CG – T14 – Particle Systems

L:CC, MI:ERSI

Miguel Tavares Coimbra

(course and slides designed by Verónica Costa Orvalho)

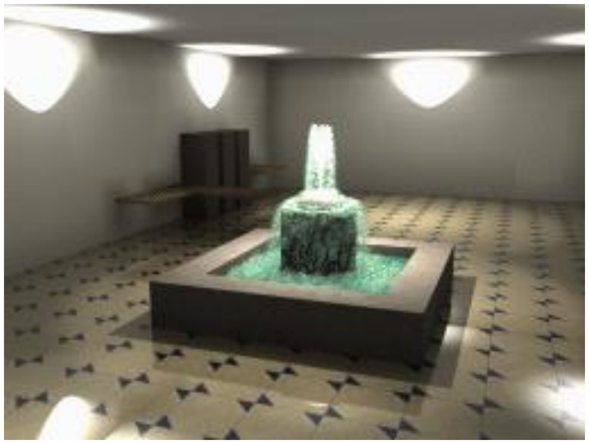
U. PORTO FC

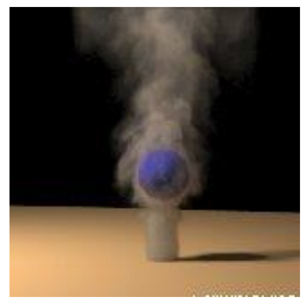
introduction

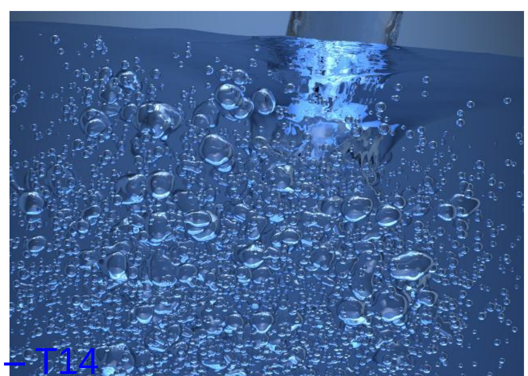
Particles systems **what for**?

solution to modeling amorphous, dynamic and fluid objects like clouds, smoke, water, explosions and fire.

How can we do it?

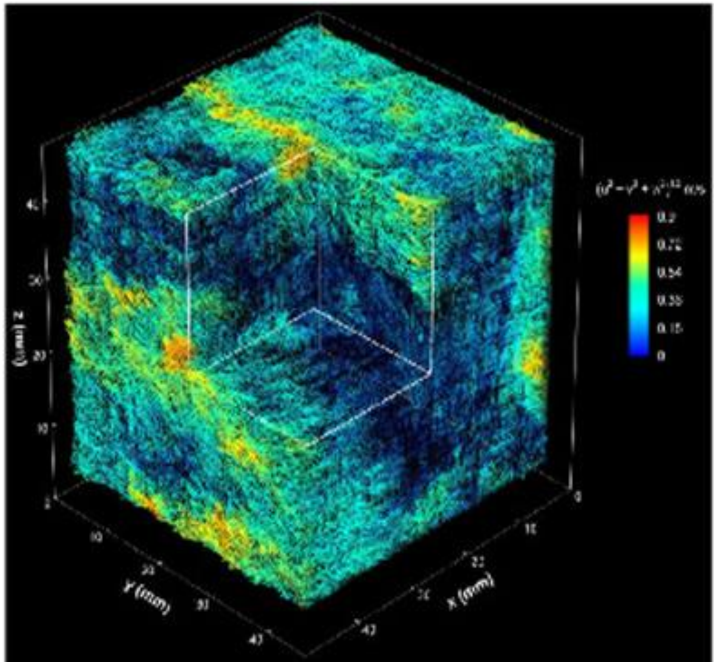


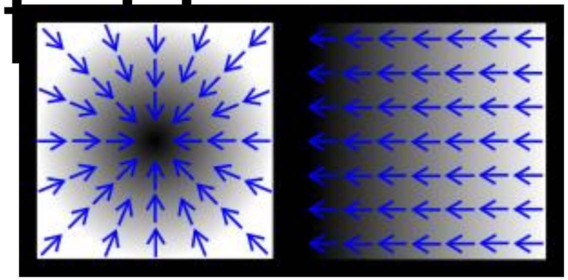


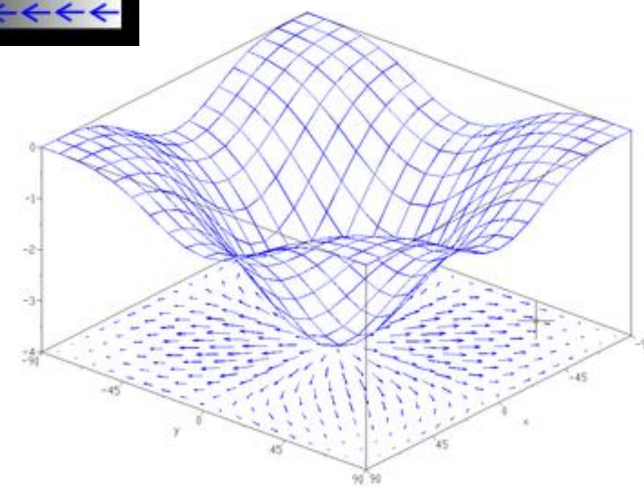


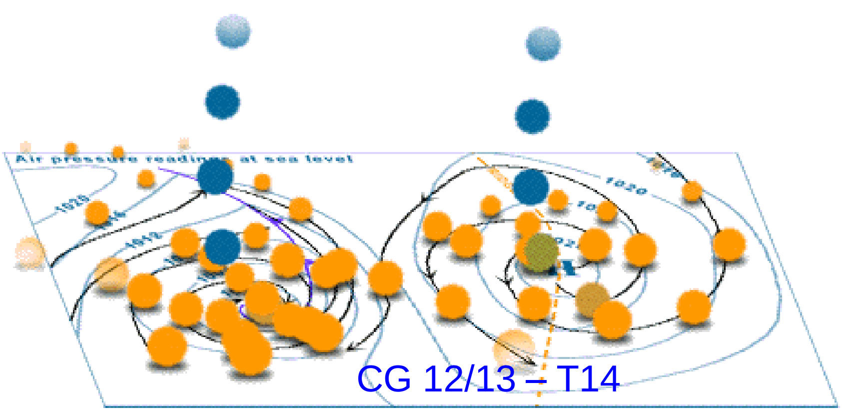
Ron Fedkiw
Jeong-Mo Hong

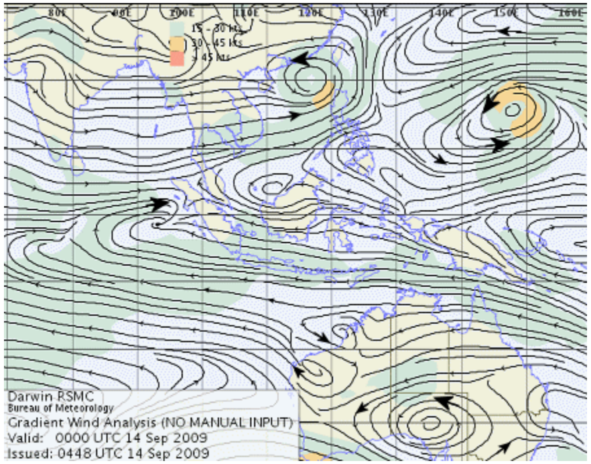
gradient, vector fields











videos

Boids + music >> Craig Reynolds interpretation
http://vimeo.com/6068511

representing objects with particles

- An object is represented as **clouds of primitive particles** that define its volume rather than by polygons or patches that define its boundary
- A particle system is **dynamic**, particles changing form and moving with the passage of time.
- Object is **not deterministic**, its shape and form are not completely specified

Basic Model of Particle Systems

1) New particles are **generated** into the system
2) Each new particle is **assigned** its individual **attributes**
3) Any particle that has existed past its prescribed lifetime is **extinguished**
4) The **remaining** particles are **moved** and **transformed** according to their dynamic attributes
5) An image of the **particles** is **rendered** in the frame buffer, often using special purpose algorithms.

Particle generation

- Particles are generated using processes with an element of randomness.

- One way to control the number of particles created is by the particles generated per frame:

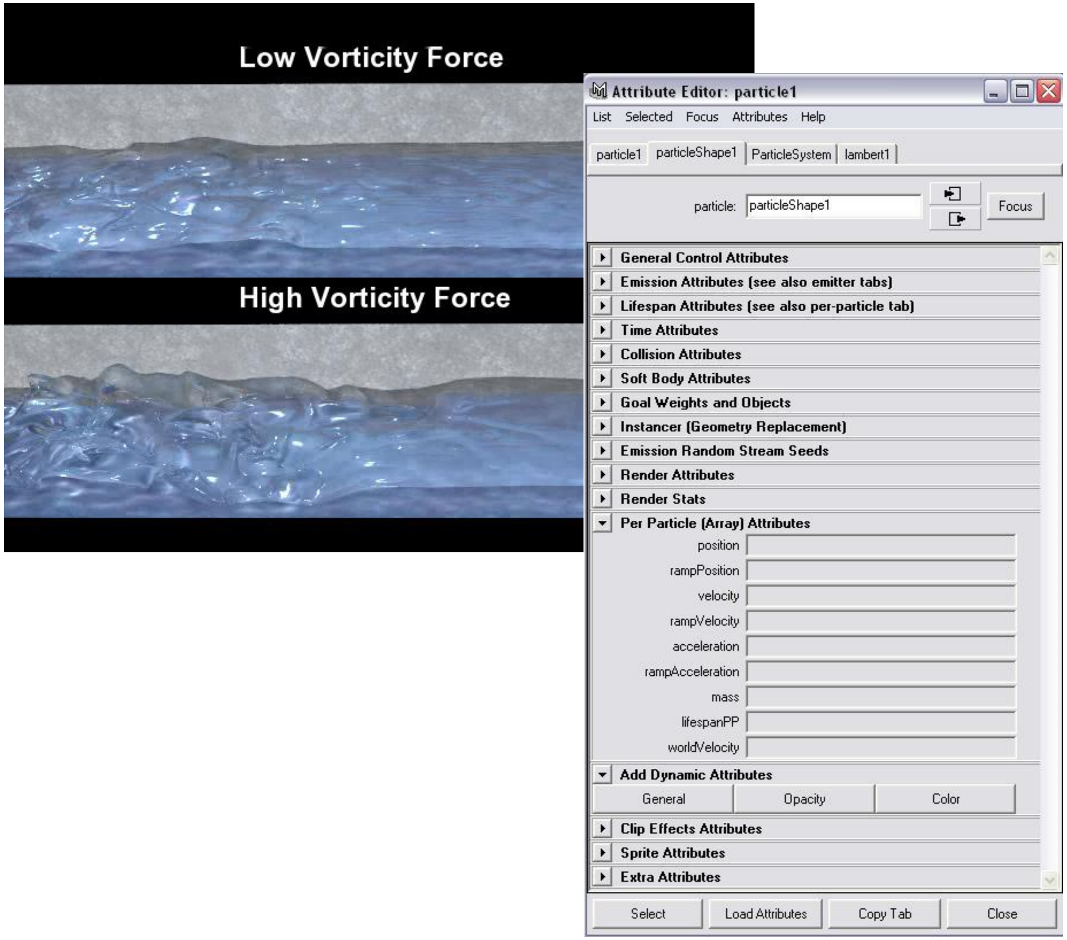
$$Nparts_f = MeanParts_f + Rand() \times VarianceParts_f$$

- Another method generates a certain number of particles per screen area:

$$Nparts_f = (MeanParts_{SAf} + Rand() \times VarianceParts_{SAf}) \times ScreenArea$$

- With this method the number of new particles depends on the screen size of the object.

Particle attributes



Alias|Wavefront's Maya

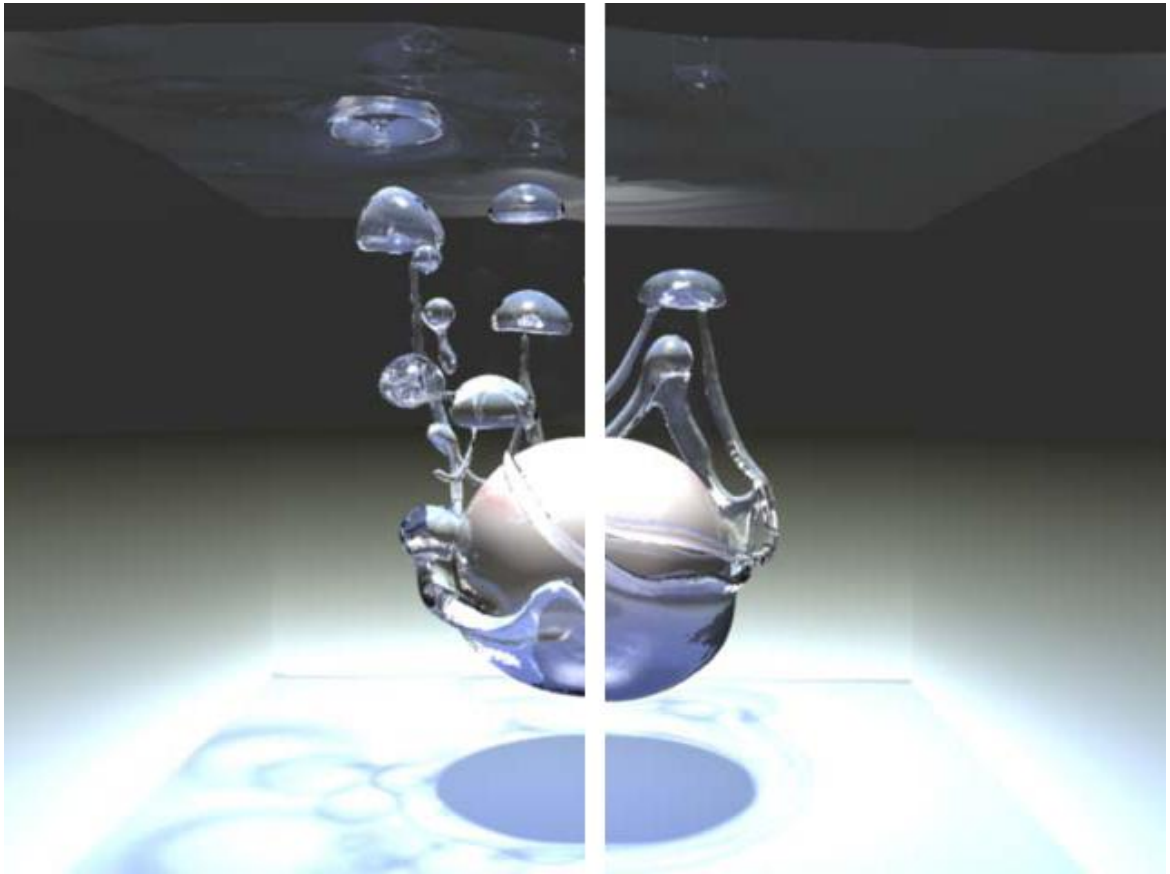
Each Particle Has:

. Position
. Velocity
. Color
. Lifetime
. Age
. Shape
. Size
. Transparency

Particle extinction

- When generated, given a lifetime in frames.
- Lifetime decremented each frame, particle is killed when it reaches zero.
- Kill particles that no longer contribute to image (transparency below a certain threshold, etc.).

Particle rendering



Particles can obscure other objects behind them, can be transparent, and can cast shadows on other objects. The objects may be polygons, curved surfaces, or other particles.

Types of particle system?

- Stateless Particle System
  - A particle data is computed from birth to death by a closed form function defined by a set of start values and a current time. (does not react to dynamic environment)
- State Preserving Particle System
  - Uses numerical iterative integration methods to compute particle data from previous values and changing environmental descriptions.

Types of particle system?

1) Stateless Particle System

A particle data is **computed** from birth to death by a **closed form function** defined by a set of start values and a current time.

. **does not react to dynamic environment**

. **no storage of varying data**

2) State Preserving Particle System

Uses **numerical iterative integration** methods to compute particle data from previous values.

. **changing environmental descriptions.**

particle life cycle

1) Generation

Particles are generated randomly within a predetermined location

2) Particle Dynamics

The attributes of a particle may vary over time. Based upon equations depending on attribute

3) Extinction

Age: Time the particle has been alive
Lifetime: Maximum amount of time the particle can live.

4) Premature Extinction

Running out of bounds
Hitting an object (ground)
Attribute reaches a threshold (particle becomes transparent)

particle rendering

1) Rendered as a graphics primitive

2) Particles that map to the same pixels are additive

- Sum the colors together
- No hidden surface removal
- Motion blur is rendered by streaking based on the particles position and velocity

rendering passes

Algorithm:

1) Process Birth and Deaths
2) Update Velocities
3) Update Positions
4) Sort Particles (optional, takes multiple passes)
5) Transfer particle positions from pixel to vertex memory
6) Render particles

data storage

1) Two Textures (position and velocity)

- Each holds an x,y,z component
- Conceptually a 1d array
- Stored in a 2d texture (why)

2) Use texture pair and double buffering to compute new data from previous data

- Total number of textures needed ?

3) Storage Types

- Velocity can be stored using 16bit floats

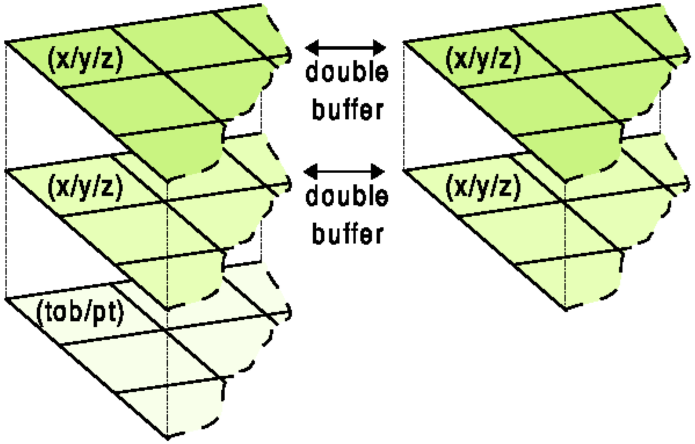
4) Size, Color, Opacity, etc.

- Simple attributes, can be added later, usually computed using the stateless method

Position texture

Velocity texture

Static per particle data,
e.g. time of birth (tob)
particle type (pt), ...



U.PORTO FC

birth and death

Birth = allocation of a particle

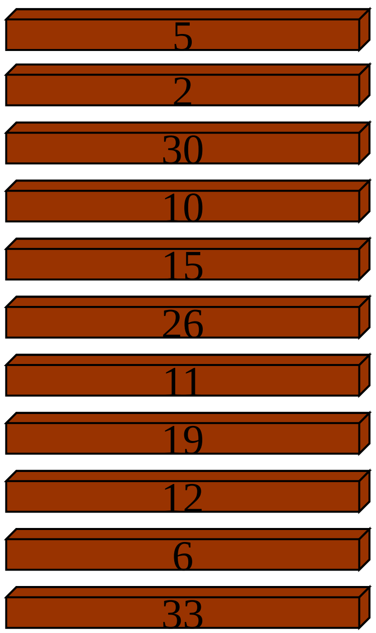
- Associate new data with an available index in the attributes textures
- Serial process – offloaded to CPU
- Initial particle data determined on CPU also

Death = deallocation of a particle

- Must be processed on CPU and GPU
  - CPU – frees the index associated with particle
  - GPU – extra pass to move any dead particles to unseen areas
    (i.e. infinity, or behind the camera)
  - In practice particles fade out or fall out of view
    (Clean-up rarely needs to be done)

allocation on CPU

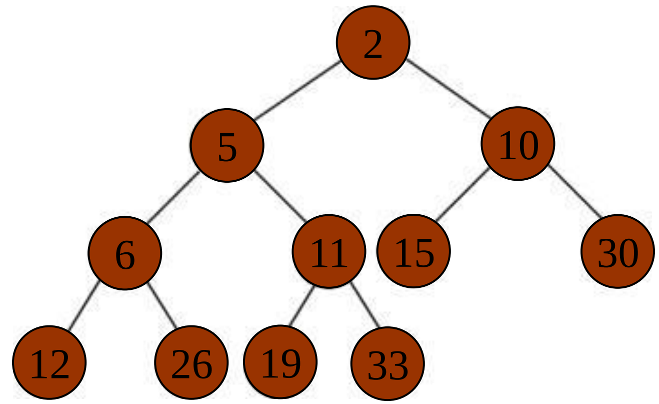
Stack



Easier!

Why?

Heap



Optimize heap to always return smallest available index

Better!

Update velocities

Velocity Operations

1) Global Forces

- Wind
- Gravity

2) Local Forces

- Attraction
- Repulsion

3) Velocity Damping

4) Collision Detection

$F = \Sigma f0 \ldots fn$
$F = ma$
$a = F/m$

If $m = 1$, then
$F = a$

Update velocities

Local Forces: flow field

Stokes Law of drag force on a sphere

$F_d = 6\Pi\eta r(v-v_{fl})$

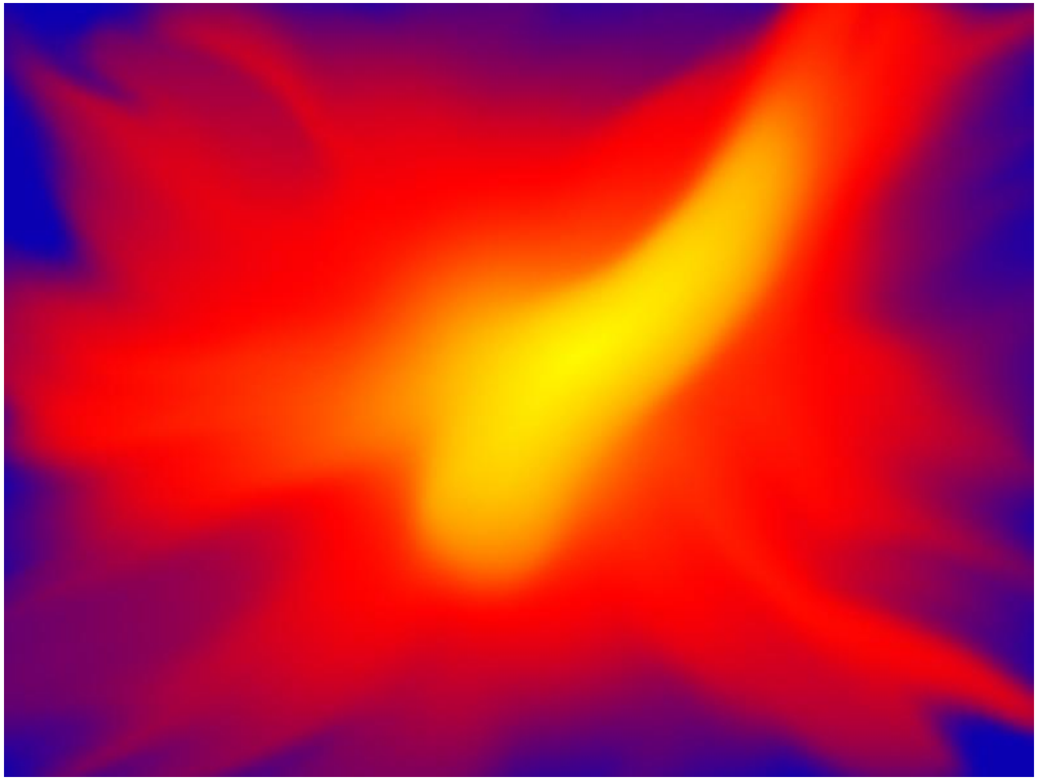
η = viscosity

r = radius of sphere

C = 6Πηr (constant)

v = particle velocity

v_{fl} = flow velocity



Sample Flow Field

Update velocities

damping

. Imitates viscous materials or air resistance

. Implement by downward scaling velocity

un-damping

. Self-propelled objects (bee swarms)

. Implement by upward scaling velocity

Update velocities

Collision against simple objects

. walls

. Bounding spheres

Collision against complex objects

. terrain

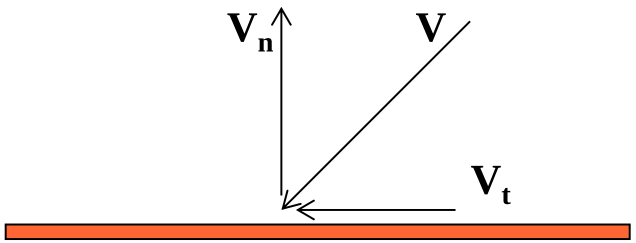
. complex objects (eg. 3D key)

. terrain is usually modeled as a texture-based height field

http://www.youtube.com/watch?v=W7tPTHV2mYk

Update velocities

compute collision reaction



$v_n = (v_{bc} * n)v_{bc}$
$v_t = v_{bc} – v_n$

v_{bc} = velocity before collision
v_n = normal component of velocity
v_t = tangental component of velocity

$$V = (1\text{-}\mu)v_t – \varepsilon v_n$$

μ = dynamic friction (affects tangent velocity)
ε = resilience (affects normal velocity)

Update position

Euler integration

$p = p_{prev} + v * \Delta t$

Verlet (simpler velocity updates)

$p_{i+1} = p_i + (p_i - p_{i-1}) + a * \Delta t^2$

numerical method used to integrate Newton's equations of motion.
used to calculate the trajectories of particles in real-time simulations.

Doesn't use the velocity!

calculates the position of the next time step from the position of the previous and current time steps

Summary

- Model volumes using particles instead of polygons
- Stateless vs. State particle systems
- Particle life-cycle
  - Generation, dynamics, extinction
- Be creative!