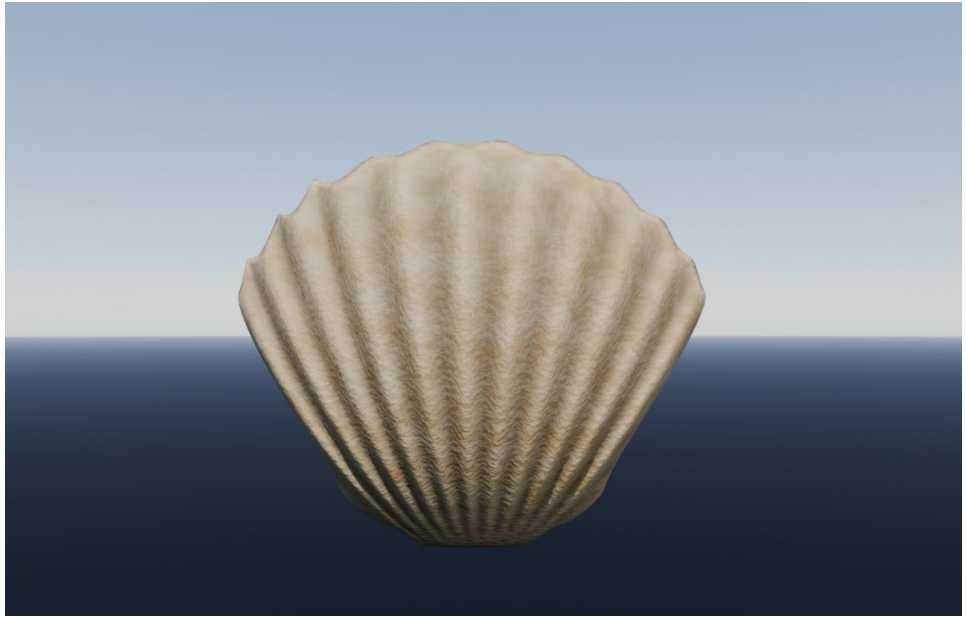
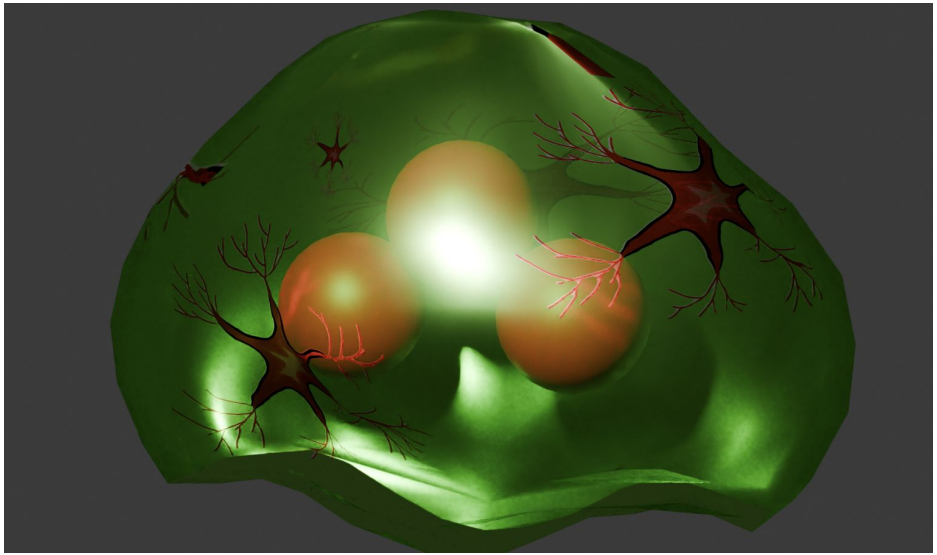
Simulation Part 1

Slides by Sarah Jobalia and Kevin Li, with modifications by Kate Baker

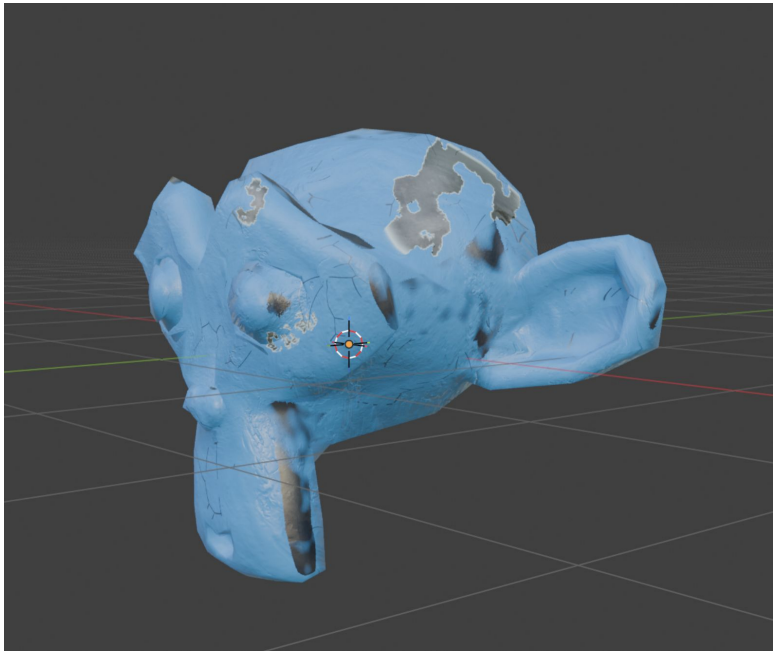
Student Reference Texture Highlights!



Student Reference Texture Highlights!



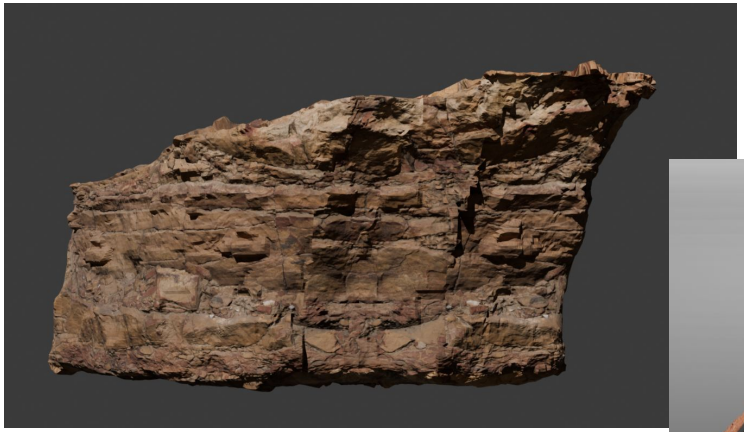
Zia Svendsen



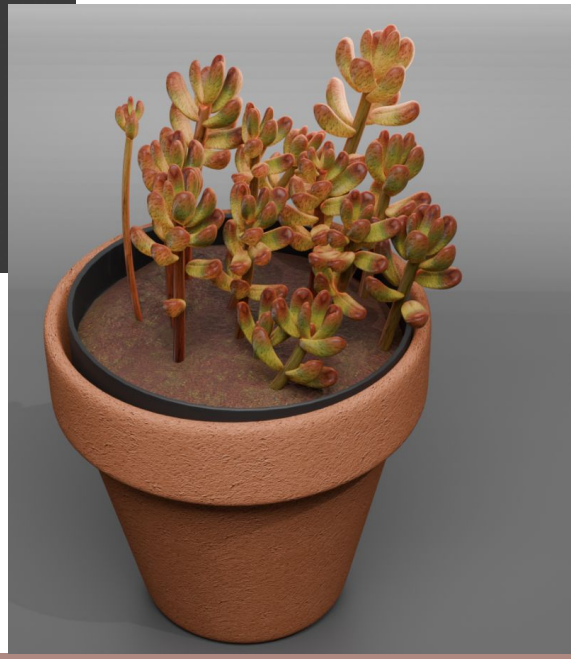
Joshua Starks

Student Reference Texture Highlights!

Steve Chia



Marcus Polson



Ji Ryoo

Student Reference Texture Highlights!

Callie Liao

Sole Mayeski

Housekeeping

- HW5 released (Last one!)
- Cat Hicks additional resources posted on Ed
- Matt Larsen guest lecture – Scientific Visualization!
- No post processing (no composite nodes)
- Project proposal grades out
  - 9.99/10 ~= 10/10
  - ^ To get your attention in case we really wanted you to see some feedback

Lecture Outline

- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

Doug James, Simulation / Interactivity

Karen Liu, Animation / Robotics

Lecture Outline

- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

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

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

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

- We can use simulation to capture a specific frame of a simulation or animation

- Blender has many tools to do so
  - E.g. simulate splashing water and save the geometry from a specific frame to export

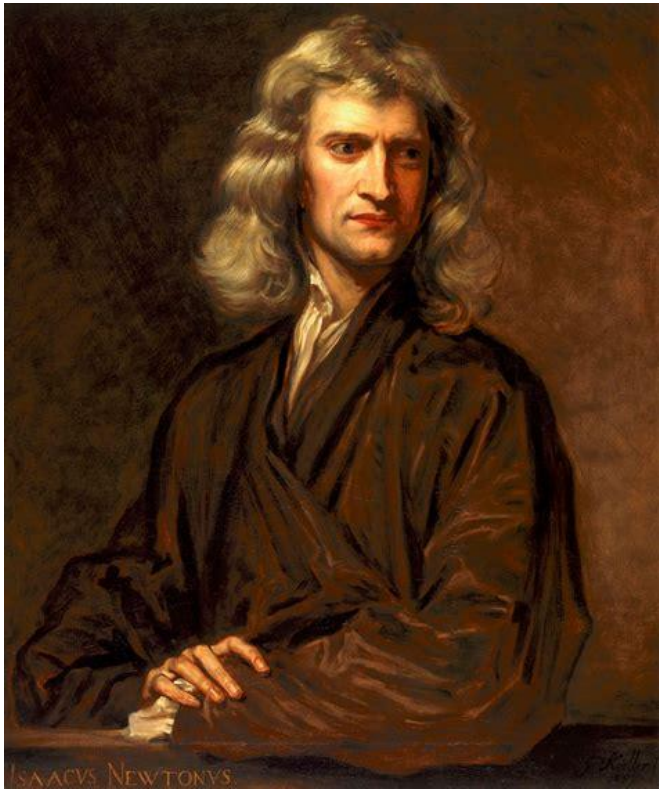
Ryan Eberhardt, 2017

Lecture Outline

- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

Physical Systems

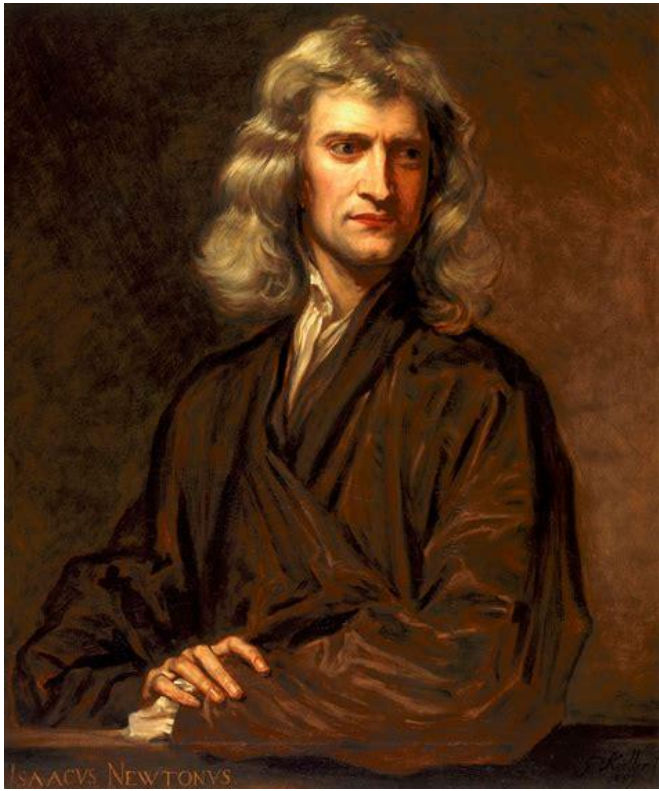
- We model an object in space at some time with a **physical state**:
  - **Mass** of the object
  - **Position** of the object
  - **Velocity** of the object
  - **External forces** on the object



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 an initial state at time *t*, compute the next state at time *t + 1*



Physical Systems

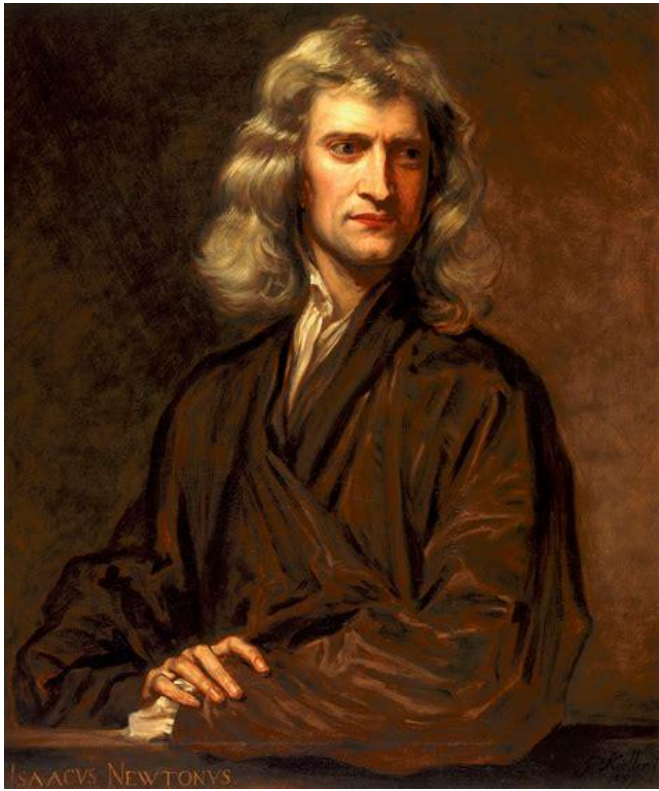
- Goal of simulation
  - Given an initial state at time *t*, compute the next state at time *t + 1*

- 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

$$a = F/m$$

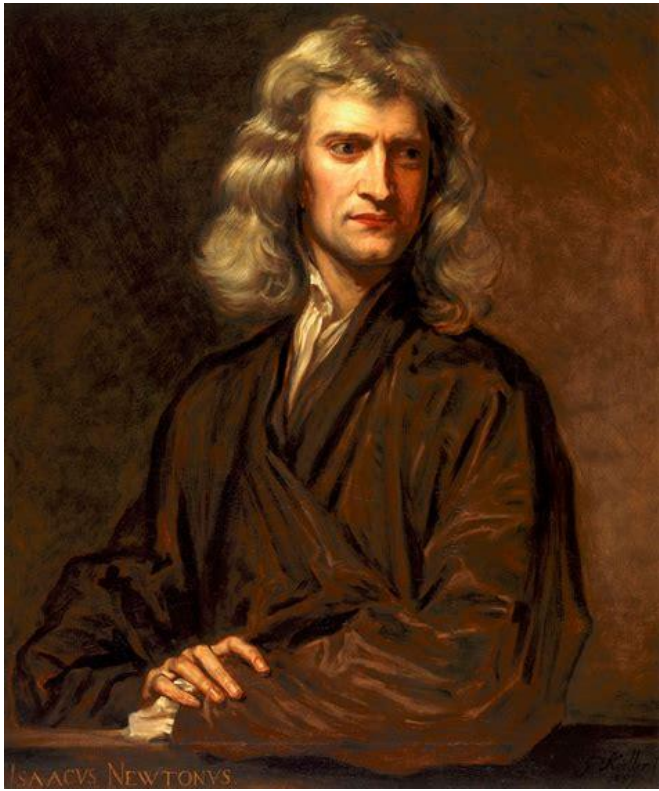


Physical Systems

- Newton's 2nd Law of Physics:

$$F = ma$$

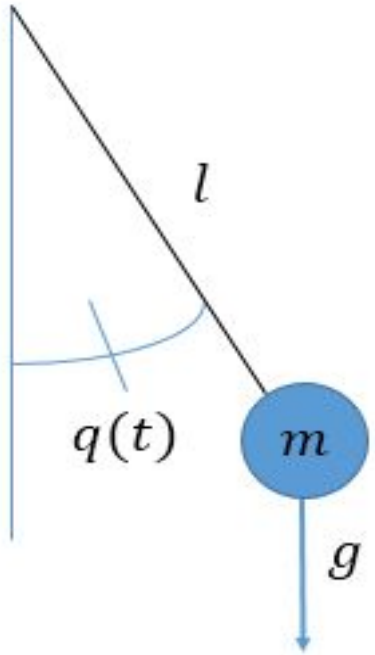
- Types of forces:
  - Constant forces (e.g. gravity)
  - Time dependent forces (e.g. wind)
  - Position dependent forces (e.g. magnets)
  - Velocity dependent forces (e.g. friction)
  - Position and velocity dependent forces (e.g. springs)



Simple Example: The Pendulum

- Physical state of the 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 generates an acceleration a(t)

- We can define the state of our system depending on the application
  - E.g. using angle for position



Simple Example: The Pendulum

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

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

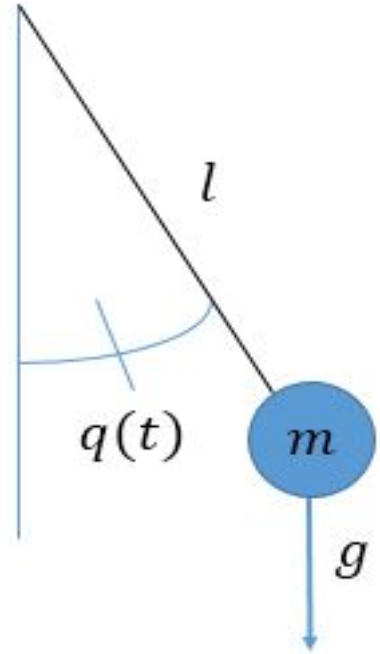
(derived from physics using force of gravity)

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

(velocity final = velocity initial + a * t)

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

(position final = position initial + v * t)



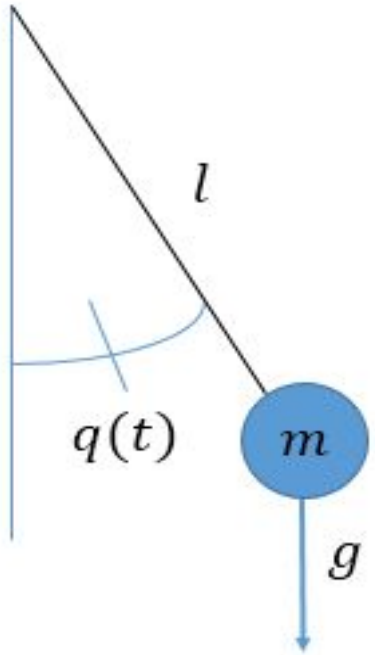
Simple Example: The Pendulum

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

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

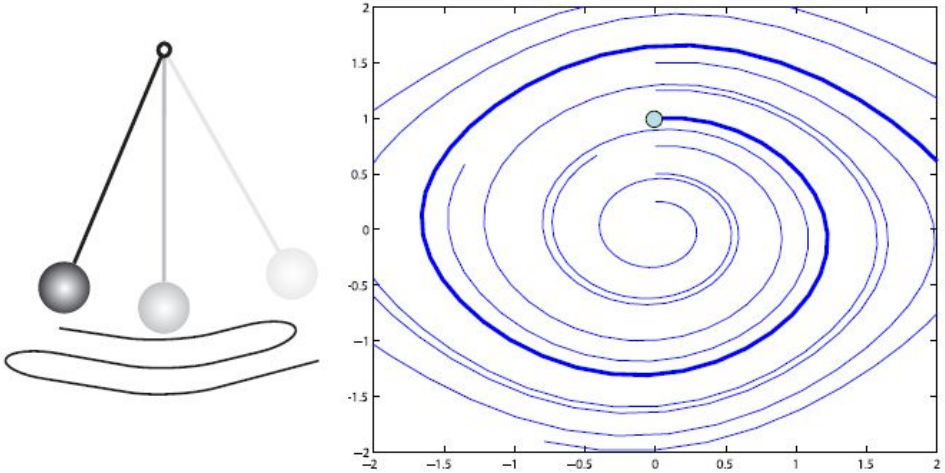
$$q_{k+1} = q_k + (\Delta t)v_k$$

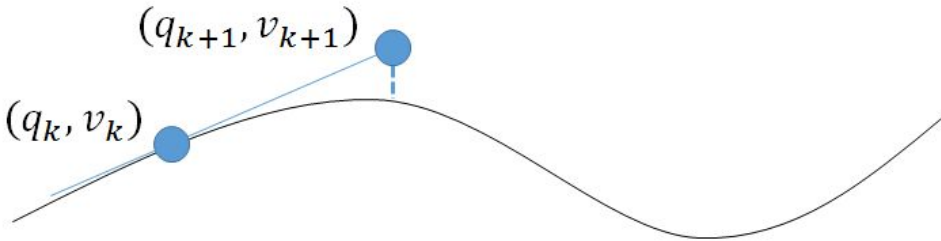
- Explicit Euler one of many “time integrators”



Time Integrators

- Not all time integrators are stable
- For Explicit Euler, might have to increase the amplitude of the pendulum every update
- Our equations are approximations (1st order Taylor series)





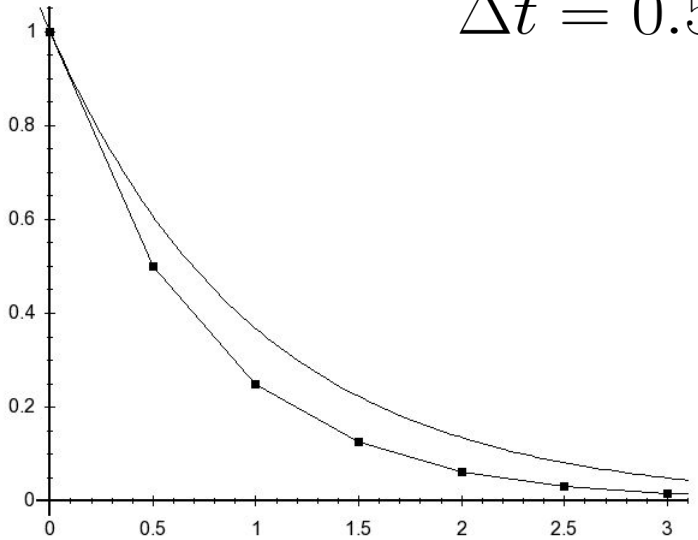
$$q_{k+1} = q_k + (\Delta t) v_k$$

Time Integrators

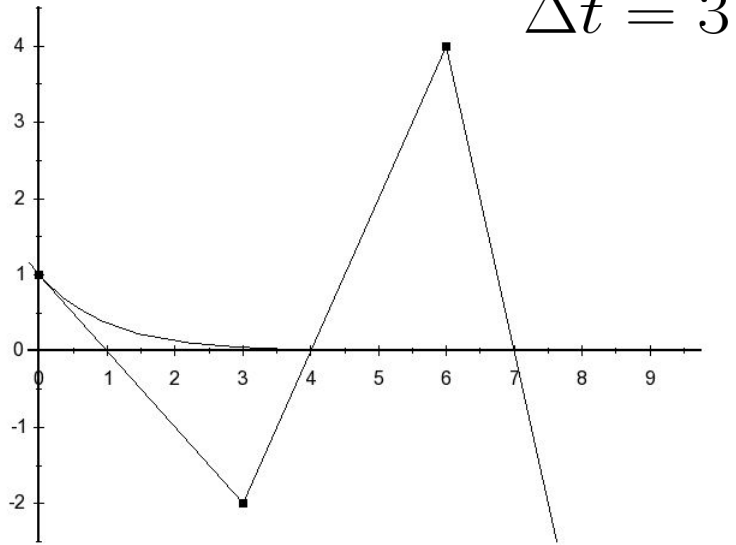
- The key to stability is in the **time step** Δt

- Example: Forward Euler on the equation $v(t) = -q(t)$

$\Delta t = 0.5$



$\Delta t = 3$



Time Integrators

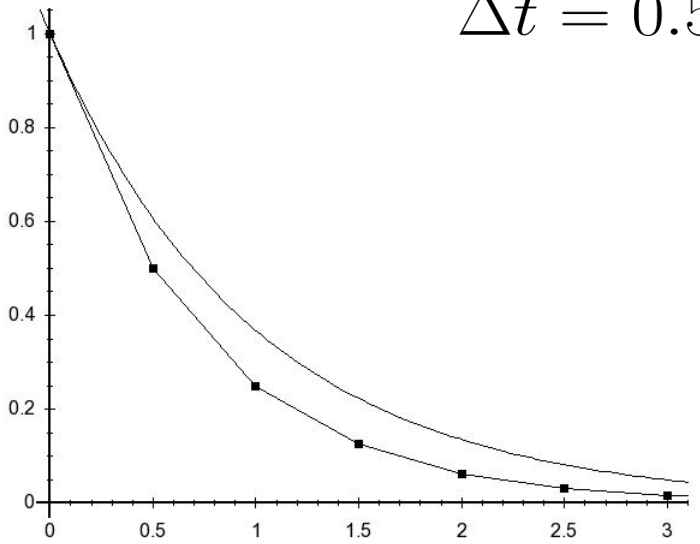
- The key to stability is in the **time step** Δt
- Example: Forward Euler on the equation $v(t) = -q(t)$

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

- Exponential decay when:

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

$\Delta t = 0.5$



Time Integrators

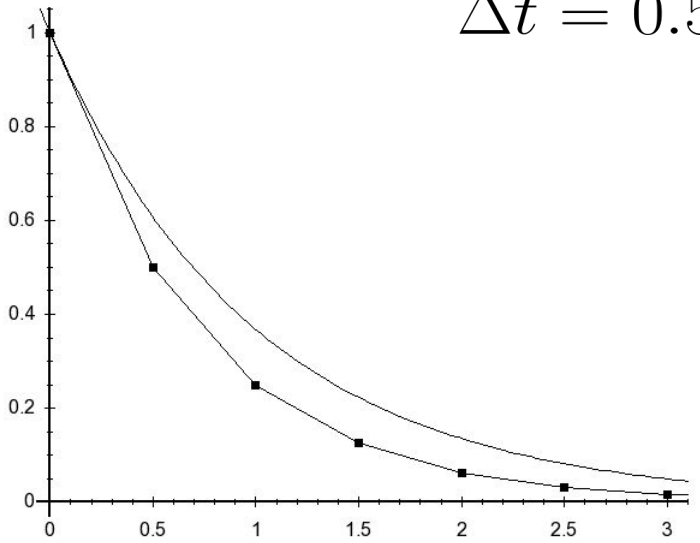
- The key to stability is in the **time step** Δt
- Example: Forward Euler on the equation $v(t) = -q(t)$
  Stable when:

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

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

- 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

$\Delta t = 0.5$

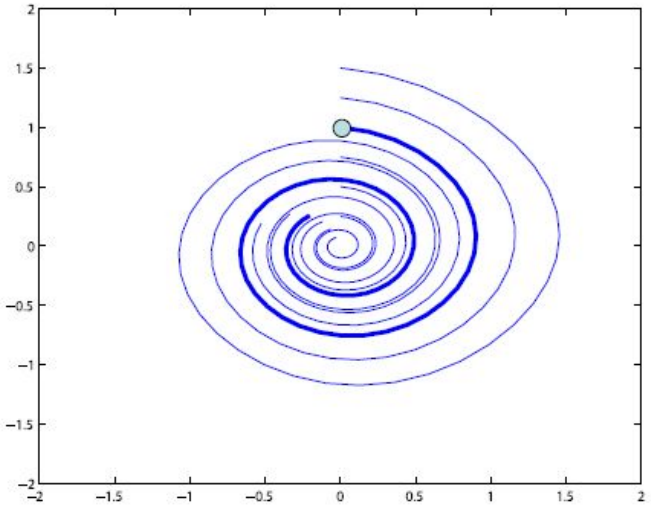


Other Time Integrators

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

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

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



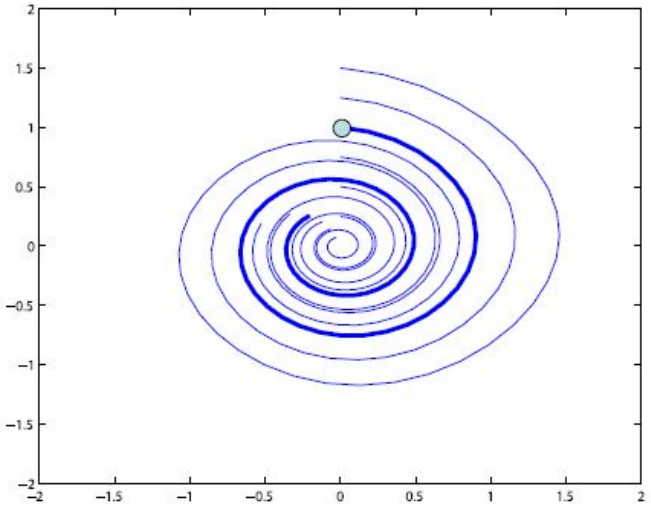
Other Time Integrators

- 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:

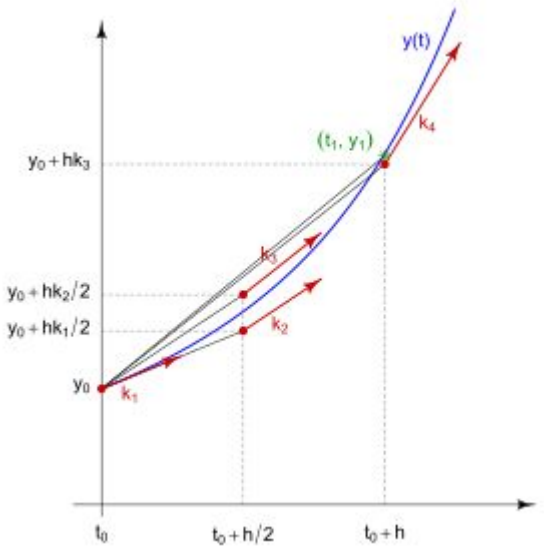
$$q_{k+1} = \frac{1}{2}(q_k + q_{k+2})$$

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:

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

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

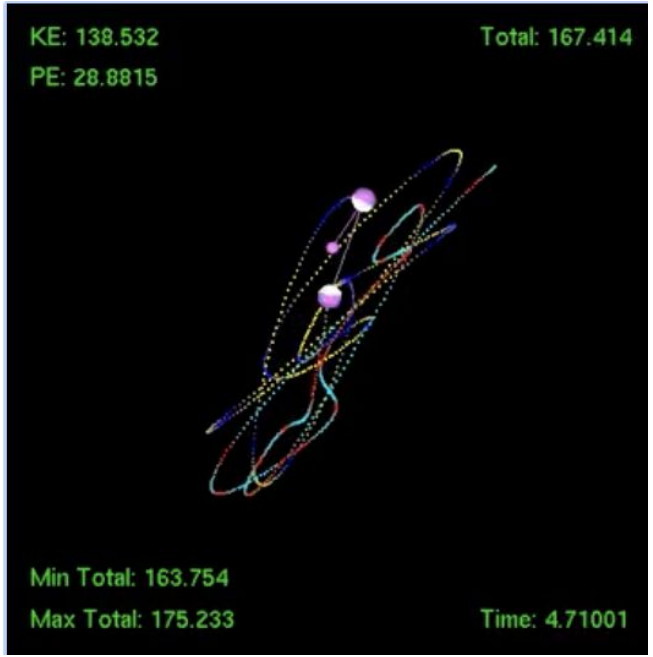


Pendulum Simulation in Action

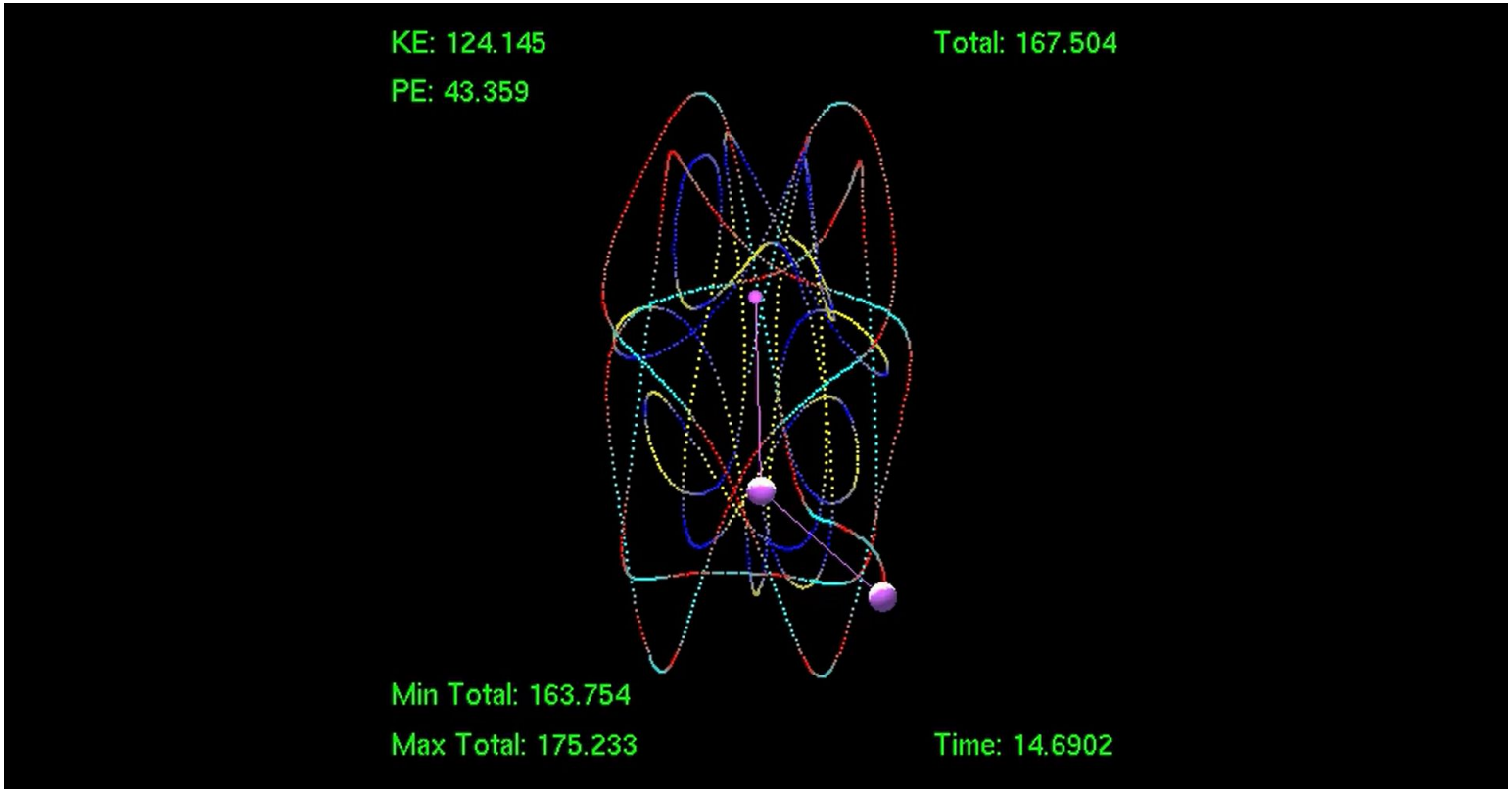
https://drive.google.com/file/d/1CuD_AHgjv5KBP7J-6R-MMCuVstm7Q-DR/view?usp=drive_link

Pendulum Simulation in Action

- PE + KE stays constant in perfect world
- But our computers have numerical imprecision
- Many schemes like explicit/implicit Euler have different ways of handling the error



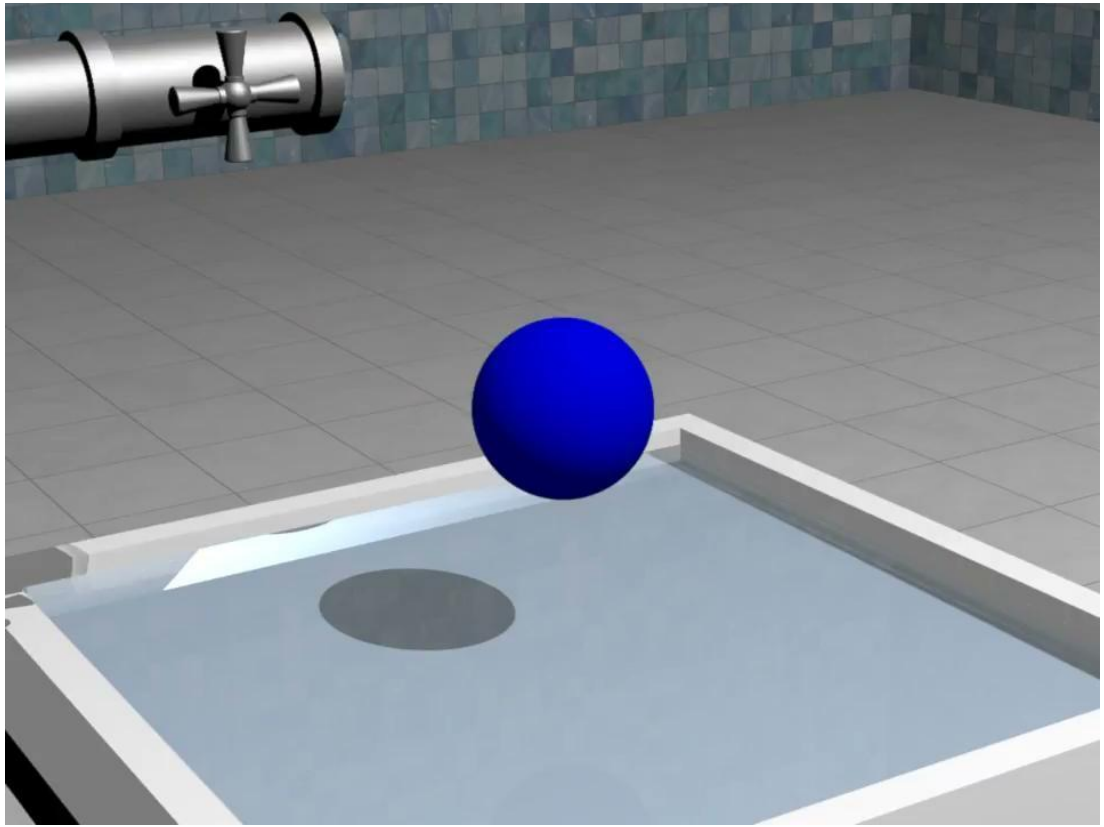
Pendulum Simulation in Action



Lecture Outline

- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

Fluid Simulation

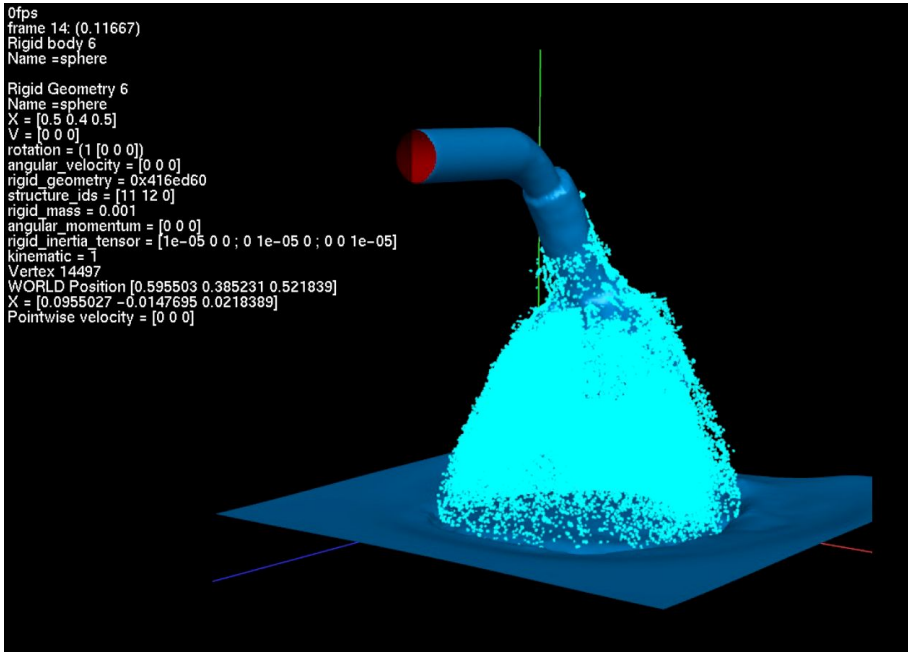


https://drive.google.com/file/d/1-elKs0--Ykc3B2ygcCvNa74E6G-OOItx/view?usp=drive_link

Fluid Simulation

- Refers to the simulation of liquids and gases as they move around in some space over time due to some force(s)

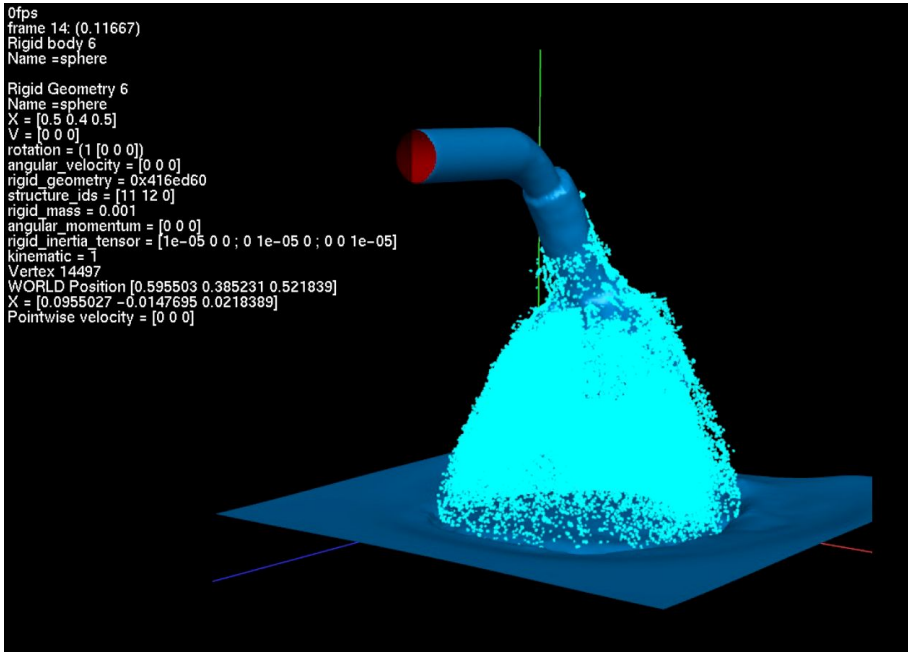
- A huge research space
- Many methods have been developed



Fluid Simulation

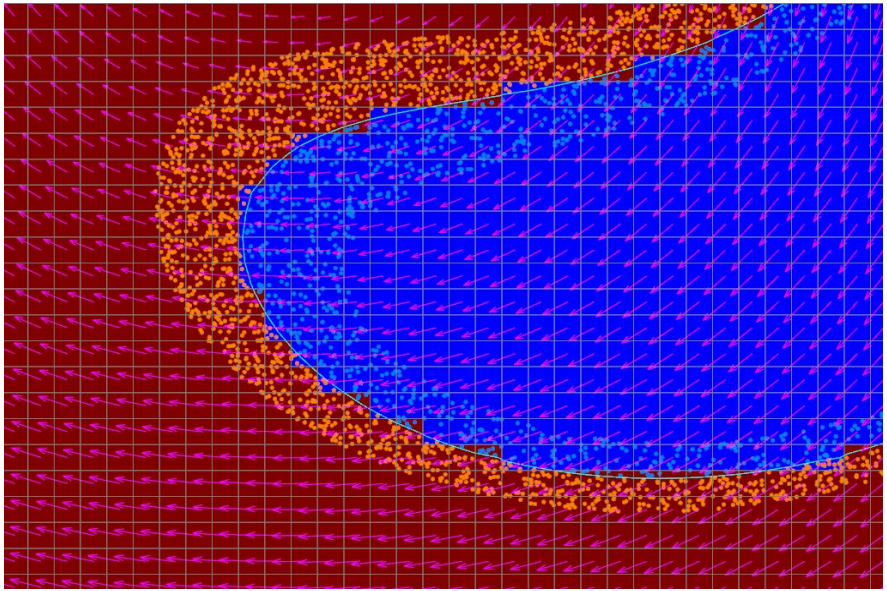
- **Lagrangian fluid simulation** tracks lots of **particles** as they move through space and time

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



Fluid Domain

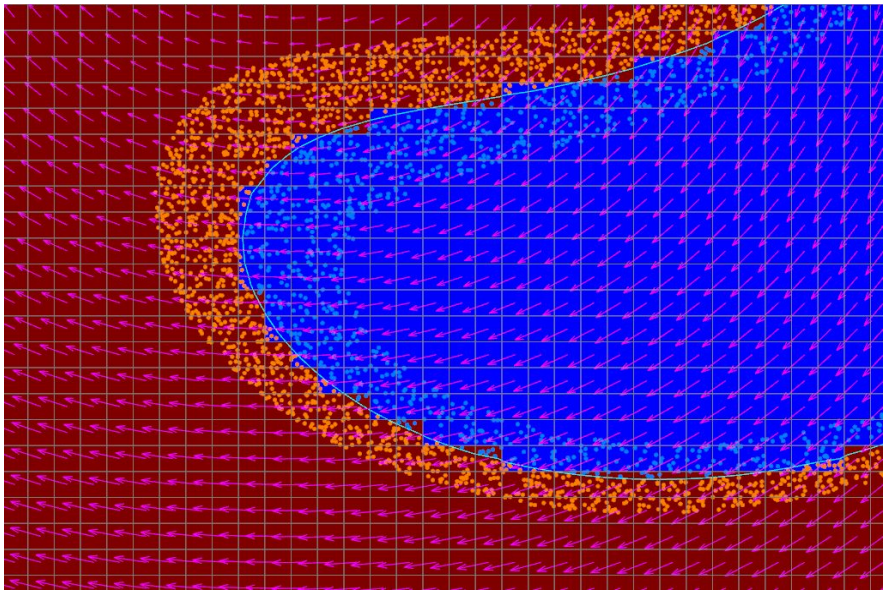
- We track the particles in space using a **fluid domain**
- In 2D we use a rectangular grid, in 3D we use a bounding volume



Fluid Domain

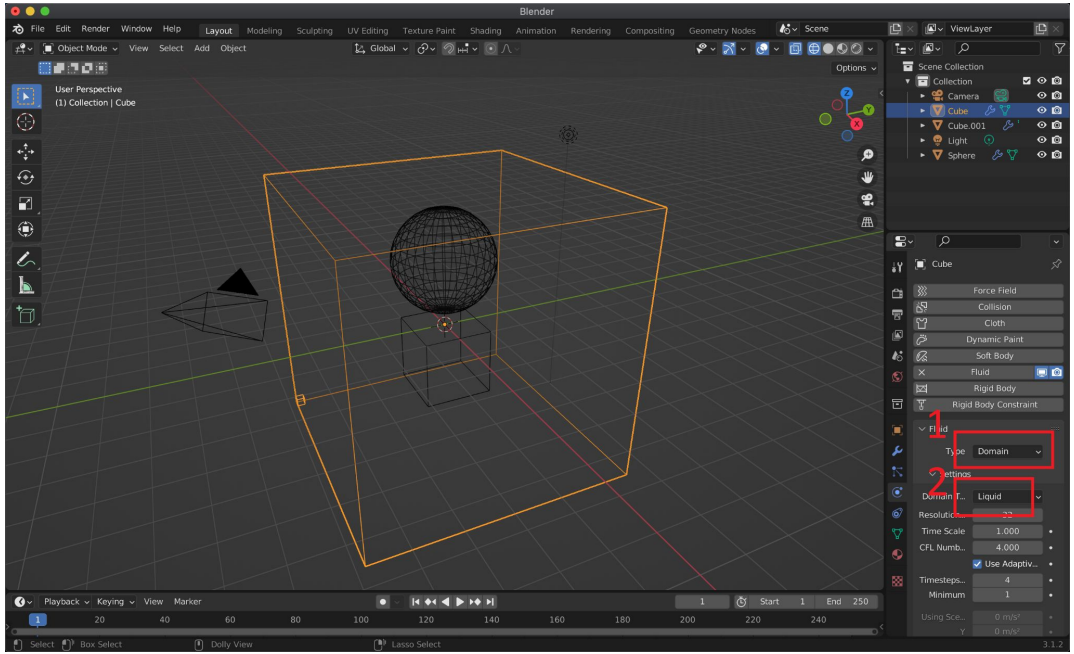
- We track the particles in space using a **fluid domain**
- In 2D we use a rectangular grid, in 3D we use a bounding volume

- 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



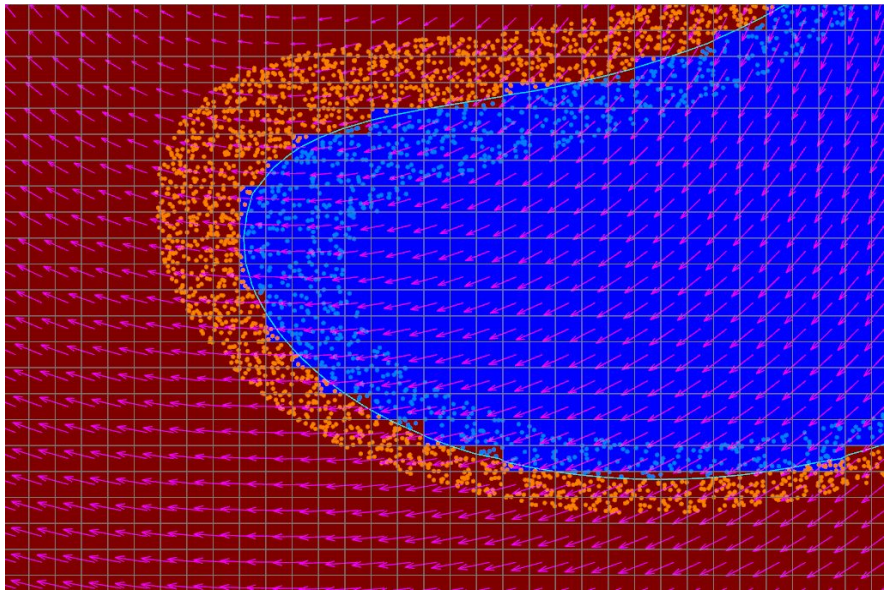
Fluid Domain

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



Eulerian Fluid Domain

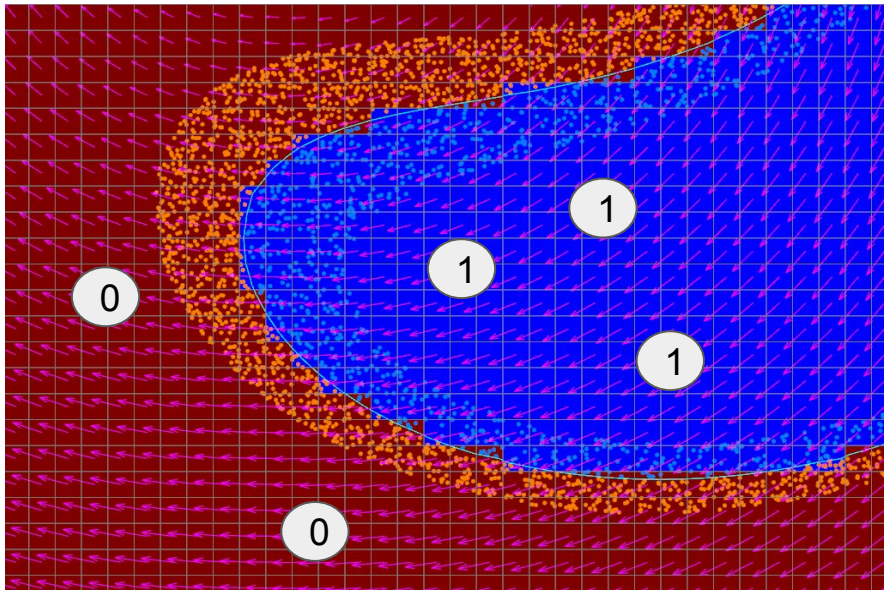
- We sometimes also represent the fluid with the fluid domain itself by marking grid cells as “on” or “off” if they’re in the fluid of interest



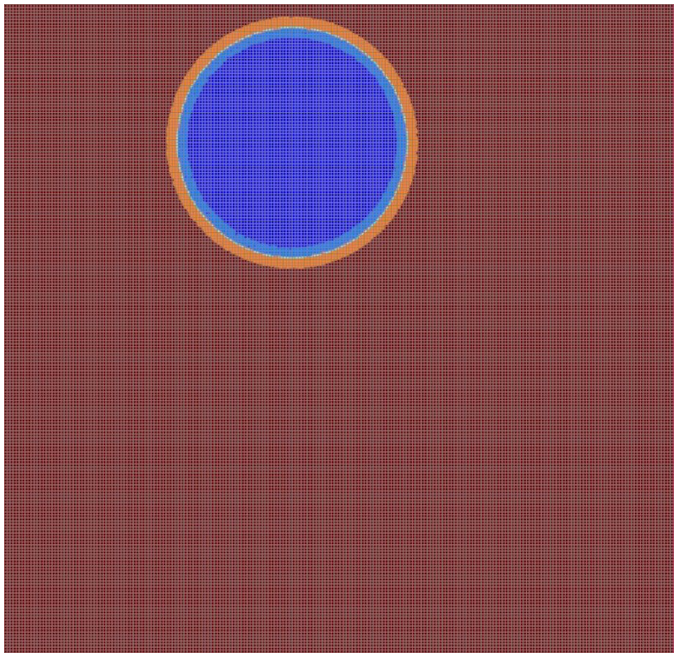
Eulerian Fluid Domain

- 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

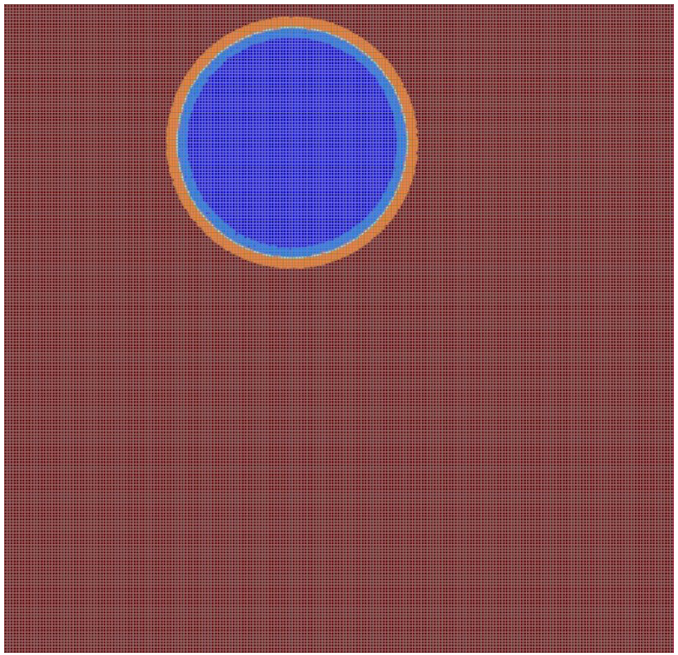


- How do we know if we're "inside" the fluid we care about?



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



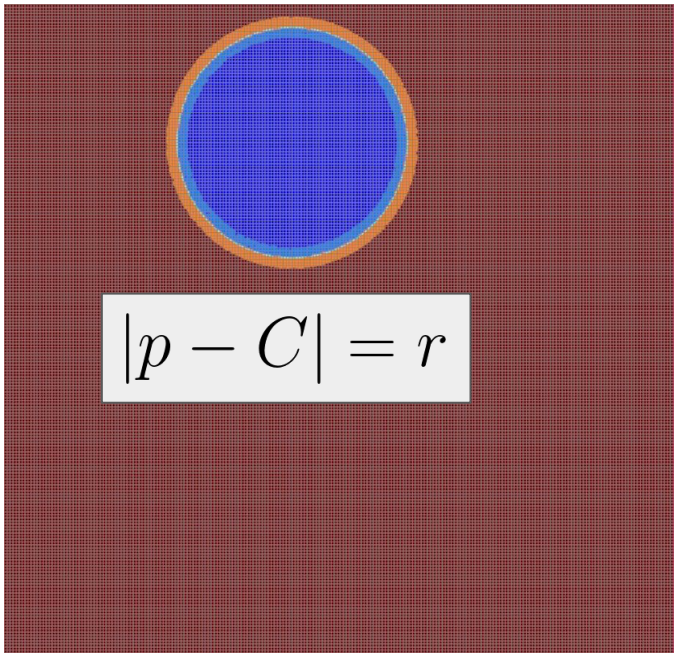
Implicit Surfaces

- 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**:

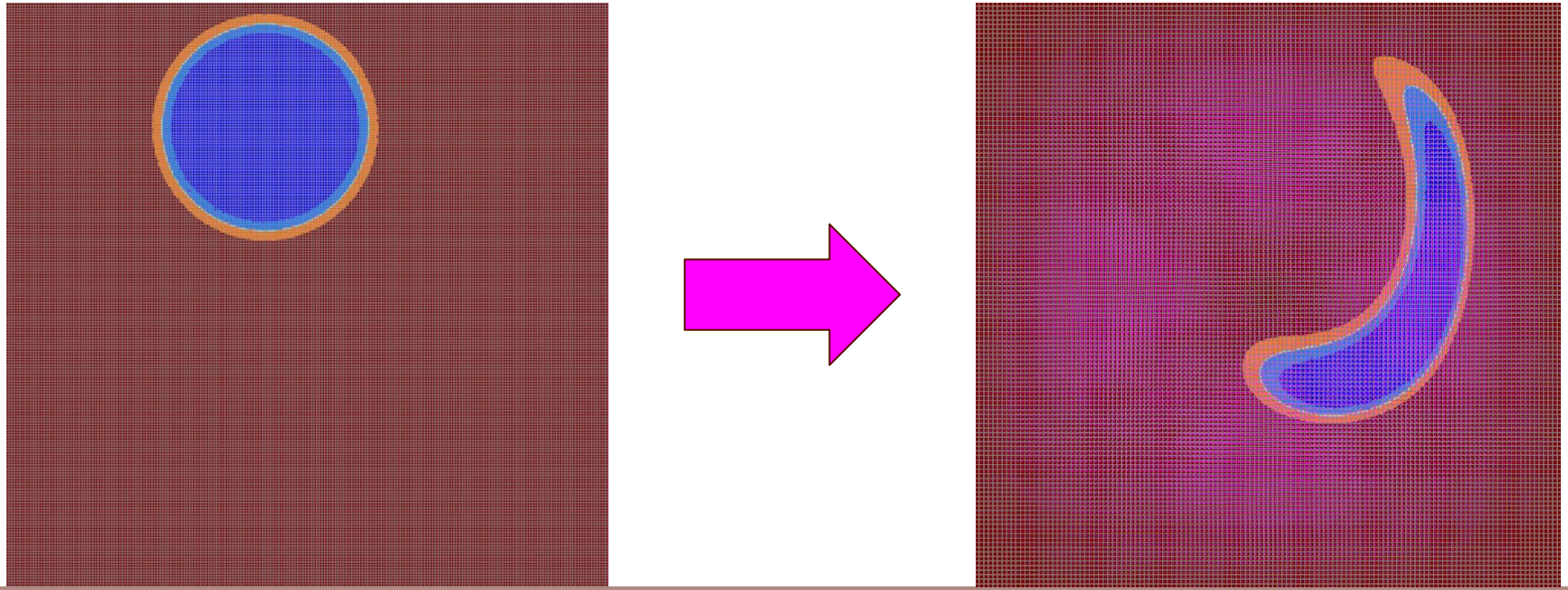
$$\phi(p) = \sqrt{(p - C) \cdot (p - C)} - r$$

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



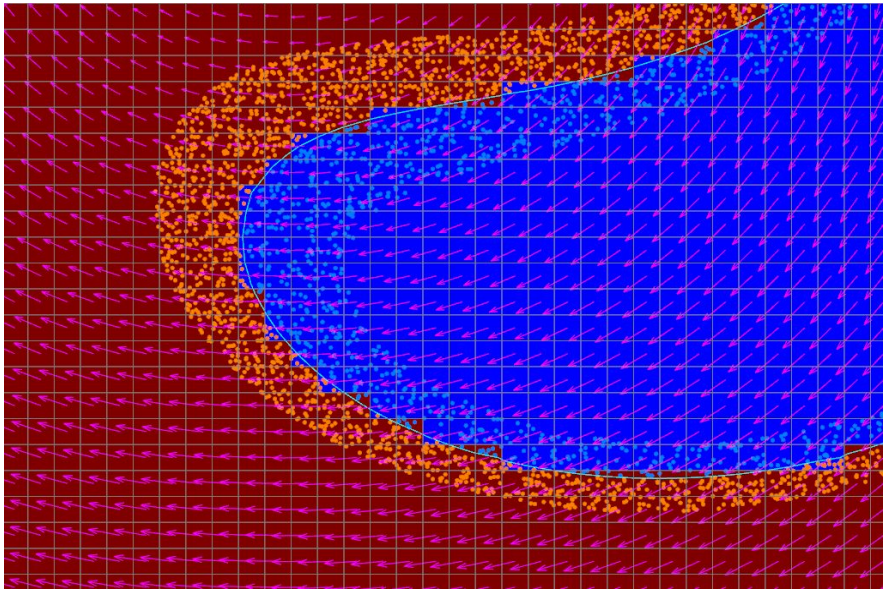
Implicit Surfaces

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



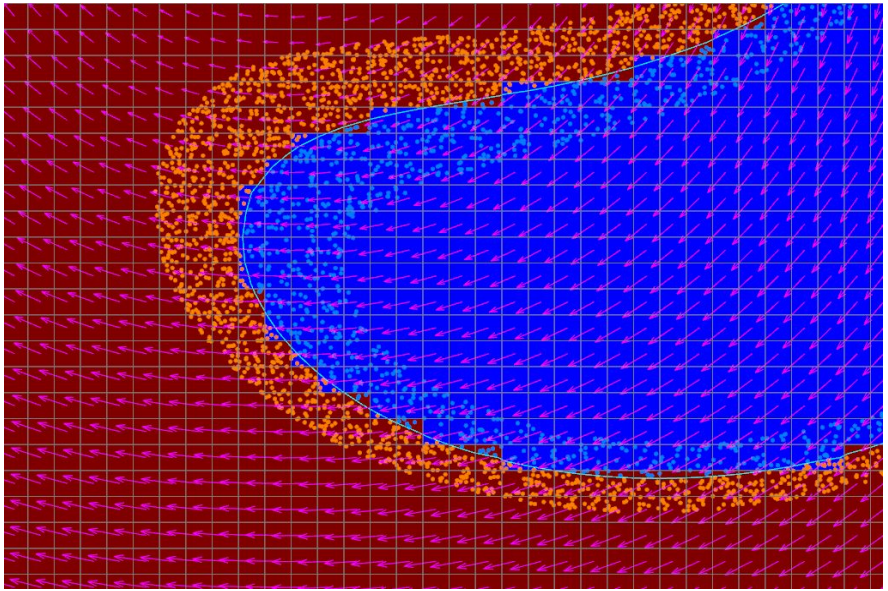
Mix of Lagrangian & Eulerian

- How could we compromise for efficiency and performance?



Mix of Lagrangian & Eulerian

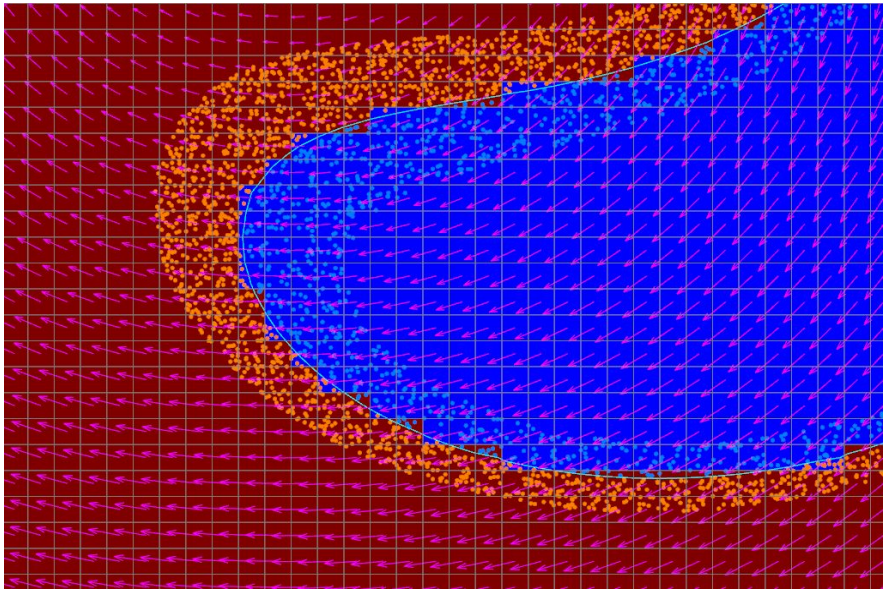
- Can use **Lagrangian marker particles** for the **fluid boundary**, then use an **Eulerian grid** to represent the rest of the fluid domain



Mix of Lagrangian & Eulerian

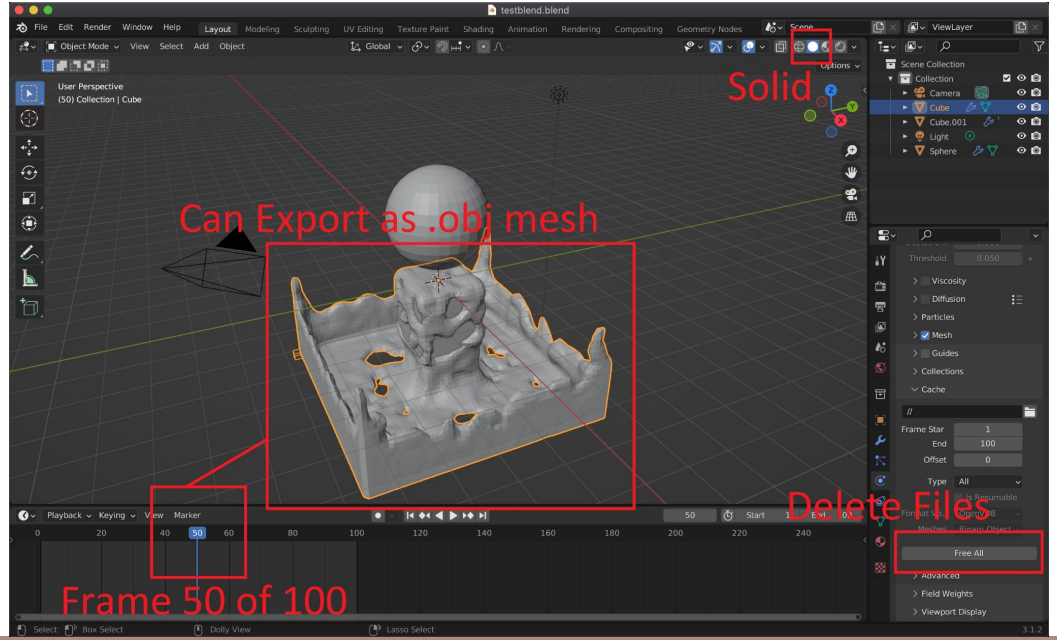
- Can use **Lagrangian marker particles** for the **fluid boundary**, then use an **Eulerian grid** to represent the rest of the fluid domain

- 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



Generating a Mesh

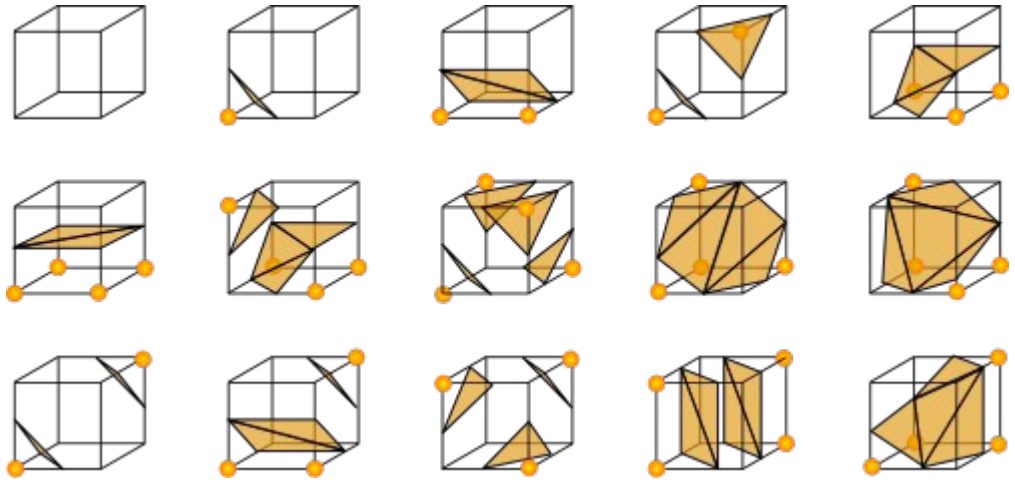
- After doing our fluid simulation, we want to turn it into a mesh (e.g. many triangles)



Generating a Mesh

- After doing our fluid simulation, we want to turn it into a mesh (e.g. many triangles)

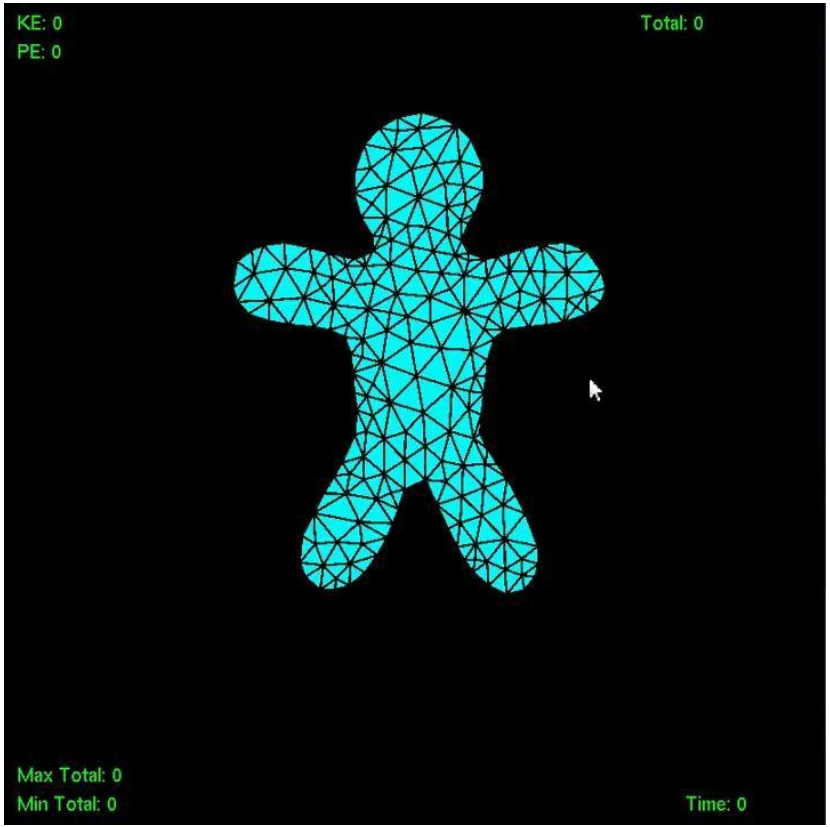
- One such algorithm is **marching cubes**
- Check each grid cell of the fluid domain
- 2^8 = 256 configurations of polygons based on which corners of the cell are “inside” e.g. water



Lecture Outline

- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

Elasticity



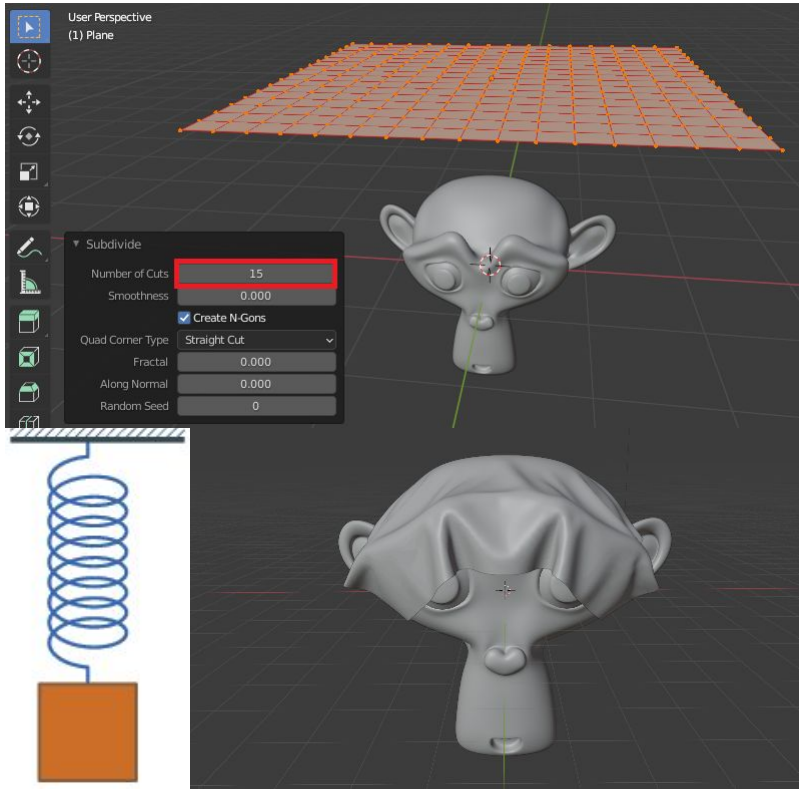
https://drive.google.com/file/d/1QW-xRso_LhAtCgOBVhWDr302dJ0QUzda/view?usp=drive_link

Elasticity for Cloth Simulation

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

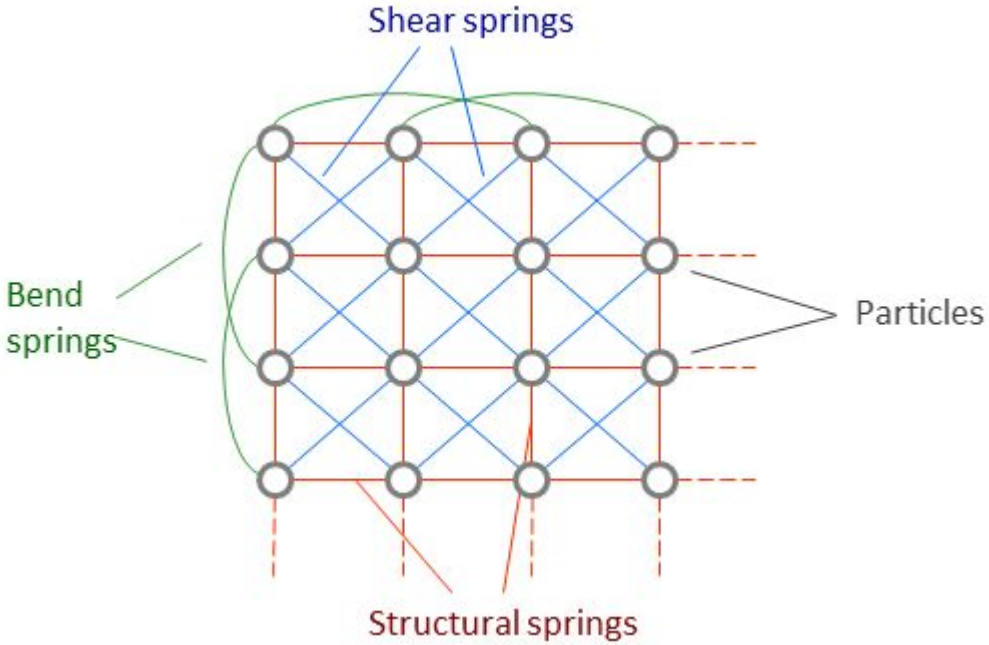
$$F_{spring} = -kq(t) - k_d v(t)$$

- 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



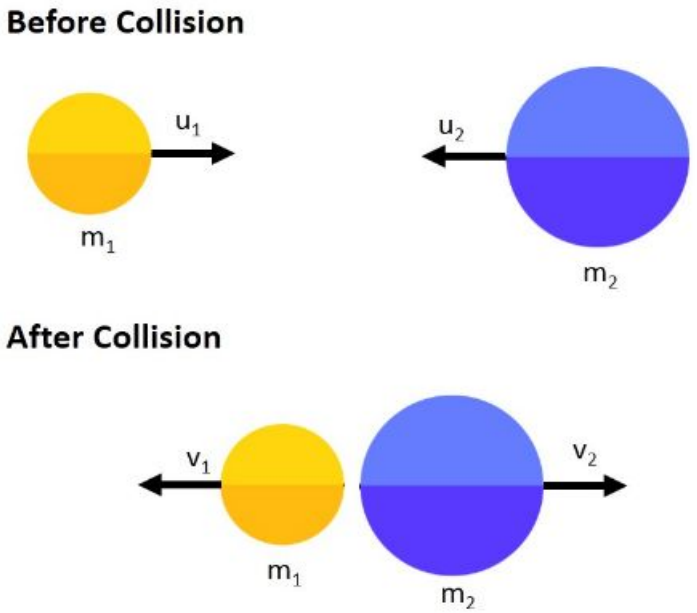
Lecture Outline

- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

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**

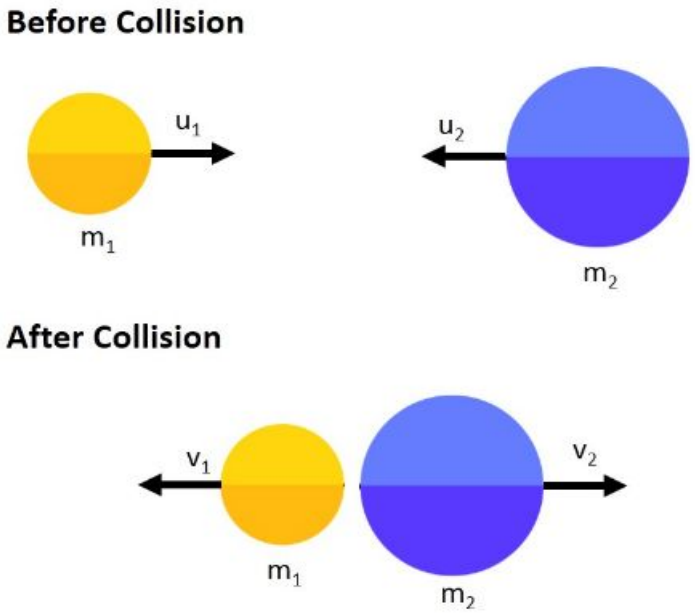


$$m_a u_a + m_b u_b = m_a v_a + m_b v_b$$

Simulating Collisions: 1D

- Define the **coefficient of restitution**

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



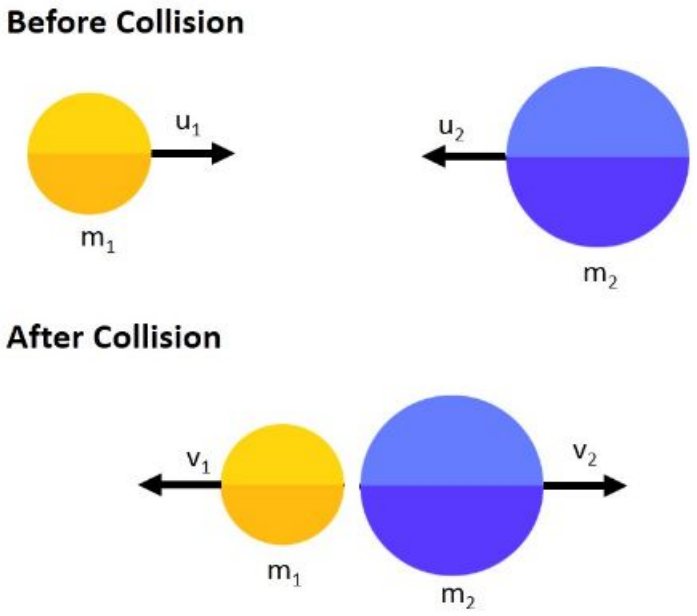
$$m_a u_a + m_b u_b = m_a v_a + m_b v_b$$

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**



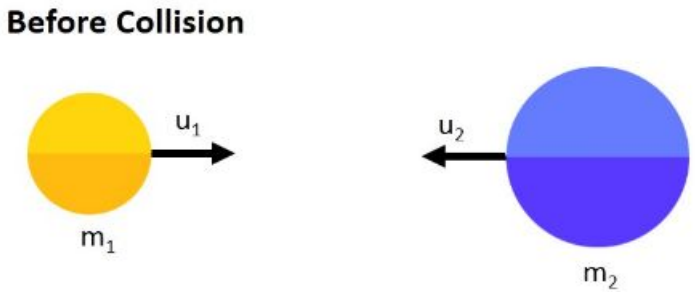
$$m_a u_a + m_b u_b = m_a v_a + m_b v_b$$

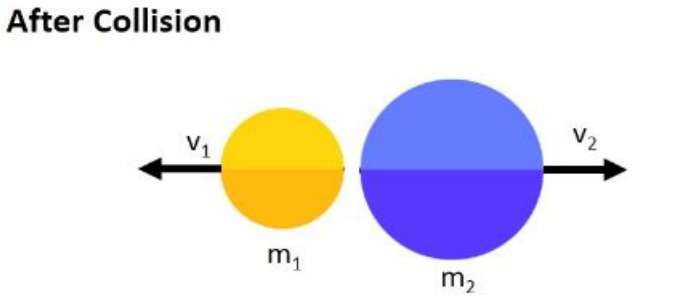
Simulating Collisions: 1D

- 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**

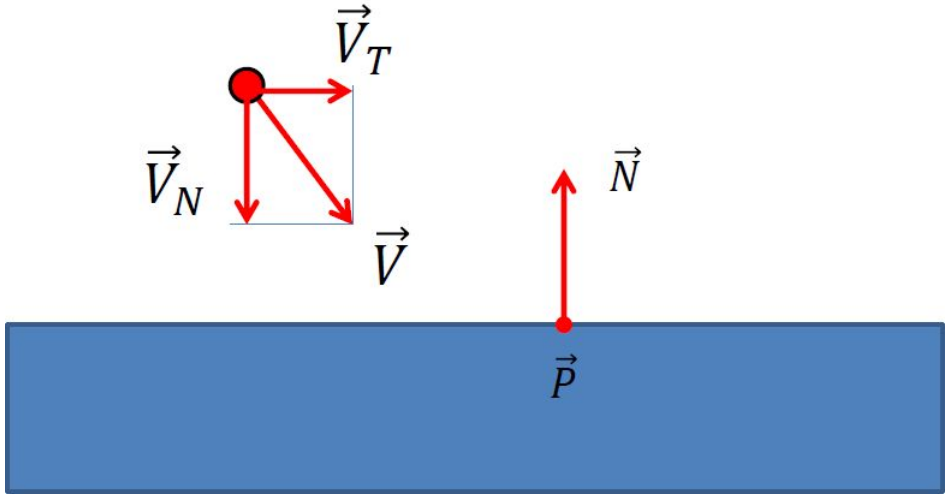
Simulating 3D Collisions: Point-Plane

- 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

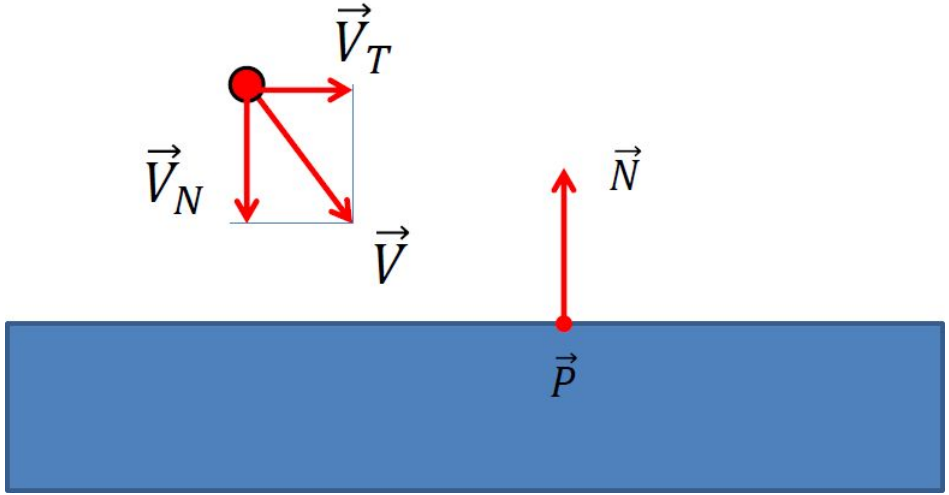
Simulating Collisions: Point-Plane

- Note: collisions with planes only involve the **normal component** of the velocity



Simulating 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

Lecture Outline

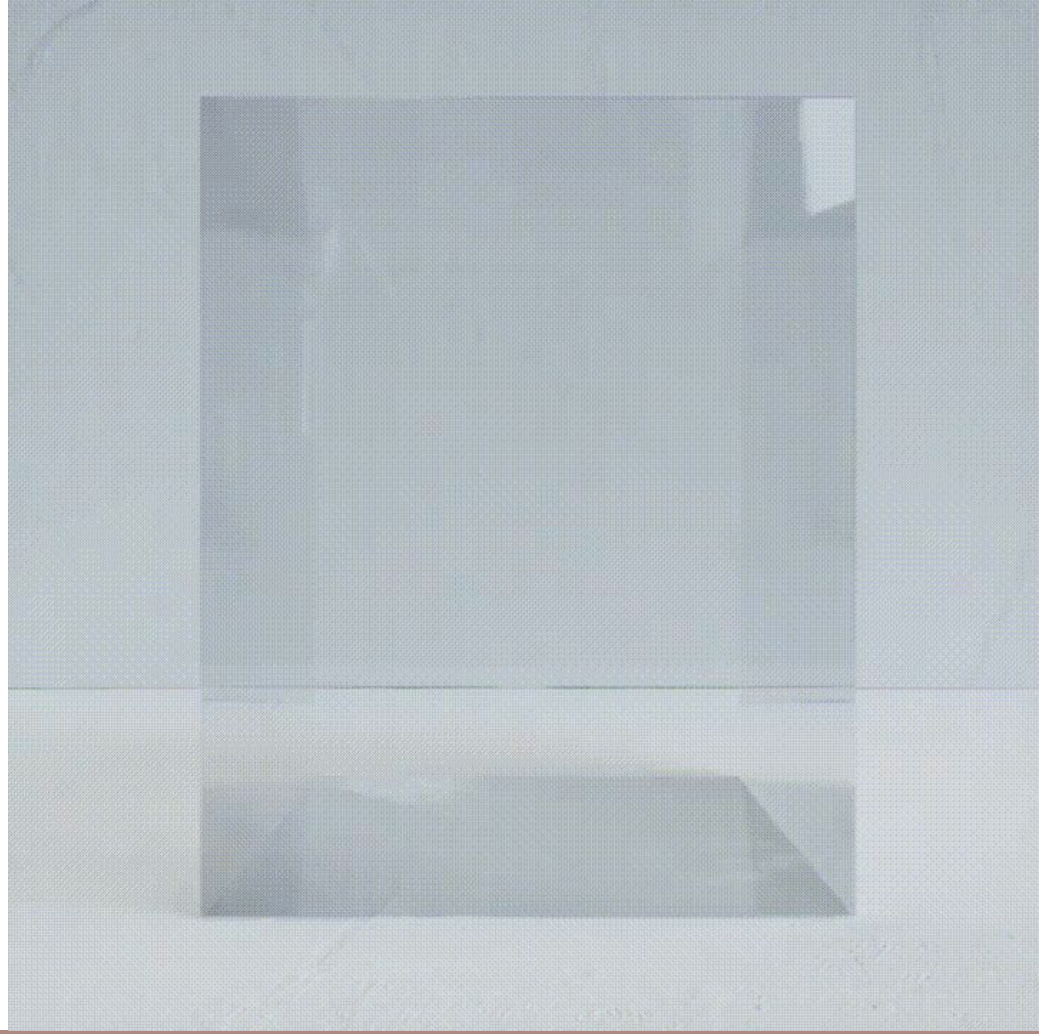
- Motivation
- Forces + time integrators
- Fluid simulation + domain, generating a mesh
- Simulating cloth
- Simulating collisions
- Artistry in simulation, flocking

Artistry in Simulation

- You can simulate anything… including things that defy physics!

Artistry in Simulation

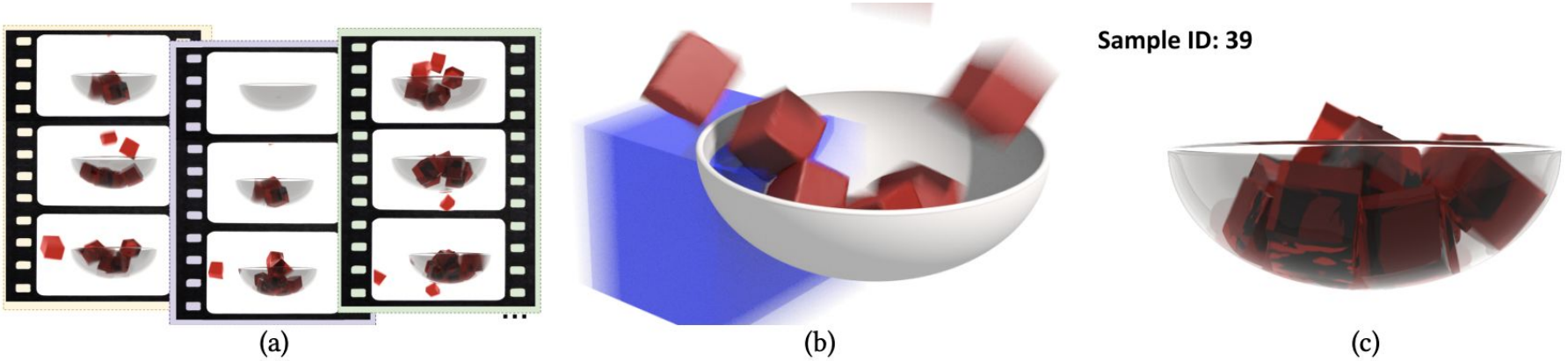
Esteban Diacono



Artistry in Simulation

Artistry in Simulation

- Or you can pick the desired outcome from many simulations



Purvi Goel, Unified Many-Worlds Browsing of Arbitrary Physics-based Animations, SIGGRAPH

Artistry in Simulation

- Or you can simulate old movie effects

The Shining, 1980

Artistry in Simulation

- Or you can simulate old movie effects

The Shining, 1980

Ready Player One, 2018

Artistry in Simulation

Spiderman: Into the Spiderverse, 2018

Flocking Simulation

What do birds do?

Flocking Simulation

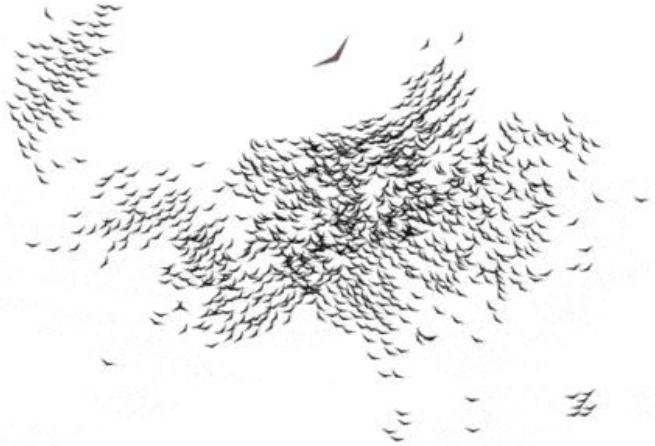
What do birds do?

- Birds like to follow each other
- Birds don't want to be too close to each other
- When flying, birds look for spaces with low air resistance
- Every little bit, birds need a rest
- If birds see food, they will probably go eat it

Flocking Simulation

What do birds do?

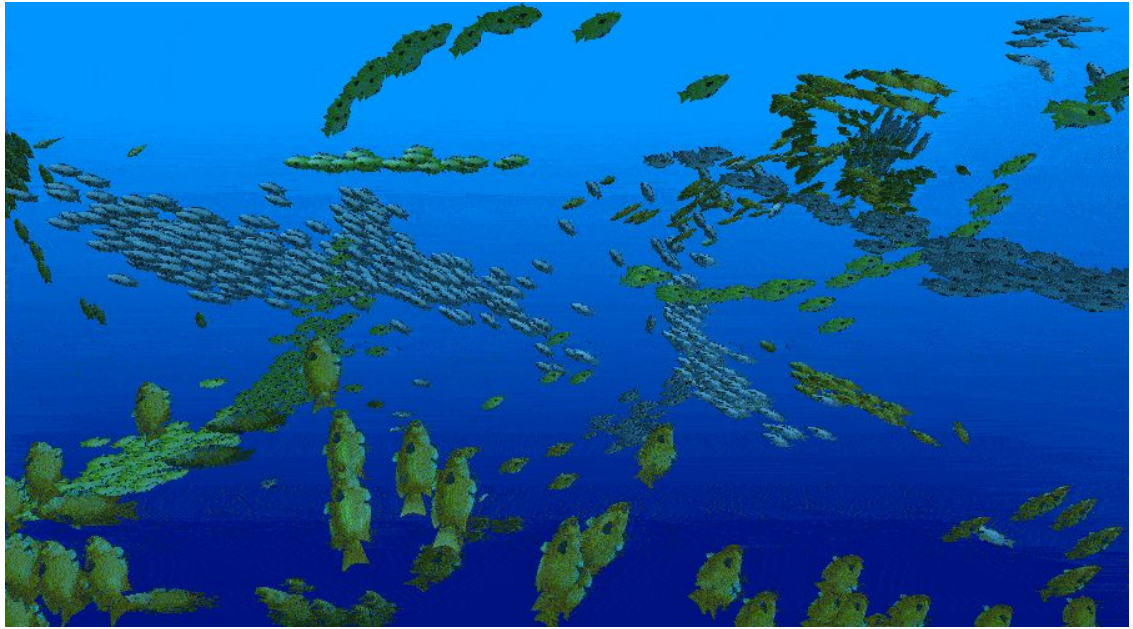
- **Birds like to follow each other**
- **Birds don’t want to be too close to each other**
- **When flying, birds look for spaces with low air resistance**
- Every little bit, birds need a rest
- If birds see food, they will probably go eat it



Flocking Simulation

What do fish do?

- Fish like to follow each other
- Fish don’t want to be too close to each other
- Fish like to swim with fish of the same species



Flocking Simulation

What do zombies do?

Flocking Simulation

What do zombies do?

- Zombies can only walk on a surface, or climb on a surface with over 30% elevation
- Zombies aren't very good at climbing
- When zombies fall, they take a minute, then get back up and keep going
- Zombies want to get to the other side of the wall

Flocking Simulation

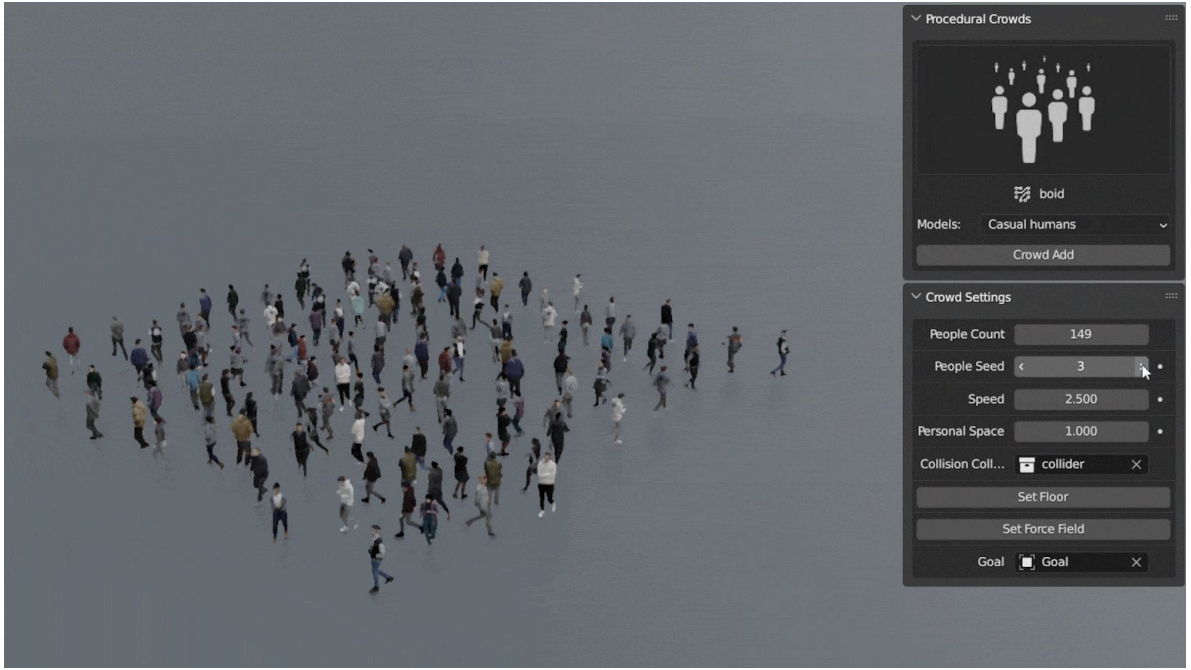
What do zombies do?

- Zombies can only walk on a surface, or climb on a surface with over 30% elevation
- Zombies aren't very good at climbing
- When zombies fall, they take a minute, then get back up and keep going
- Zombies want to get to the other side of the wall

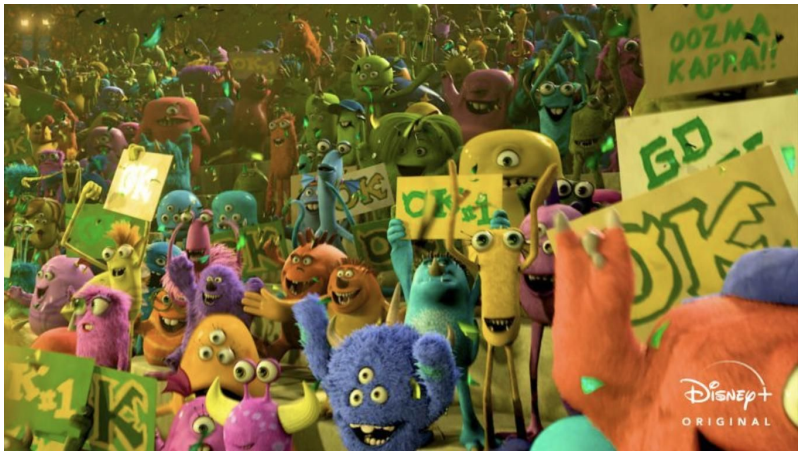
Flocking Simulation

What do people do?

- People can only walk on surfaces
- Often people have a destination in mind
- People don’t want to be too close to strangers



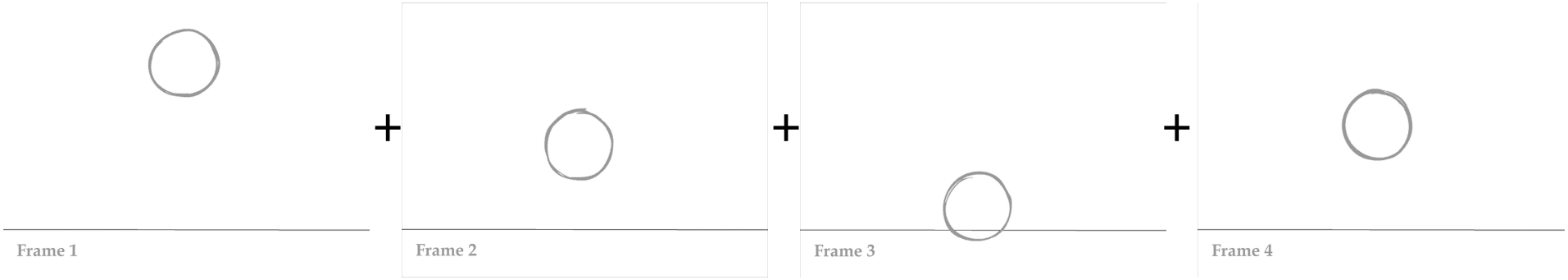
Flocking Simulation



Additional Resources + Sources

- Collisions – Blender 5.2
- Fluid Simulation Research Papers
- Sag-Fee Hair Simulation (University of Utah!)
- Python Numerical Methods (Implicit/Explicit Combinations)
- Check out extra slides at the end for simulation math examples!

Simulation



Envato Tuts Animation tutorial

=

Simulation

F = -9.81
V = 0
P = .04

A = -9.81
V_next = -.39
P_next = .024

Frame 1

+

F = -9.81
V = -.39
P = .024

A = -9.81
V_next = -.78
P_next = -.007

Frame 2

+

F = -9.81 **+ momentum**
V = -.78
P = 0

A = **N/A**
V_next = **+**.78
P_next = .031

Frame 3

+

F = -9.81
V = .78
P = .031

A = -9.81
V_next = .39
P_next = .04

Frame 4

Envato Tuts Animation tutorial

=

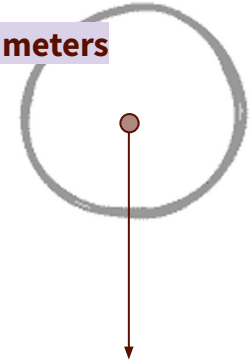
Frame 1

Simulation Math

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (0, -9.81 * mass)N

time step = .04 seconds

acceleration = force / mass

change in Velocity = acceleration * time step

New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

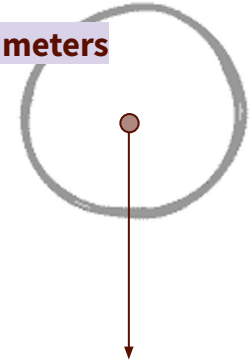
Frame 1

Simulation Math

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (0, -9.81 * mass)N

time step = .04 seconds

acceleration = force / mass
acceleration = (0, -9.81 * 1.0g) N / 1.0g
acceleration = (0, -9.81) m/s^2

change in Velocity = acceleration * time step

New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

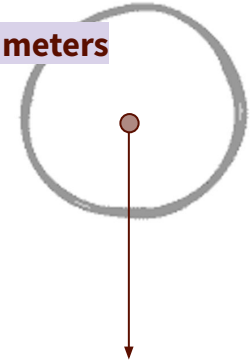
Frame 1

Simulation Math

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (0, -9.81 * mass)N

time step = .04 seconds

acceleration = force / mass
acceleration = (0, -9.81 * 1.0g) N / 1.0g
acceleration = (0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (0, -9.81)m/s^2 * .04s
change in Velocity = (0, -.39)m/s

New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

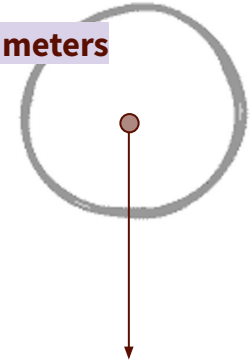
Frame 1

Simulation Math

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (0, -9.81 * mass)N

time step = .04 seconds

acceleration = force / mass
acceleration = (0, -9.81 * 1.0g) N / 1.0g
acceleration = (0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (0, -9.81)m/s^2 * .04s
change in Velocity = (0, -.39)m/s

New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (0, -.39)m/s
New velocity = (0, -.39)m/s

change in position = new velocity * time step

New position = Old Position + change in position

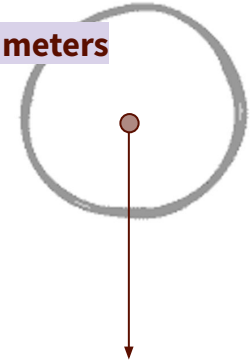
Frame 1

Simulation Math

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (0, -9.81 * mass)N

time step = .04 seconds

acceleration = force / mass
acceleration = (0, -9.81 * 1.0g) N / 1.0g
acceleration = (0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (0, -9.81)m/s^2 * .04s
change in Velocity = (0, -.39)m/s

New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (0, -.39)m/s
New velocity = (0, -.39)m/s

change in position = new velocity * time step
change in position = (0, -.39)m/s * .04s
change in position = (0, -.016)m

New position = Old Position + change in position

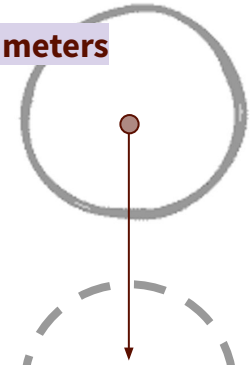
Frame 1

Simulation Math

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (0, -9.81 * mass)N

time step = .04 seconds

acceleration = force / mass
acceleration = (0, -9.81 * 1.0g) N / 1.0g
acceleration = (0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (0, -9.81)m/s^2 * .04s
change in Velocity = (0, -.39)m/s

New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (0, -.39)m/s
New velocity = (0, -.39)m/s

change in position = new velocity * time step
change in position = (0, -.39)m/s * .04s
change in position = (0, -.016)m

New position = Old Position + change in position
New position = (0, .04) m + (0, -.016)m
New position = (0, .024)m

Frame 1

Simulation Math

F = -9.81
V = 0
P = .04

A = -9.81
V_next = -.39
P_next = .024

Frame 1

+

F = -9.81
V = -.39
P = .024

A = -9.81
V_next = -.78
P_next = -.007

Frame 2

+

F = -9.81 + momentum
V = -.78
P = 0

A = N/A
V_next = +.78
P_next = .031

Frame 3

+

F = -9.81
V = .78
P = .031

A = -9.81
V_next = .39
P_next = .04

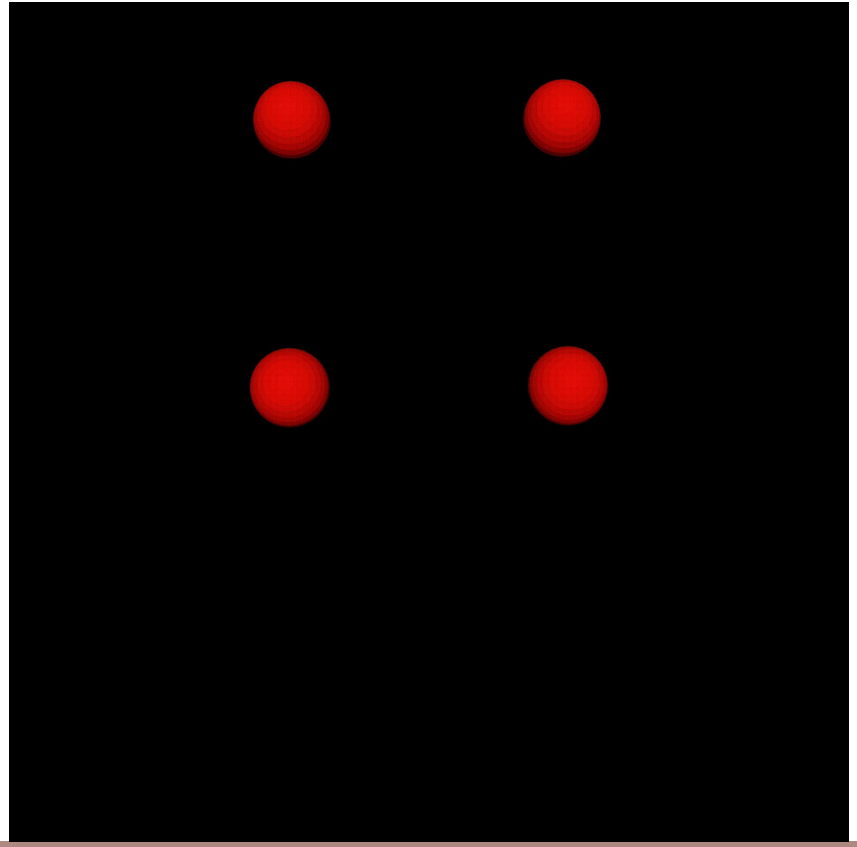
Frame 4

Envato Tuts Animation tutorial

=

Frame 1

Gravity Simulation

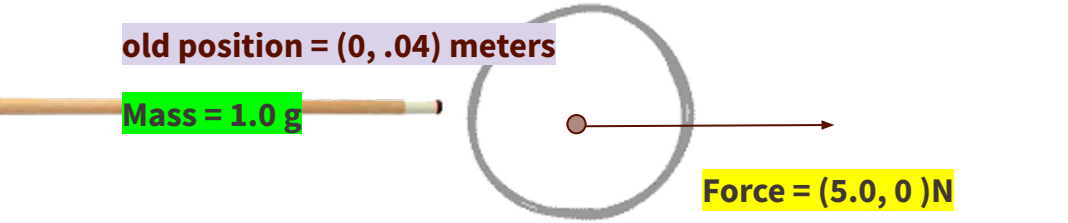


Simulation Math Part 2

old velocity = (0, 0) meters/second

old position = (0, .04) meters

Mass = 1.0 g



Force = (5.0, 0)N

time step = .04 seconds

acceleration = force / mass

change in Velocity = acceleration * time step

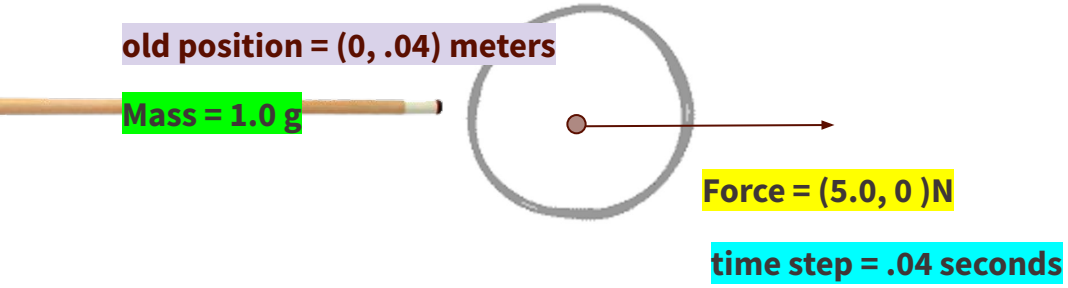
New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Part 2



acceleration = force / mass
acceleration = (5.0, 0) N / 1.0g
acceleration = (5.0, 0) m/s^2

change in Velocity = acceleration * time step

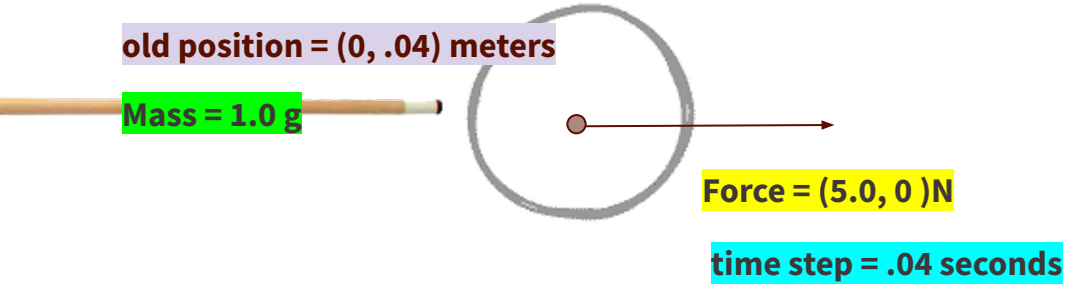
New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Part 2



acceleration = force / mass
acceleration = (5.0, 0) N / 1.0g
acceleration = (5.0, 0) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, 0)m/s^2 * .04s
change in Velocity = (.2, 0)m/s

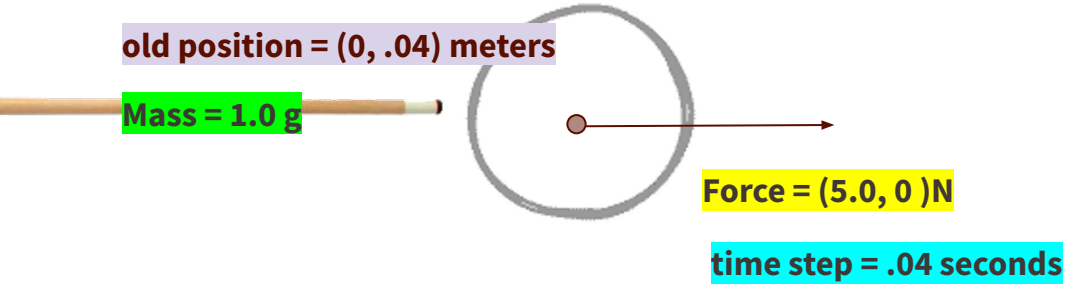
New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Part 2



acceleration = force / mass
acceleration = (5.0, 0) N / 1.0g
acceleration = (5.0, 0) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, 0)m/s^2 * .04s
change in Velocity = (.2, 0)m/s

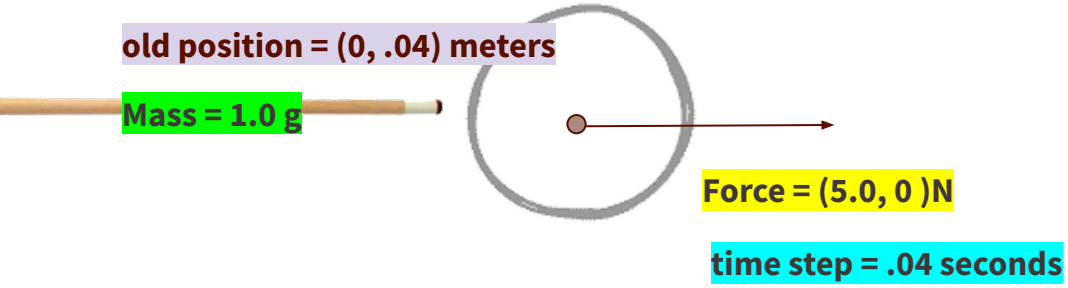
New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (.2, 0)m/s
New velocity = (.2, 0)m/s

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Part 2



acceleration = force / mass
acceleration = (5.0, 0) N / 1.0g
acceleration = (5.0, 0) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, 0)m/s^2 * .04s
change in Velocity = (.2, 0)m/s

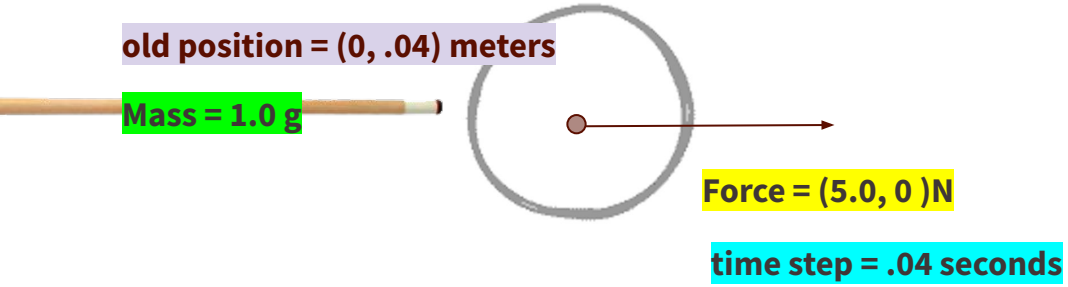
New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (.2, 0)m/s
New velocity = (.2, 0)m/s

change in position = new velocity * time step
change in position = (.2, 0)m/s * .04s
change in position = (.008, 0)m

New position = Old Position + change in position

Frame 1

Simulation Math Part 2



acceleration = force / mass
acceleration = (5.0, 0) N / 1.0g
acceleration = (5.0, 0) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, 0)m/s^2 * .04s
change in Velocity = (.2, 0)m/s

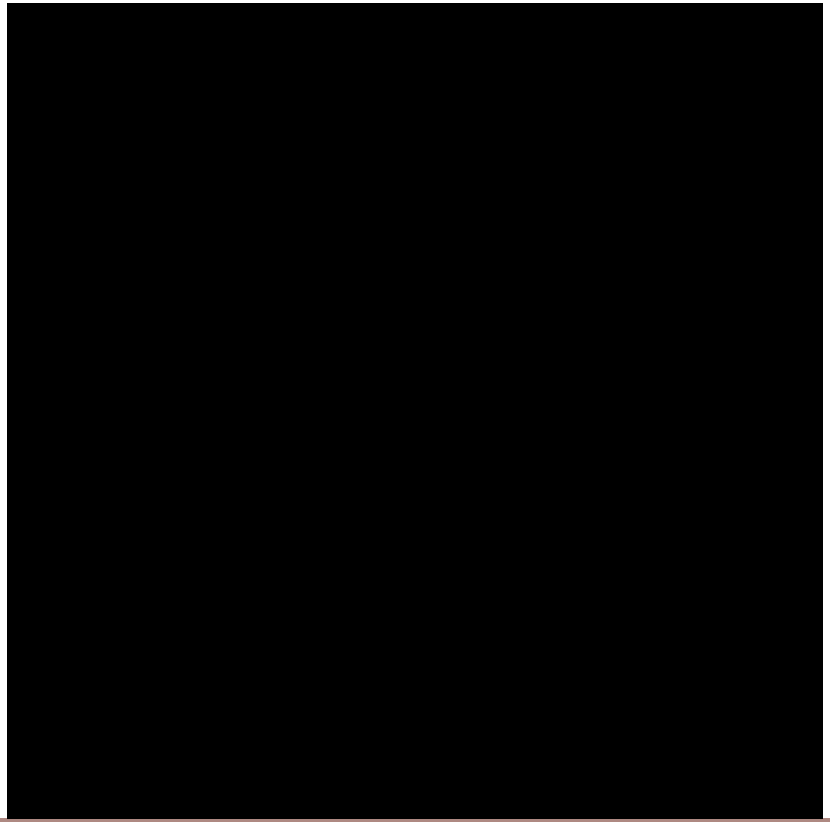
New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (.2, 0)m/s
New velocity = (.2, 0)m/s

change in position = new velocity * time step
change in position = (.2, 0)m/s * .04s
change in position = (.008, 0)m

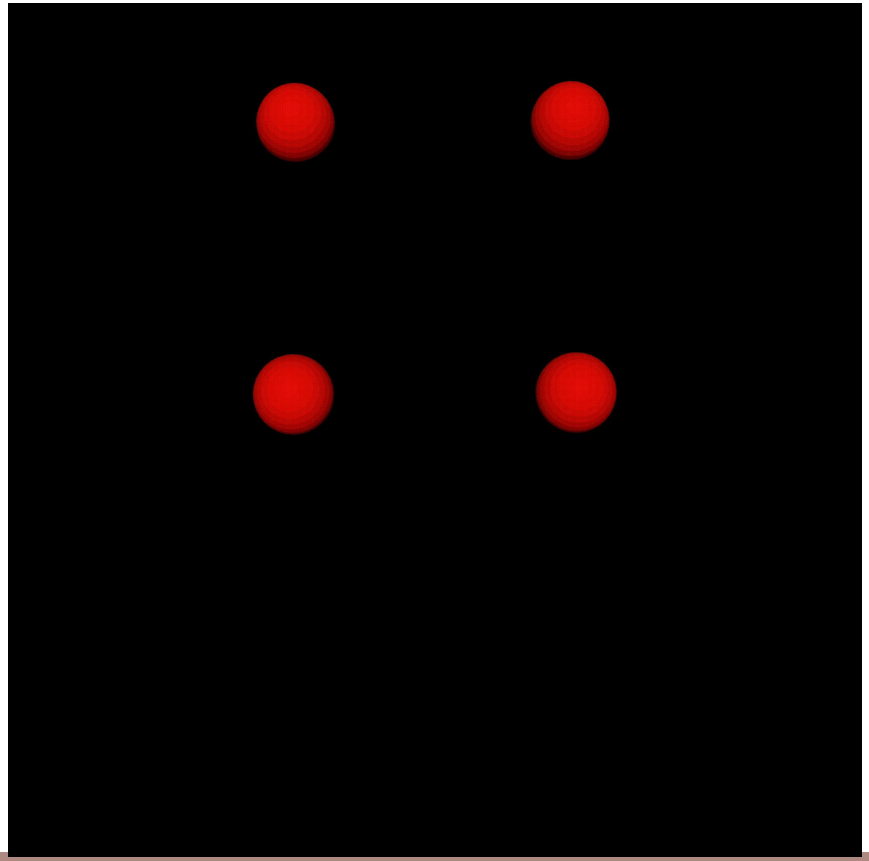
New position = Old Position + change in position
New position = (0, .04) m + (.008, 0)m
New position = (.008, .04)m

Frame 1

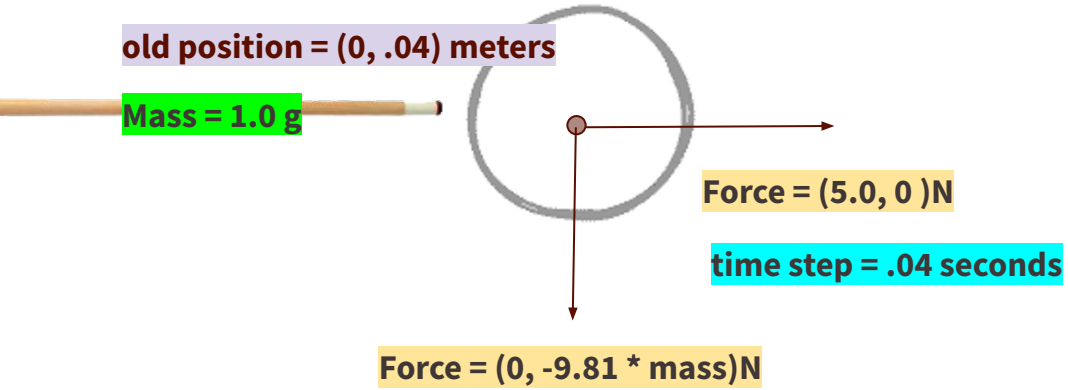
Attraction Simulation



Attraction Simulation



Simulation Math Multiple Forces



acceleration = force / mass

change in Velocity = acceleration * time step

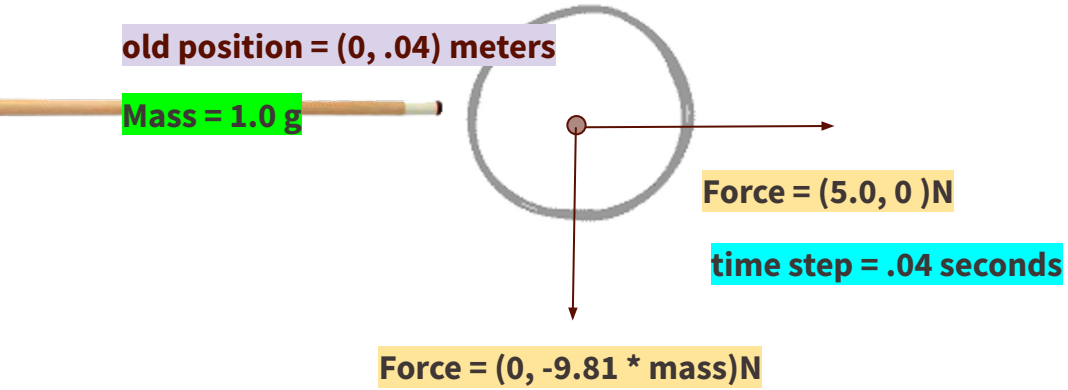
New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Multiple Forces



acceleration = force / mass
acceleration = (5.0, -9.81* 1.0g) N / 1.0g
acceleration = (5.0, -9.81) m/s^2

change in Velocity = acceleration * time step

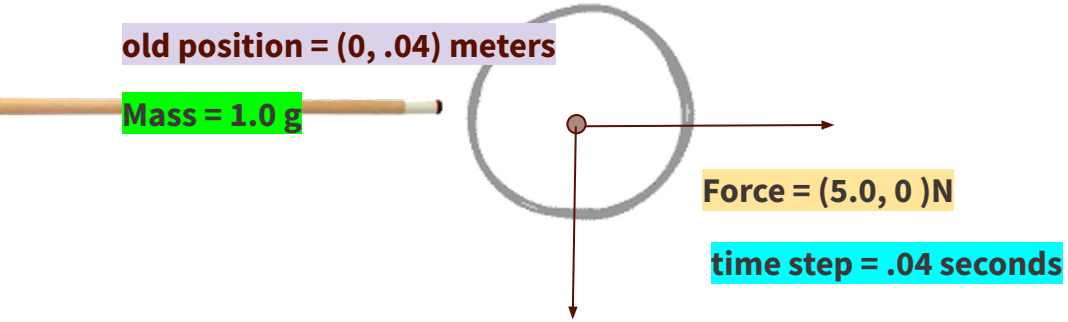
New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Multiple Forces



acceleration = force / mass
acceleration = (5.0, -9.81* 1.0g) N / 1.0g
acceleration = (5.0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, -9.81)m/s^2 * .04s
change in Velocity = (.2, -.39)m/s

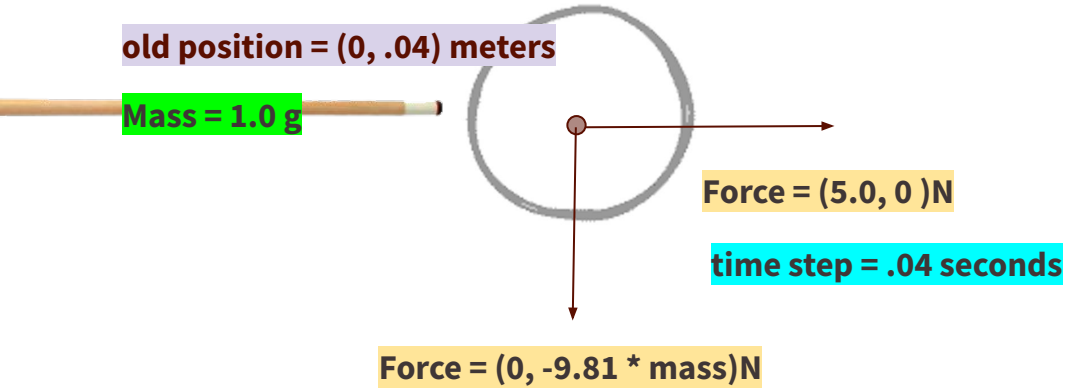
New velocity = old velocity + change in velocity

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Multiple Forces



acceleration = force / mass
acceleration = (5.0, -9.81* 1.0g) N / 1.0g
acceleration = (5.0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, -9.81)m/s^2 * .04s
change in Velocity = (.2, -.39)m/s

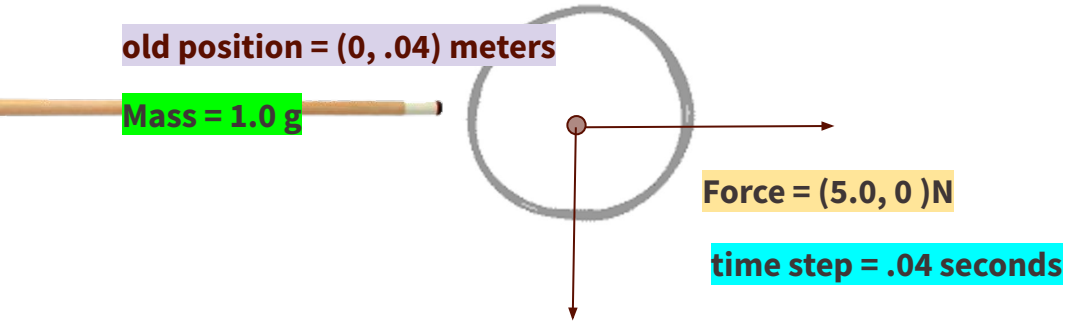
New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (.2, -.39)m/s
New velocity = (.2, -.39)m/s

change in position = new velocity * time step

New position = Old Position + change in position

Frame 1

Simulation Math Multiple Forces



acceleration = force / mass
acceleration = (5.0, -9.81* 1.0g) N / 1.0g
acceleration = (5.0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, -9.81)m/s^2 * .04s
change in Velocity = (.2, -.39)m/s

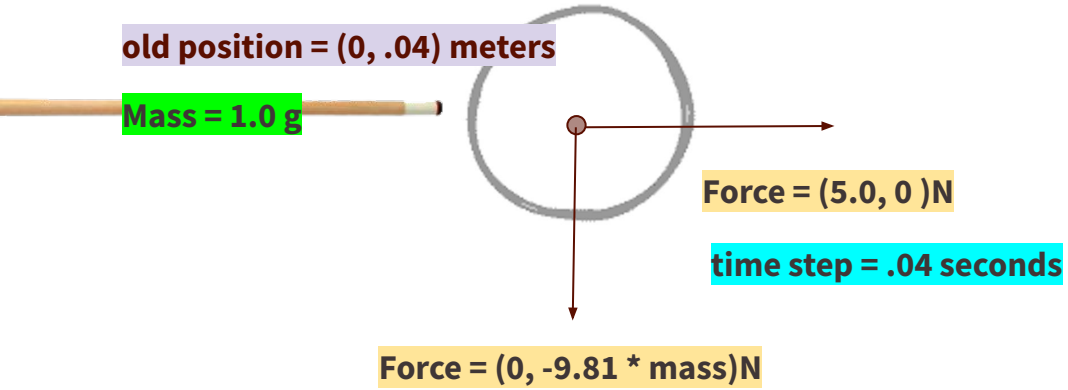
New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (.2, -.39)m/s
New velocity = (.2, -.39)m/s

change in position = new velocity * time step
change in position = (.2, 0)m/s * .04s
change in position = (.008, -.016)m

New position = Old Position + change in position

Frame 1

Simulation Math Multiple Forces



acceleration = force / mass
acceleration = (5.0, -9.81* 1.0g) N / 1.0g
acceleration = (5.0, -9.81) m/s^2

change in Velocity = acceleration * time step
change in Velocity = (5.0, -9.81)m/s^2 * .04s
change in Velocity = (.2, -.39)m/s

New velocity = old velocity + change in velocity
New velocity = (0,0)m/s + (.2, -.39)m/s
New velocity = (.2, -.39)m/s

change in position = new velocity * time step
change in position = (.2, 0)m/s * .04s
change in position = (.008, -.016)m

New position = Old Position + change in position
New position = (0, .04) m + (.008, -.016)m
New position = (.008, .024)mo

Frame 1

Multiple Forces Simulation