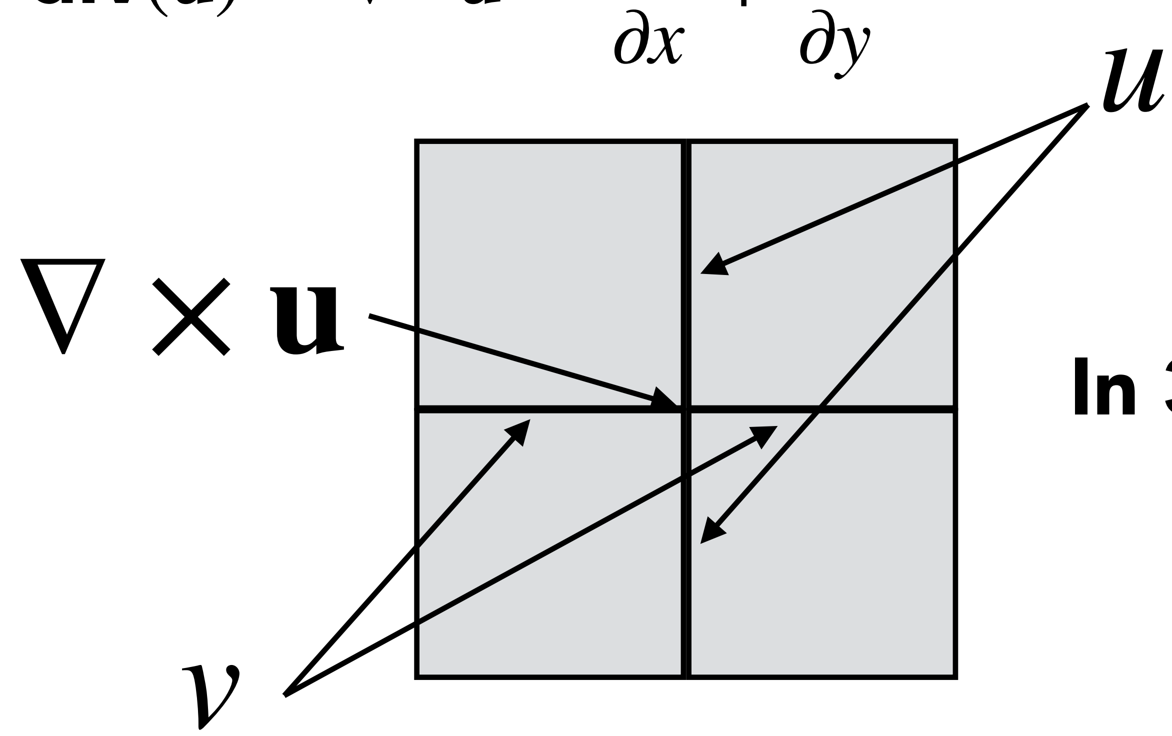
Divergence:
Net flow in or out of region



Curl:
Circulation around point

$$\textbf{grad}(p) = \nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

$$\textbf{div}(\mathbf{u}) = \nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

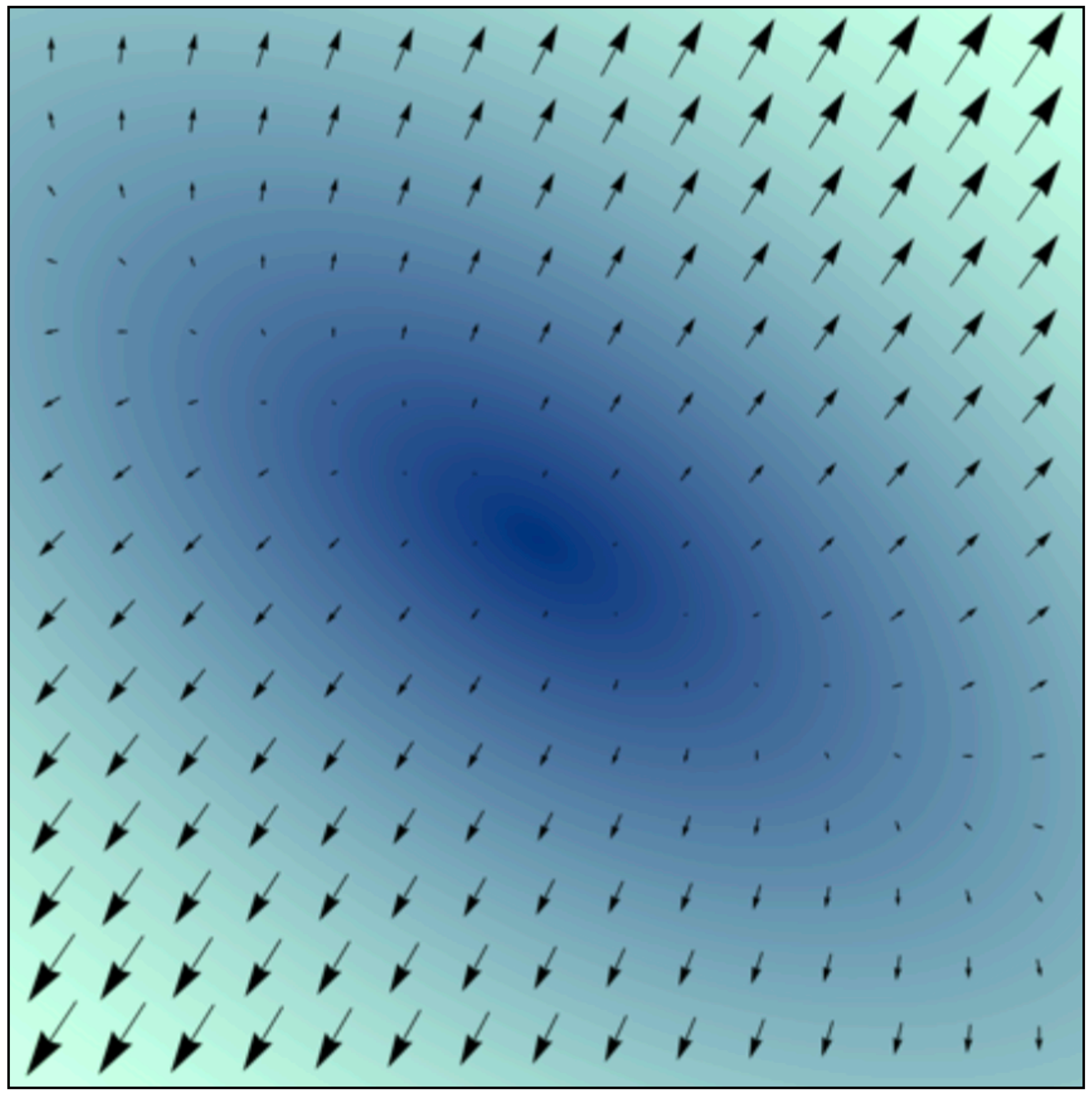


$$\textbf{curl}(\mathbf{u}) = \nabla \times \mathbf{u} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

In 3D, cell faces and edges

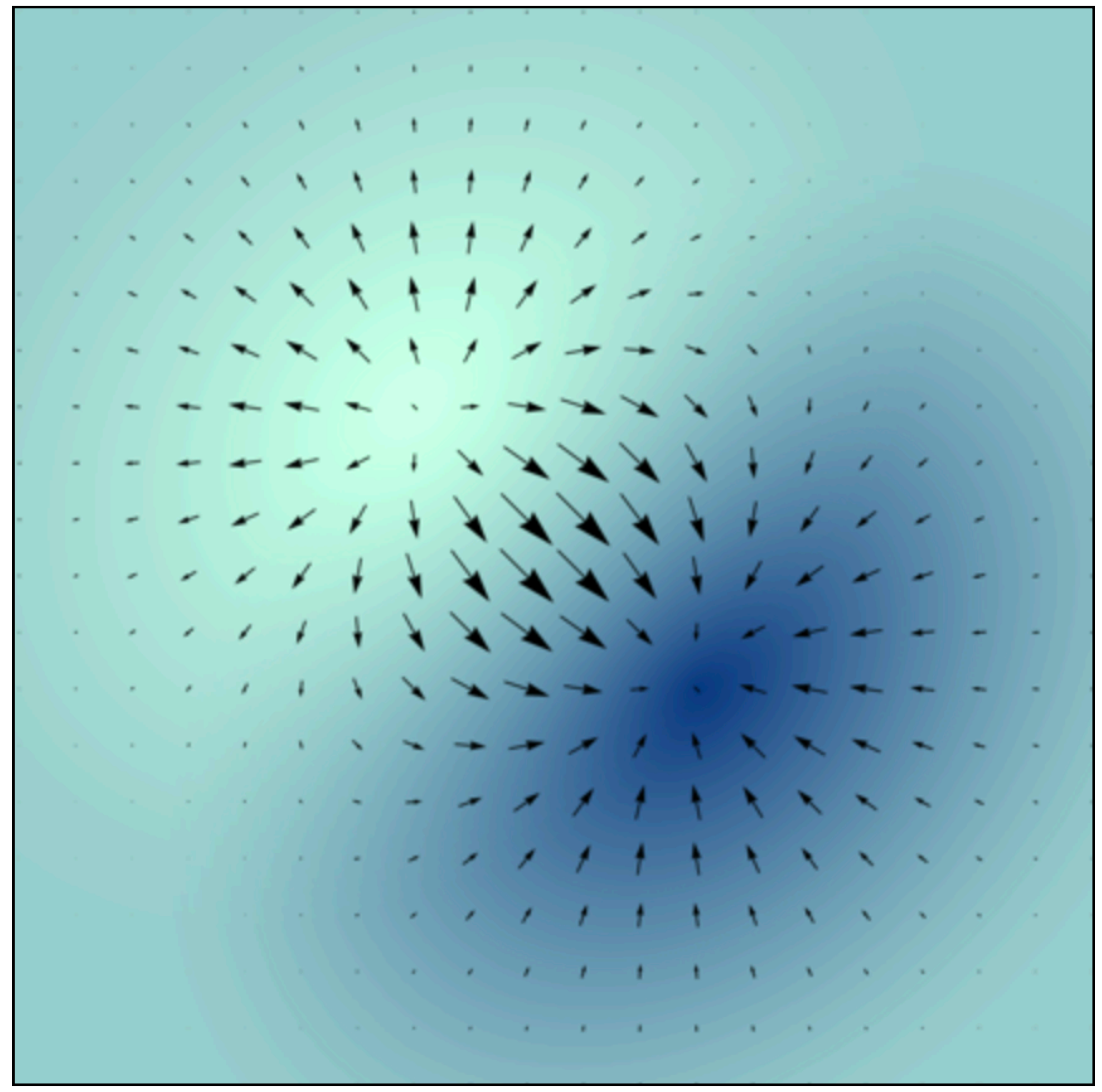
Vector Fields

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \qquad \mathbf{v} = \mathbf{v}(x, y) \quad p = p(x, y)$$



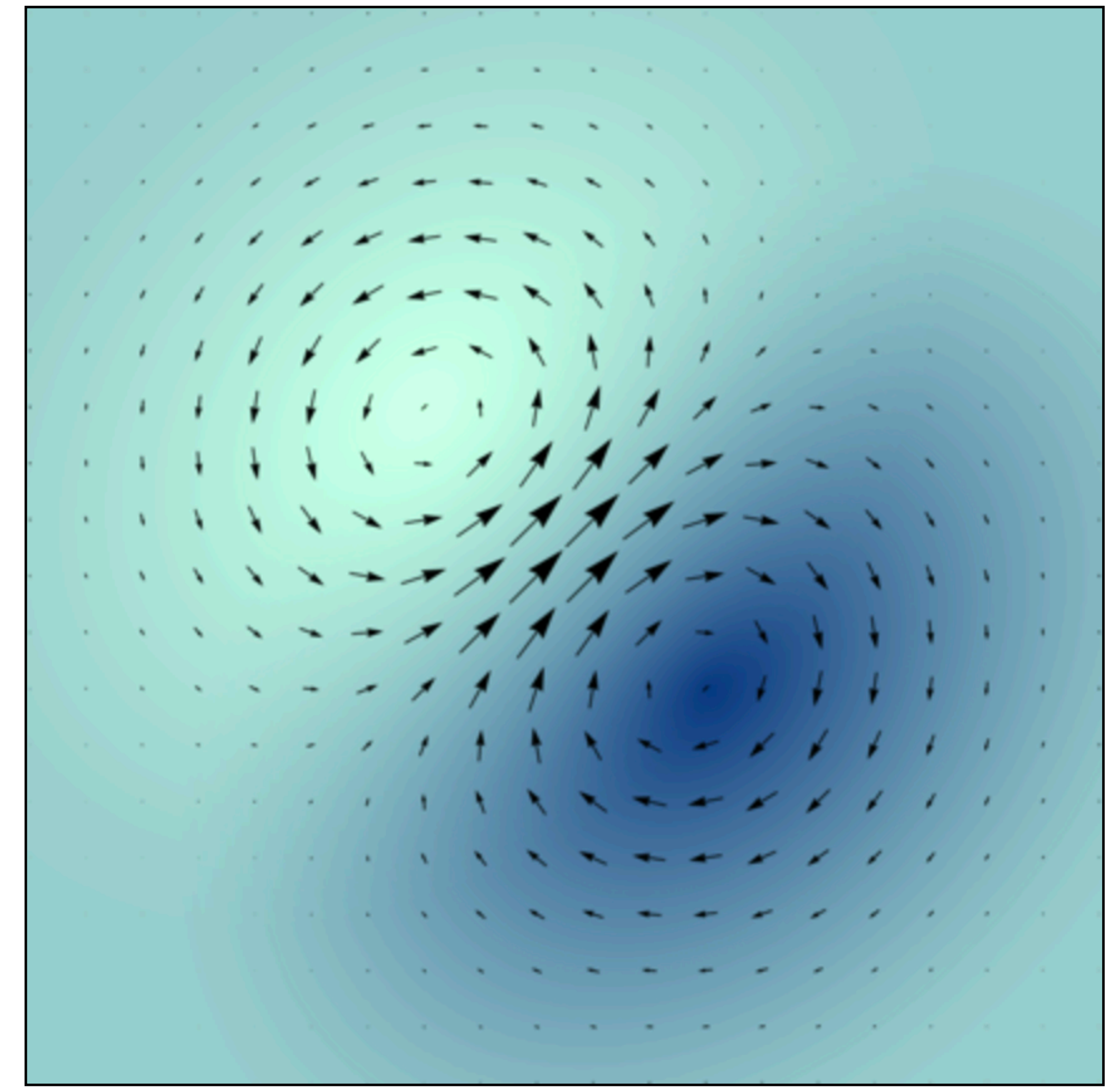
Gradient:
Direction of greatest change

$$\textbf{grad}(p) = \nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$



Divergence:
Net flow in or out of region

$$\textbf{div}(\mathbf{u}) = \nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

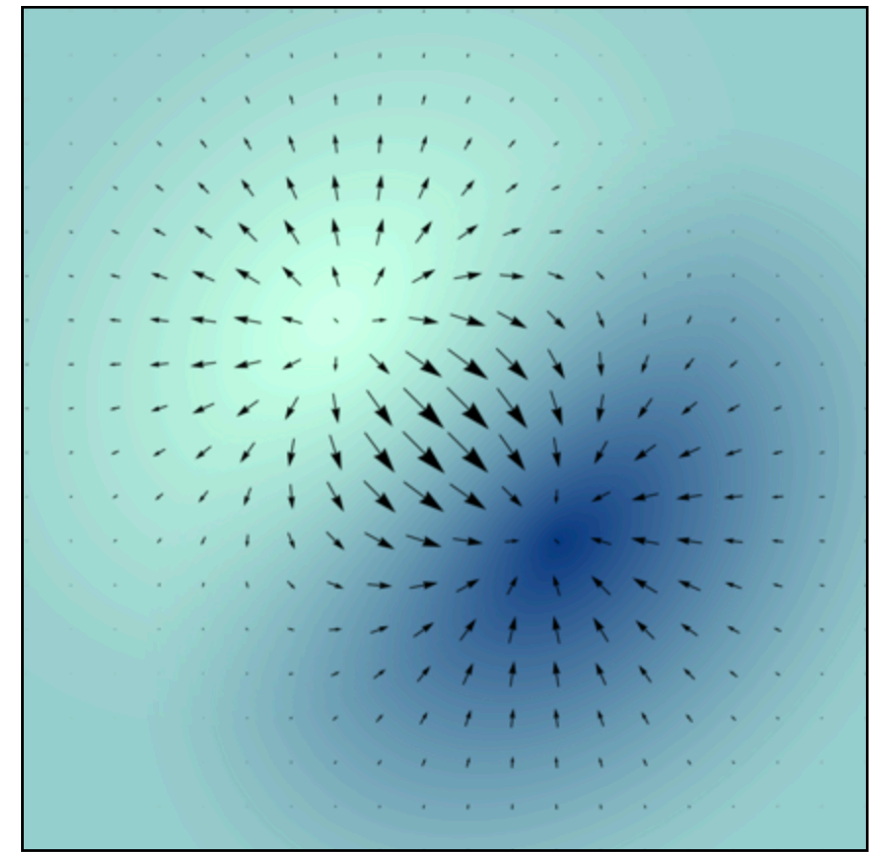


Curl:
Circulation around point

$$\textbf{curl}(\mathbf{u}) = \nabla \times \mathbf{u} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

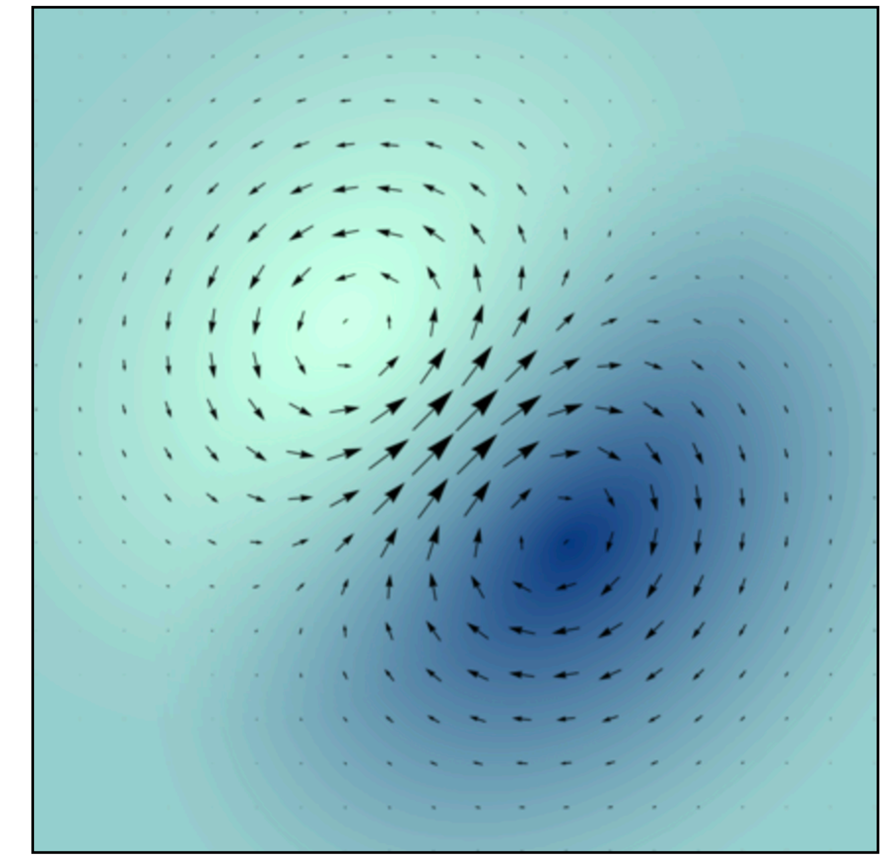
In 3D, curl is vector stored at edges

Vector Fields



Divergence:
Net flow in or out of region

$$\textbf{div}(\mathbf{u}) = \nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

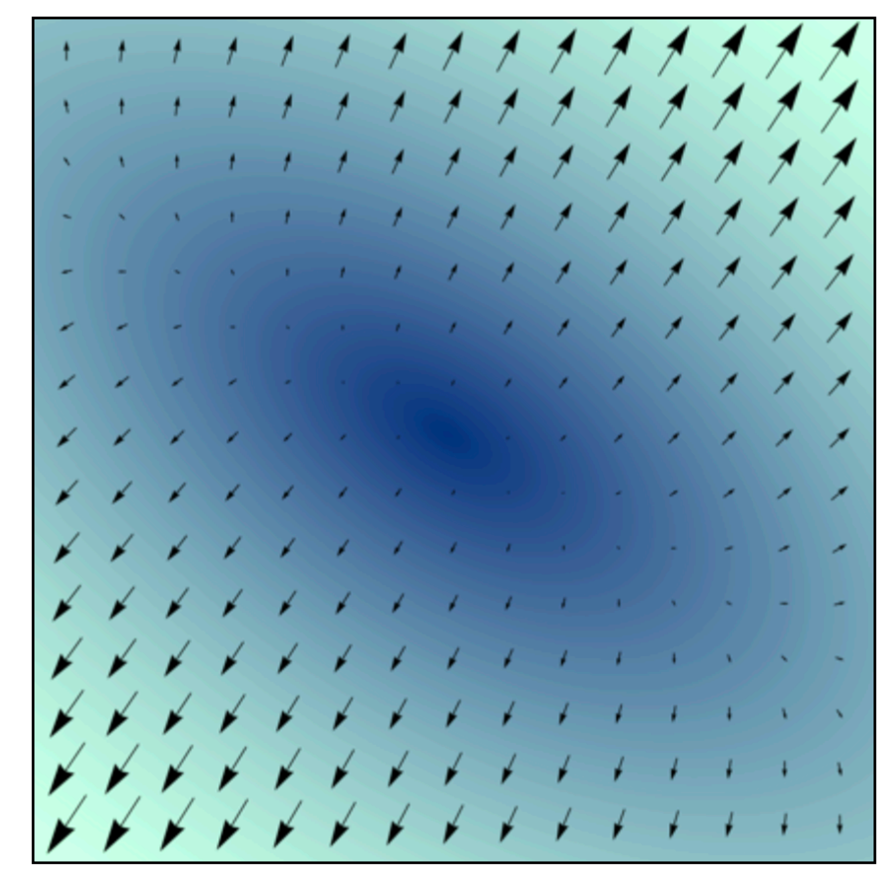


Curl:
Circulation around point

$$\textbf{curl}(\mathbf{u}) = \nabla \times \mathbf{u} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

Laplacian:
Difference from the neighborhood average

$$\nabla^2 = \nabla \cdot \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)$$



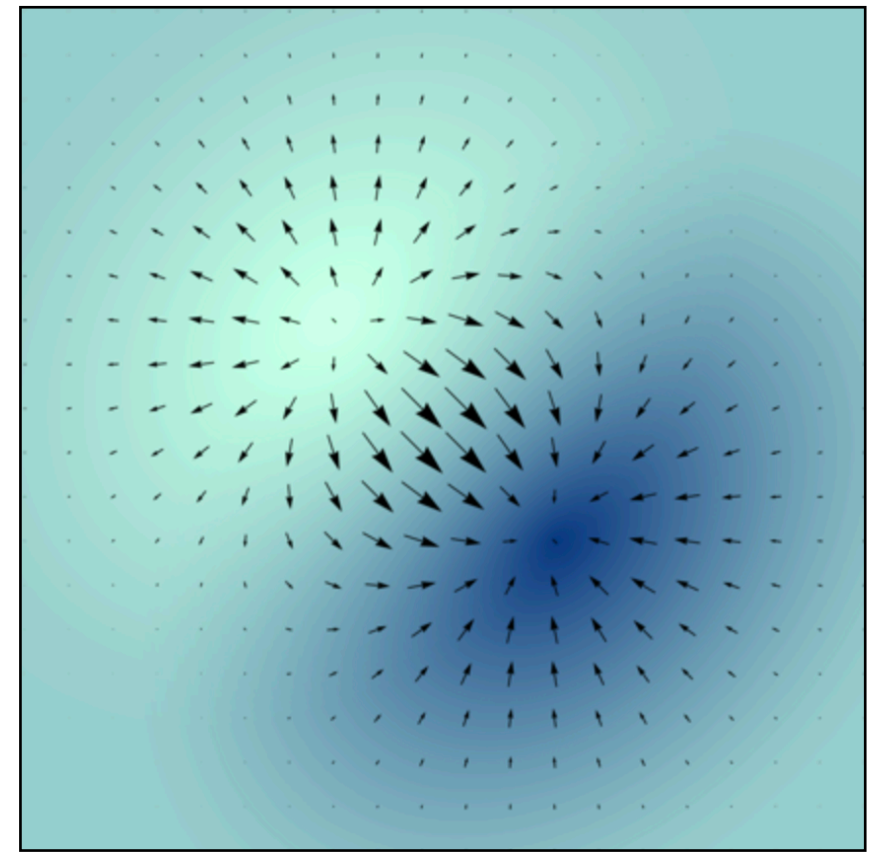
Gradient:
Direction of greatest change

$$\textbf{grad}(p) = \nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

Vector Fields

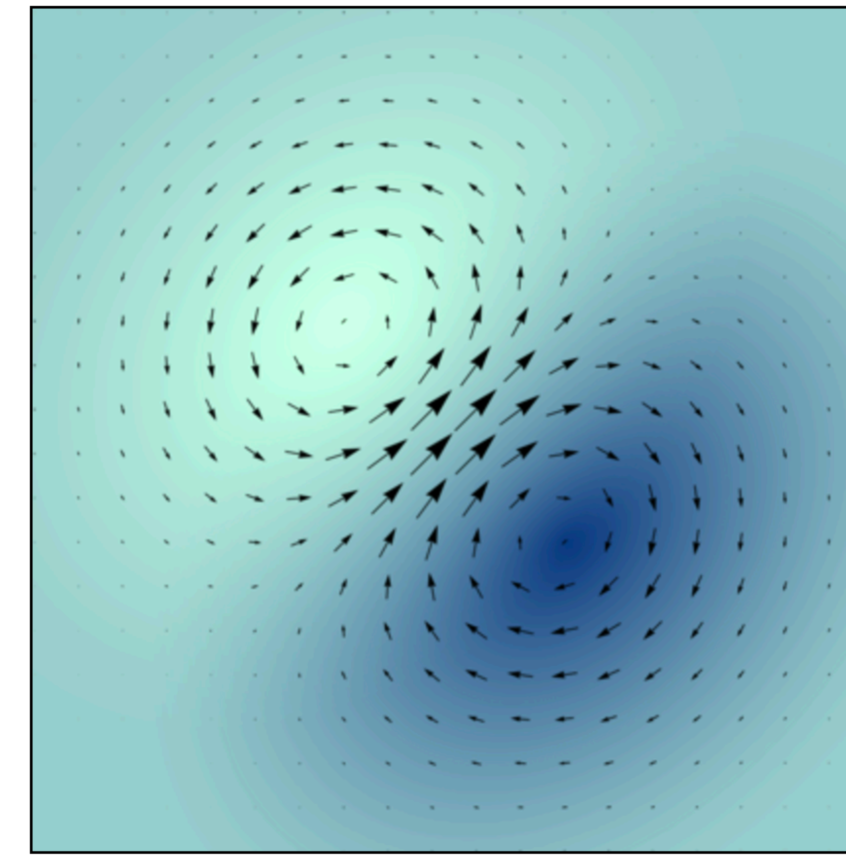
Laplacian:
Difference from the neighborhood average

$$\nabla^2 = \nabla \cdot \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)$$



Divergence:
Net flow in or out of region

$$\textbf{div}(\mathbf{u}) = \nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$



Curl:
Circulation around point

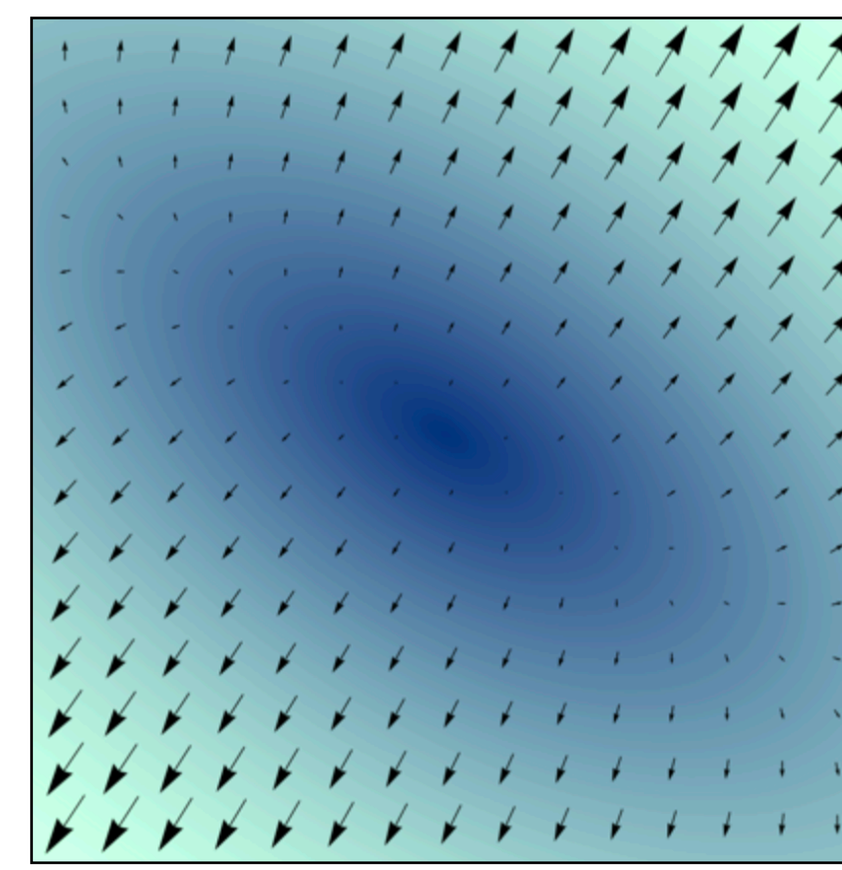
$$\textbf{curl}(\mathbf{u}) = \nabla \times \mathbf{u} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

Directional Derivative:
How a quantity changes as point of observation moves

$$(\mathbf{u} \cdot \nabla) = \left(u_x \frac{\partial}{\partial x} + u_y \frac{\partial}{\partial y} \right)$$

How fast are we moving in x direction?

How does something change as we move in the x direction?



Gradient:
Direction of greatest change

$$\textbf{grad}(p) = \nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$$

CS184/284A
Ren Ng

"Vector Analysis" - lots of fun math

Navier–Stokes Equations (N-SE)

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \frac{\nu}{\rho}\nabla^2 \mathbf{u} + \frac{\mathbf{f}_f}{\rho}$$

Change in fluid velocity

Advection

Pressure

Viscosity

Field forces (*e.g.*: gravity)

ρ **is density**

ν **is viscosity**

Bad Solver

- Store velocity ($\mathbf{u}$) and density (ρ) on staggered grid
- Compute pressure (p) as function of density
- Use N-SE to update velocities
- Update densities $\dot{\rho} \propto -(\mathbf{u} \cdot \nabla)\rho + \nabla \cdot (\mathbf{u}\,\rho)$
- Repeat until end of simulation

Problem: Pressure waves move fast so this explicit method must use very small timesteps or go unstable.

Problem: Advection term also limits time step based on speed of fluid. (Bulk speed of fluid is generally less than wave speed.)

Incompressible Fluids

Replace pressure forces with constraints

- No more pressure waves
- This is another projection method!

Divergence is net in-/out-flow

- Constrain divergence to be zero by projection
  - $\nabla \cdot \mathbf{u} = 0$

Split advection term off from the rest of N-SE and use semi-Lagrangian advection.

"Stable Fluids" by Jos Stam, SIGGRAPH 99

Incompressible Fluids

Separate problems terms from the rest:

$$\Delta \mathbf{u} = \Delta t \left(-(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \frac{\nu}{\rho} \nabla^2 \mathbf{u} + \frac{\mathbf{f}_f}{\rho} \right)$$

$$\Delta \mathbf{u} = \Delta t \left(\Delta \mathbf{u}_a + \Delta \mathbf{u}_p + \frac{\nu}{\rho} \nabla^2 \mathbf{u} + \frac{\mathbf{f}_f}{\rho} \right)$$

$$\mathbf{u}^* = \mathbf{u}^t + \Delta t \left(\frac{\nu}{\rho} \nabla^2 \mathbf{u} + \frac{\mathbf{f}_f}{\rho} \right)$$

Unprojected and unadvected new velocities

Incompressible Fluids

Separate problems terms from the rest:

$$\mathbf{u}^* = \mathbf{u}^t + \Delta t \left(\frac{\nu}{\rho} \nabla^2 \mathbf{u} + \frac{\mathbf{f}_f}{\rho} \right)$$

In general we will have $\nabla \cdot \mathbf{u}^* \neq 0$

Use pressure to correct this:

$$\nabla \cdot \left(\mathbf{u}^* + \Delta \mathbf{u}_p \right) = \nabla \cdot \mathbf{u}^* + \nabla \cdot \Delta \mathbf{u}_p = 0$$

$$\Delta \mathbf{u}_p = - \Delta t \frac{\nabla p}{\rho}$$

$$\nabla \cdot \mathbf{u}^* = \Delta t \nabla \cdot \frac{\nabla p}{\rho}$$

Incompressible Fluids

Separate problems terms from the rest:

$$\mathbf{u}^* = \mathbf{u}^t + \Delta t \left(\frac{\nu}{\rho} \nabla^2 \mathbf{u} + \frac{\mathbf{f}_f}{\rho} \right)$$

$$\nabla \cdot \mathbf{u}^* = \Delta t \nabla \cdot \frac{\nabla p}{\rho}$$

$$\frac{\Delta t \, \nabla^2}{\rho} p = \nabla \cdot \mathbf{u}^*$$

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}$$ Solve for pressure.

Density is now constant, so it can move past the divergence operator.

Incompressible Fluids

Add pressure correction to get projected, but not advected, velocities:

$$\mathbf{u}^{+} = \mathbf{u}^{*} - \frac{\Delta t \, \nabla^2}{\rho} p$$

Solving for pressure

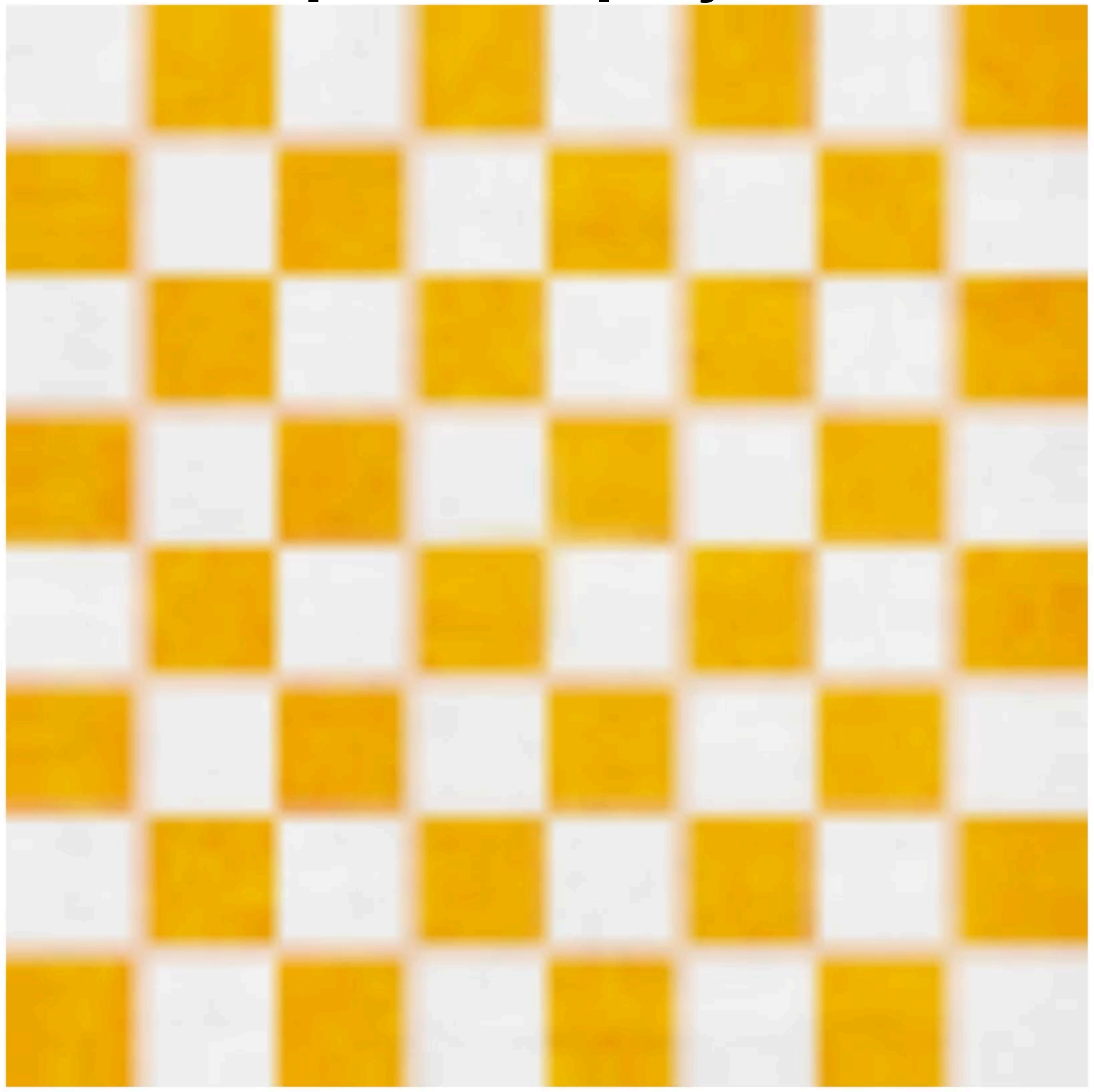
- Successive over-relaxation
  - Easy to understand and implement, but slow
- Pre-conditions conjugate gradient
  - Widely used, reasonably fast
  - [Modified] Incomplete Cholesky for preconditioned
- Other problem-specific methods

Incompressible Fluids

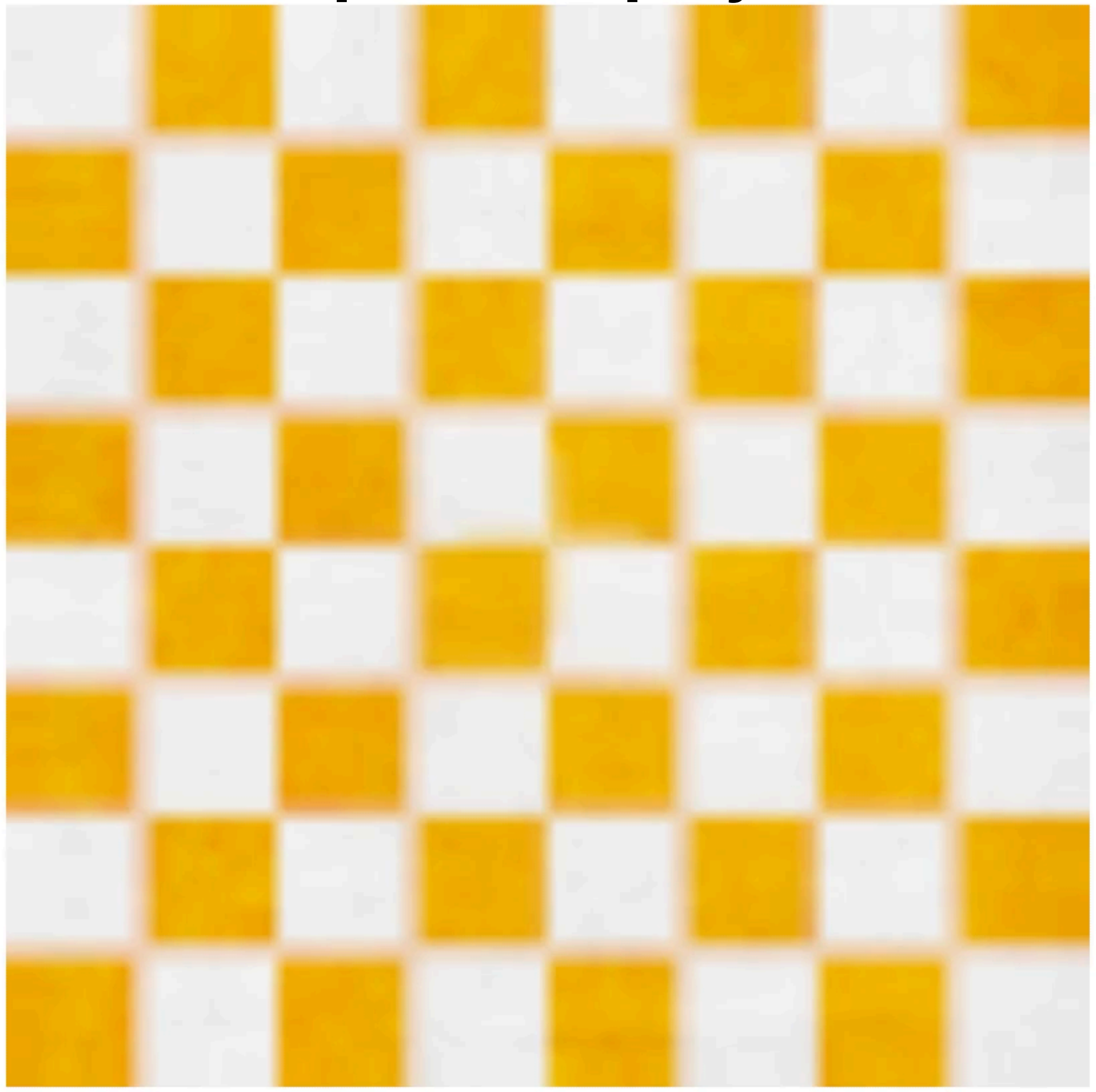
Add pressure correction to get projected, but not advected, velocities:

$$\mathbf{u}^{+} = \mathbf{u}^{*} - \frac{\Delta t \, \nabla^{2}}{\rho} p$$

No pressure projection



With pressure projection

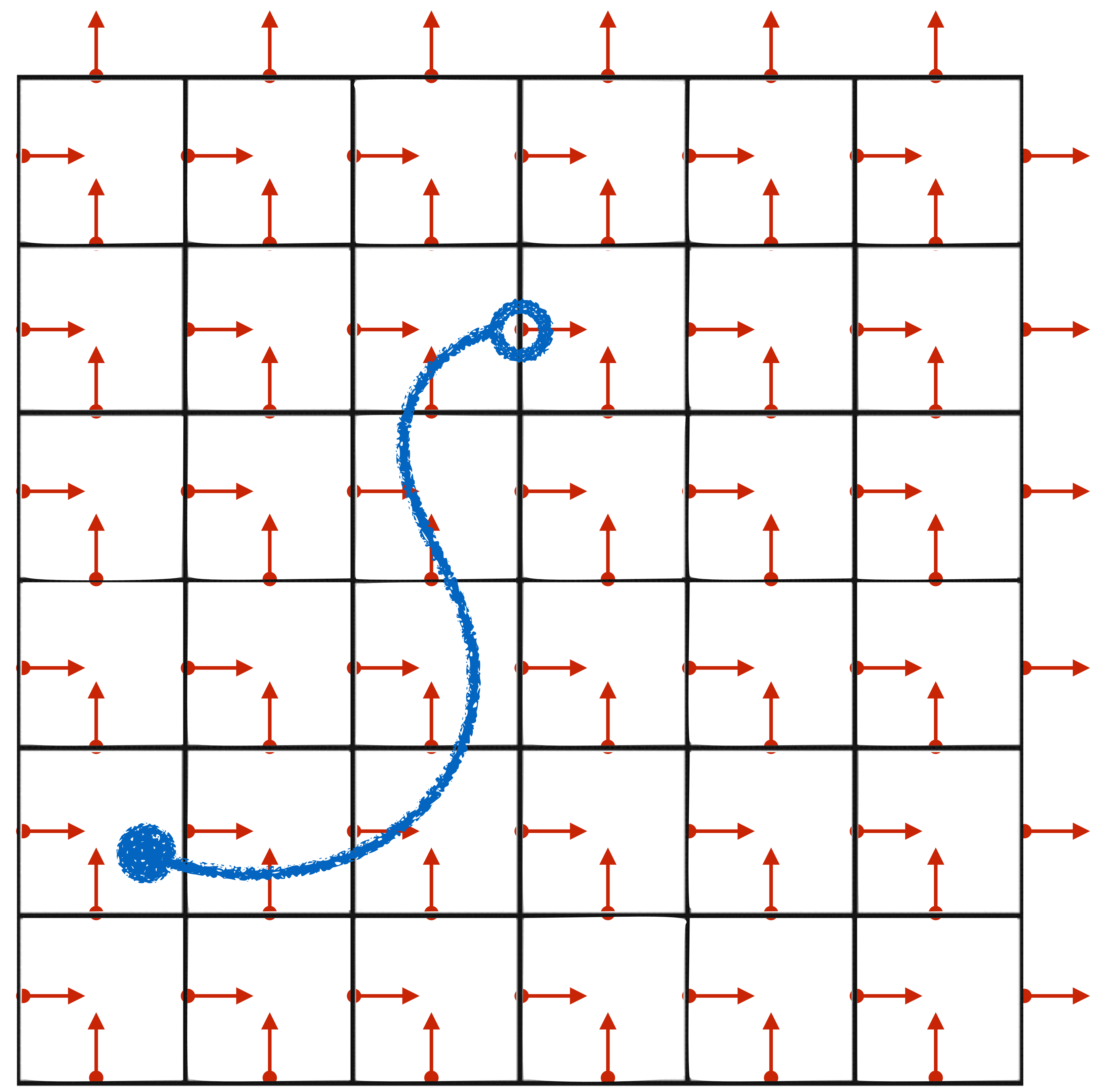


Semi-Lagrangian Advection

(A method of characteristics)

Instead of using 2nd order advection term, pick up the values and move them!

- For each location
  - Track backward through grid for Δt
  - Interpolate value
  - Copy to new location



Note: This works for other quantities besides velocity.

Note: Vector values should be rotated based on flow, but most people don't do this.

Note: Backtrace is done in one or more substeps.

Semi-Lagrangian Advection

Final velocity is:

$$\mathbf{u}^{t+\Delta t} = \text{advect}\left(\mathbf{u}^* - \frac{\Delta t\,\nabla^2}{\rho} p\right)$$

Unconditionally stable

Large steps introduce extra damping

- Viscosity term often omitted as unwanted

Stable Fluids

Demo by Amanda Ghassaei

https://apps.amandaghassaei.com/gpu-io/examples/fluid/

Things to notice:

- In pressure view you can see grid cells
  - You don't see them when simulation is rendered!
- Note how much damping there is
- Note how pressure changes as cursor is moved