Student Feedback

What is helpful in class?

- Homework Content
- Class content
- Methods of teaching
- Social engagement
- Logistics

- Detailed homework descriptions
- “Assignments”
- Live grading
- In class demos
- Interactive class activities
- In-class group interactions (Getting to know fellow students)
- Approachable teaching team (Lots of opportunities for questions and feedback)
- Website and project showcase

What can we change?

- Homework Content
- Class content
- Methods of teaching
- Social engagement
- Logistics

- Less hand-holding on homework assignments
- More clear homework instructions
- Class pace can be too fast
- Math equations can be hard to grasp
- More Blender demos
- More thorough and visual math explanations
- More text and explanations on lecture slides
- Online lectures available
- In class polling / alternate methods of getting a pulse
- More structure for group activities

Improvements going forward

- Going through assignments in class
- More time spent on art (in homework and class)
- More Blender demos / walkthroughs
- Hear about our projects
- Visualizations
- More interaction in lecture
- More group activities
- Blender resources
- Final project check ins

- Homework Content
- Class content
- Methods of teaching
- Social engagement
- Logistics

What topics need more clarity

- **Modeling**
- **Math equations**
- **Lighting/Shadows**
- **Camera angles**
- **Ray Tracing**
- **UV mapping (normal and texturing)**

What has been challenging

- **Time management (especially for homework assignments)**
- **Finding project partners**
- **Difference between math and Blender buttons**
- **Difficult Blender tutorials**

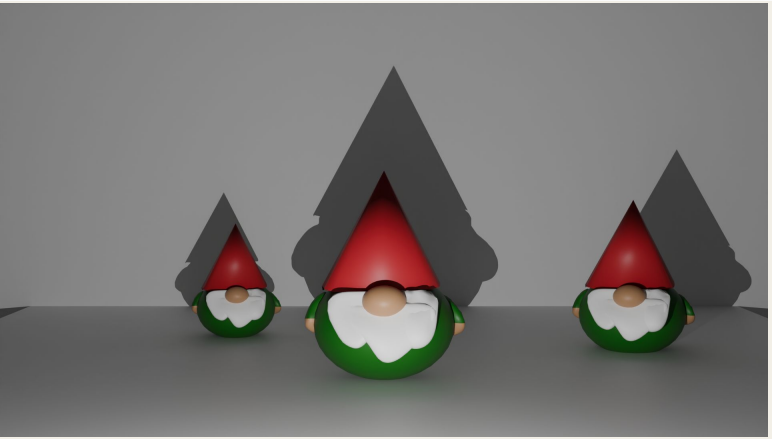
CS148 Summer 2025 - Lecture 11

07/29/2025

Simulation I

cs148.stanford.edu

Homework 4 Student Lighting & Textures



Sólveig Jónsdóttir

Jiaqi Ye

Yicheng Wang & Hanjun Luo

Oliver Loo

Homework 4 Student Lighting & Textures

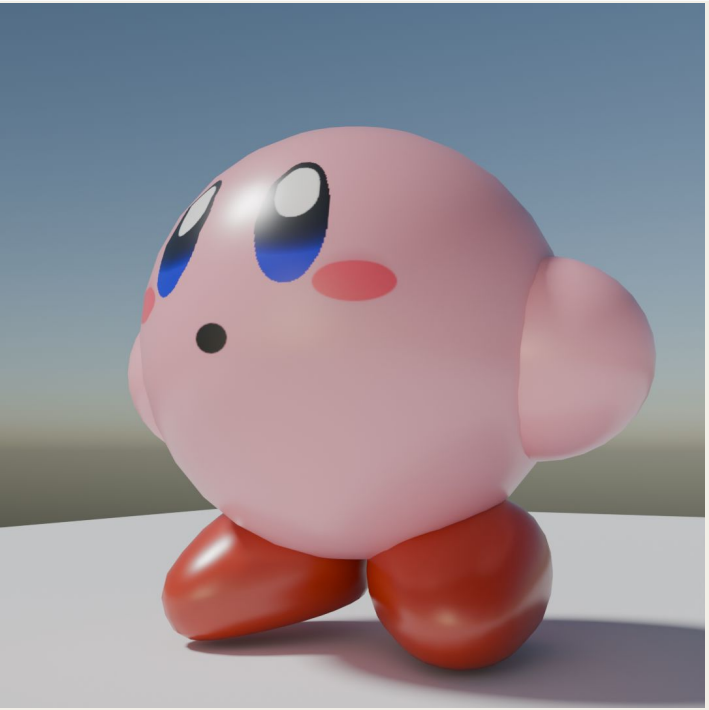
Grace Zhao

Abby Chen

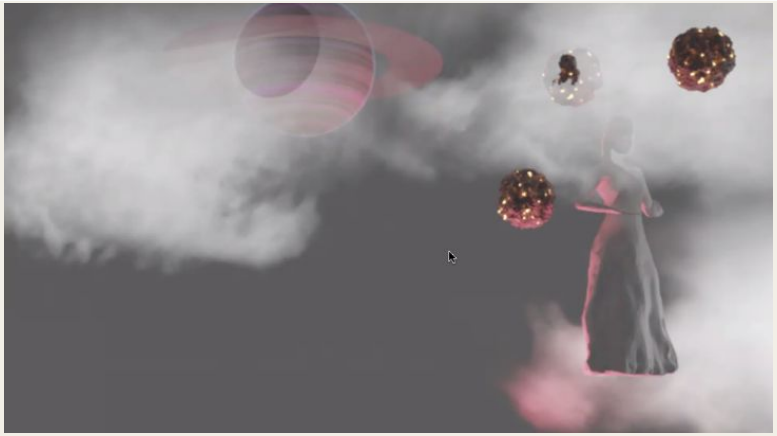
Seigo Hayami

Moritz Rosenthal & Michael Wang

Homework 4 Student Lighting & Textures



Chris Kim



Angelina Krinos

Siyuan Liu

Animation vs Simulation

- Animation: create specific **keyframes** to **snapshot** a scene at various **points in time**, then **connect** them by “filling the gaps” with **smooth motion**

- Simulation: use real life **physics** to **model** the behavior of **natural phenomena** as they **evolve dynamically** in **time**

- Both allow us to generate **change in a scene over time** for e.g. movies and video games, scientific computing, etc
- Even for generating a **still image**, we might want to **capture a specific frame of an animation or simulation**

Motivation

Motivation

- Even for generating a **still image**, we might want to **capture a specific frame of a simulation or animation**

- Towards your final project:
  - Blender has many tools for simulating physics and animating keyframes
  - e.g. simulate splashing water and save the geometry from a specific frame to then export and give materials!

Ryan Eberhardt, 2017

Lecture Outline

- Today will be **very high level**
- You will not need to implement any of this!
- Both simulation and animation are big topics deserving their own classes:

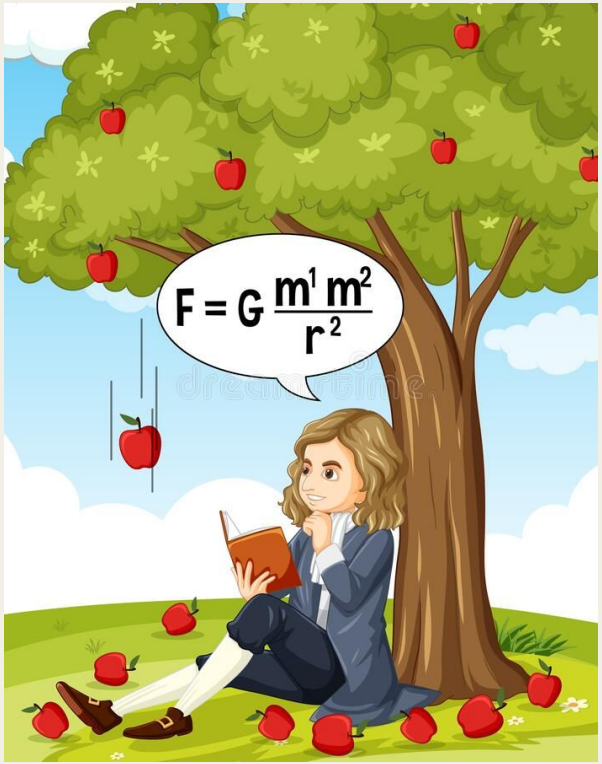
Doug James, Simulation / Interactivity

Karen Liu, Animation / Robotics

Physical Systems

- We model an object in space at some time with a **physical state:**
  - Mass of the object?
  - Position of the object?
  - Velocity of the object?
  - External forces on the object?

- Goal of simulation:
  - Given initial state at time t
  - Compute next state at time t + 1



Force = Mass x Acceleration (F = MA)

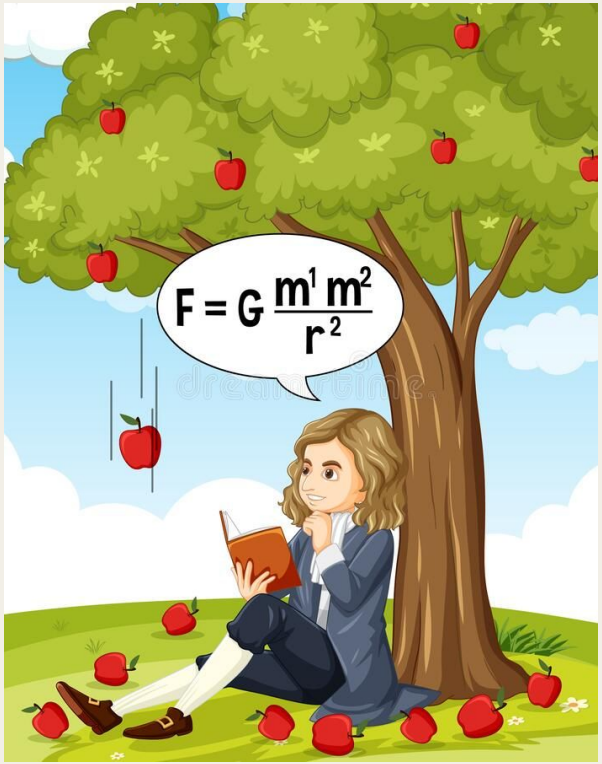
- Goal of simulation:
  - Given initial state at time t
  - Compute next state at time t + 1

- Newton's 2nd Law of Physics:

$$F = ma$$

- Applying a **force** on an object of some **mass** will cause it to **move** with some **acceleration**

$$a = F/m$$

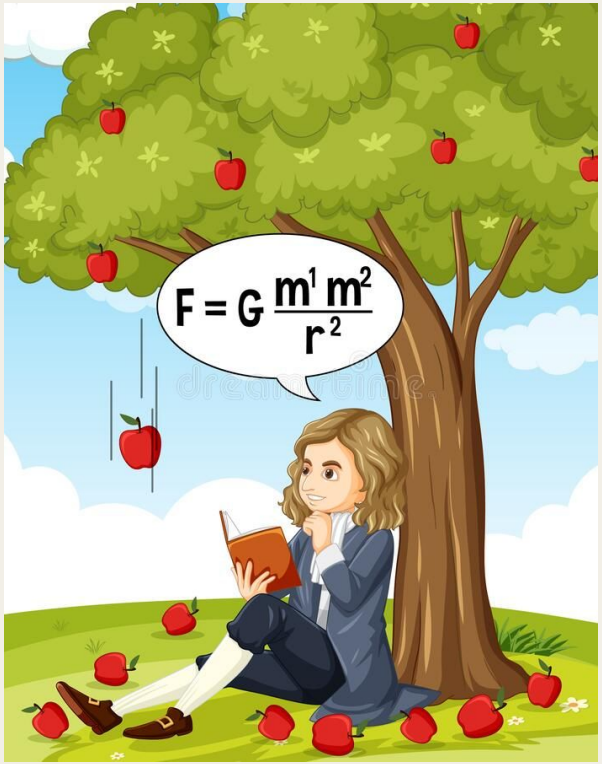


Types of Forces

- Newton's 2nd Law of Physics:

$$F = ma$$

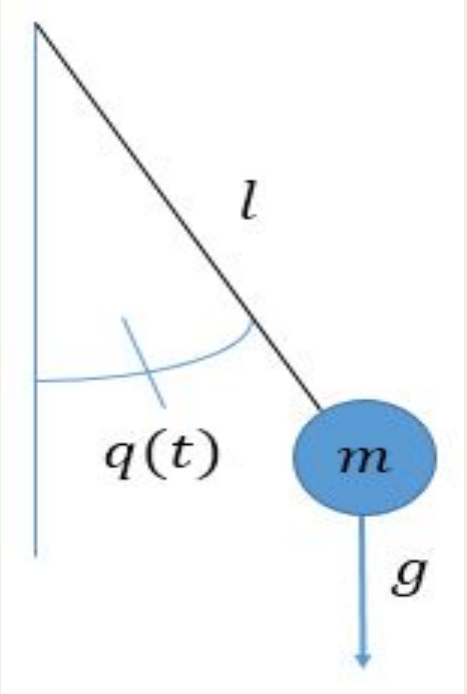
- Constant forces, e.g. gravity
- Time dependent forces, e.g. wind
- Position dependent forces, e.g. magnets
- Velocity dependent forces, e.g. friction
- Position & Velocity dependent forces, e.g. springs



Simple Example: The Pendulum

- **Physical state** of a pendulum at time t:
  - Mass of the object? m
  - Position of the object? q(t)
  - Velocity of the object? v(t)
  - External forces on the object? a(t)
    (the force of gravity leads to an acceleration a(t))

- What we use to define the state of our system can vary depending on the application
  - e.g. here, we use angle for “position”



Simple Example: The Pendulum

- Using our state variables, write our **equations of motion** from t to t + **time step** Δt:

$$a(t) = \ddot{q} = \frac{d^2q}{dt^2} = -\frac{g}{l}\sin q$$

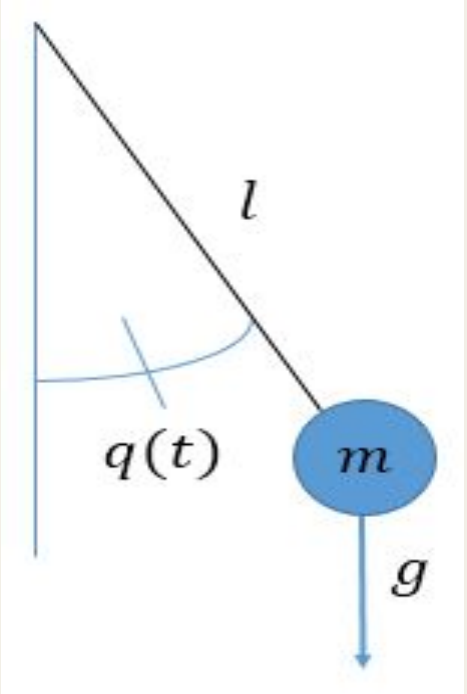
(derived from physics using force of gravity)

$$v(t + \Delta t) = v(t) + (\Delta t)a(t)$$

(velocity final = velocity initial + a * t)

$$q(t + \Delta t) = q(t) + (\Delta t)v(t)$$

(position final = position initial + v * t)



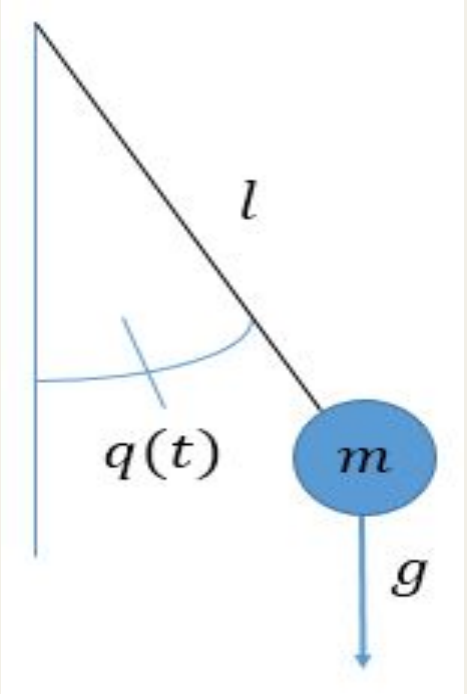
Simple Example: The Pendulum

- This **numerical method** for **updating** our **state** variables from state k to k+1 is called **Explicit Euler** (Forward Euler):

$$v_{k+1} = v_k - (\Delta t)\frac{g}{l}\sin q_k$$

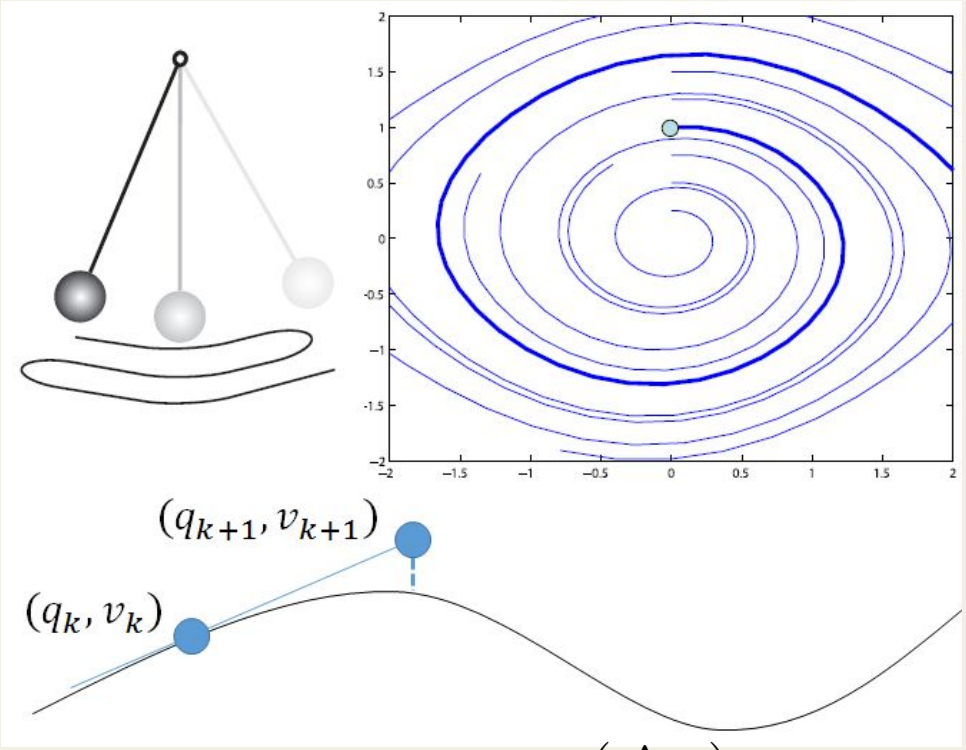
$$q_{k+1} = q_k + (\Delta t)v_k$$

- Explicit Euler one of many “time integrators”



Be Careful with Time Integrators

- Not all time integrators are **stable**
- For Explicit Euler, we may increase the amplitude of the pendulum every update!
- Why? Our equations are approximations (1st order Taylor's series)

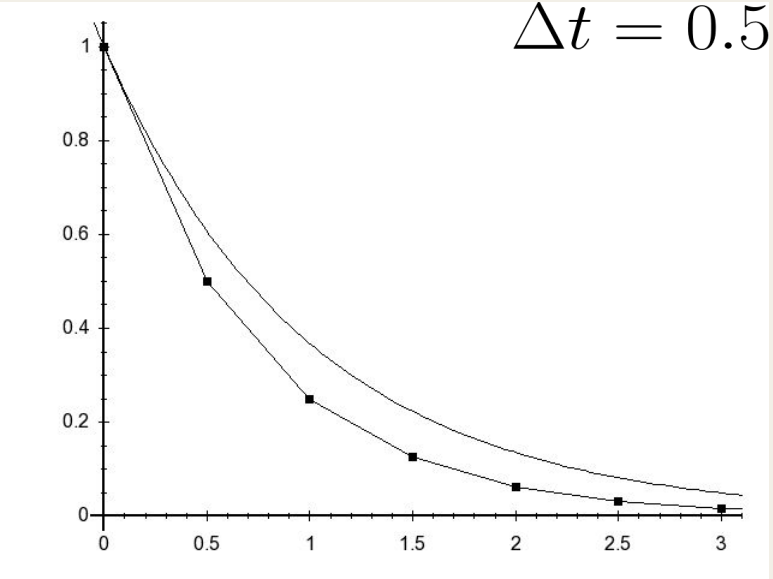


$$q_{k+1} = q_k + (\Delta t) v_k$$

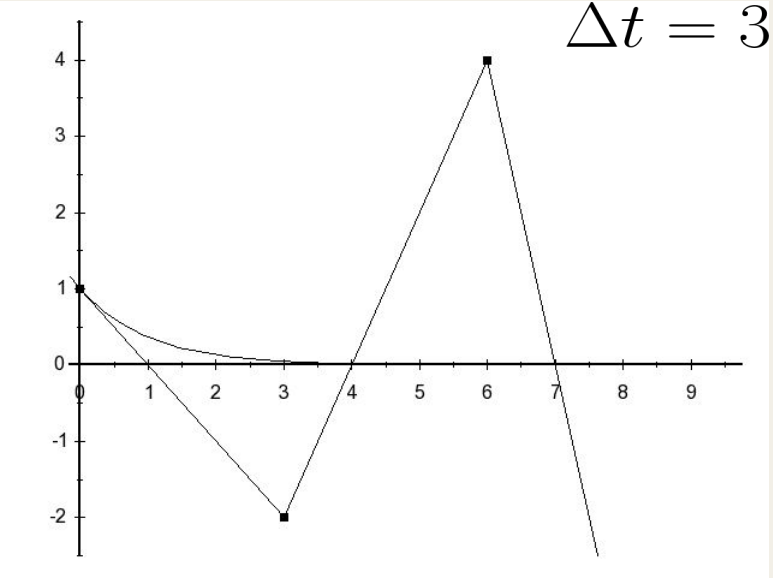
Be Careful with Time Integrators

- The key to stability is in the **time step** Δt
- Example: Forward Euler on the equation $v(t) = -q(t)$

$\Delta t = 0.5$



$\Delta t = 3$



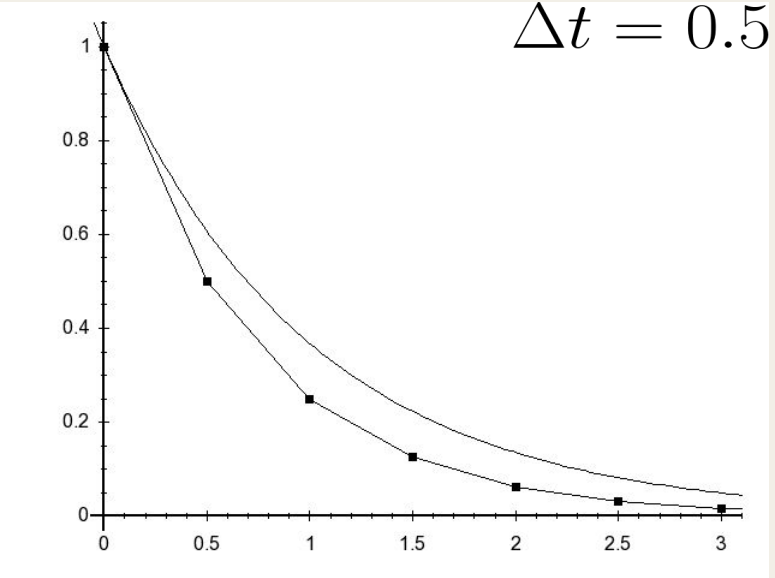
Be Careful with Time Integrators

- The key to stability is in the **time step** Δt
- Example: Forward Euler on the equation $v(t) = -q(t)$

$$q_{k+1} = q_k + (\Delta t)v_k$$
$$= q_k - (\Delta t)q_k$$
$$= (1 - \Delta t)q_k$$

- Exponential decay when:

$$|1 - \Delta t| < 1$$
$$-2 < -\Delta t < 0$$



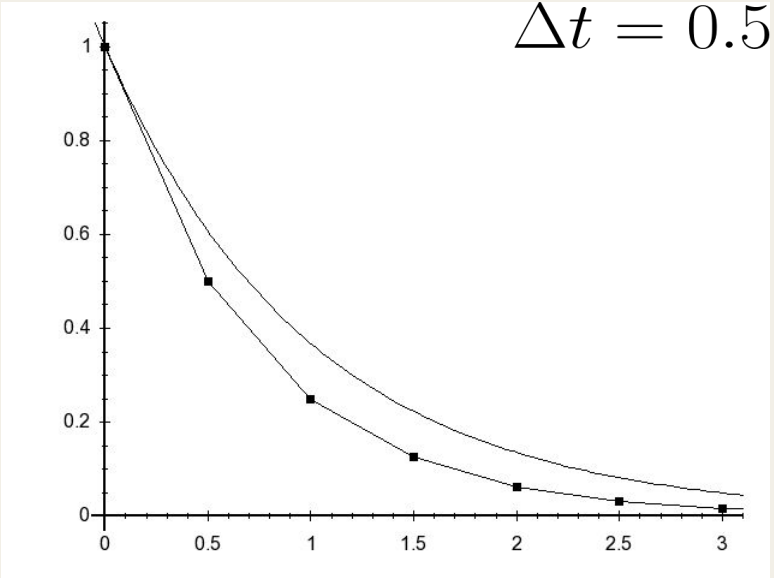
Be Careful with Time Integrators

- The key to stability is in the **time step** Δt
- Example: Forward Euler on the equation $v(t) = -q(t)$
  Stable when:

$$|1 - \Delta t| < 1$$

$$-2 < -\Delta t < 0$$

- Can do similar analyses for other applications of Explicit Euler to find the **time step restriction**
- Some applications use **adaptive time steps** that change over time



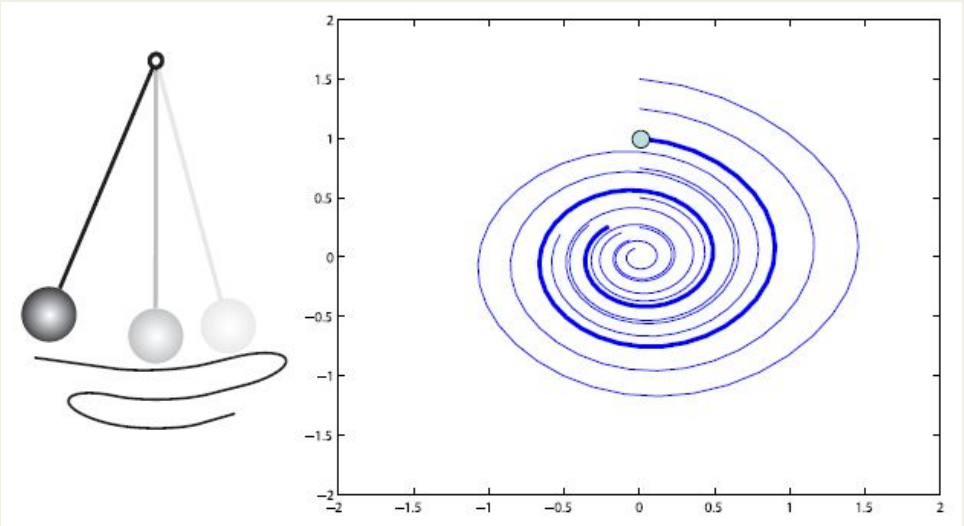
Other Time Integrators

- There are many time integrators besides Explicit Euler, and some are still being developed for various applications!
- Consider **Implicit Euler** (aka Backward Euler):

$$q_{k+1} = q_k + (\Delta t) v_{k+1}$$

$$v_{k+1} = v_k + (\Delta t) a_{k+1}$$

- Can do analysis to see this is stable for ALL time steps!
- But requires implicit solve and may cause damping



Other Time Integrators

- The Runge-Kutta methods are a series of popular time integrators

- Intuition with Runge-Kutta 2 (RK2):
  Take 2 Forward Euler steps:

$$q_{k+1} = q_k + (\Delta t) v_k$$

$$q_{k+2} = q_{k+1} + (\Delta t) v_{k+1}$$

  Then average:

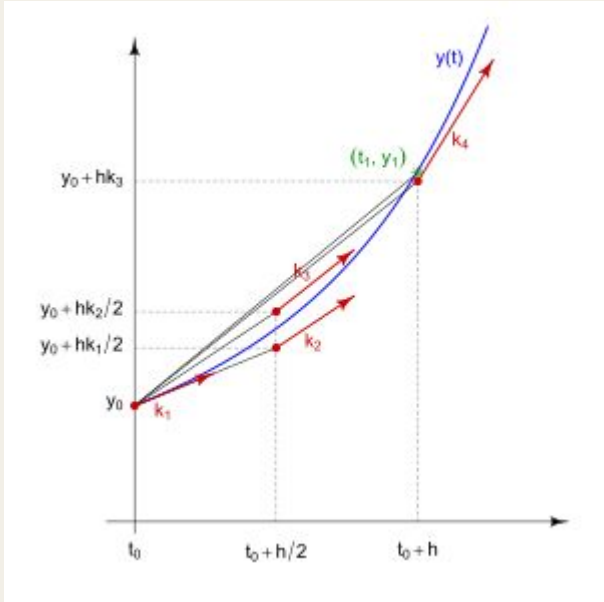
$$q_{k+1} = \frac{1}{2}(q_k + q_{k+2})$$

Other Time Integrators

- The Runge-Kutta methods are a series of popular time integrators
- Most popular is Runge-Kutta 4 (RK4): Weighted-average of 4 steps:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\,h,$$
$$t_{n+1} = t_n + h$$

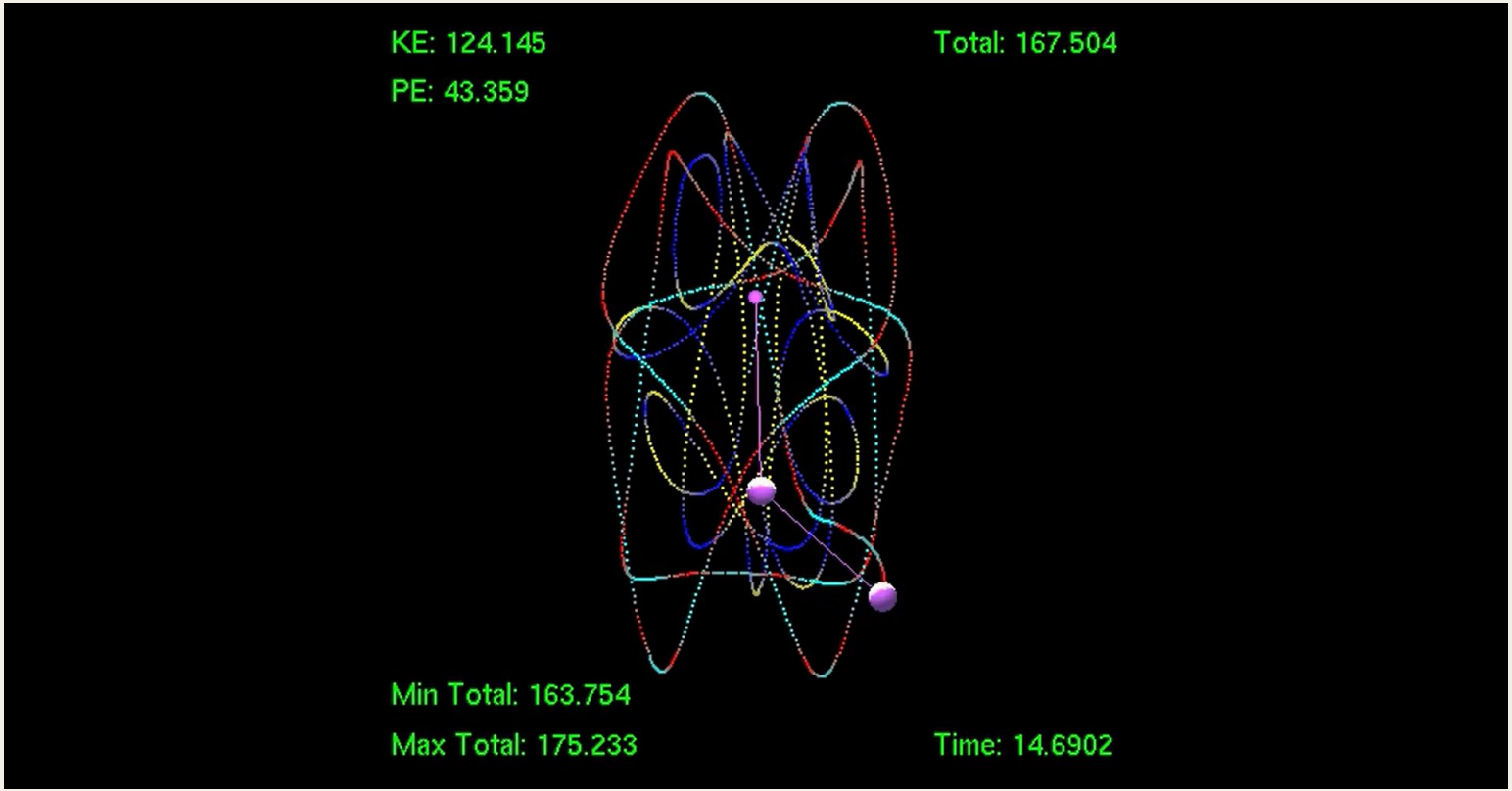
$$k_1 = f(t_n, y_n),$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right),$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right),$$
$$k_4 = f(t_n + h, y_n + hk_3).$$



Pendulum Simulation in Action

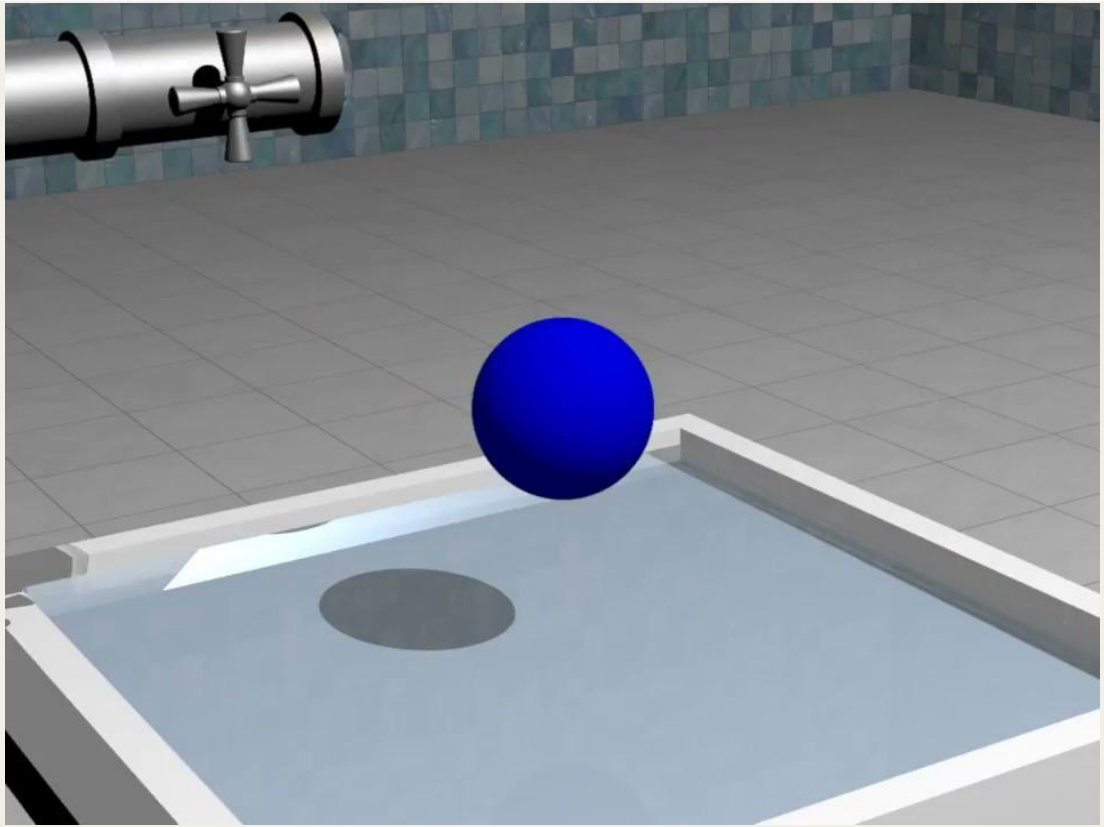
https://drive.google.com/file/d/1CuD_AHgjv5KBP7J-6R-MMCuVstm7Q-DR/view?usp=drive_link

Pendulum Simulation in Action



Questions?

Fluid Simulation

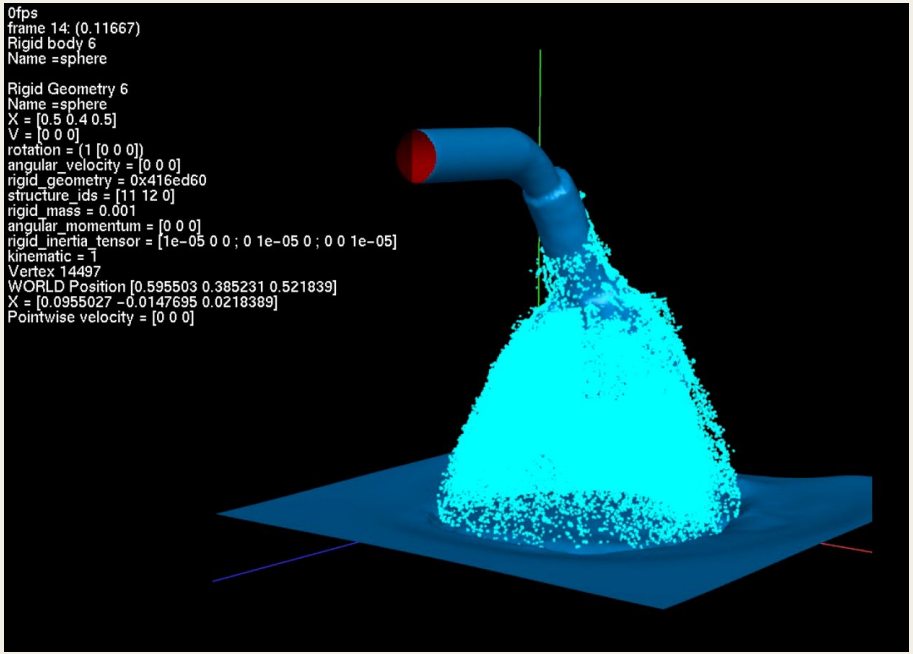


https://drive.google.com/file/d/1-elKs0--Ykc3B2ygcCvNa74E6G-OOItx/view?usp=drive_link

Fluid Simulation

- **Fluid simulation** refers to the simulation of liquids and gases as they move around in some space over time due to some force(s)
- Still ongoing research!

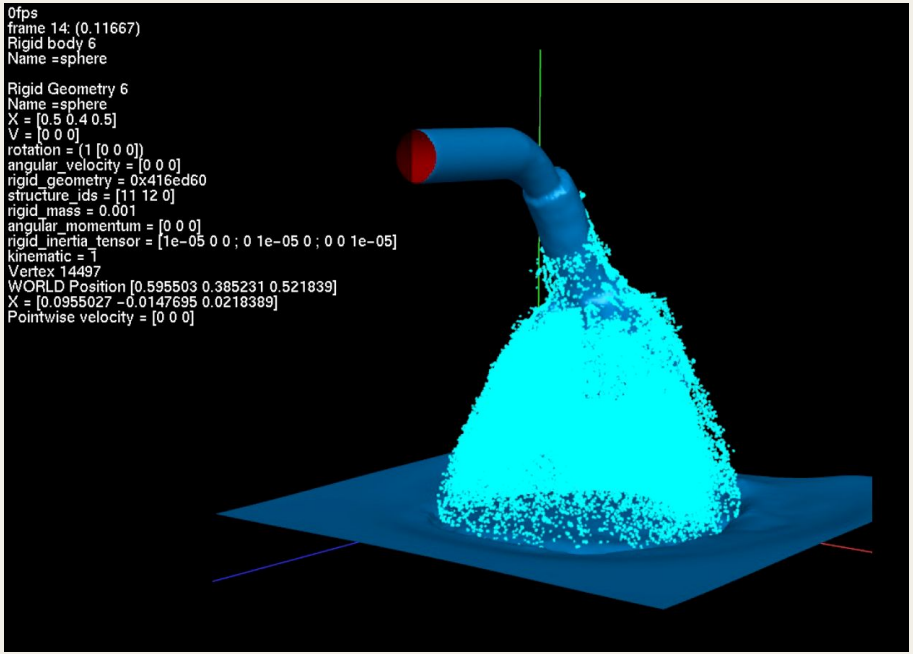
- Many methods have been developed over the years
- We'll talk about this from a **very high level**



Fluid Simulation: Lagrangian Particle System

- **Lagrangian fluid simulation** tracks lots of **particles** as they move through space and time

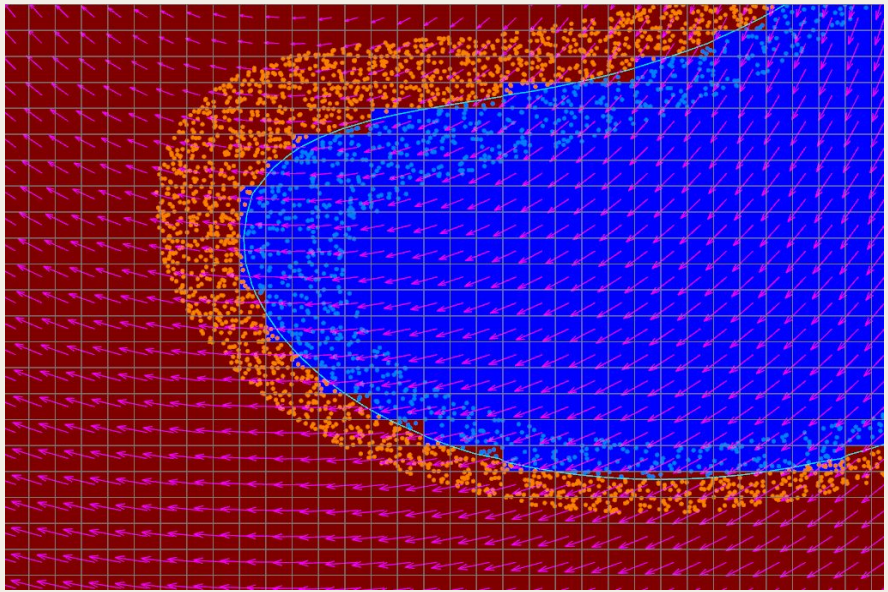
- Each particle has its own physical state at time t of:
  - Mass
  - Position
  - Velocity
  - Force / Acceleration



Fluid Simulation: Fluid Domain

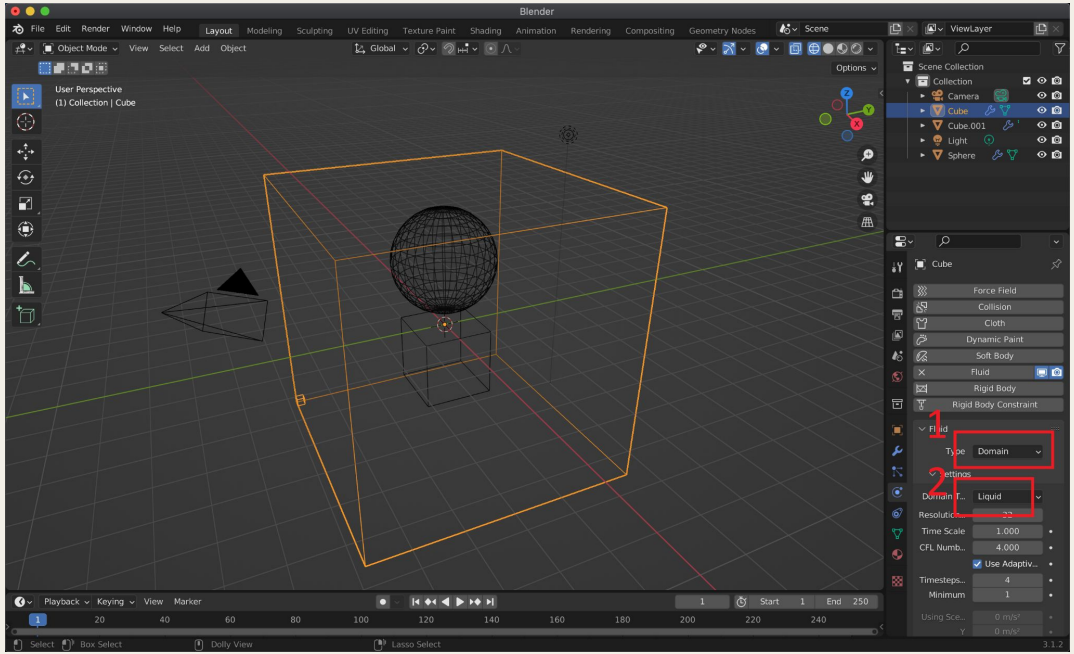
- We track the particles in space using a **fluid domain**
- In 2D we use a rectangular grid, in 3D we use a bounding volume

- 2D Example –
- Each **grid cell** stores **velocity vectors** along edges
- Essentially, the domain stores a **velocity field**
- These velocities are used to move/**advect** the particles in the cell to the next state



Fluid Simulation: Fluid Domain

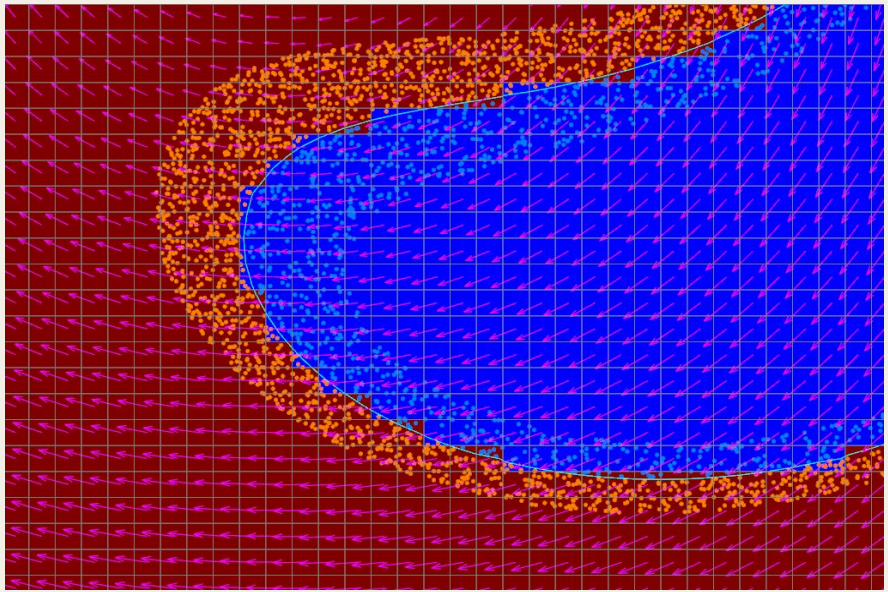
- We track the particles in space using a **fluid domain**
- Blender fluid domain with a gravity force field



Fluid Simulation: Eulerian Fluid Domain

- We sometimes also represent the fluid with the fluid domain itself by marking grid cells as “on” or “off” if they’re in the fluid of interest

- **Eulerian fluid simulation** tracks the fluid with the **grid**
- In the example –
  blue grid cell = water
  red grid cell = air
- Check if a grid cell is “inside” the water domain and mark the cell blue if so



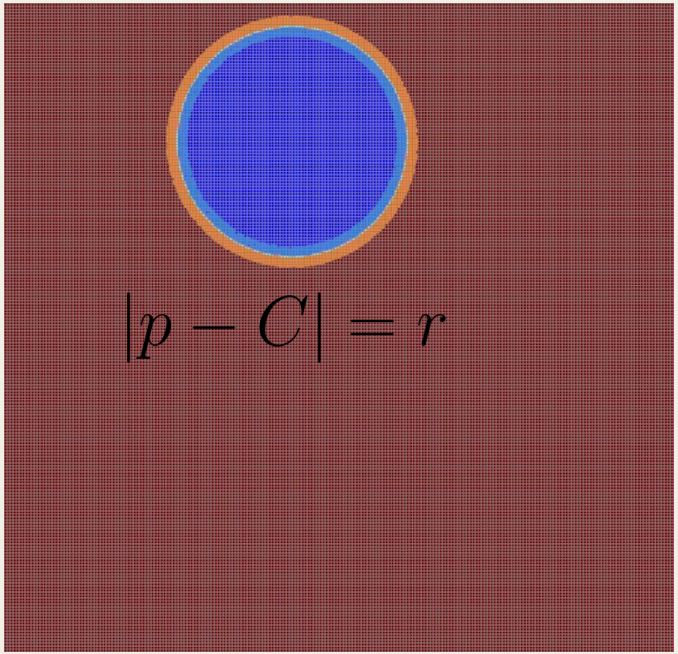
Fluid Simulation: Implicit Surfaces

- How do we know if we're "inside" the fluid we care about?
- Recall **implicit surfaces**, e.g. when we ray traced implicit spheres

- Consider starting with a very simply shaped fluid, e.g. a circle of water
- Can define the **implicit function** as a **signed distance function**:

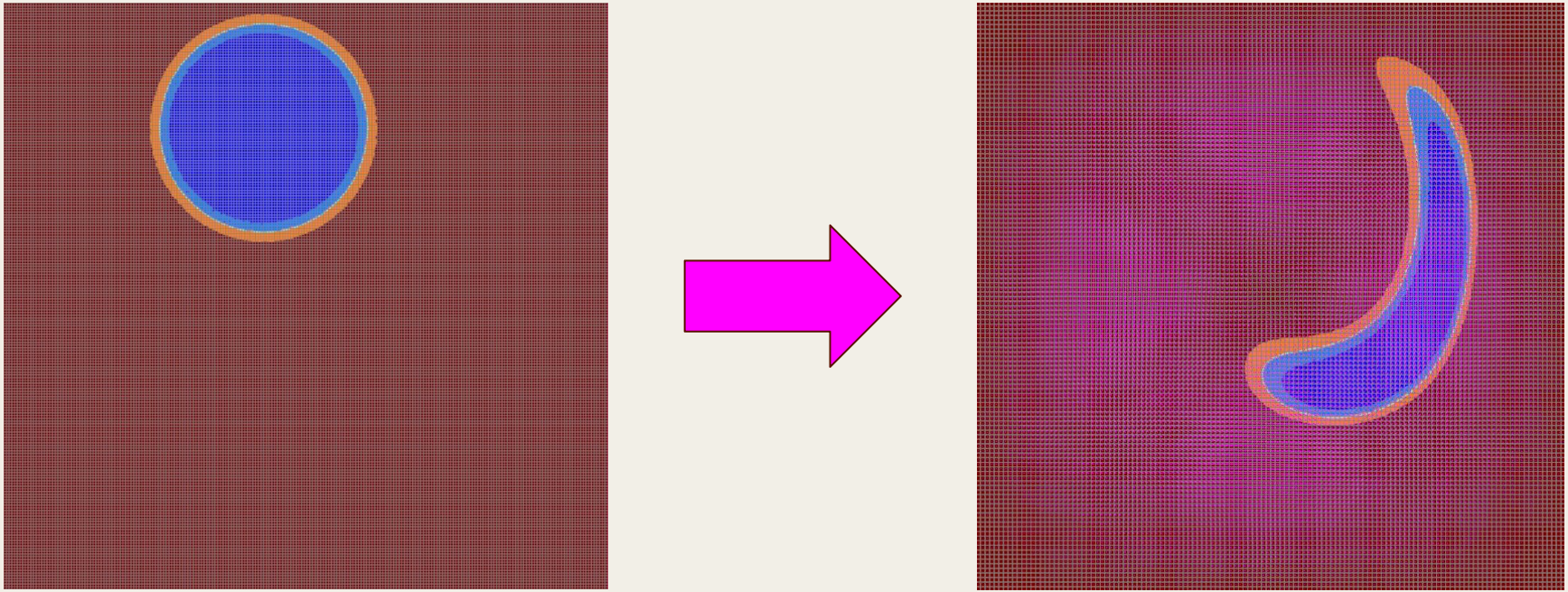
$$\phi(p) = \sqrt{(p - C) \cdot (p - C)} - r$$

- If = 0, then on boundary of water/air
- If < 0, then inside circle (is in water)
- If > 0, then outside circle (is in air)



Fluid Simulation: Implicit Surfaces

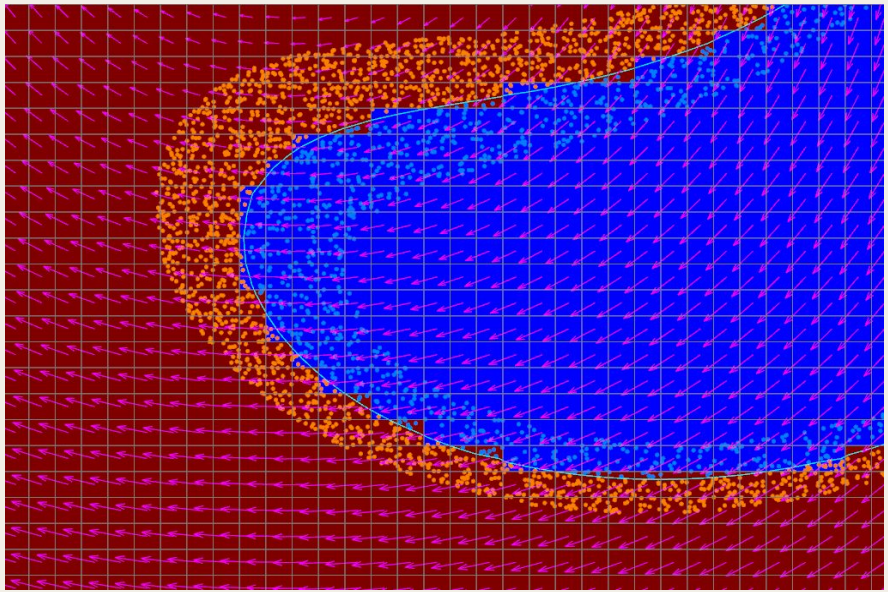
- Storing the **implicit function** values on each grid cell turns the grid into a **signed distance field** that can move with the **velocity field**



Fluid Simulation: Mix of Lagrangian & Eulerian

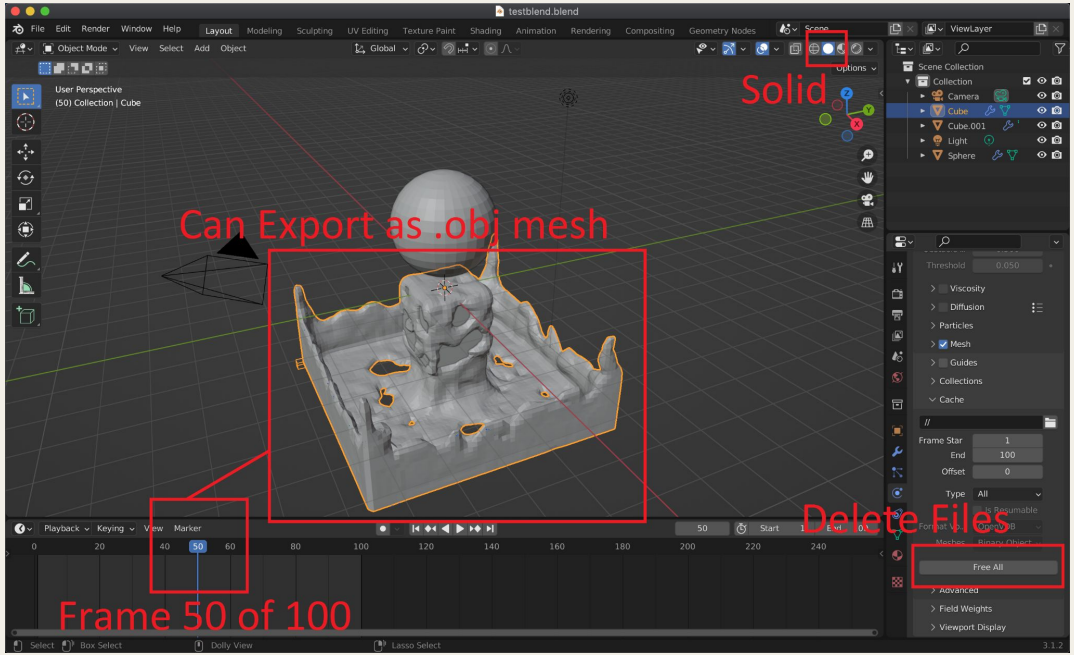
- Can use **Lagrangian marker particles** for the **fluid boundary**, then use an **Eulerian grid** to represent the rest of the fluid domain

- Example –
- Orange and light blue marker particles track the boundary between water and air
- Only need to store in memory a “few” marker particles, then just use an inexpensive grid to track the “inside” of the water



Fluid Simulation: Generating a Mesh

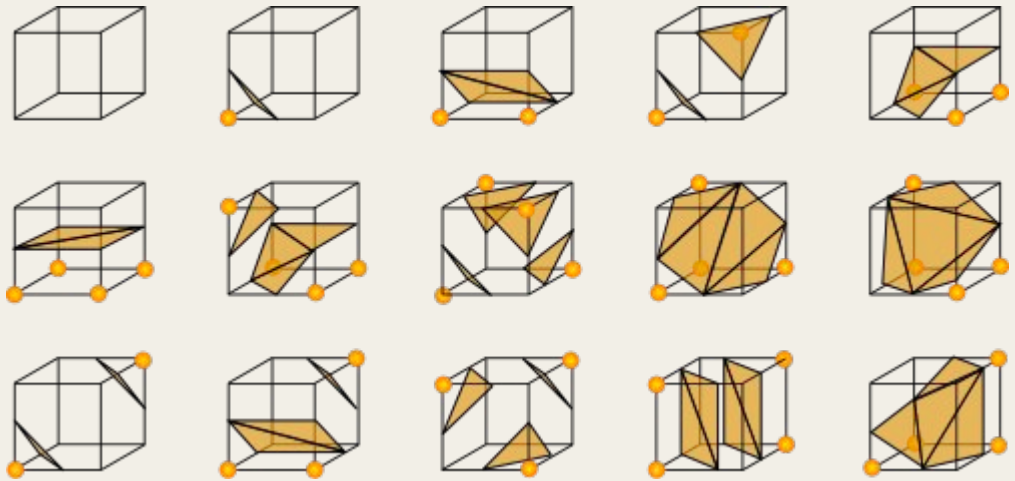
- After doing our fluid simulation, how do we turn it into a mesh?
- Think back to tessellating a sphere – similar idea, but more complex



Fluid Simulation: Generating a Mesh

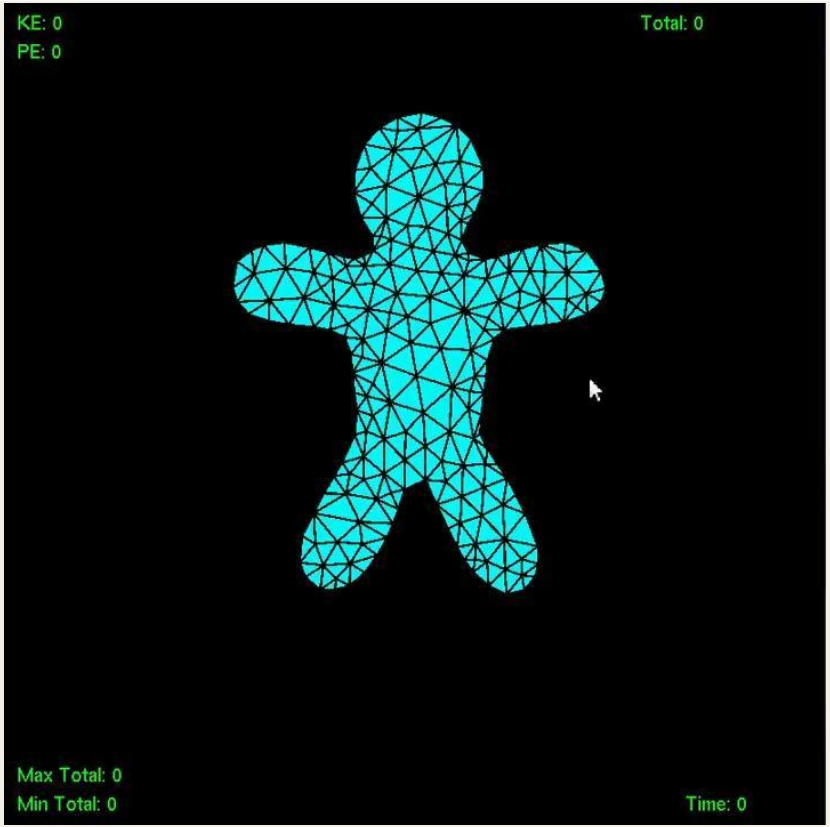
- After doing our fluid simulation, how do we turn it into a mesh?
- Think back to tessellating a sphere – similar idea, but more complex

- One such algorithm is **marching cubes**
- Check each grid cell of the fluid domain
- 2^8 = 256 configurations of polygons based on which corners of the cell are “inside” e.g. water



Questions?

Elasticity in Action



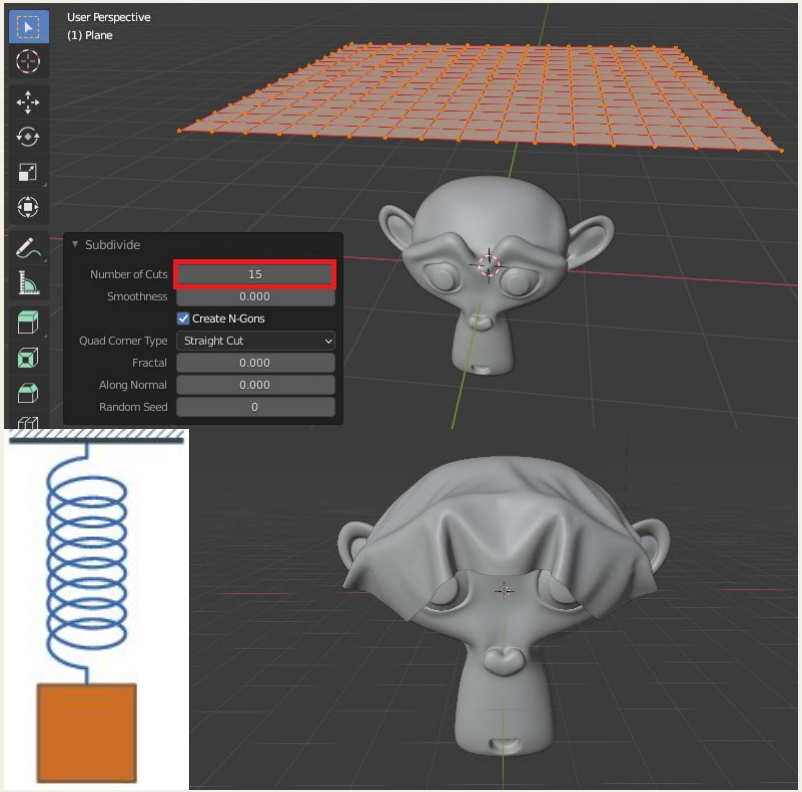
https://drive.google.com/file/d/1QW-xRso_LhAtCgOBVhWDr302dJ0QUzda/view?usp=drive_link

Elasticity for Cloth Simulation

- **Spring forces** are position and velocity dependent forces of the form:

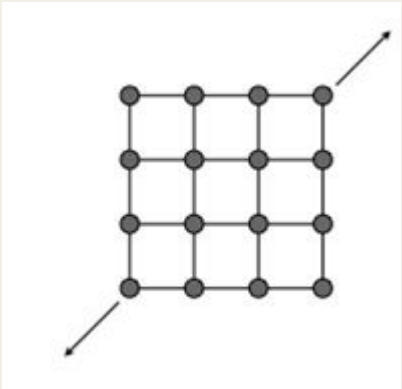
$$F_{spring} = -kq(t) - k_d v(t)$$

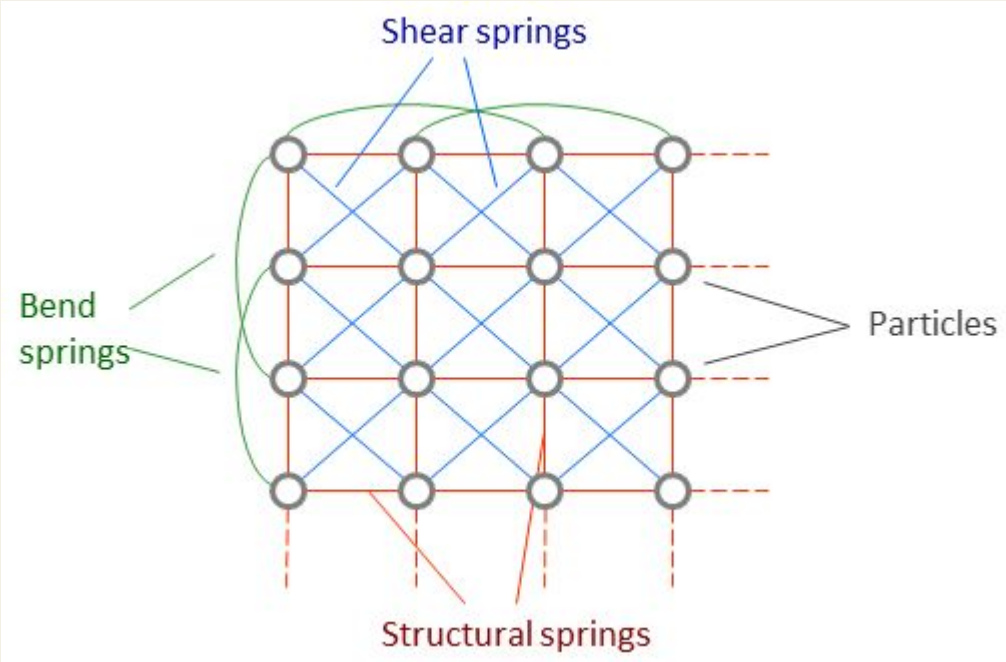
- Simplest **cloth simulation** models the cloth as a grid of **particles** that connect via a **network of springs**
- Also still ongoing research!



Elasticity for Cloth Simulation

- **Structural springs** connect particles
- **Shear springs** resist shear motion



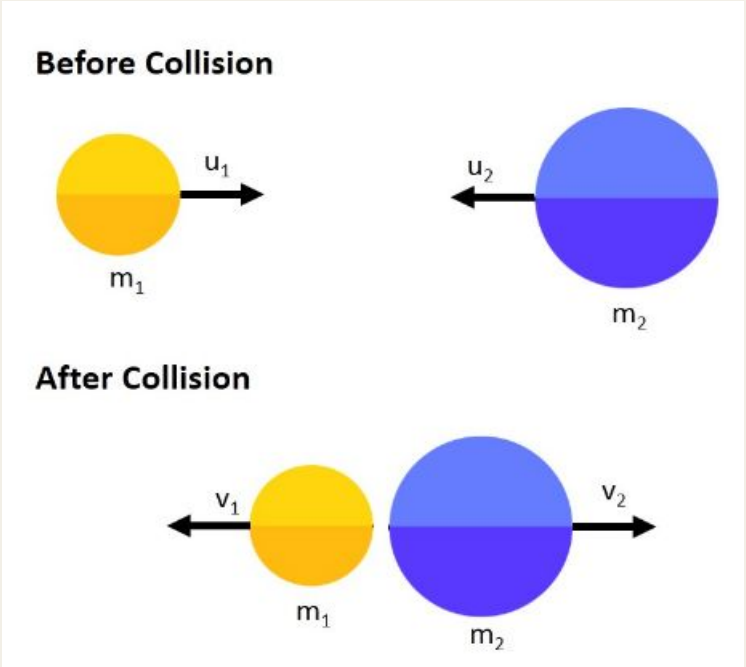


- **Bend / Flexion springs** resist out-of-plane motion

Simulating Collisions: 1D

- Suppose 2 particles a & b each with their own mass and velocity collide
- Newton's 3rd law says every action has an equal and opposite reaction, i.e. the force on object a equals the force on object b

- Leads to the idea of **conservation of momentum**



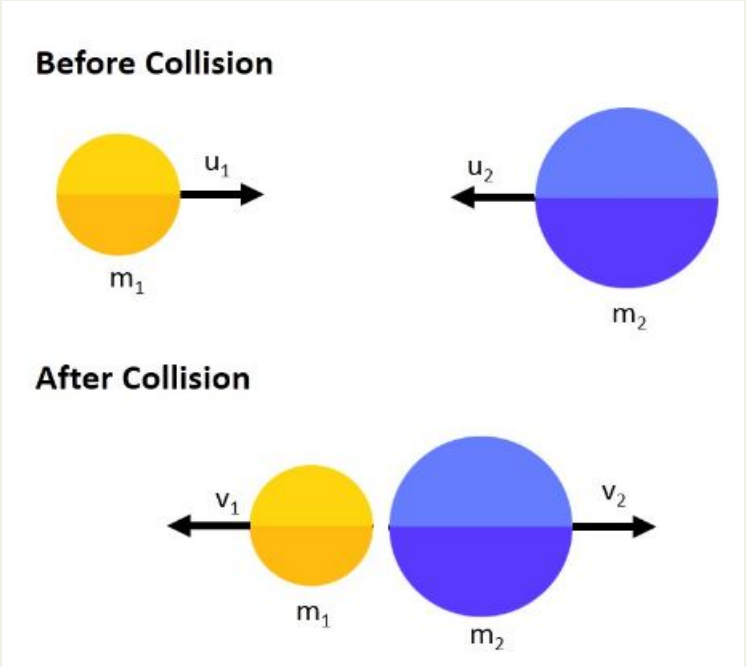
$$m_a u_a + m_b u_b = m_a v_a + m_b v_b$$

Simulating Collisions: 1D

- Define the **coefficient of restitution**

$$c_R = -\frac{v_b - v_a}{u_b - u_a}$$

- If 0, then objects stick together after collision, i.e. **inelastic collision**
- If 1, then objects bounce off each other without losing any energy, i.e. completely **elastic collision**
- If between 0 and 1, then some of the **energy is lost to heat, etc**



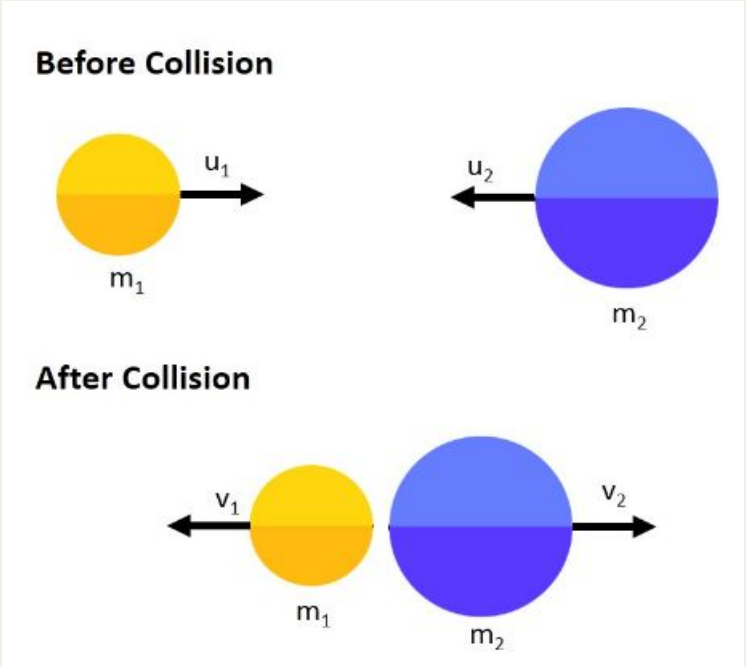
$$m_a u_a + m_b u_b = m_a v_a + m_b v_b$$

Simulating Collisions: 1D

- Set the **coefficient of restitution** to some desired model value:

$$c_R = -\frac{v_b - v_a}{u_b - u_a}$$

- We know the velocities before the collision
- Use equation for cR and equation for conservation of momentum to **solve for the new velocities** after collision



$$m_a u_a + m_b u_b = m_a v_a + m_b v_b$$

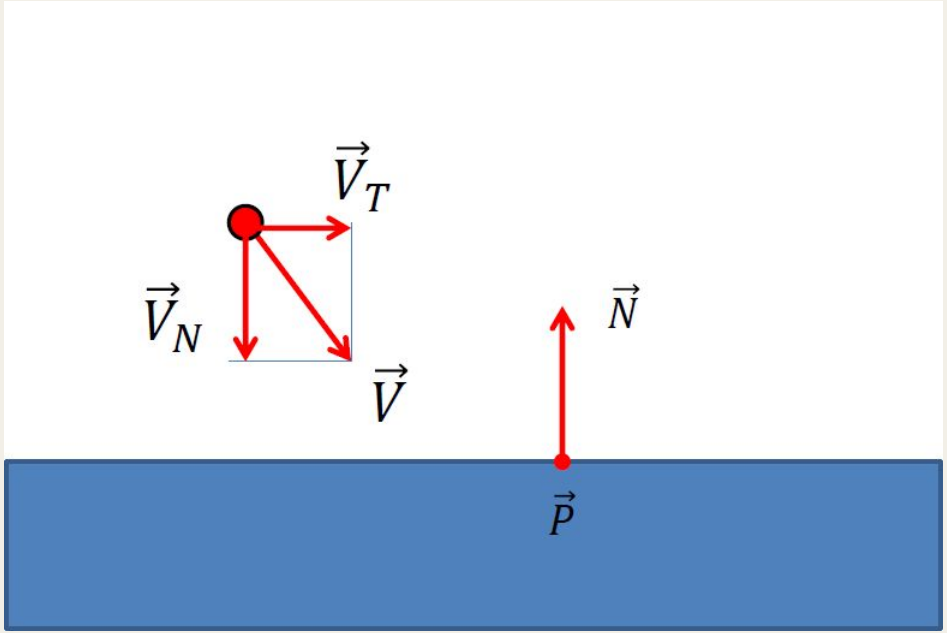
Simulating 3D Collisions: Point-Plane

- Remember most of our geometry will be triangles, so most collisions will be **point-plane intersections**

- Reuse ideas from ray-plane intersections!
  - A particle's initial position is like the origin of a ray
  - A particle's velocity is like the direction of a ray

Simulating 3D Collisions: Point-Plane

- Note: collisions with planes only involve the **normal component** of the velocity
- This simplifies point-plane collisions in 3D into a 1D collision problem!
- Solve using just the normal components of velocities



- **Tangential components** stay the same unless **friction** is involved

Questions?