Physical Simulation

Things We Can Simulate

- Point Masses
- Collision Detection and Response
- Rigid Bodies
- Articulated Systems and Constraints
- Soft Bodies
- Fluid Dynamics

Point Masses

Remember that particle systems are functionally a collection of point masses that obey some set of rules

What rules might particles in a physical simulation follow?

Newton's Equations of Motion

Describe motion over time by modeling force in relationship to object trajectory

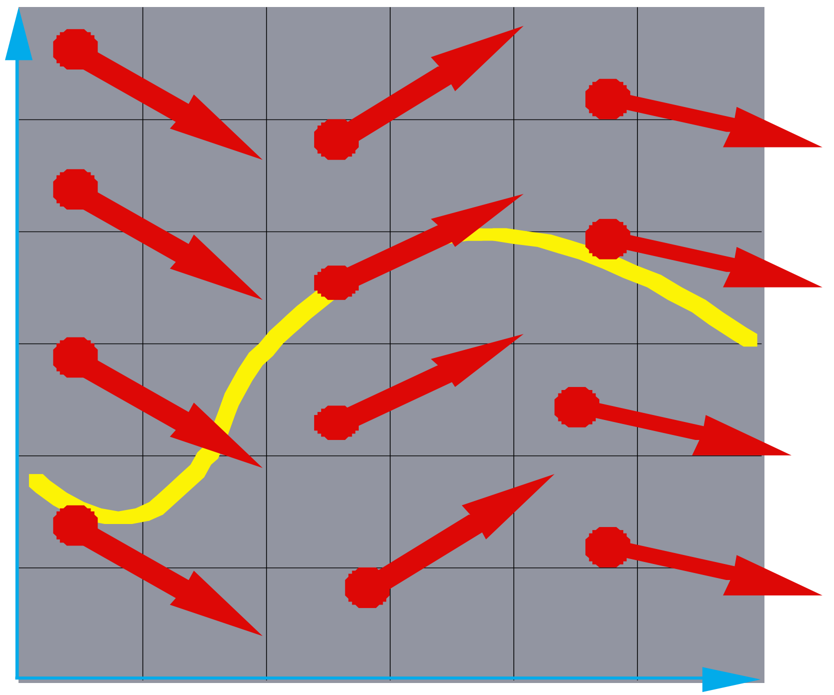
- F = ma

Integrating over time captures a system's physical behaviors

How to discretize?

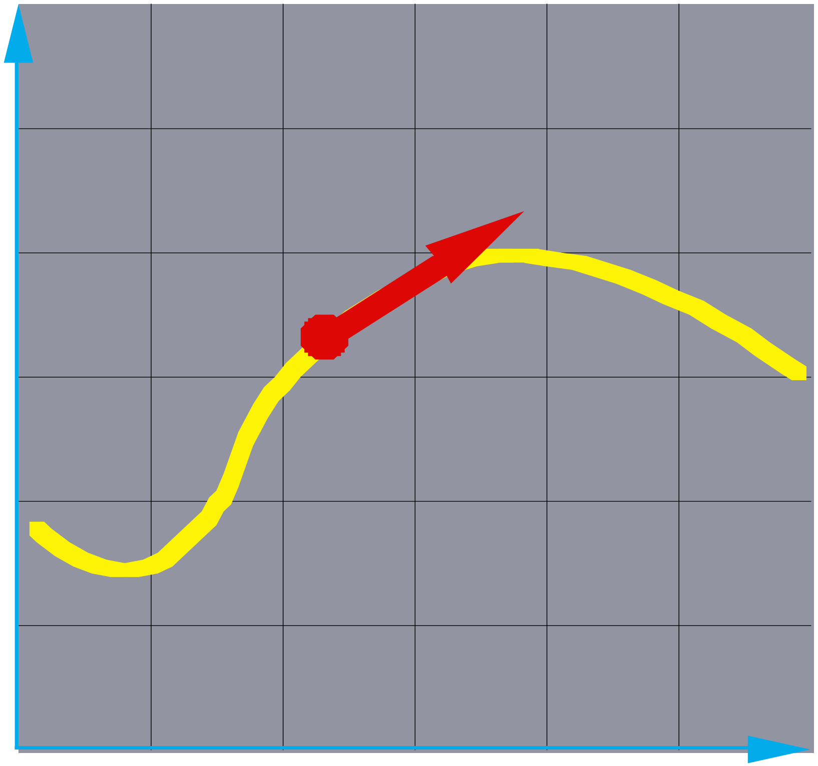
Vector Field

At any point in space, function **g(x, t)** defines a vector field dictating velocity for x at time t



Particle in a Vector Field

- Particle has a position and a velocity based on the vector field
- How to calculate a new position?

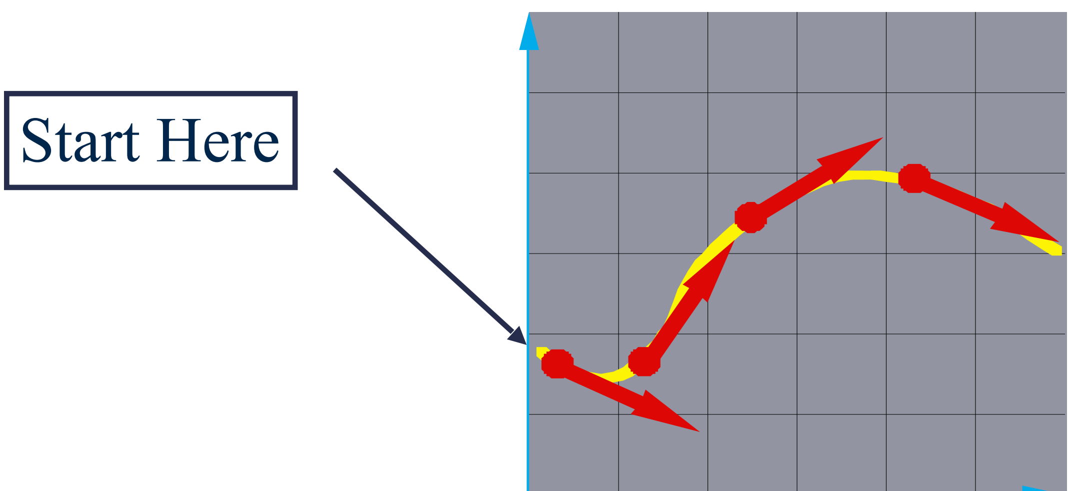


Differential Equations

$$\dot{x} = g(\vec{x}, t)$$

is a first-order differential equation!

Solve for **x** over time by starting at initial point and stepping along the vector field



Euler's Method

- Take linear time steps (Δt) along flow:

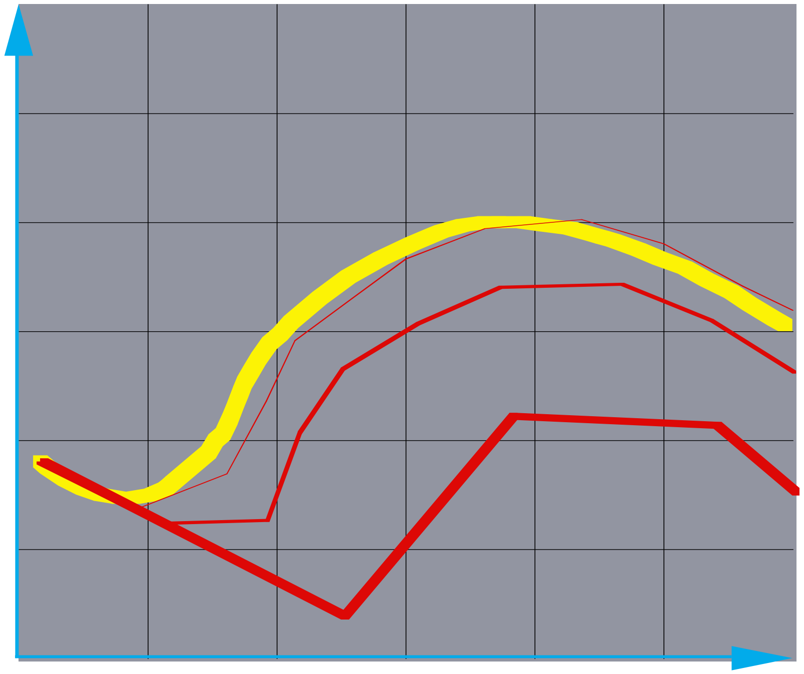
$$\vec{\mathbf{x}}(t+\Delta t) = \vec{\mathbf{x}}(t) + \Delta t \cdot \dot{\mathbf{x}}(t) = \vec{\mathbf{x}}(t) + \Delta t \cdot g(\vec{\mathbf{x}},t)$$

- Write as a time iteration:

$$\vec{\mathbf{x}}^{i+1} = \vec{x}^{i} + \Delta t \cdot \vec{\mathbf{v}}^{i}$$

Euler across Time Steps

What do you notice about Euler's method?



Explicit Euler Properties

- Simplest numerical method
- Bigger steps lead to bigger errors

Particle in a Force Field

Now consider a particle with mass in a force field **f**

We can write out Newton's law as follows:

$$\vec{f} = m\vec{a} = m\ddot{x}$$

Since **f** depends on particle position, velocity and time:

$$\ddot{x} = \frac{\vec{f}(\vec{x}, \dot{x}, t)}{m}$$

Second Order Equations

$$\ddot{x} = \frac{\vec{f}(\vec{x}, \dot{x}, t)}{m}$$

is a second order differential equation… we’d rather not deal with this!

Rather than solve directly, create a pair of coupled first order equations:

$$\begin{bmatrix} \dot{x} = \vec{v} \\ \dot{v} = \frac{\vec{f}(\vec{x}, \vec{v}, t)}{m} \end{bmatrix}$$

Differential Equation Solver

Since $\begin{bmatrix} \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} \vec{v} \\ \frac{\vec{f}}{m} \end{bmatrix}$

Euler's method:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \dot{x}(t)\Delta t$$

$$\dot{x}(t + \Delta t) = \dot{x}(t) + \ddot{x}(t)\Delta t$$

With substitutions:

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

$$\dot{x}(t + \Delta t) = \dot{x}(t) + \frac{\vec{f}(\vec{x}, \dot{x}, t)}{m}\Delta t$$

Euler Iterative Form

$$\vec{x}^{i+1} = \vec{x}^{i} + \vec{v}^{i}\Delta t$$
$$\vec{v}^{i+1} = \vec{v}^{i} + \frac{\vec{f}^{i}}{m}\Delta t$$

Still performs poorly for large time steps!

Ideally we want a more stable integrator…

Many Integrators Exist!

- Runge-Kutta
- Implicit Integration
- Semi-implicit Euler
- Verlet

- Vary in terms of complexity and computation

Verlet Integration

A better solver with greater stability and no additional computational overhead (popular in realtime applications)

Three versions:

- Position
- Velocity
- Leapfrog

Verlet Flavors

Position Verlet

- Uses 2 previous positions to model velocity

Leapfrog

- Alternately updates position and velocity

Velocity Verlet

- Updates position and velocity in same time step

Position Verlet

Handles velocities implicitly:

$$\overrightarrow{x}^{i+1} = \overrightarrow{x}^{i} + (\overrightarrow{x}^{i} - \overrightarrow{x}^{i-1}) + \dot{v}\Delta t^2$$

$$\overrightarrow{x}^{i-1} = \overrightarrow{x}^{i}$$

- Requires constant time steps and two steps to start
- Simple and cheap to implement

Applying Forces

Each particle experiences a force/forces

Common forces:

- Constant (gravity)
- Position/time dependent (force fields)
- Velocity dependent (drag)
- Combinations (damped springs)

Force Examples

Gravity: $\vec{f}_{grav} = m\vec{G}$

Viscous drag: $\vec{f}_{drag} = -k\vec{v}$

One body spring: $\vec{f} = -k_{spring}(|\Delta\vec{x}| - r)$

One body damped spring:

$$\vec{f} = -[k_{spring}(|\Delta\vec{x}| - r) + k_{damp}|\vec{v}|]$$

Collision Detection and Response

Collision Detection: Determine when an intersection has happened

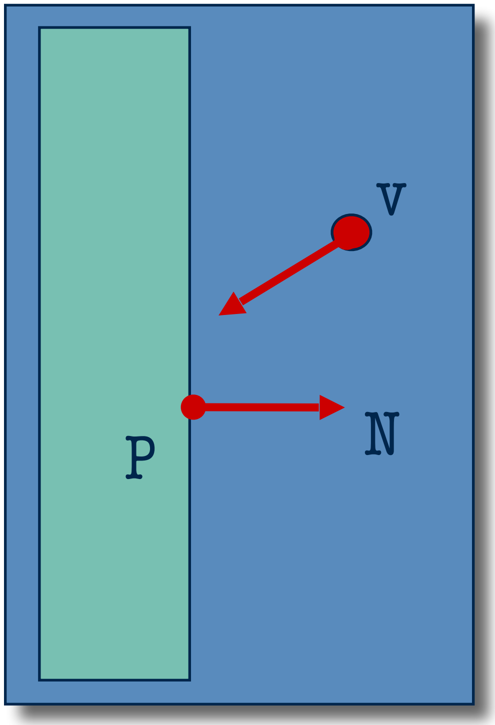
Collision Response: Determine what to do when intersection detected

Collision Detection

A very familiar problem!

(think ray-tracing)

Must also consider particle velocity



Collision Response

- After Contact (a posteriori)
  - Run simulation
  - “Roll back” if intersection occurs
- Before Contact (a priori)
  - Predict time of collision
  - Update position accordingly
- Resting Contact
  - Two objects are in contact with each other
  - A surprisingly difficult special case!

Rigid Bodies

Extends idea of point-mass

- Bodies can be interconnected
- Bodies are rigid relative to each other

Articulated Systems and Constraints

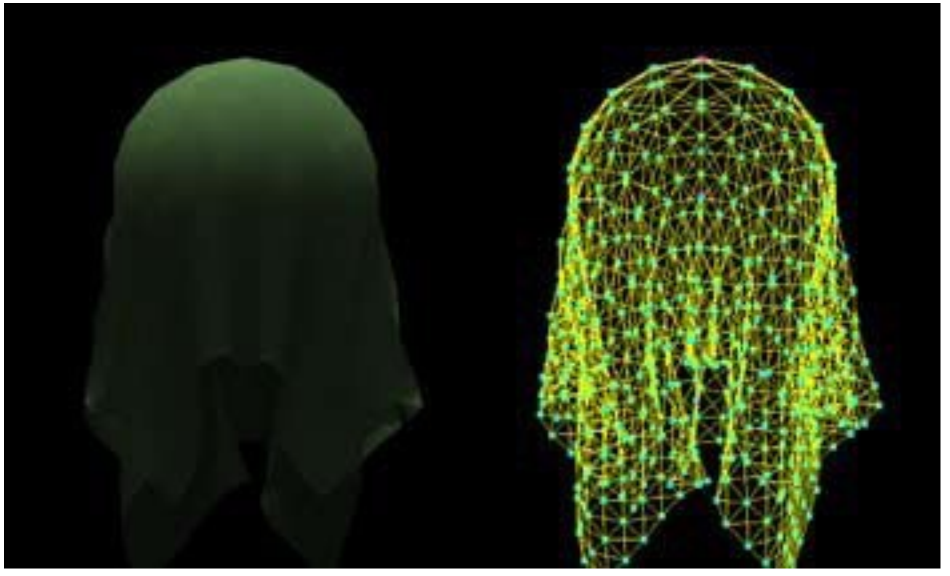
- Not all rotations are physically plausible
- Solve by limiting joint movement with constraints
- Use of **inverse kinematics** to solve for all joint angles based on final position of child bones

Soft Bodies

Distance between particles is not fixed

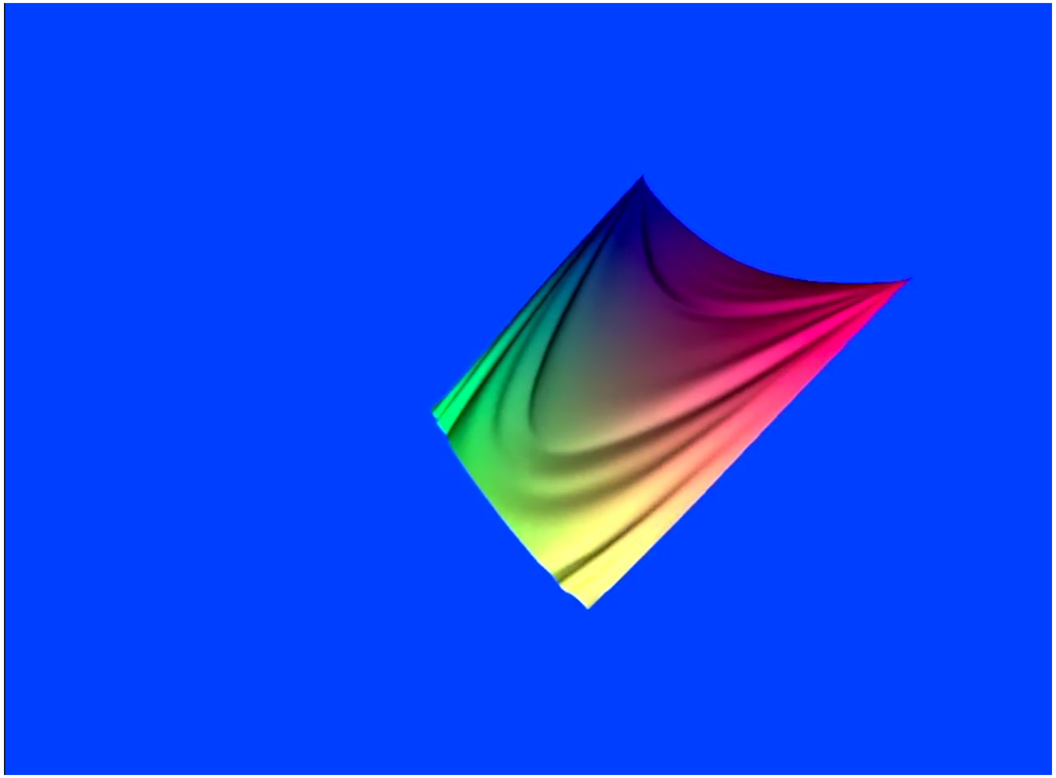
Generally a very expensive computation

Easier to simulate as a system of rigid body springs



Cloth Simulation Demo

https://www.youtube.com/watch?v=UhmZ3uigDvo



Fluid Dynamics

Describes the flow of fluids and gases

Models properties such as:

- Flow velocity
- Pressure
- Density
- Temperature

Navier-Stokes

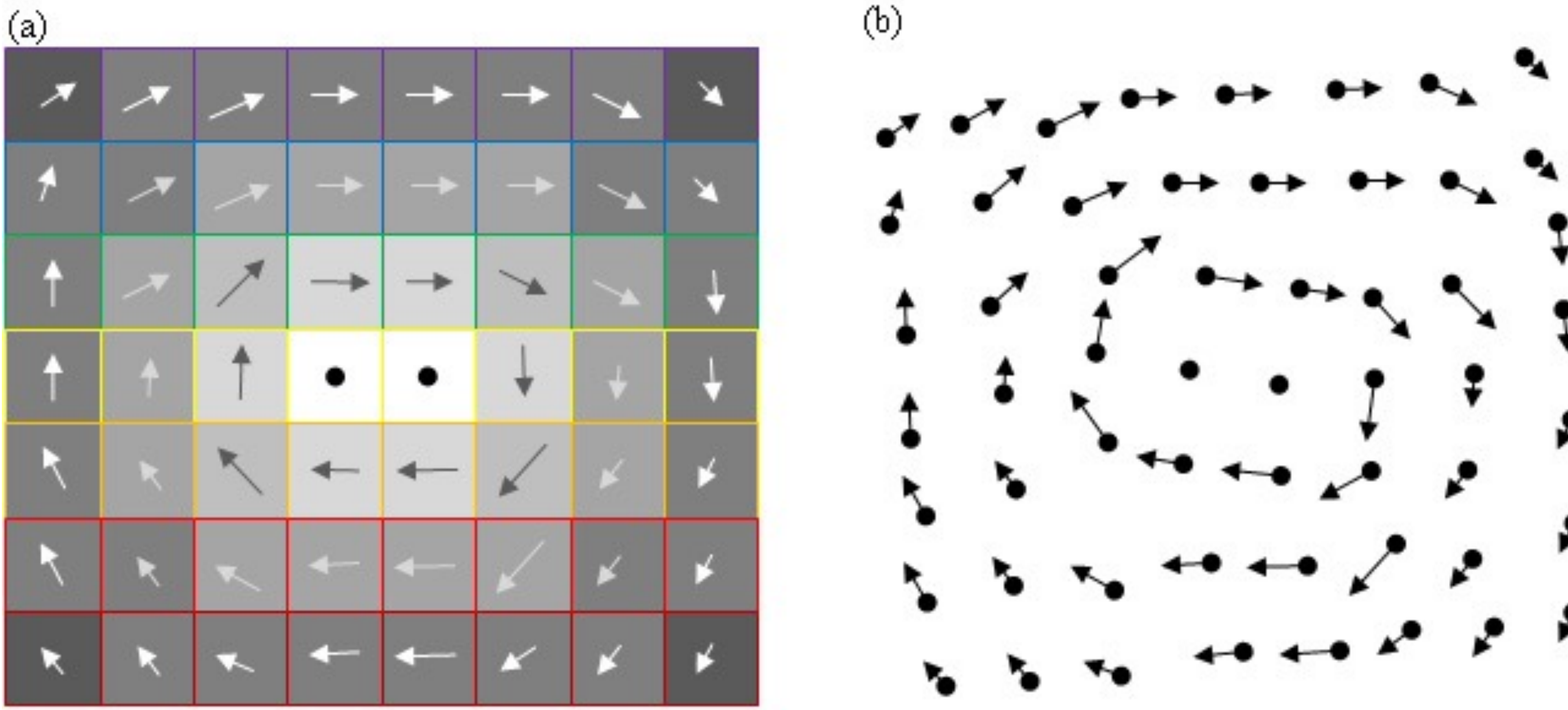
Applies Newton’s second law to fluid motion to calculate flow velocity

Requires solving for fluid’s:

- Diffusion (change in concentration)
- Advection (transport of material)

Can solve using grid-based or particle-based methods

Grid vs Particle



Boundaries

Define interactions between fluid and other objects/types of fluid

Common ones:

- Slip (does not cross boundary)
- No-slip (at rest at boundary)
- Inflow (enters boundary with velocity)
- Outflow (can leave boundary)

Boundary Conditions

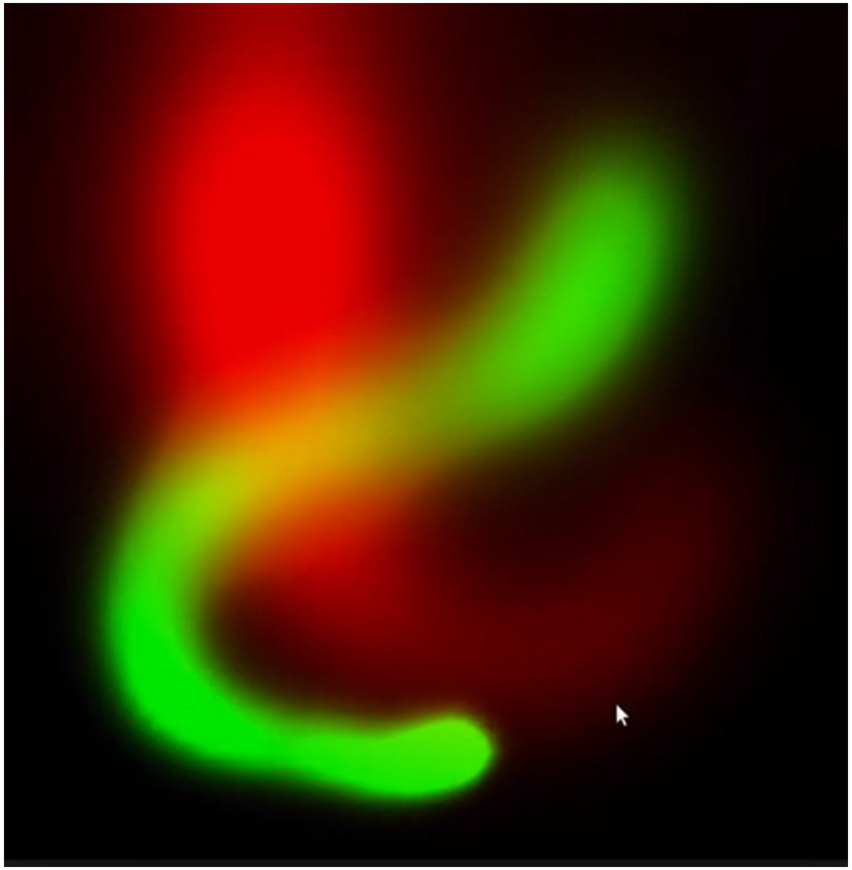
Usually specified as values for function at boundary

- e.g. velocity = 0 for no-slip

Can also define boundary in terms of pressure or other domains

Must resolve violations of boundary conditions if they occur during advection step

Fluid Demo



https://vimeo.com/247574785

Curl-Noise

Non-physically-based method for approximating fluid flow

- Create a vector field using Perlin noise
- Take curl (rotation) of this field to generate a divergence-free (doesn't shrink or expand) velocity field

A popular method in real-time applications!

https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph2007-curlnoise.pdf

Curl Noise Demo

https://www.youtube.com/watch?v=8TNZS2AkFNs