Passive Dynamics and Particle Systems

COS 426, Fall 2022

PRINCETON UNIVERSITY

Animation & Simulation

- Animation
  - Make objects change over time according to scripted actions

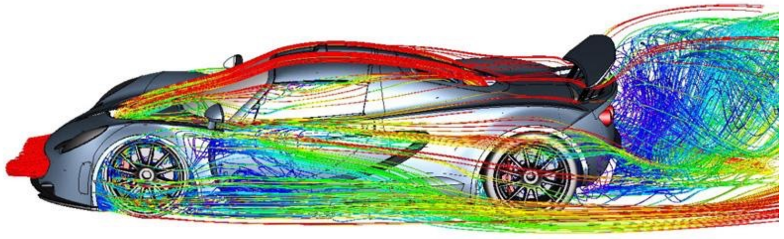
Pixar

Animation & Simulation

- Animation
  - Make objects change over time according to scripted actions

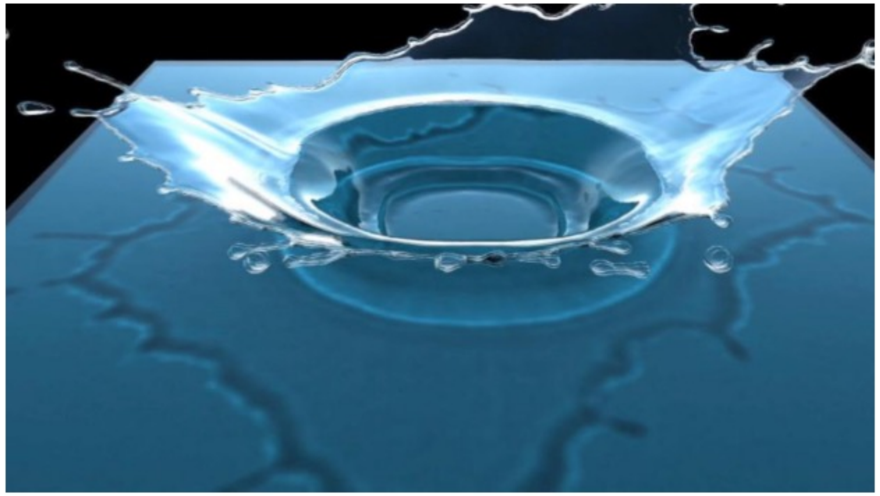
Pixar

- Simulation / dynamics
  - Predict how objects change over time according to physical laws



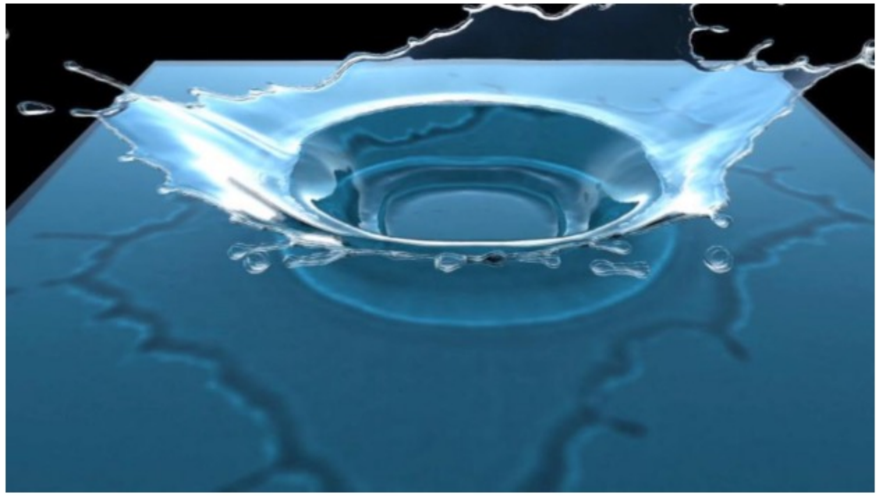
Animation & Simulation

- Keyframing:
  - Manually specify a few poses; computer interpolates.
  - Good for characters and simple motion.
  - But many physical systems are too complex!



Simulation

1. Identify/derive mathematical model (ODE, PDE)
2. Develop computer model
3. Simulate



Simulation

- Equations known for a long time
  - Motion (Newton, 1660)
  - Elasticity (Hooke, 1670)
  - Fluids (Navier, Stokes, 1822)

$$d/dt(m\mathbf{v}) = \mathbf{f}$$

$$\boldsymbol{\sigma} = \mathbf{E}\boldsymbol{\varepsilon}$$

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -k\nabla\rho + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}$$

1938: Zuse Z1

0.2 ops

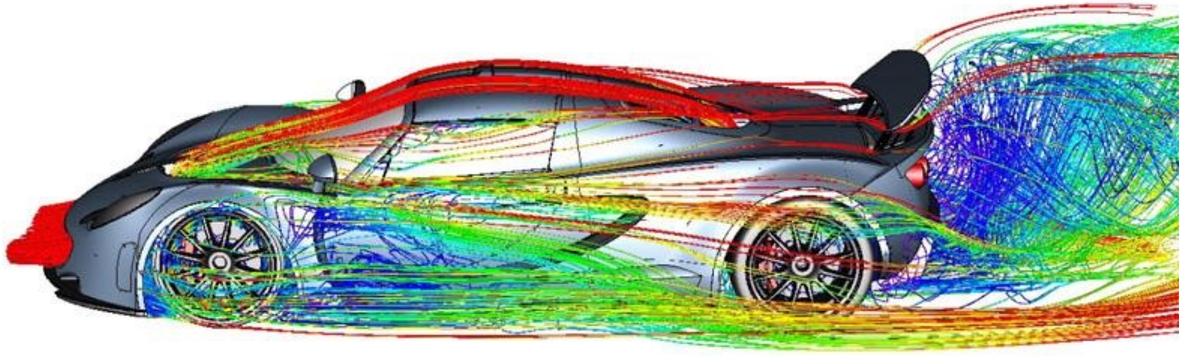
10^{18}

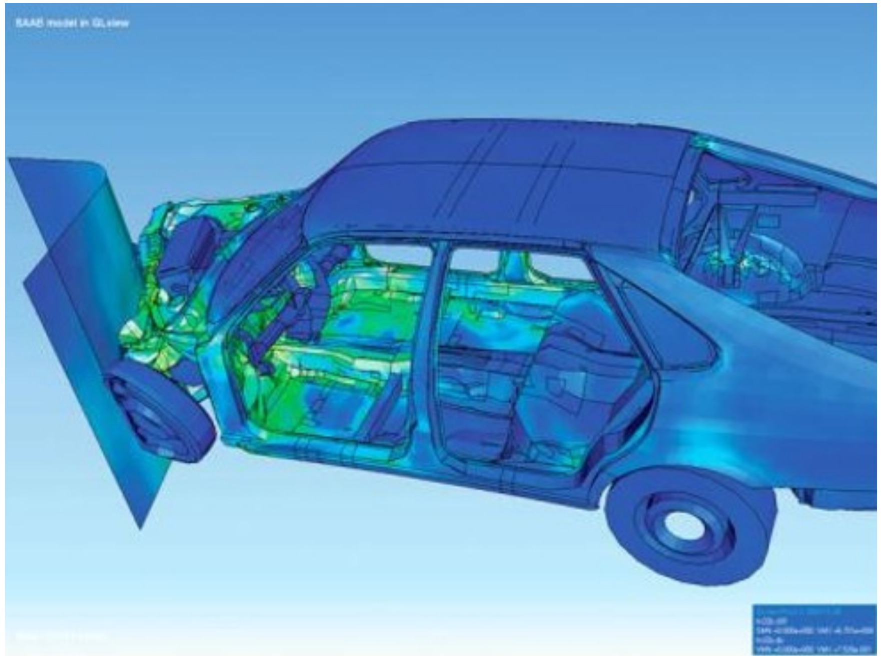
2014: Tianhe-2 @ NUDT (China)

54,902 teraflops (3.12M cores)

Physically-based Simulation

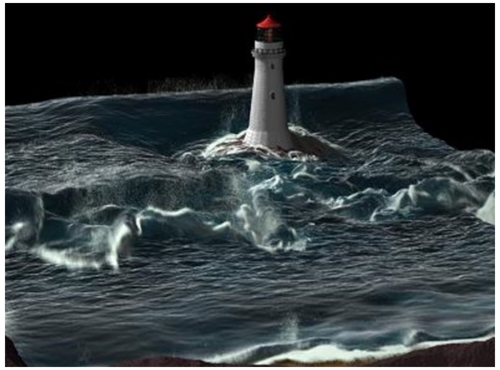
- Computational Sciences
  - Goal: reproduction of physical phenomena
  - Predictive capability
  - Substitute for expensive experiments





Physically-based Simulation

- Computational Sciences
  - Goal: reproduction of physical phenomena
  - Predictive capability
  - Substitute for expensive experiments
- Computer Graphics
  - Goal: imitation of physical phenomena
  - Visually plausible behavior
  - Speed, stability, art-directability



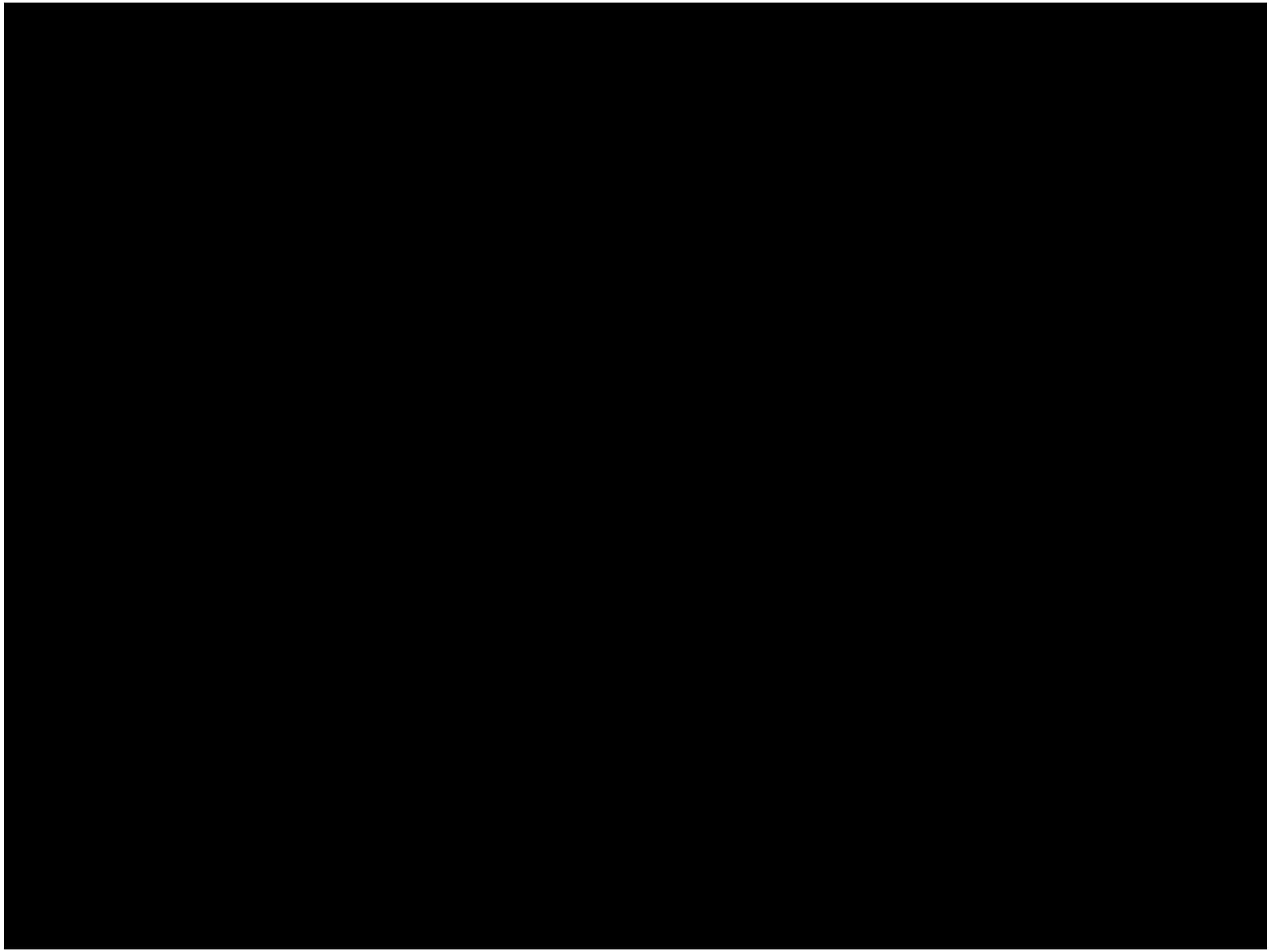
Simulation: Speed

https://www.youtube.com/watch?v=8jD1bz4N3_0

Simulation: Stability

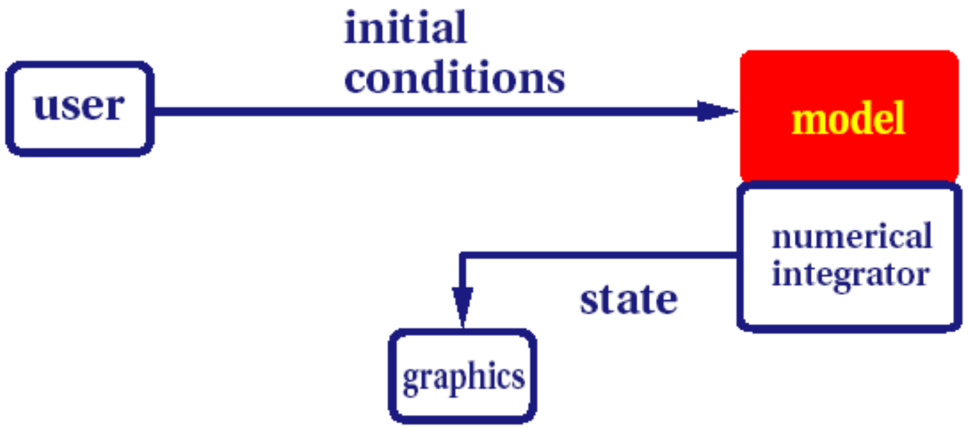
https://www.youtube.com/watch?v=tT81VPk_ukU

Simulation: Art-directability



Dynamics

Passive--no muscles or motors

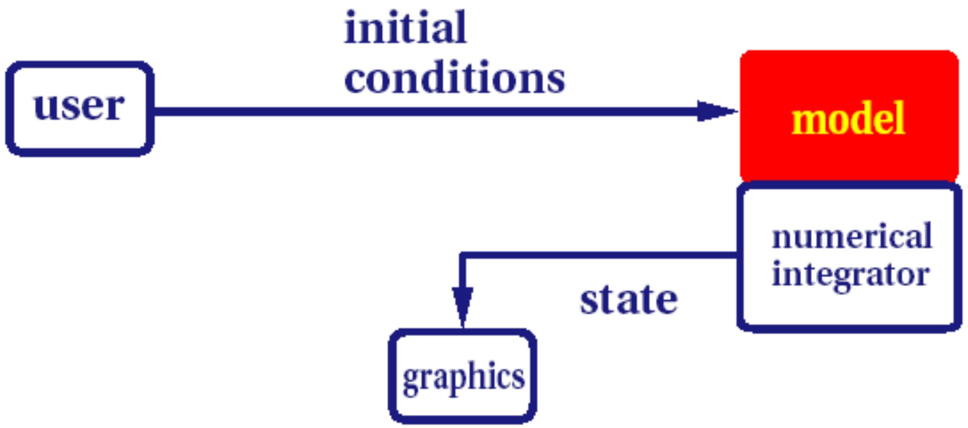


particle systems
leaves
water spray
clothing

Hodgins

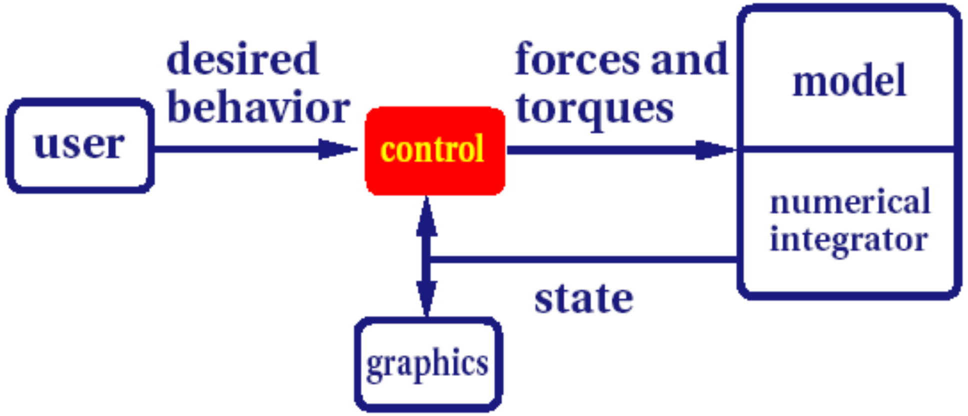
Dynamics

Passive--no muscles or motors



particle systems
leaves
water spray
clothing

Active--internal source of energy



running human
trotting dog
swimming fish

Hodgins

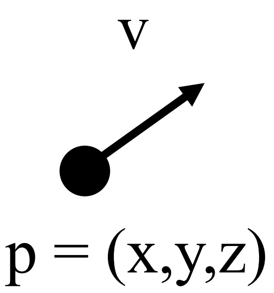
Passive Dynamics

- Physical laws
  - Newton’s laws
  - Hooke’s law
  - Etc.
- Physical phenomena
  - Gravity
  - Momentum
  - Friction
  - Collisions
  - Elasticity
  - Fracture

McAllister

Particle Systems

- A particle is a point mass
  - Position
  - Velocity
  - Mass
  - Drag
  - Elasticity
  - Lifetime
  - Color



- Use many particles to model complex phenomena
  - Keep array of particles
  - Newton's laws

Particle Systems

- For each frame:
  - For each simulation step (Δt)
    - Create new particles and assign attributes
    - Update particles based on attributes and physics
    - Delete any expired particles
  - Render particles

Creating Particles

- Where to create particles?
  - Predefined source
  - Where particle density is low
  - etc.

Reeves

Creating Particles

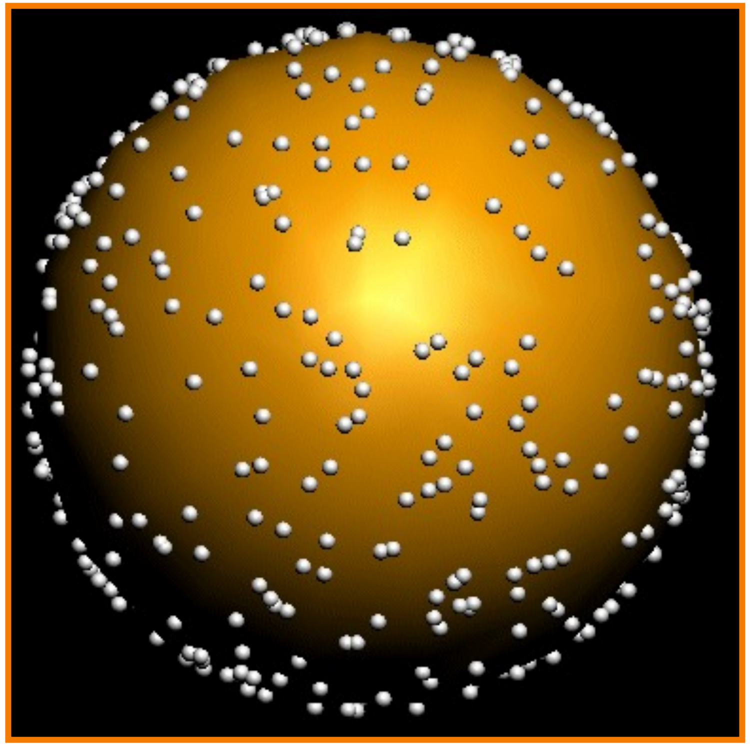
- Where to create particles?
  - Predefined source
  - Where particle density is low
  - etc.

Reeves

Creating Particles

- Example: particles emanating from shape
  - Line
  - Box
  - **Circle**
  - **Sphere**
  - Cylinder
  - Cone
  - Mesh

McAllister

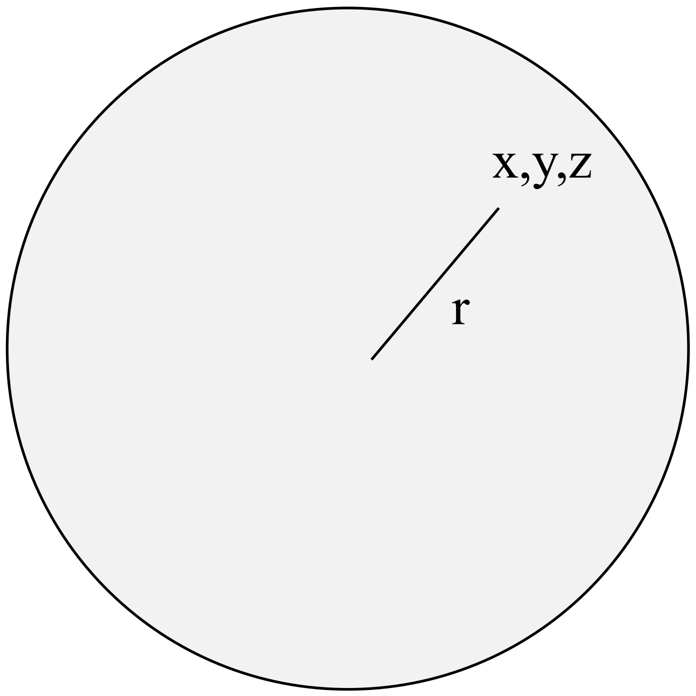


Creating Particles

- Example: particles emanating from sphere
  - Selecting random position on surface of sphere

<u>Rejection Sampling</u>:

```
// pick random point in sphere
do {
   x,y,z = random(-1,1)
   r_sq = x^2+y^2+z^2
} while (r_sq > 1)
```



Creating Particles

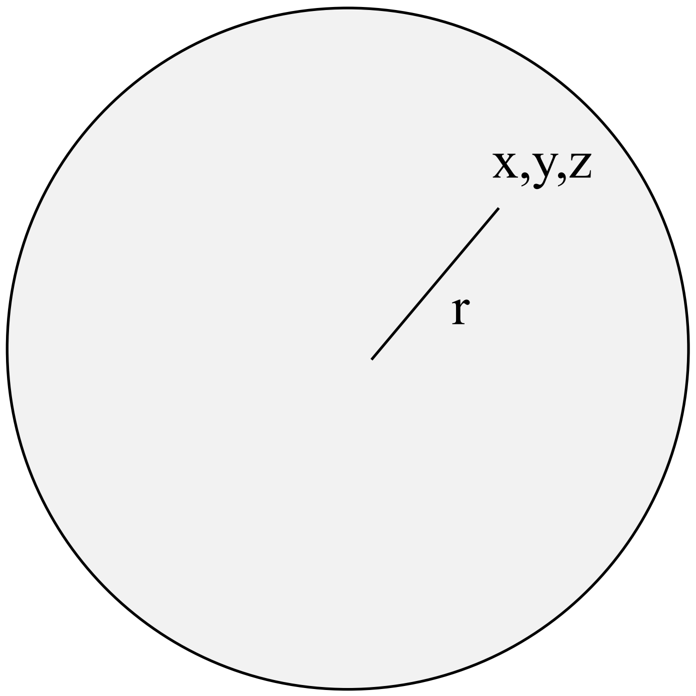
- Example: particles emanating from sphere

  - Selecting random position on surface of sphere

Rejection Sampling:

```
// pick random point in sphere
do {
   x,y,z = random(-1,1)
   r_sq = x^2+y^2+z^2
} while (r_sq > 1)

// normalize length
r = sqrt(r_sq)
x /= r
y /= r
z /= r
```

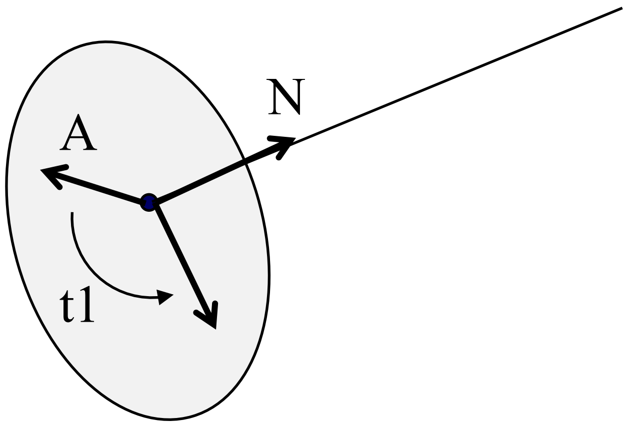


Creating Particles

- Example: particles emanating from sphere

    - Selecting random direction within angle cutoff of normal

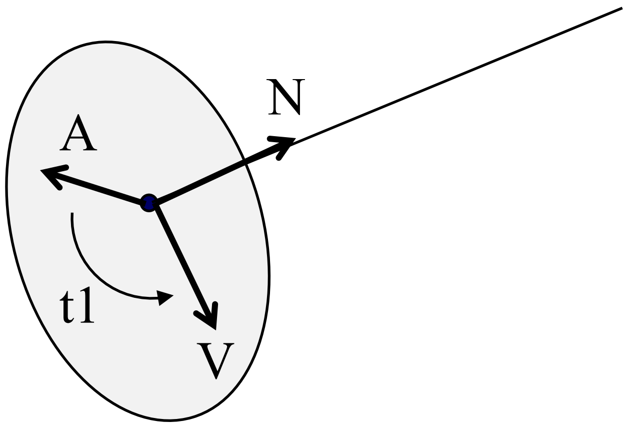
1. N = surface normal
2. A = any vector on tangent plane
3. t1 = random [0, 2π)



Creating Particles

- Example: particles emanating from sphere
  - Selecting random direction within angle cutoff of normal

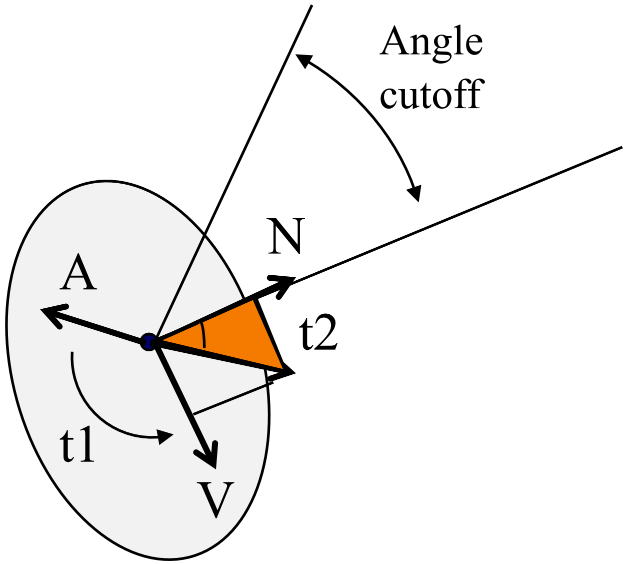
1. N = surface normal
2. A = any vector on tangent plane
3. t1 = random [0, 2π)
4. V = rotate A around N by t1



Creating Particles

- Example: particles emanating from sphere
  - Selecting random direction within angle cutoff of normal

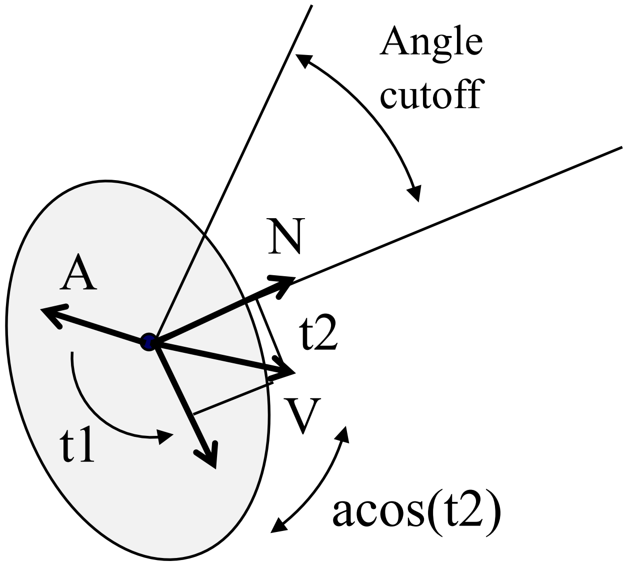
1. N = surface normal
2. A = any vector on tangent plane
3. t1 = random [0, 2π)
4. V = rotate A around N by t1
5. t2 = random [0, sin(angle cutoff))



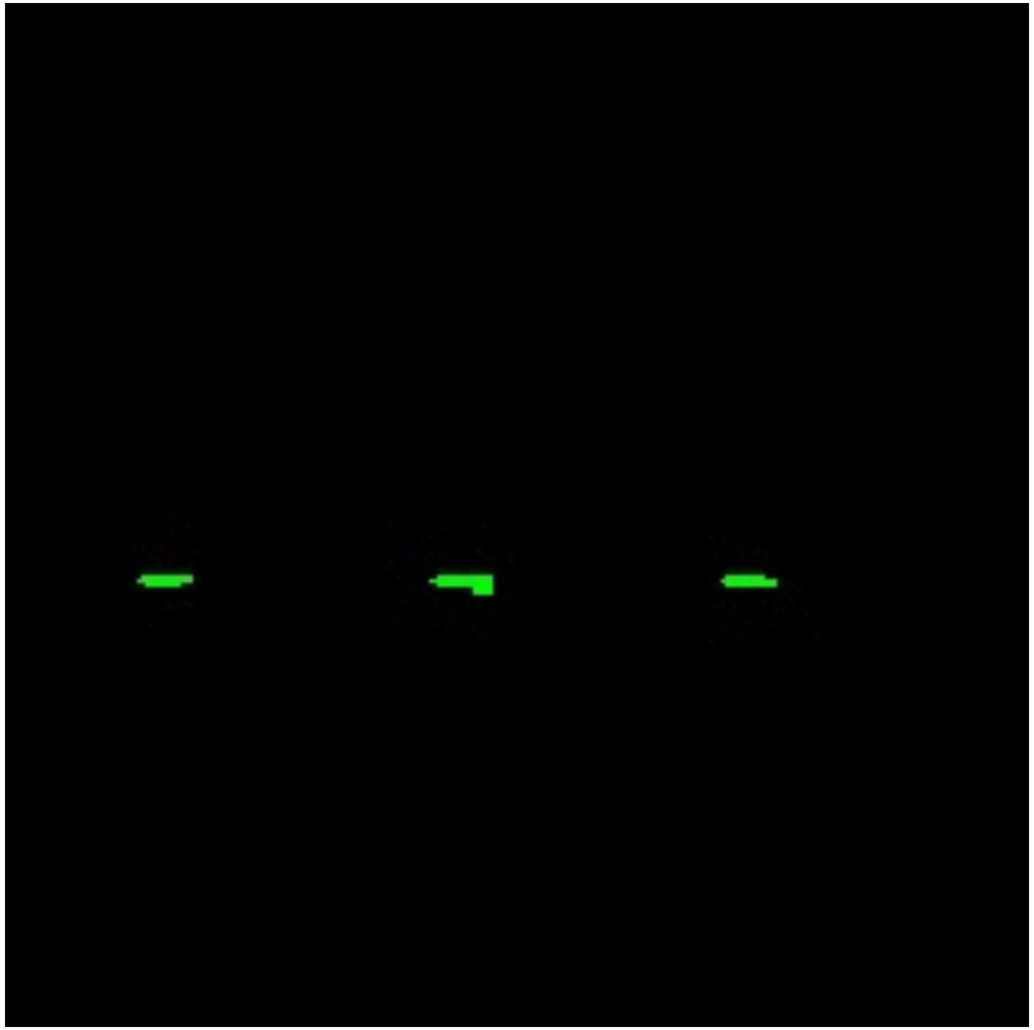
Creating Particles

- Example: particles emanating from sphere
  - Selecting random direction within angle cutoff of normal

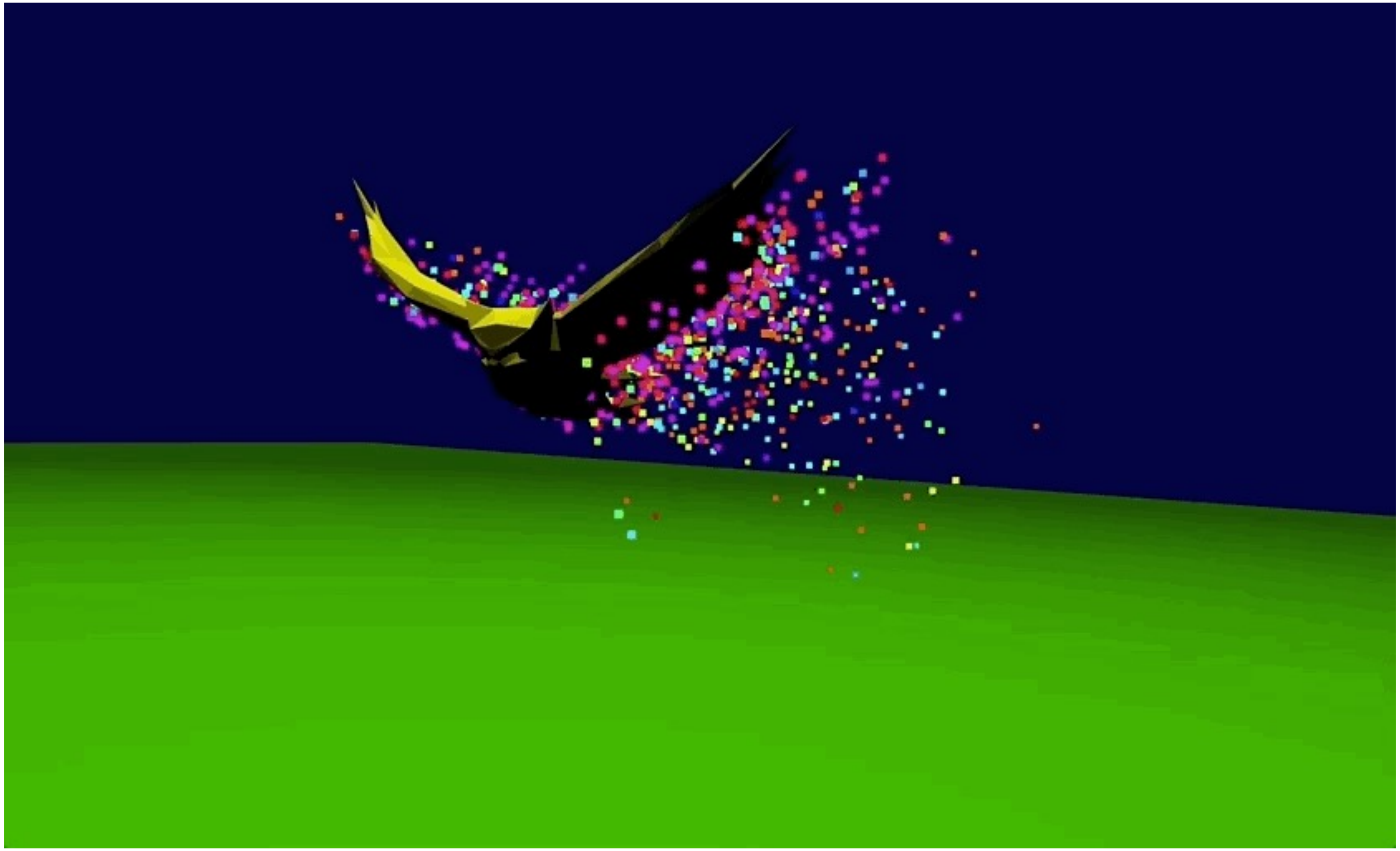
1. N = surface normal
2. A = any vector on tangent plane
3. t1 = random $[0, 2\pi)$
4. V = rotate A around N by t1
5. t2 = random [0, sin(angle cutoff))
6. V = rotate V around VxN by acos(t2)



Example: Fountains



Example: Emission from Surface



Jacob Zimmer, COS 426 2018

Particle Systems

- For each frame:
  - For each simulation step (Δt)
    - Create new particles and assign attributes
    - Update particles based on attributes and physics
    - Delete any expired particles
  - Render particles

Equations of Motion

- Newton's Law for a point mass
  - f = ma
  - And remember: dx/dt = v and dv/dt = a

Equations of Motion

- Newton's Law for a point mass
  - f = ma
  - And remember: dx/dt = v   and   dv/dt = a

$$\begin{cases} \dot{x} = v \\ \dot{v} = \dfrac{f}{m} \end{cases}$$

Equations of Motion

- Newton's Law for a point mass
  - f = ma
  - And remember: dx/dt = v and dv/dt = a

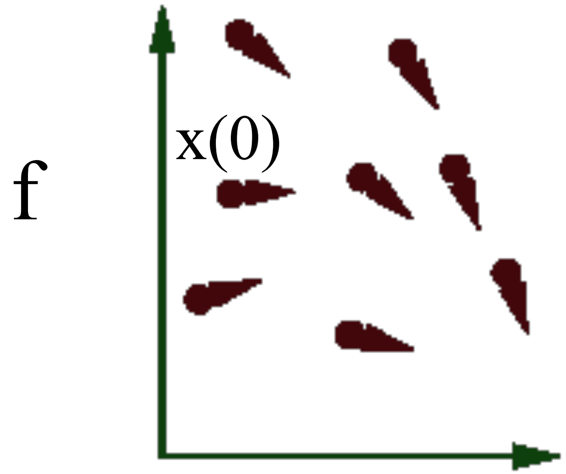
$$\begin{cases} \dot{x} = v \\ \dot{v} = \dfrac{f}{m} \end{cases}$$

- Computing particle motion requires solving second-order differential equation

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

Solving the Equations of Motion

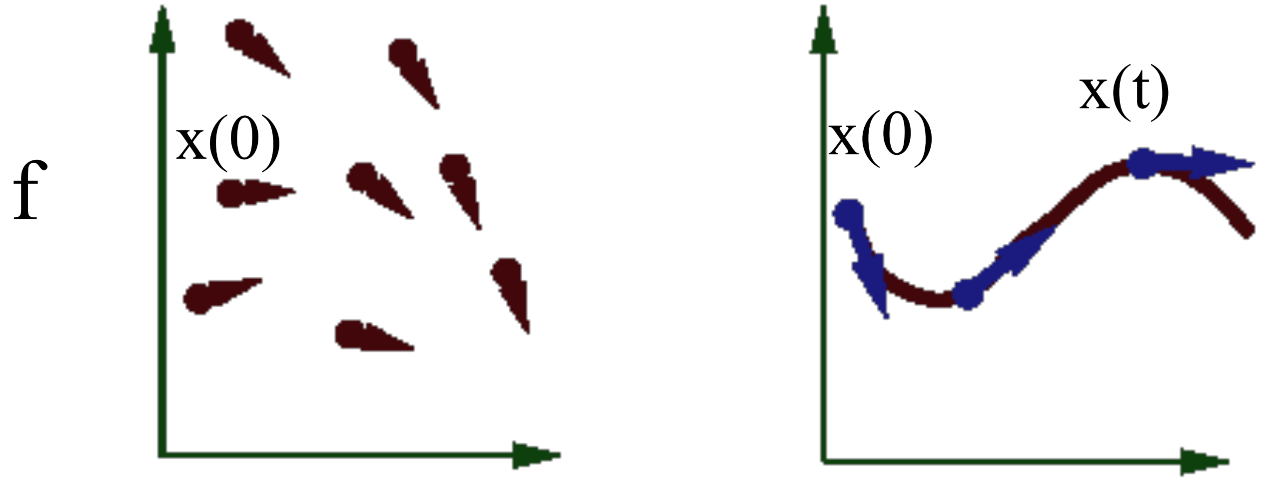
- Initial value problem
  - Know x(0), v(0)
  - Can compute force (and therefore acceleration) for any position / velocity / time



Hodgins

Solving the Equations of Motion

- Initial value problem
  - Know x(0), v(0)
  - Can compute force (and therefore acceleration) for any position / velocity / time
  - Compute x(t) by forward integration



Hodgins

Solving the Equations of Motion

- Forward (explicit) Euler integration

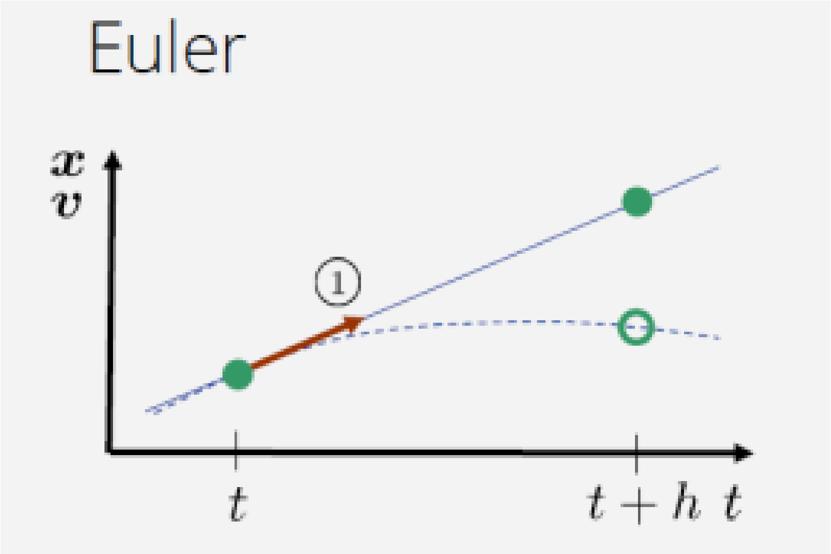
Euler Step (1768)

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

- **Idea**: start at initial condition and take a step into the direction of the tangent.
- Iteration scheme: $y_n \rightarrow f(t_n, y_n) \rightarrow y_{n+1} \rightarrow f(t_{n+1}, y_{n+1}) \rightarrow \ldots$

Solving the Equations of Motion

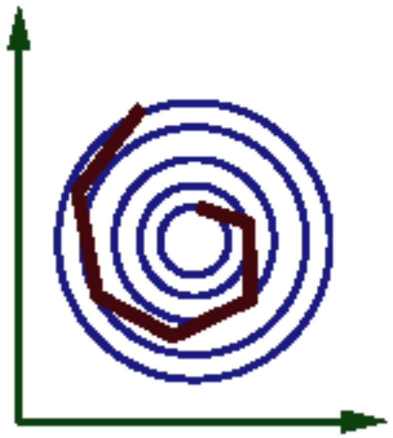
- Forward (explicit) Euler integration
  - $x(t+\Delta t) \leftarrow x(t) + \Delta t \, v(t)$
  - $v(t+\Delta t) \leftarrow v(t) + \Delta t \, f\big(x(t), v(t), t\big) / m$



Teschner

Solving the Equations of Motion

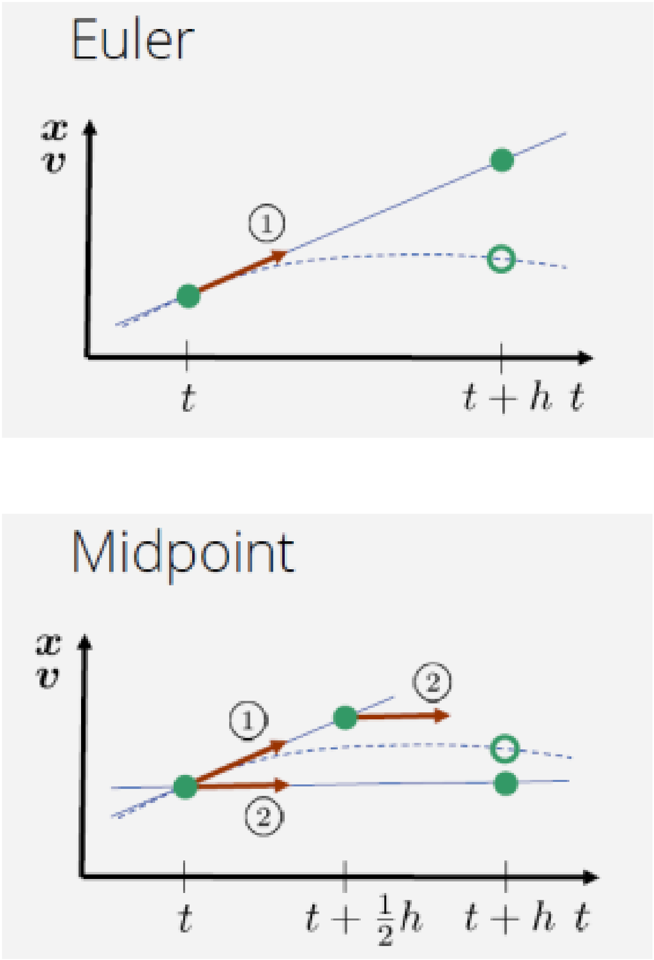
- Forward (explicit) Euler integration
  - $x(t+\Delta t) \leftarrow x(t) + \Delta t\, v(t)$
  - $v(t+\Delta t) \leftarrow v(t) + \Delta t\, f(x(t), v(t), t) / m$
- Problem:
  - Accuracy decreases as Δt gets bigger



Hodgins

Solving the Equations of Motion

- Midpoint method
  1. Compute an Euler step
  2. Evaluate f at the midpoint of Euler step
  3. Compute new position / velocity using midpoint velocity / acceleration



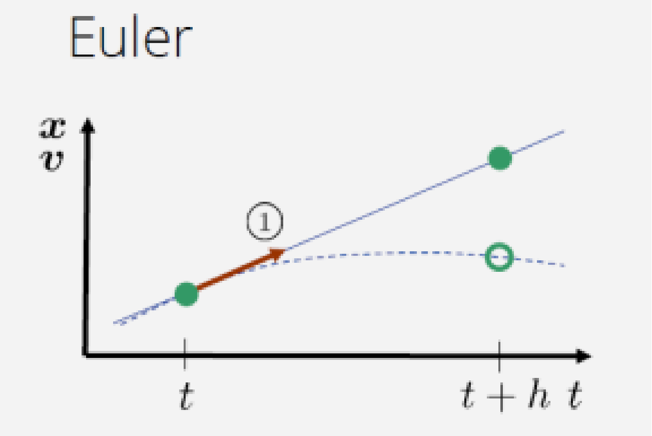
Teschner

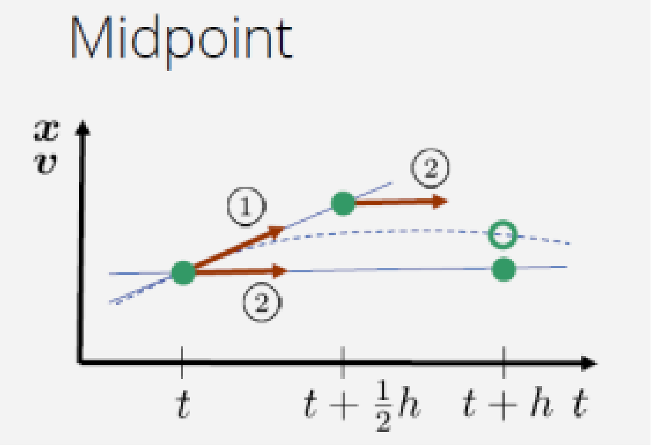
Solving the Equations of Motion

- Midpoint method
  1. Compute an Euler step
  2. Evaluate f at the midpoint of Euler step
  3. Compute new position / velocity using midpoint velocity / acceleration

$x_{mid} \leftarrow x(t) + \Delta t / 2 * v(t)$

$v_{mid} \leftarrow v(t) + \Delta t / 2 * f(x(t), v(t), t) / m$





Teschner

Solving the Equations of Motion

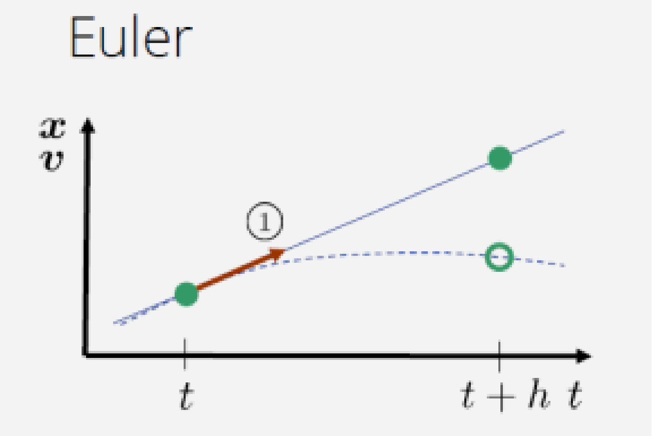
- Midpoint method
  1. Compute an Euler step
  2. Evaluate f at the midpoint of Euler step
  3. Compute new position / velocity using midpoint velocity / acceleration

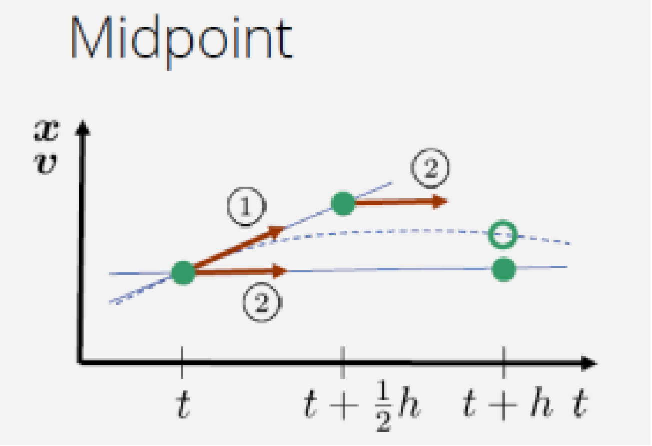
$x_{mid} \leftarrow x(t) + \Delta t / 2 * v(t)$

$v_{mid} \leftarrow v(t) + \Delta t / 2 * f(x(t), v(t), t) / m$

$x(t+\Delta t) \leftarrow x(t) + \Delta t\ v_{mid}$

$v(t+\Delta t) \leftarrow v(t) + \Delta t\ f(x_{mid}, v_{mid}, t) / m$

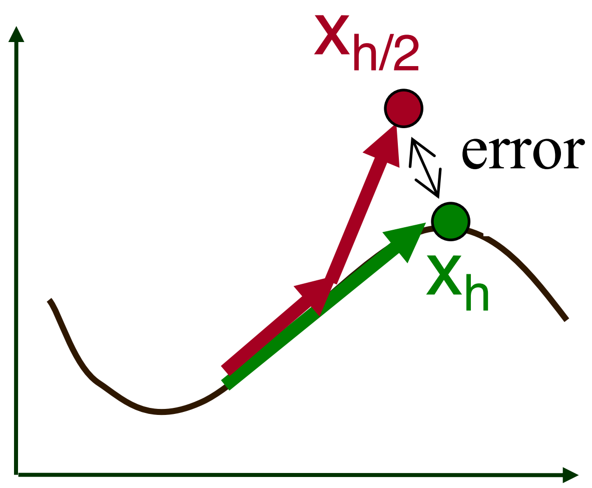


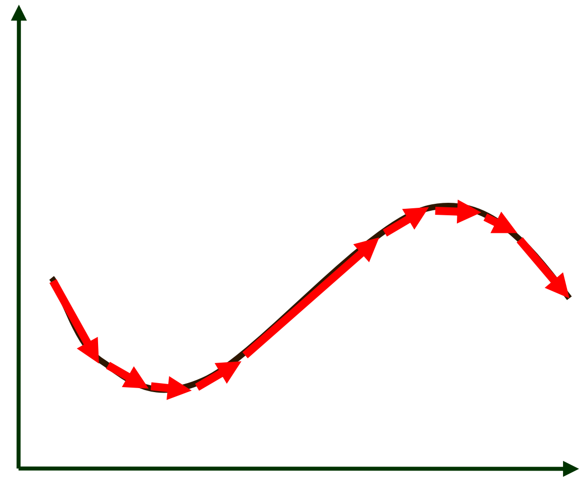


Teschner

Solving the Equations of Motion

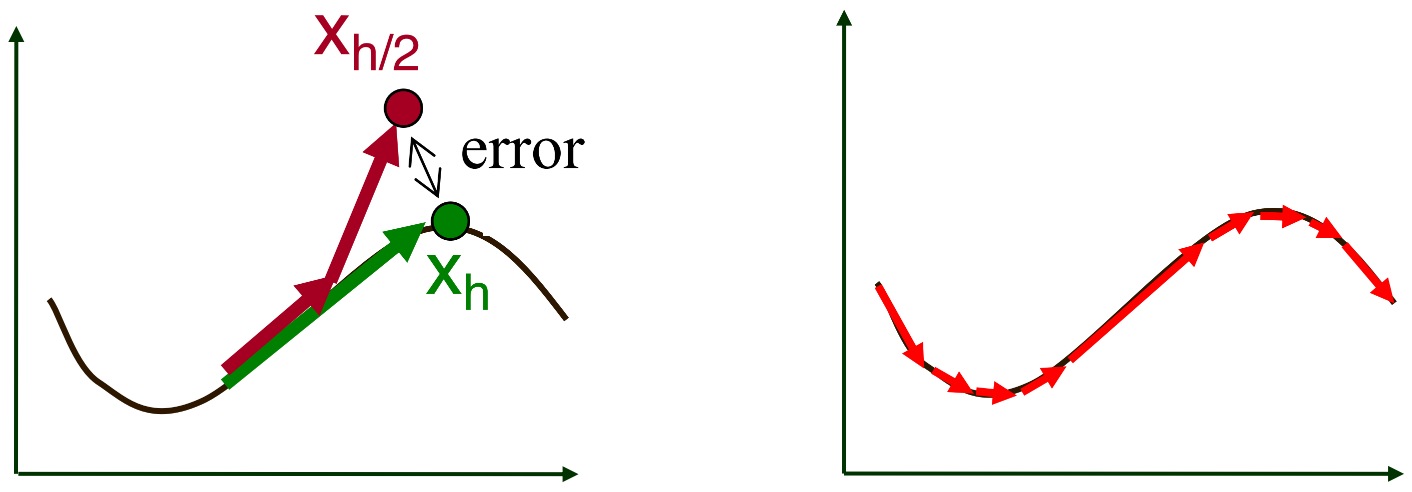
- Adaptive step size
  - Repeat until error is below threshold
    1. Compute x_h by taking one step of size h
    2. Compute $x_{h/2}$ by taking 2 steps of size h / 2
    3. Compute error = | x_h - $x_{h/2}$ |





Solving the Equations of Motion

- Adaptive step size
  - Repeat until error is below threshold
    1. Compute x_h by taking one step of size h
    2. Compute $x_{h/2}$ by taking 2 steps of size h / 2
    3. Compute error = | x_h - $x_{h/2}$ |
    4. If (error < threshold) break
    5. Else, reduce step size and try again



Particle System Forces

- Force fields
  - Gravity, wind, pressure
- Viscosity/damping
  - Drag, friction
- Collisions
  - Static objects in scene
  - Other particles
- Attraction and repulsion
  - Springs between neighboring particles (mesh)
  - Gravitational pull, charge

Particle System Forces

- Gravity
  - Force due to gravitational pull (of earth)
  - g = acceleration due to gravity (m/s^2)

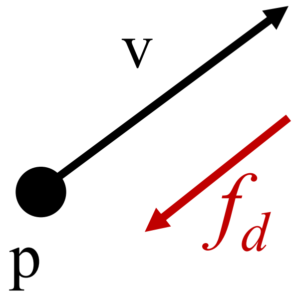
$$f_g = mg$$

g = (0, -9.80665, 0)

Particle System Forces

- Drag
  - Force due to resistance of medium
  - k_{drag} = drag coefficient (kg/m)

$$f_d = -k_{drag} v^2$$

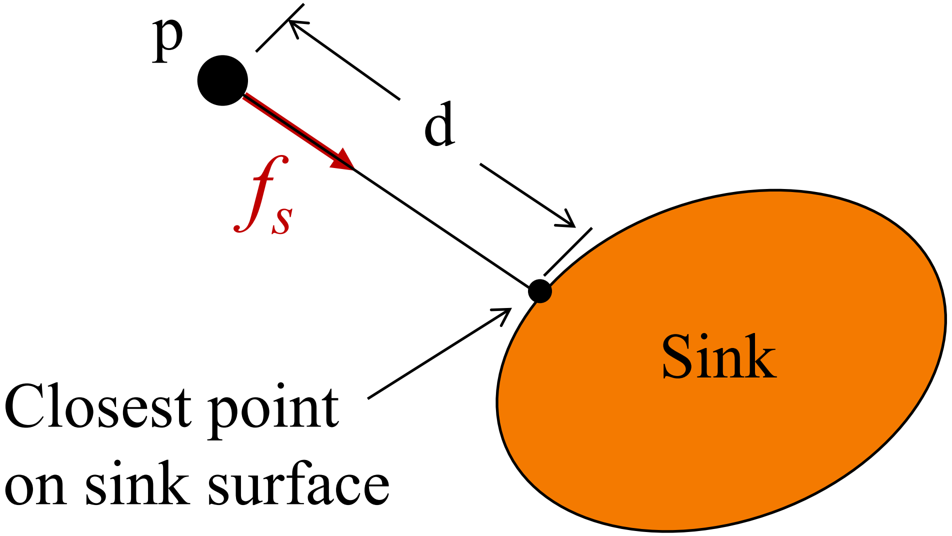


  - Air resistance taken as proportional to v^2

Particle System Forces

- Sinks
  - Force due to attractor in scene

$$f_s = \frac{\text{intensity}}{c_a + l_a \cdot d + q_a \cdot d^2}$$

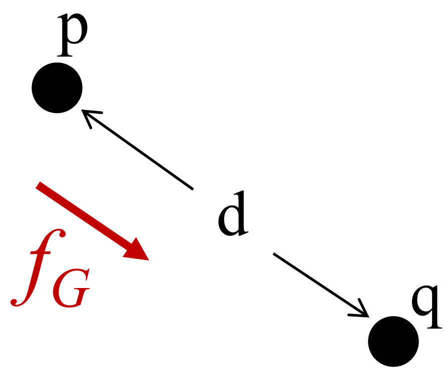


Particle System Forces

- Gravitational pull of other particles
  - Newton's universal law of gravitation

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

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

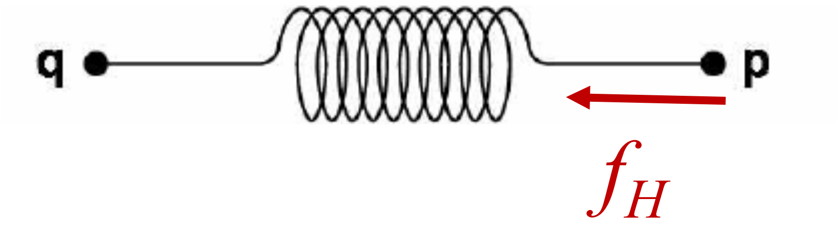


Particle System Forces

- Springs
  - Hooke's law

$$f_H(p) = k_s\left(d(p,q) - s\right)\ D$$

$$D = (q - p)/\|q - p\|$$
$$d(p,q) = \|q - p\|$$
$$s = \text{resting length}$$
$$k_s = \text{spring coefficient}$$



Particle System Forces

- Springs
  - Hooke's law with damping

$$f_H(p) = \left[k_s\left(d(p,q) - s\right) + k_d\left(v(q) - v(p)\right)\cdot D\right] D$$

$$D = (q-p)/\|q-p\|$$

$$d(p,q) = \|q-p\|$$

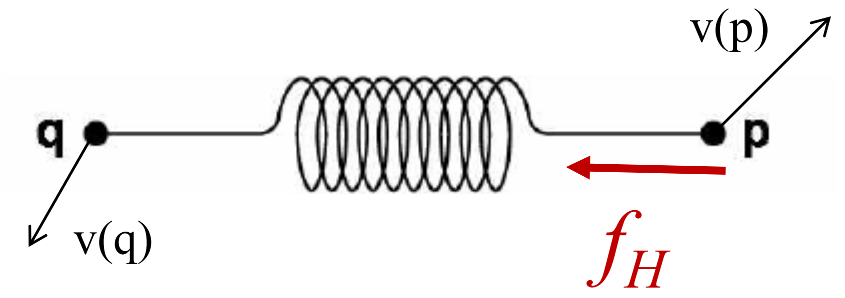
s = resting length

k_s = spring coefficient

k_d = damping coefficient

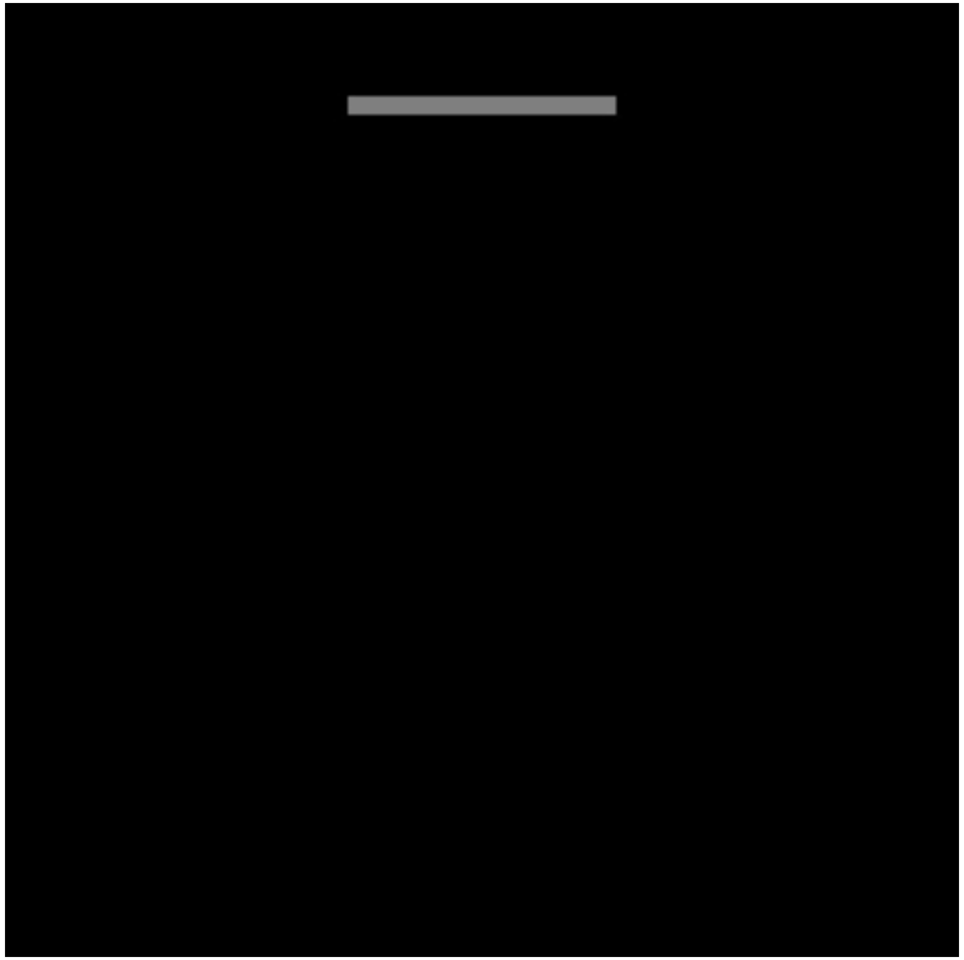
$v(p)$ = velocity of p

$v(q)$ = velocity of q



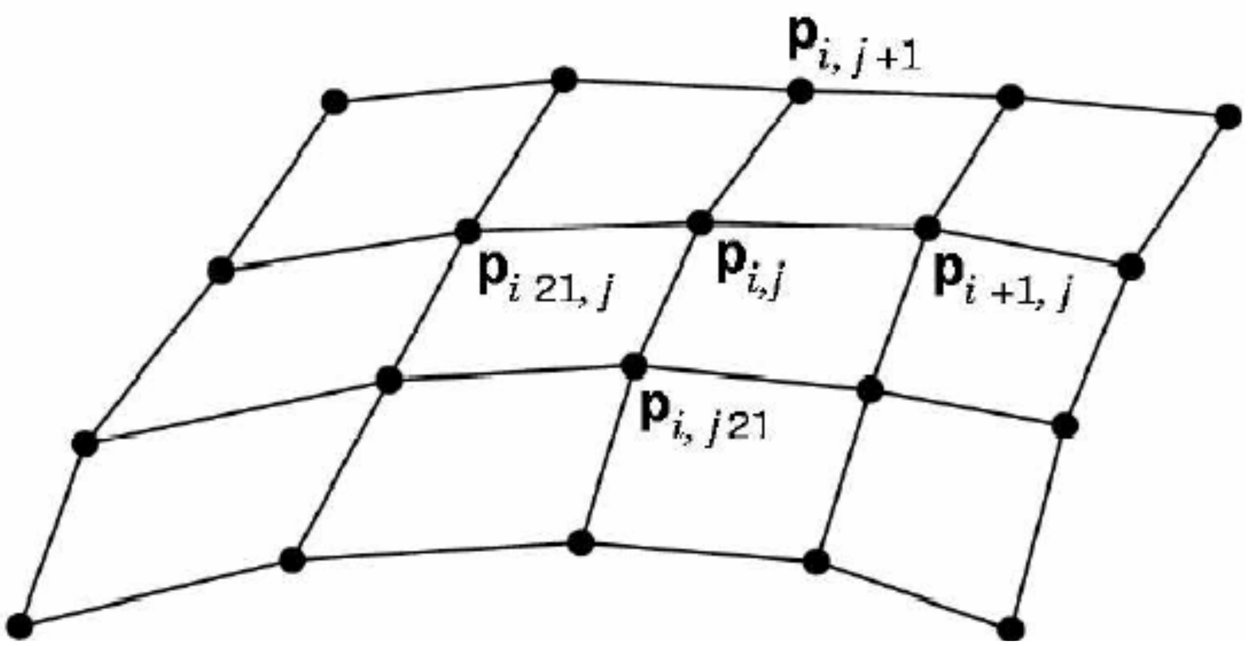
$$k_d \sim 2\sqrt{mk_s}$$

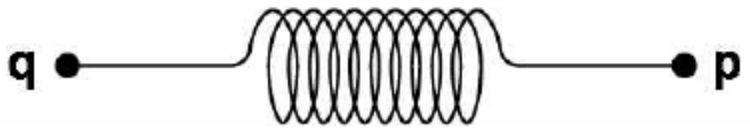
Example: Rope



Particle System Forces

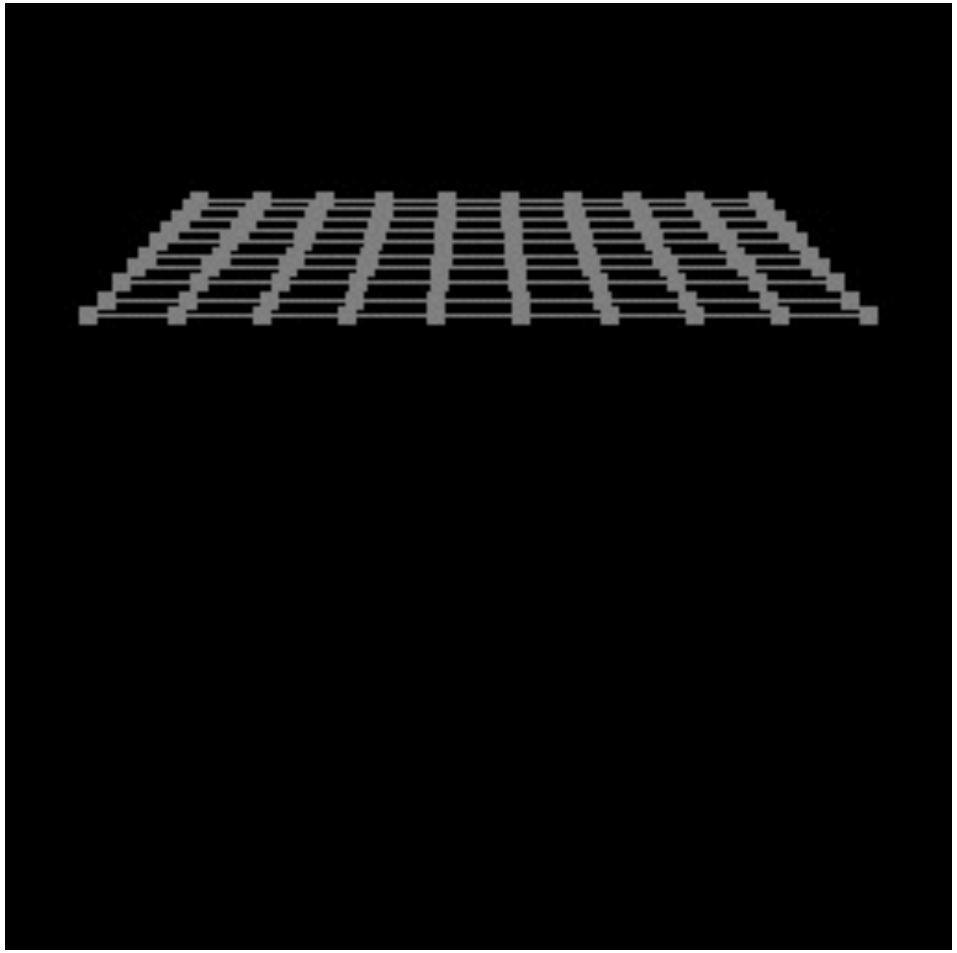
- Spring-mass mesh





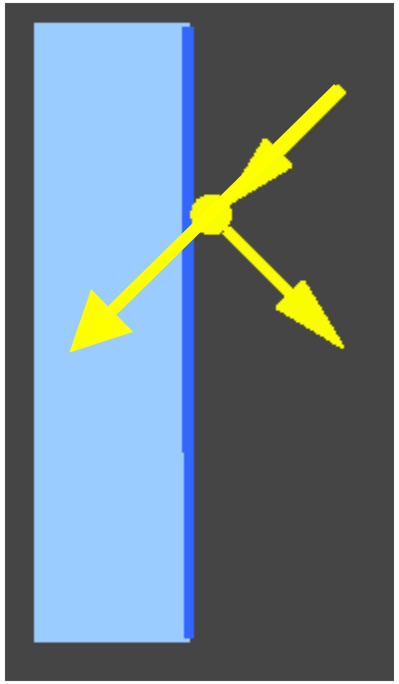
Hodgins

Example: Cloth



Particle System Forces

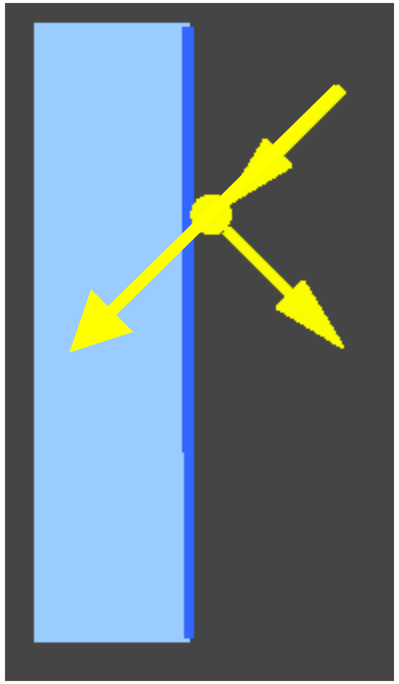
- Collisions
  - Collision detection
  - Collision response



Witkin

Particle System Forces

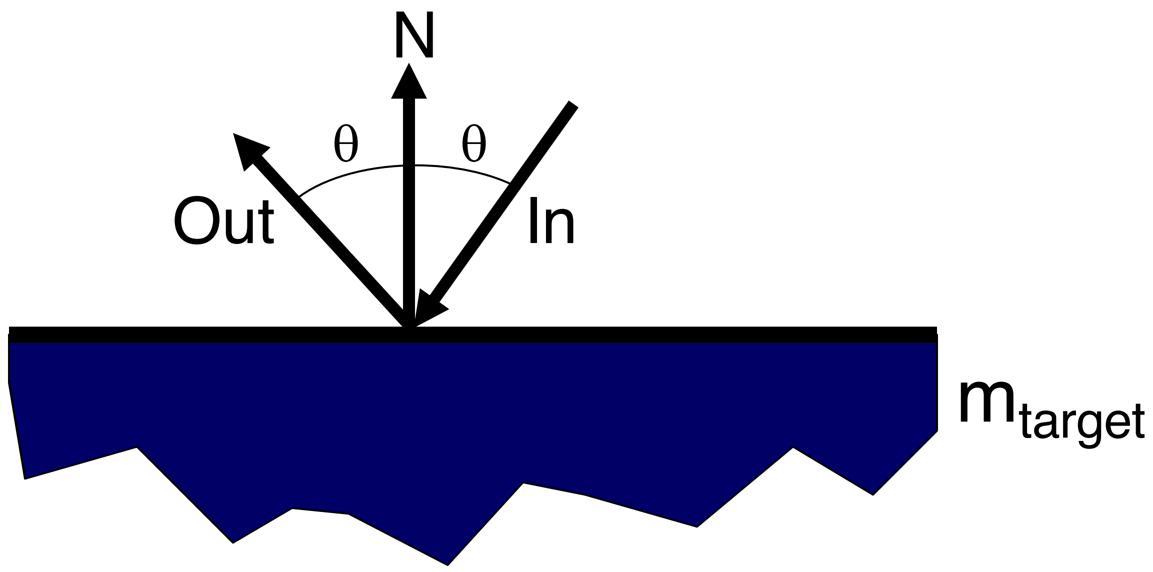
- Collision detection
  - Intersect ray with scene
  - Compute up to Δt away from time of time of first collision, and then continue from there



Witkin

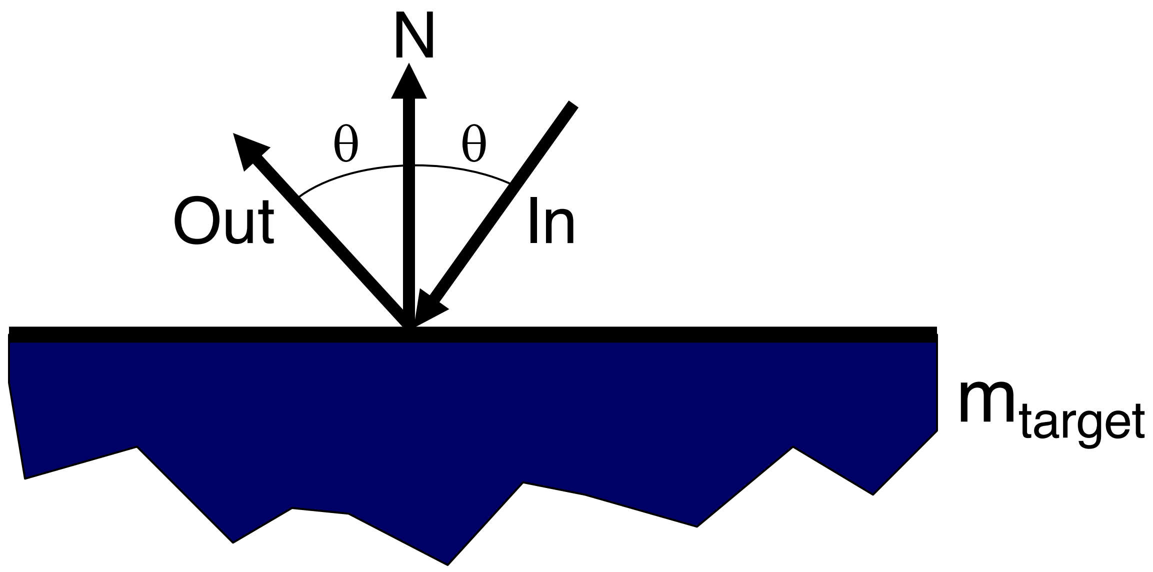
Particle System Forces

- Collision response
  - No friction: elastic collision
    (for $m_{target} >> m_{particle}$: specular reflection)



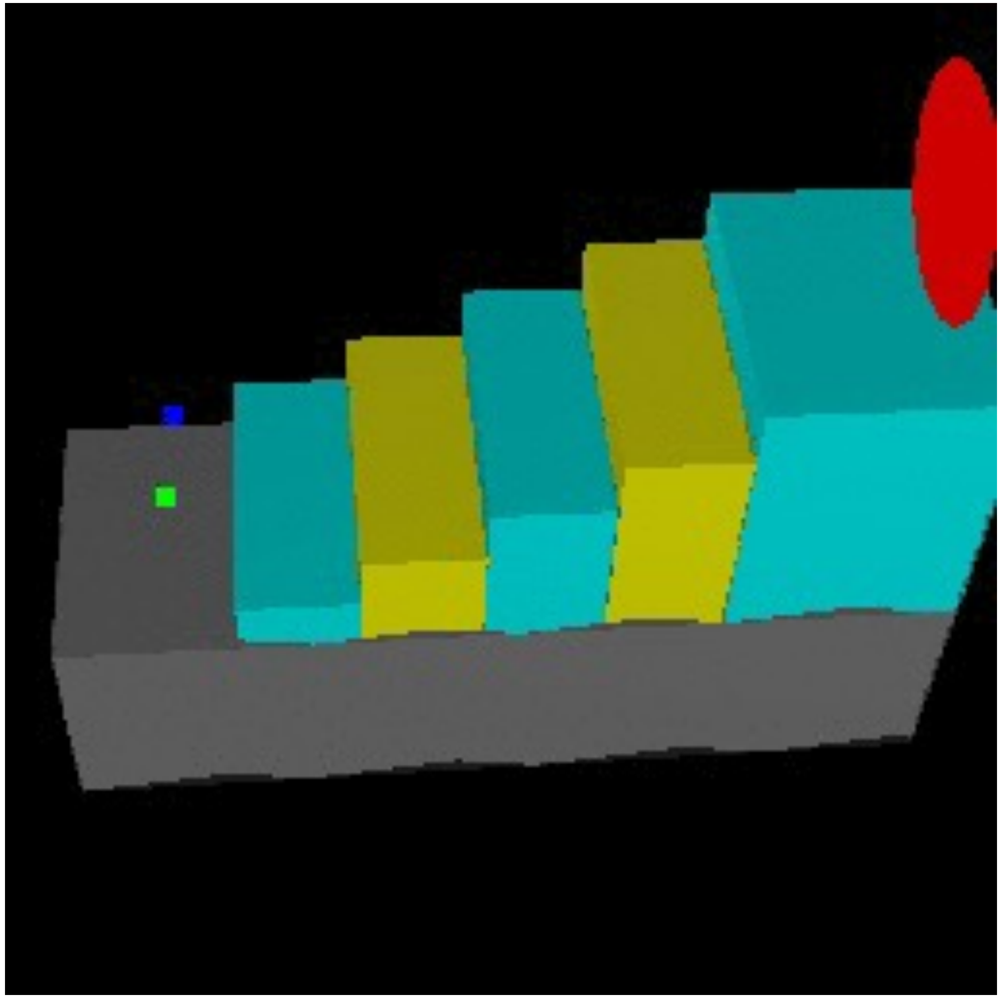
Particle System Forces

- Collision response
  - No friction: elastic collision
    (for $m_{target} >> m_{particle}$: specular reflection)



  - Otherwise, total momentum conserved, energy dissipated if inelastic

Example: Bouncing



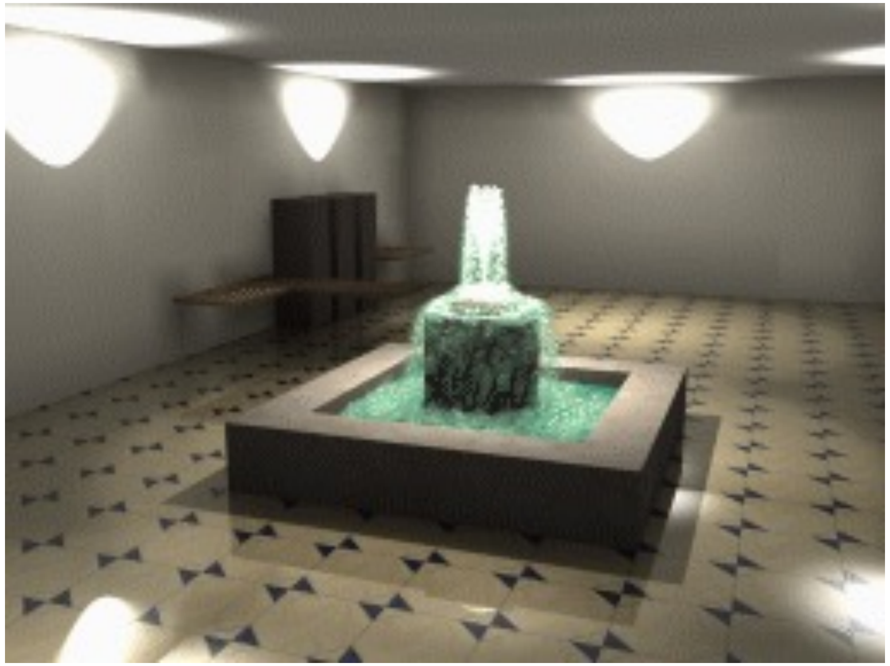
Ning Jin
COS 426, 2013

Particle Systems

- For each frame:
  - For each simulation step (Δt)
    - Create new particles and assign attributes
    - Update particles based on attributes and physics
    - Delete any expired particles
  - Render particles

Deleting Particles

- When to delete particles?
  - When life span expires
  - When intersect predefined sink surface
  - Where density is high
  - Random



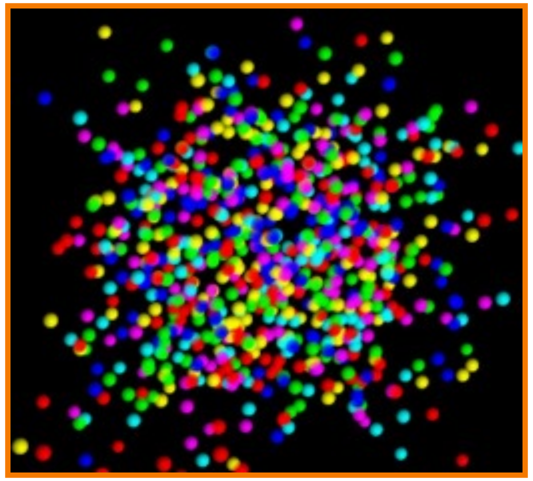
McAllister

Particle Systems

- For each frame:
  - For each simulation step (Δt)
    - Create new particles and assign attributes
    - Update particles based on attributes and physics
    - Delete any expired particles
  - Render particles

Rendering Particles

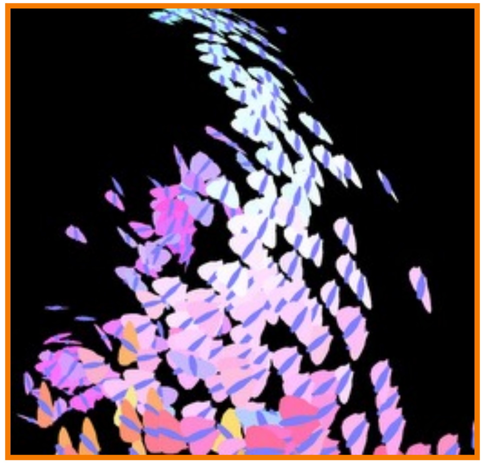
- Rendering styles
  - ➢ Points
  - ◦ Polygons
  - ◦ Shapes
  - ◦ Trails
  - ◦ etc.



McAllister

Rendering Particles

- Rendering styles
  - Points
  - ➢ Textured polygons: sprites
  - Shapes
  - Trails
  - etc.



Rendering Particles

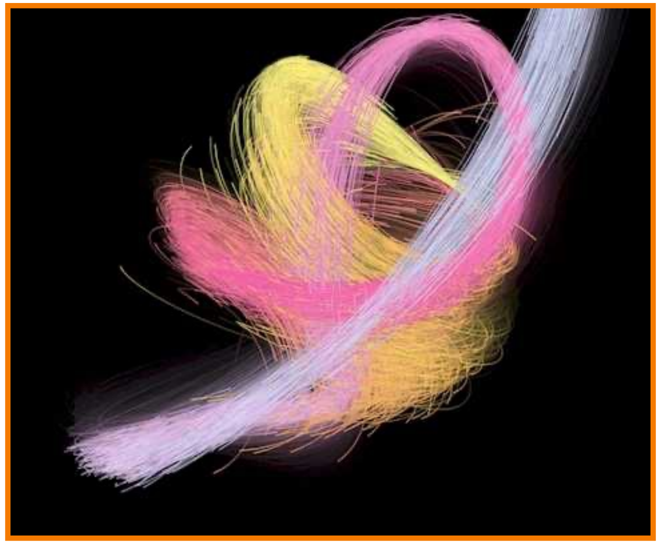
- Rendering styles
  - Points
  - Polygons
  - ➢ Shapes
  - Trails
  - etc.

McAllister

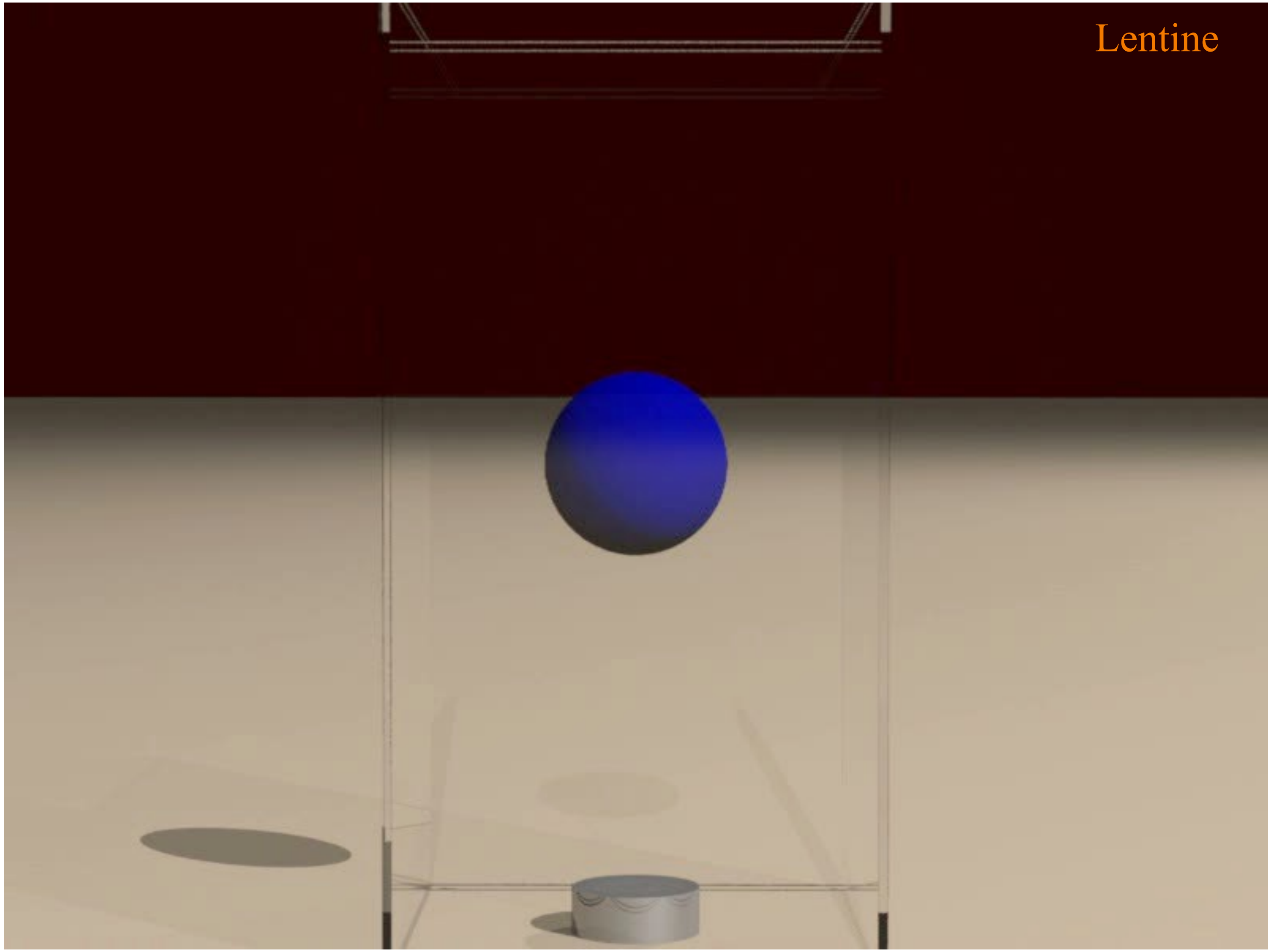
Rendering Particles

- Rendering styles
  - Points
  - Polygons
  - Shapes
  - Trails
  - etc.

McAllister

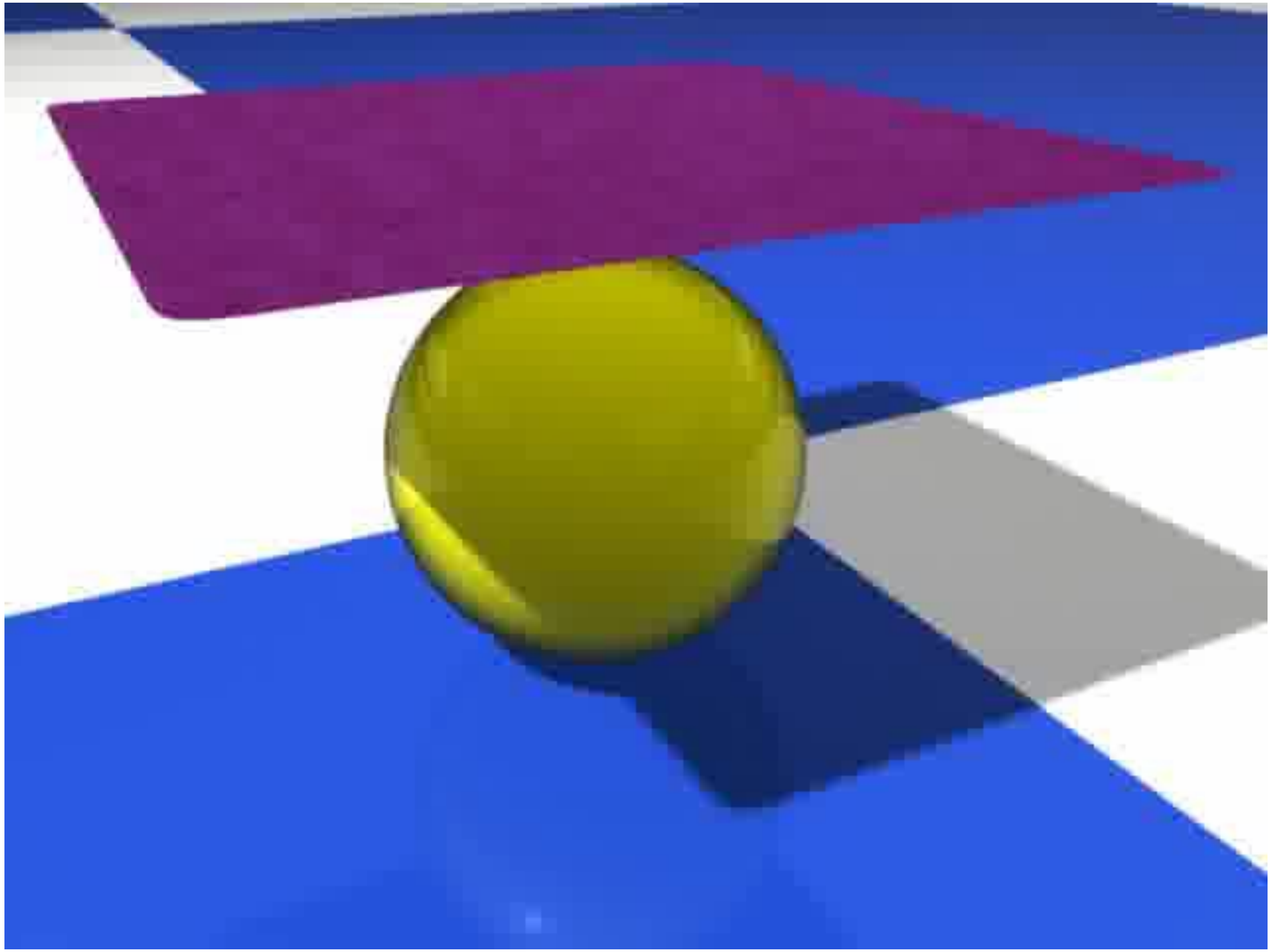


Example: “Smoke”



Example: Fire

Example: Cloth



Bender

Summary

- Particle systems
  - Lots of particles
  - Simple physics
- Interesting behaviors
  - Smoke
  - Cloth
- Solving motion equations
  - For each step, first sum forces, then update position and velocity