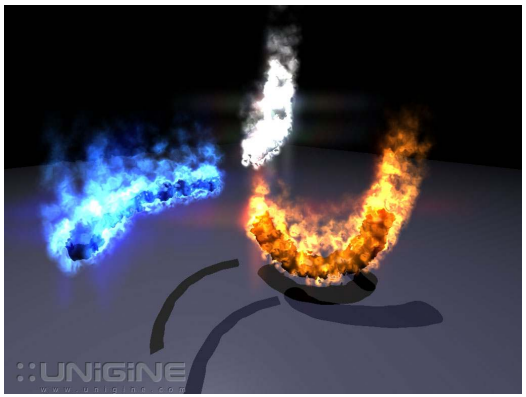
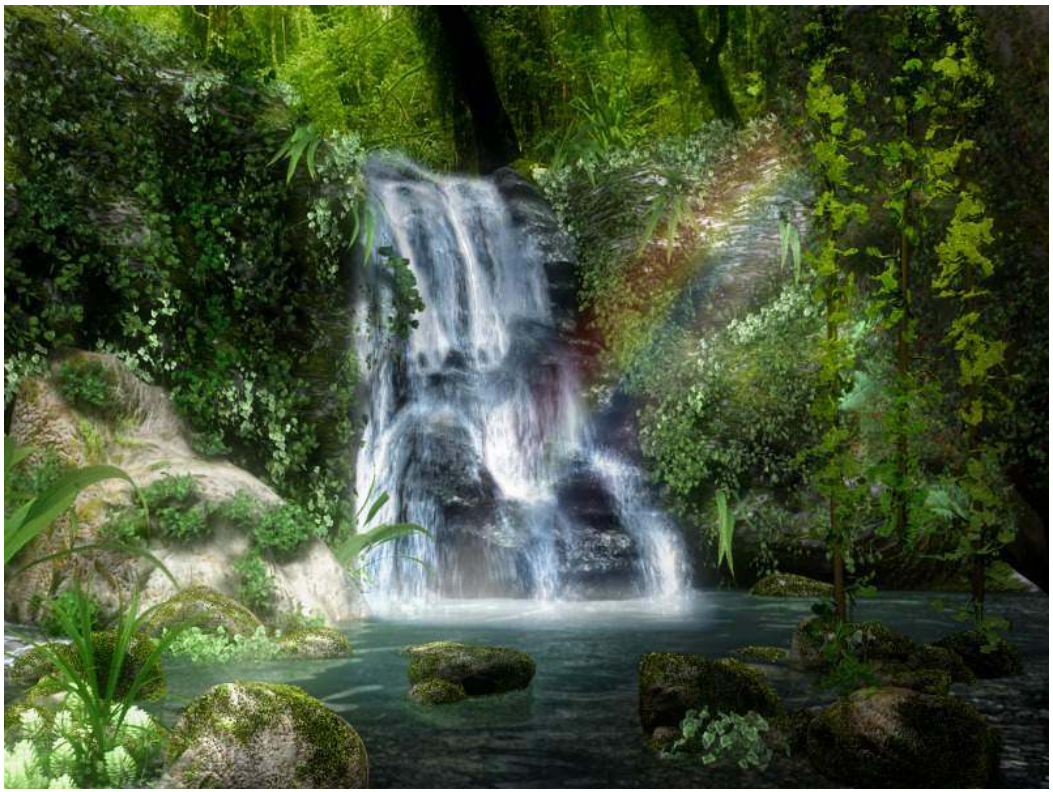
Virtual Reality & Physically-Based Simulation
Particle Systems

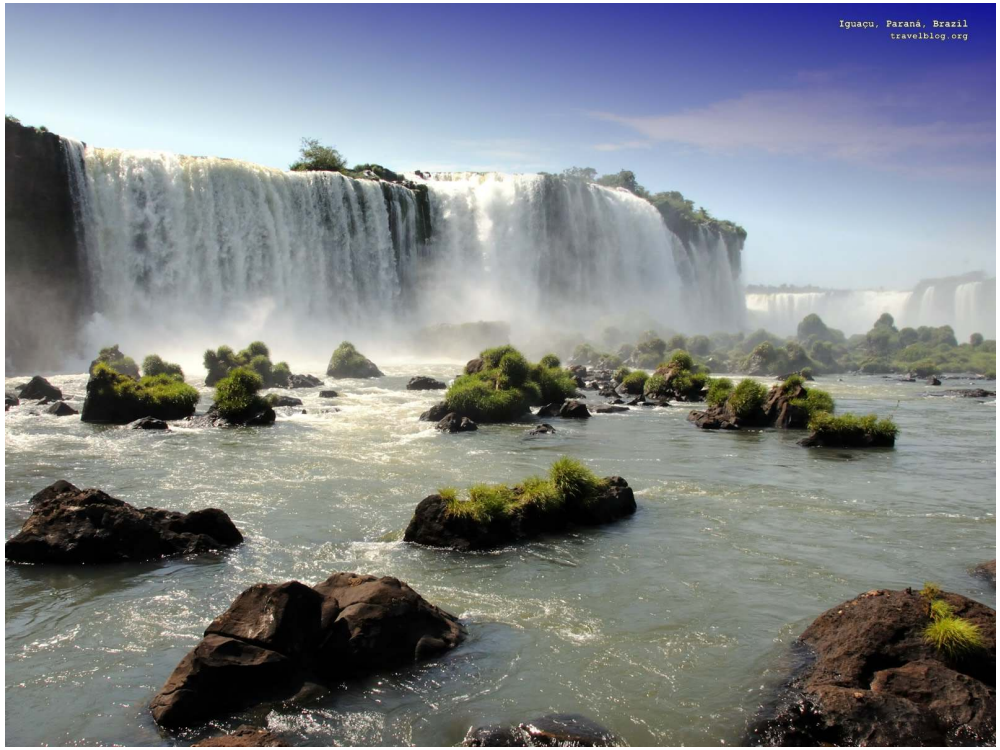


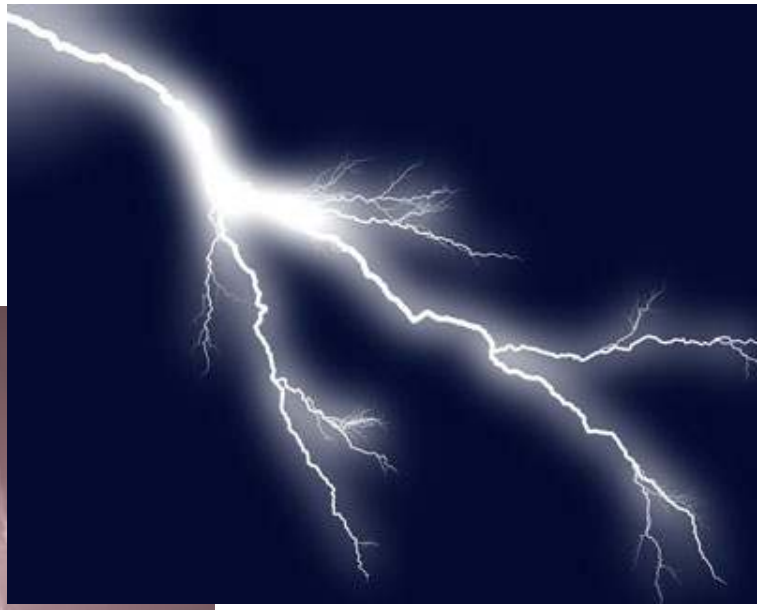
G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Obvious Systems of "Particles"

Particle Systems for Natural Phenomena

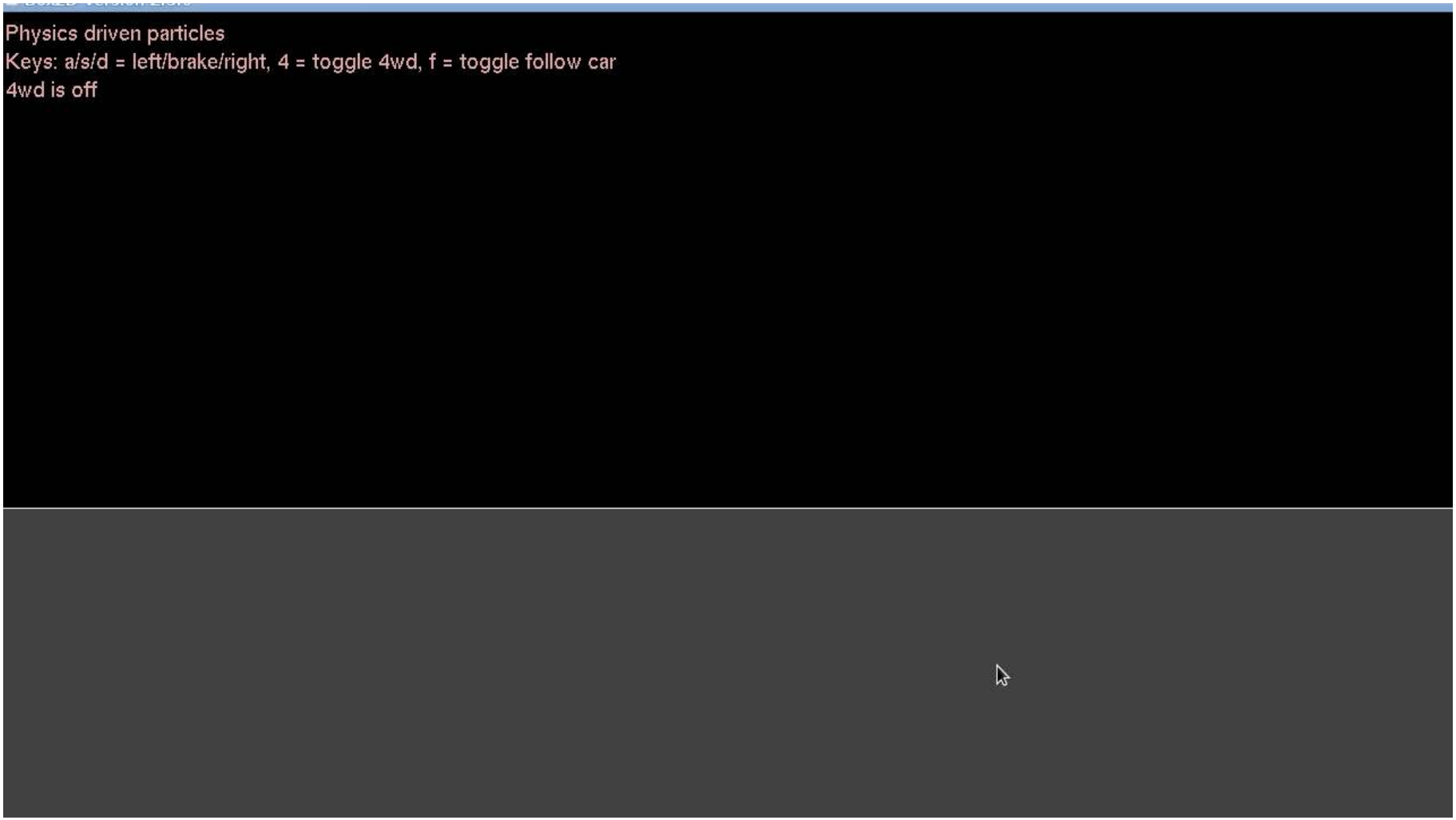



Iguaçu, Paraná, Brazil
travelblog.org

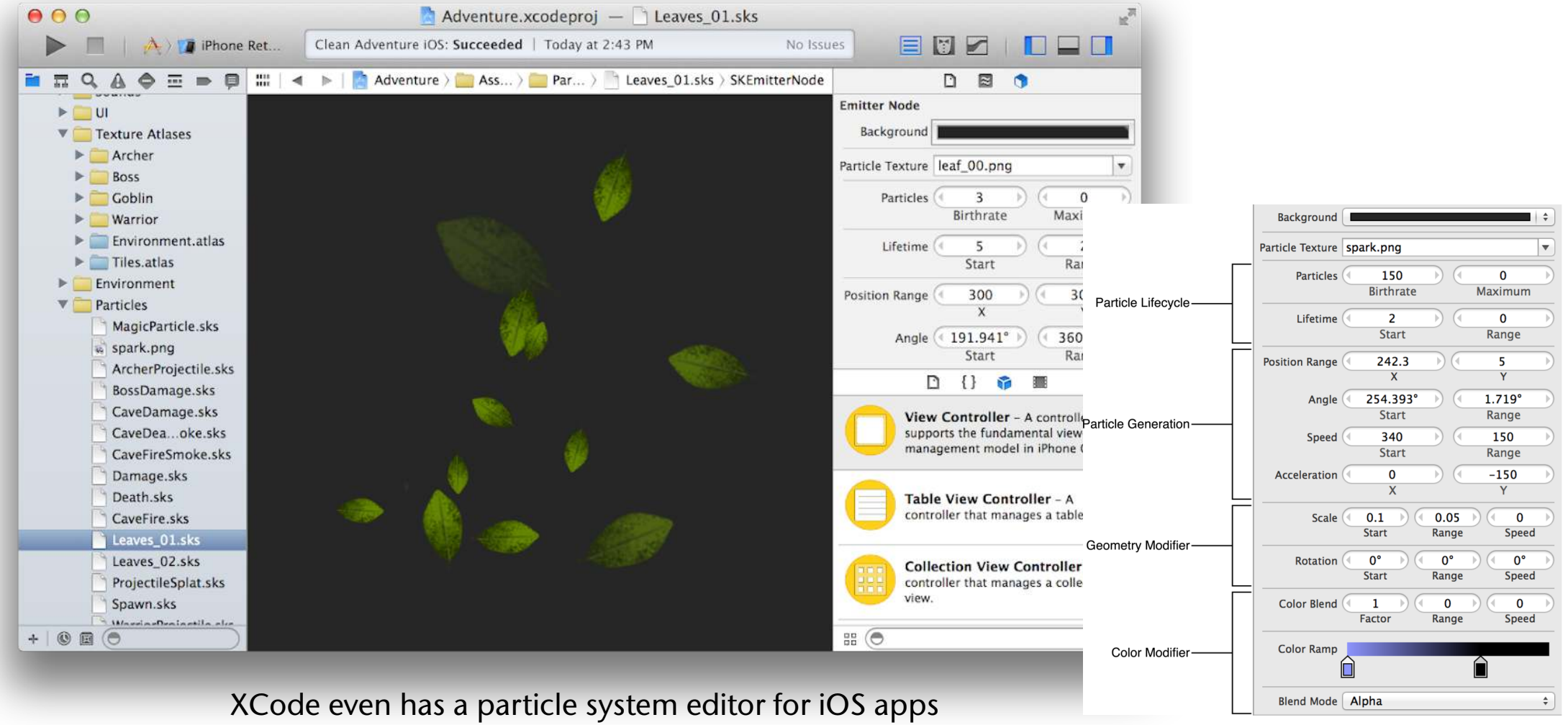




And Special Effects (in Games and Apps)

© UniGine


Physics driven particles
Keys: a/s/d = left/brake/right, 4 = toggle 4wd, f = toggle follow car
4wd is off

Particle Systems Have Become Ubiquitous

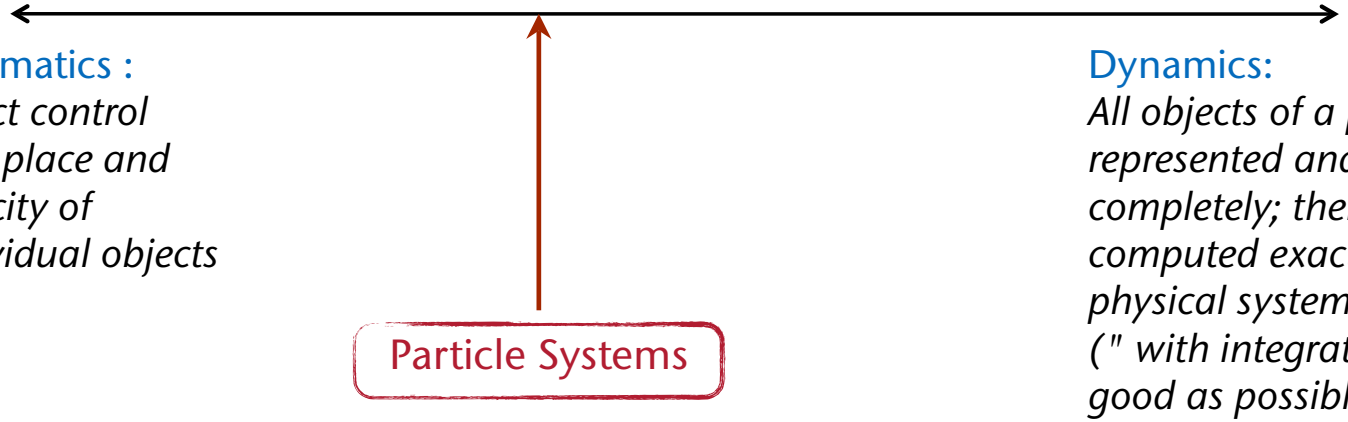


XCode even has a particle system editor for iOS apps

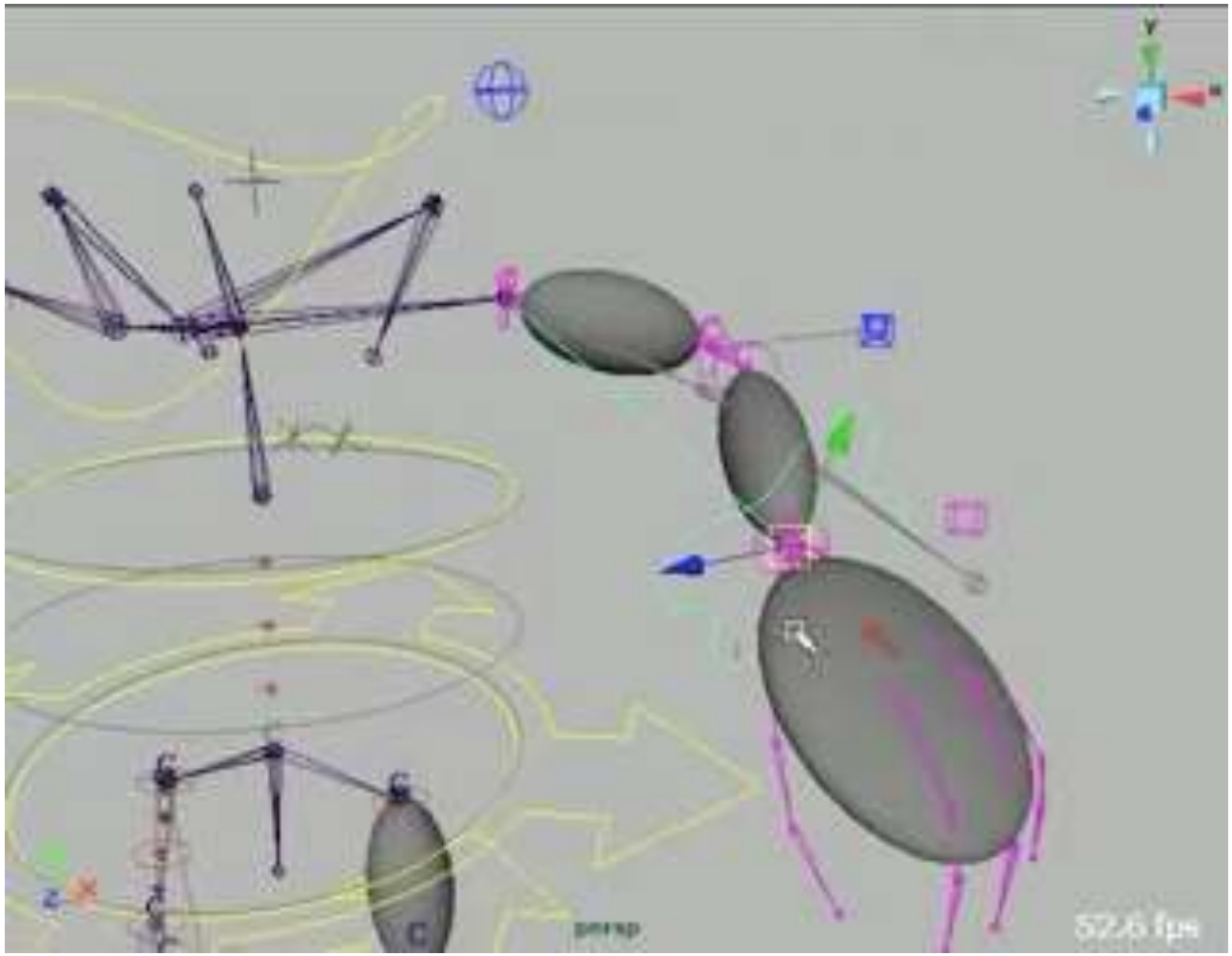
Kinematics vs Dynamics

- Kinematics = computation of the position & velocity of bodies over time without considering forces
- Dynamics = computation of forces & torques acting on bodies, and the motions of the objects resulting from them

- In computer graphics we always move within a continuum:



Example of Pure Kinematics: F*orward* & *Inverse Kinematics*

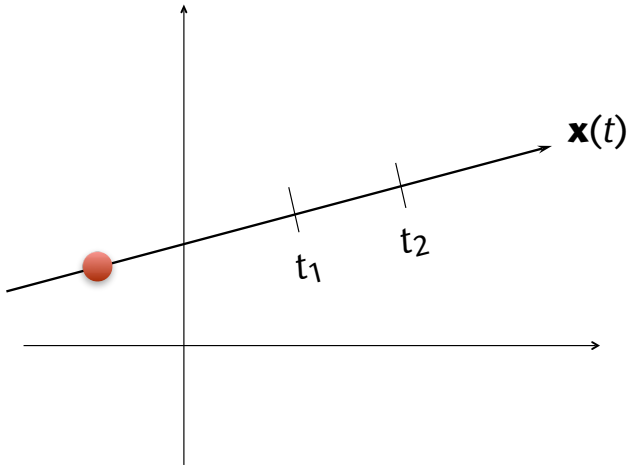


Dynamics of a Point Mass

- Definition Particle:
  A particle is an ideal point with a mass m and a velocity $\mathbf{v}$.
  It does *not* possess an orientation.
- Path of a particle = $\mathbf{x}(t)$
- Velocity:

$$\mathbf{v} = \frac{\text{distance}}{\text{time}} = \frac{\mathbf{x}(t_2) - \mathbf{x}(t_1)}{t_2 - t_1}$$

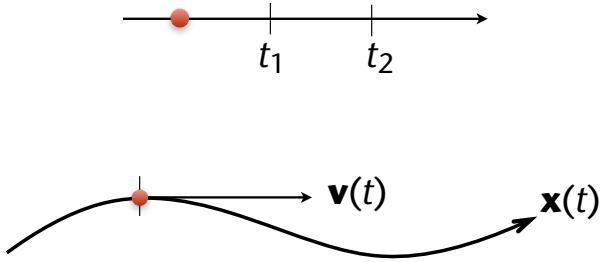
  - Unit: m/s
  - Note: velocity of particle = vector
    position of particle = point !



- The momentary velocity:

$$\mathbf{v}(t_1) = \lim_{t_2 \to t_1} \frac{\mathbf{x}(t_2) - \mathbf{x}(t_1)}{t_2 - t_1}$$

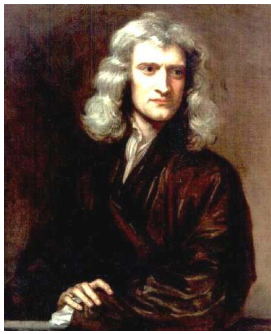
$$= \frac{d}{dt}\mathbf{x}(t_1) = \dot{\mathbf{x}}(t_1)$$



- Examples:
  - Point moves on a circular path → $\|\dot{\mathbf{x}}\|$ is constant
  - Point accelerates on a straight line → $\frac{\dot{\mathbf{x}}}{\|\dot{\mathbf{x}}\|}$ is constant
- Acceleration at some point in time : $\mathbf{a}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}(t) = \dot{\mathbf{v}}(t) = \frac{\mathbf{F}(t)}{m}$

Newtons Second Law

Newton's Laws



1. Law (law of inertia):

> A body, which no (unbalanced) forces act upon, continues to move with constant velocity.

- A resting body is just a special case of this law.

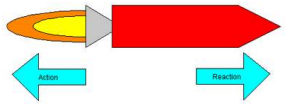
2. Law (law of action):

> If a force **F** acts on a body with mass m , then the body accelerates, and its acceleration is given by
>
> $$\mathbf{F} = m \cdot \mathbf{a}$$

- In other words: force and acceleration are proportional to each other; (the proportionality factor happens to be m). In particular, both force and acceleration have the same direction.

3. Law (law of reaction):

When one body exerts a force **F** on a second body,
the second body simultaneously exerts a force –**F** on the first body.



- In school, you learn: "action= reaction"

4. Law (law of superposition):

When a number of forces $\mathbf{F}_1, \ldots, \mathbf{F}_n$ act on a point or body, they can be accumulated by vector addition yielding one resulting force:

$$\mathbf{F} = \mathbf{F}_1 + \ldots + \mathbf{F}_n .$$

Historical Digression

- Newton published these laws in his original book

  Principia Mathematica

  (1687) like this:

  - *Lex I. Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus illud a viribus impressis cogitur statum suum mutare.*
  - *Lex II. Mutationem motus proportionalem esse vi motrici impressae, et fieri secundum lineam rectam qua vis illa imprimitur.*

[13]

AXIOMATA
SIVE
LEGES MOTUS

Lex. I.

Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus a viribus impressis cogitur statum illum mutare.

Projectilia perseverant in motibus suis nisi quatenus a resistentia aeris retardantur & vi gravitatis impelluntur deorsum. Trochus, cujus partes cohærendo perpetuo retrahunt sese a motibus rectilineis, non cessat rotari nisi quatenus ab aere retardatur. Majora autem Planetarum & Cometarum corpora motus suos & progressivos & circulares in spatiis minus resistentibus facilius conservant diutius.

Lex. II.

Mutationem motus proportionalem esse vi motrici impressæ, & fieri secundum lineam rectam qua vis illa imprimitur.

Si vis aliqua motum quemvis generet, dupla duplum, tripla triplum generabit, sive simul & semel, sive gradatim & successive impressa fuerit. Et hic motus quoniam in eandem semper plagam cum vi generatrice determinatur, si corpus antea movebatur, motui ejus vel conspiranti additur, vel contrario subducitur, vel obliquo obliquæ adjicitur, & cum eo secundum utriusq; determinationem componitur.

Lex. III.

Euler Integration

- Given:
  a particle of mass m; and a force $\mathbf{F}(t)$ that acts on the particle over time
- Sought: the path $\mathbf{x}(t)$ of the particle
- The analytical approach:

$$\mathbf{v}(t) = \mathbf{v}_0 + \int_{t_0}^{t} \mathbf{a}(t)\,\mathrm{d}t$$

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{v}(t)\,\mathrm{d}t$$

- Discretization and linearization yields:

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \mathbf{a}^t \cdot \Delta t$$

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

or

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \frac{\mathbf{v}^t + \mathbf{v}^{t+1}}{2} \Delta t$$

(*approximate midpoint method*)

Example

- Your car is out of gas, and you have your friends push it
- They want to make it move by

$$0.05\,\frac{\mathrm{m}}{\mathrm{s}}$$

- Question: what is the force they need to apply to the car?
- Solution:

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

$$v(1\,\mathrm{s}) = 0\,\frac{\mathrm{m}}{\mathrm{s}} + 1\,\mathrm{s}\cdot a(0) \overset{!}{=} 0.05\,\frac{\mathrm{m}}{\mathrm{s}}$$

$$a(0) = 0.05\,\frac{\mathrm{m}}{\mathrm{s}^2}$$

$$F = m \cdot a = 0.05\,\frac{\mathrm{m}}{\mathrm{s}^2}\cdot 1000\,\mathrm{kg} = 50\,\mathrm{N}$$

(Assuming the acceleration is constant in the time interval [t, $t+\Delta t$])

The Phase Space

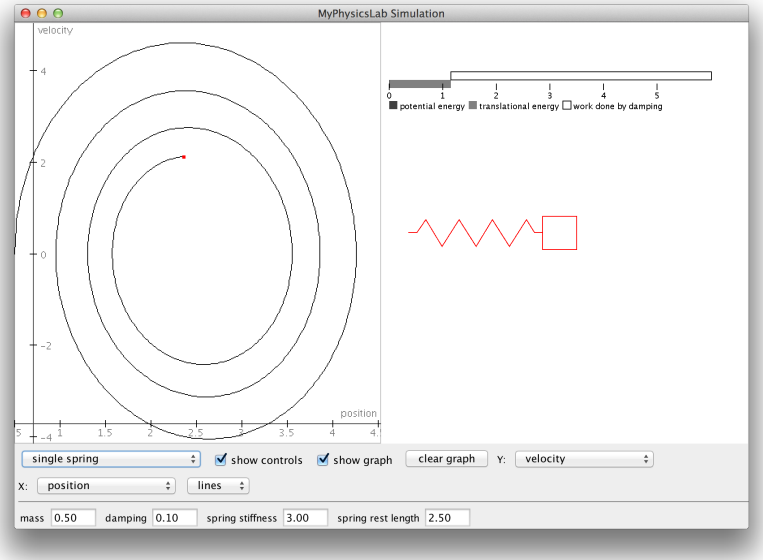
- The (physical) momentary state of a particle is described completely by

$$\mathbf{q} = (\mathbf{x}, \mathbf{v}) = (x_1, x_2, x_3, v_1, v_2, v_3) = (x_1, x_2, x_3, \dot{x}_1, \dot{x}_2, \dot{x}_3) \in \mathbb{R}^6$$

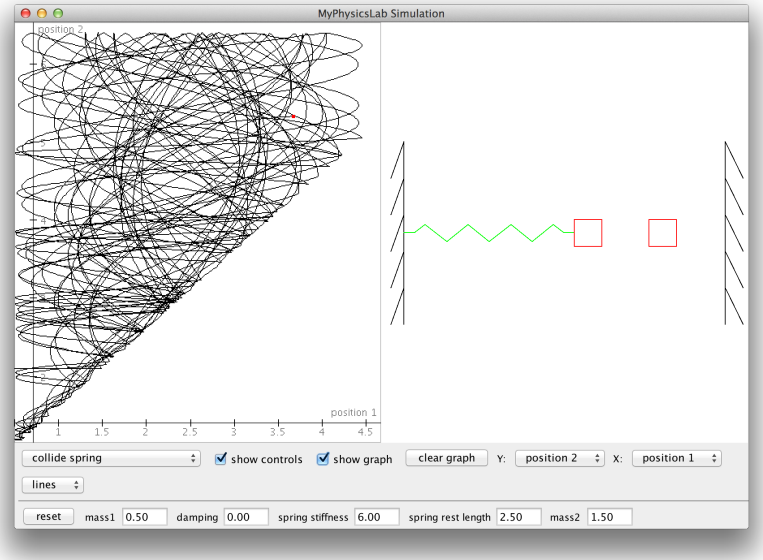
- The space of all possible states is called phase space
  - The dimension is $6n$, n = number of particles
- The motion of a particle in phase space:

$$\dot{\mathbf{q}} = (\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{v}_1, \dot{v}_2, \dot{v}_3) = \left(v_1, v_2, v_3, \frac{f_1}{m}, \frac{f_2}{m}, \frac{f_3}{m} \right)$$

- Example for a particle that can move only along the X axis and that is held in a resting position by a spring:



Single spring



"Collide spring"

www.myphysicslab.com

Particle Systems

- Definition: a particle system is comprised of
  1. A set of particles; each particle i has, at least, the following attributes:
     - Mass, position, velocity (m_i , $\mathbf{x}_i$, $\mathbf{v}_i$)
     - Age a_i
     - Force accumulator $\mathbf{F}_i$
     - Optional: color, transparency, optical size, lifespan, type, ...
  2. A set of particle sources; each one is described by
     - Shape of the particle source
     - Stochastic processes that determine the initial attributes of the particles, e.g., velocity, direction, etc.
     - Stochastic processes that determine the number of particles created per frame
  3. Other (global) parameters, e.g.
     - TTL (time to live) = max. lifespan of particles
     - Global forces, e.g. gravitation, wind, ...
     - The Algorithms, that move and renderer of particles

- Stochastic process:
  - Simplest case: average + variance; process outputs random values according to uniform distribution
  - A bit more complex: average and variance functions over time

- Remark on the geometric representation of particle sources:
  - Just an intuitive way to describe the stochastic process for the initial position of particles
  - Frequent forms: disk, cube, cone, etc.

The Execution Model

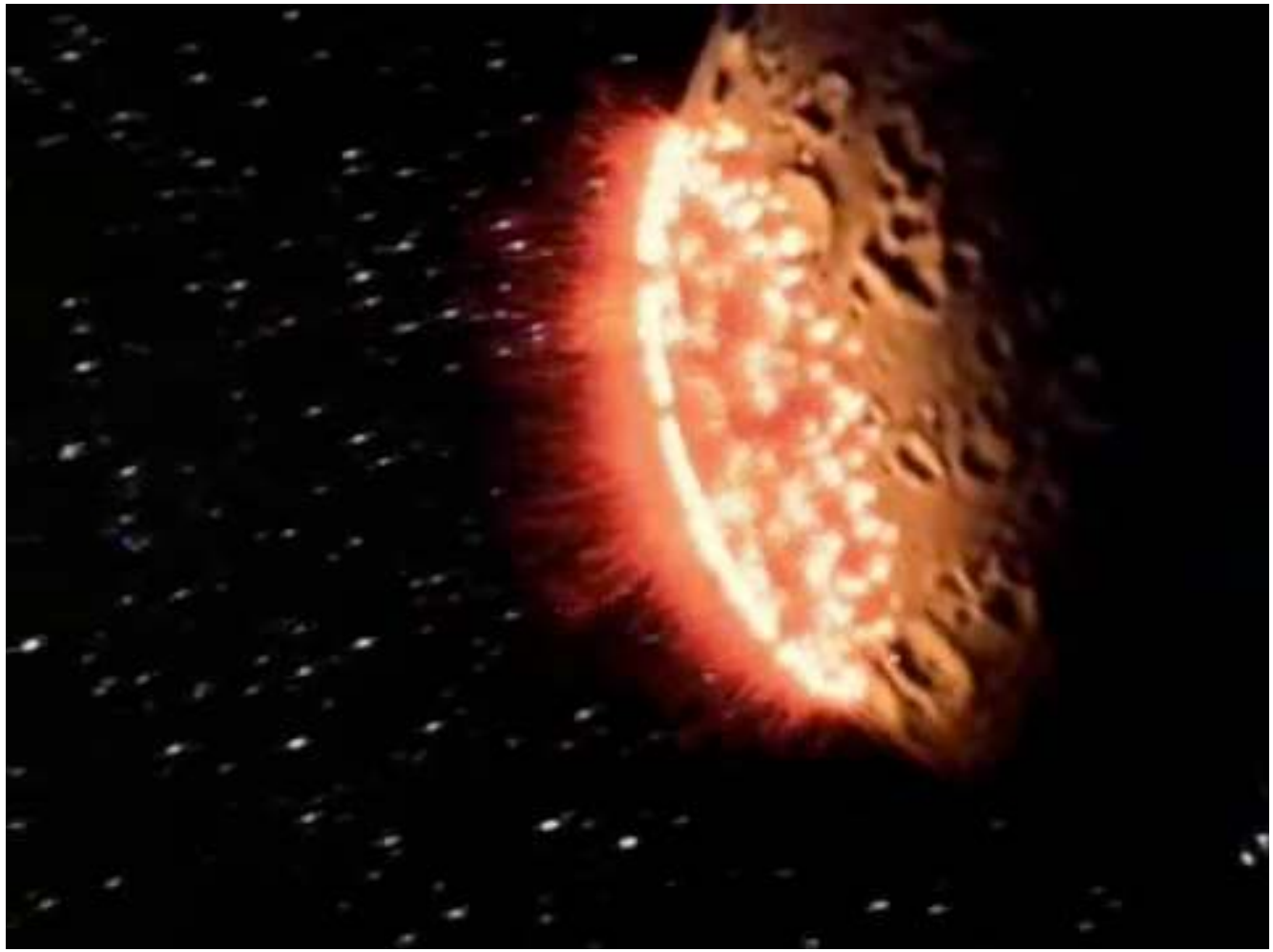
- The "main loop" of a particle system:

```
loop forever:
    render all particles
     sort all particles by depth in case of alpha blending
    Δt := rendering time
    kill all particles with age > TTL (max. life-span)
    create new particles at particle source
    reset all force accumulators
    compute all forces on each particle (accumulate them)
    compute new velocities (one Euler step with Δt)
    optionally modify velocities                          (*)
    compute new positions (another Euler step)
    optionally modify positions (e.g. b/c of constraints)
```

Remarks on the Implementation of Particle Systems

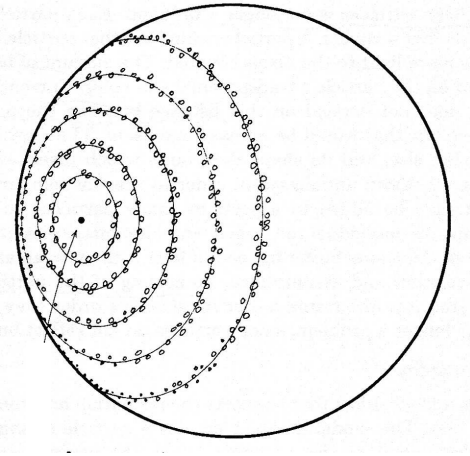
- There is lots of space for optimizations, e.g.
  - Initialize force accumulators with gravitational force
  - Don't increment the age of each particle "by hand"; instead, save the time of their creation in t_{gen} , then just test $t_{\text{current}} - t_{\text{gen}} > \text{TTL}$
    - Will be important for parallel implementation later
- Remark on (*) in the algorithm:
  - This is "non-physical", but allows for better kinematic control by the programmer
  - This is also necessary in case of collisions
- Often, we store a small history of the positions of particles, in order to create simple "motion blur" effects
- Particles can be killed by other constraints, too, e.g. distance from the source, entrance into a specific region, etc.
- For an efficient implementation, a "struct-of-array" data structure can be better! (SoA instead of AoS)

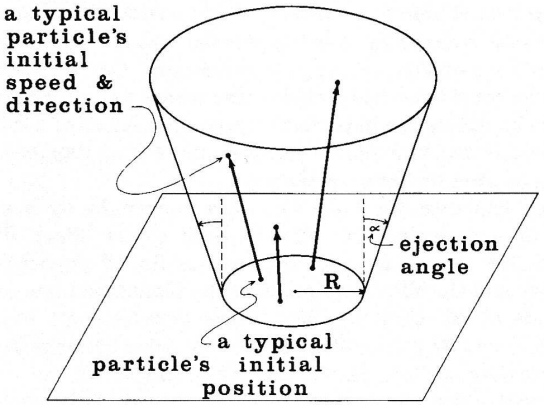
Example of a Particle System



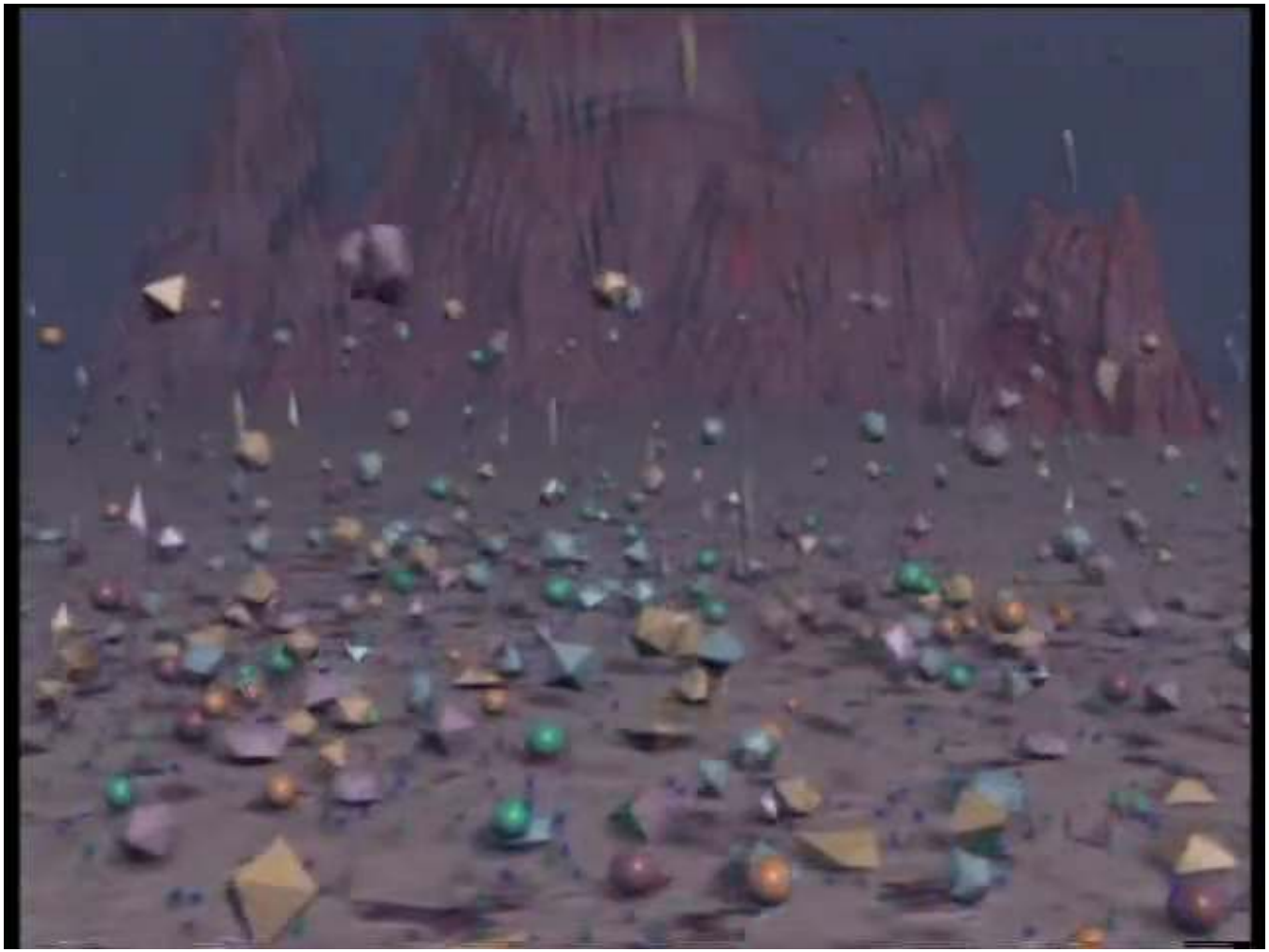
Excerpt of "Wrath of Khan" (Loren Carpenter, William Reeves, Alvy Ray Smith, et al., 1982)

- Particle source:
  - Circle on a sphere around the *point of impact*
  - Circle's radius increases over time
- Stochastic processes for particle creation:
  - Place many disks on big circle, create a few random positions within each disk
  - For each particle, create random velocity vector within a cone normal to surface of sphere
  - Some variance for each particle's lifespan
- Color of particle = *f*(*age*)





Digression: the Panspermia Hypothesis



Karl Sims, 1990

Is the Future Pre-Determined? Do We Have a Free Will?

https://www.menti.com/ibea4p4bvh

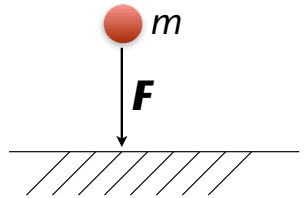
Philosophical Digression: Laplace's Demon [1814]

- *Causal/scientific determinism* postulates: if someone (the demon) knows the precise location and momentum of every atom in the universe, their future values for any given time can be calculated from the laws of classical mechanics.
- Consequences:
  - Everything would be pre-determined
  - There would be no free will
- Fortunately, such a demon is impossible

Physical Effects on Particles

- Gravity:

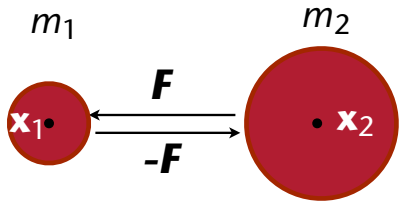
$$\mathbf{F} = m \cdot \mathbf{g} \quad , \qquad g = 9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$$



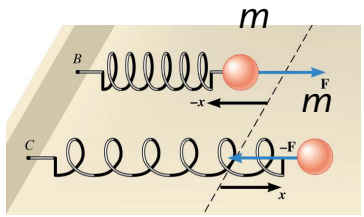
- Gravitation:

$$\mathbf{F} = G \cdot \frac{m_1 m_2}{r^2} \cdot \frac{\mathbf{x}_1 - \mathbf{x}_2}{|\mathbf{x}_1 - \mathbf{x}_2|}$$

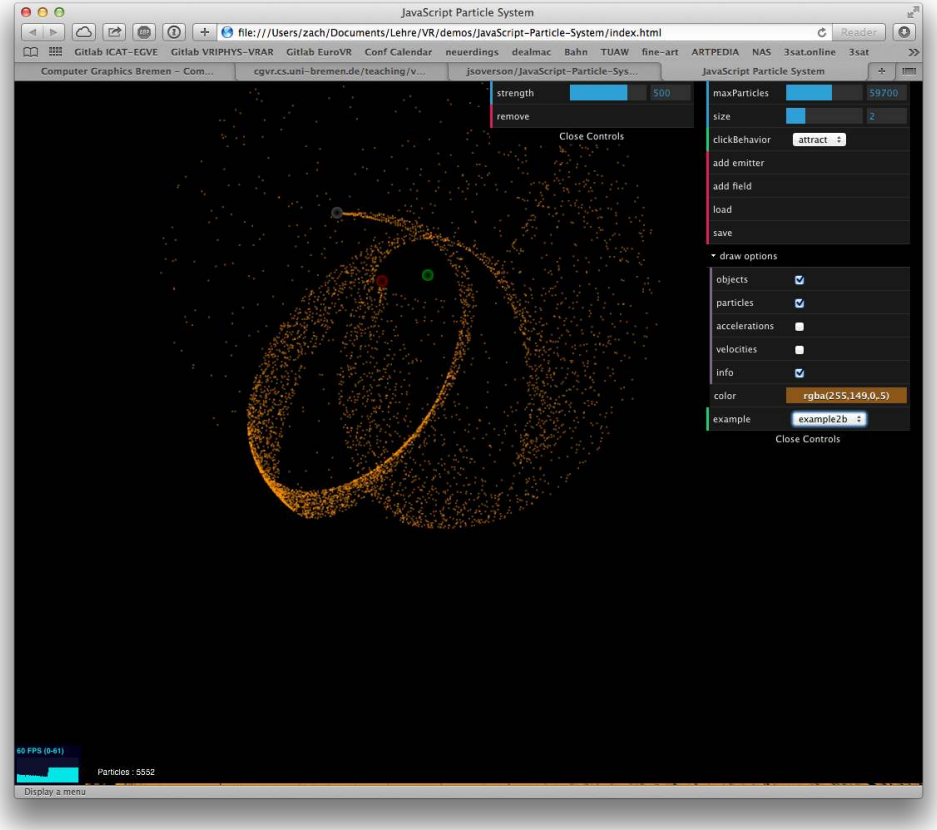
$$G = 6,67 \cdot 10^{-11}$$



- Spring force: later



A Simple Demo



http://jarrodoverson.com/static/demos/particleSystem

- Viscous drag (viskose Hemmung/Dämpfung):

$$\mathbf{F} = \text{-b } \mathbf{v}$$

in a stationary fluid/gas;

or, sometimes,

$$\mathbf{F} = 6\pi\eta r(\mathbf{v} - \mathbf{v}_{fl})$$

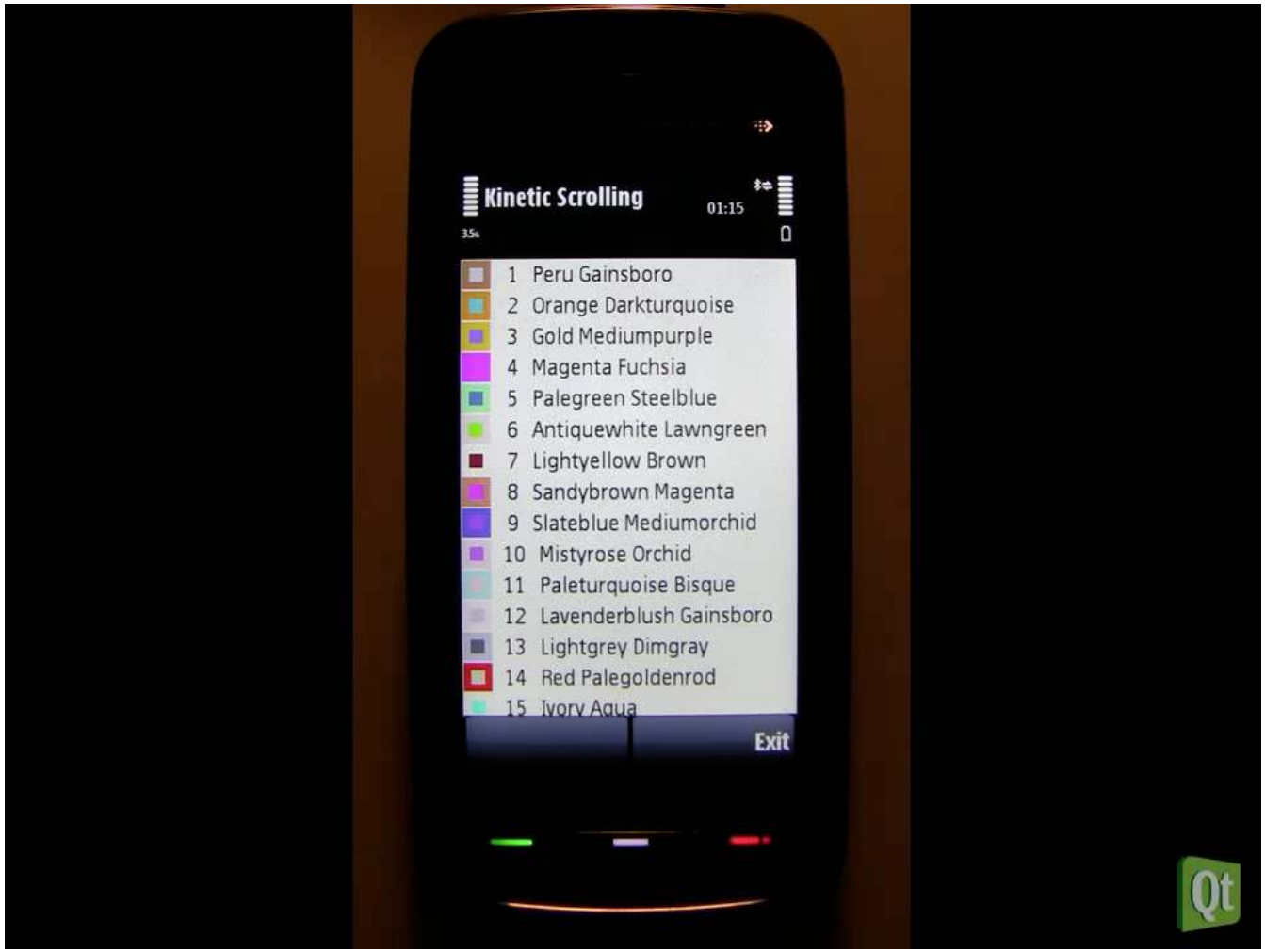
in fluids having velocity $\mathbf{v}_{fl}$ themselves, particles with radius r, viscosity η;

or, sometimes, with high velocities

$$\mathbf{F} = -\frac{1}{2}c\rho A\mathbf{v}^2$$

ρ = density, A = size of cross-sectional area, c = viscosity constant

Example: Kinetic GUI's



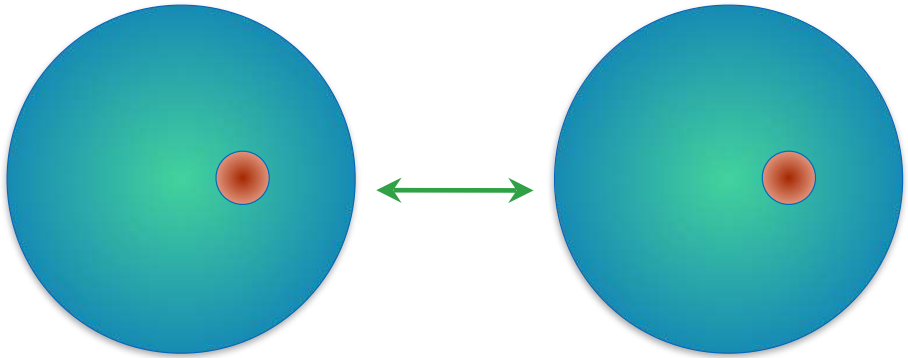
- Electromagnetic force (Lorentz force):

$$\mathbf{F} = q \cdot \mathbf{v} \times \mathbf{B}$$

where
q = charge of particle ,
$\mathbf{v}$ = velocity of particle,
$\mathbf{B}$ = magnetic field

The Lennard-Jones Force

- There are two kinds of forces between atoms:
  - A repelling force on short distances
  - An attracting force on mid-range distances (called *van-der-Waals* force or dispersion force)



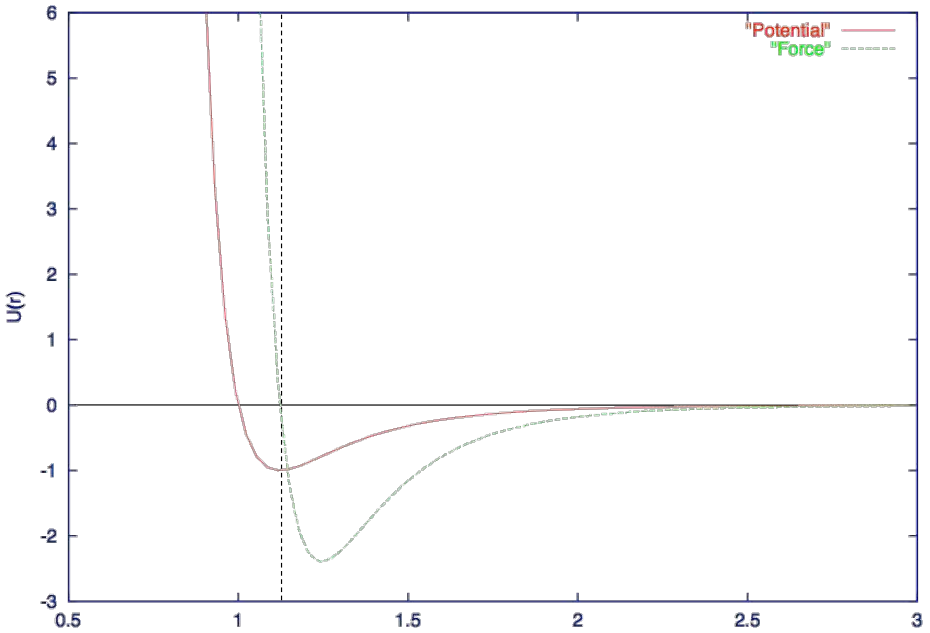
Instantaneous dipole Induced dipole

- One (arbitrary) approximation of the Lennard-Jones force:

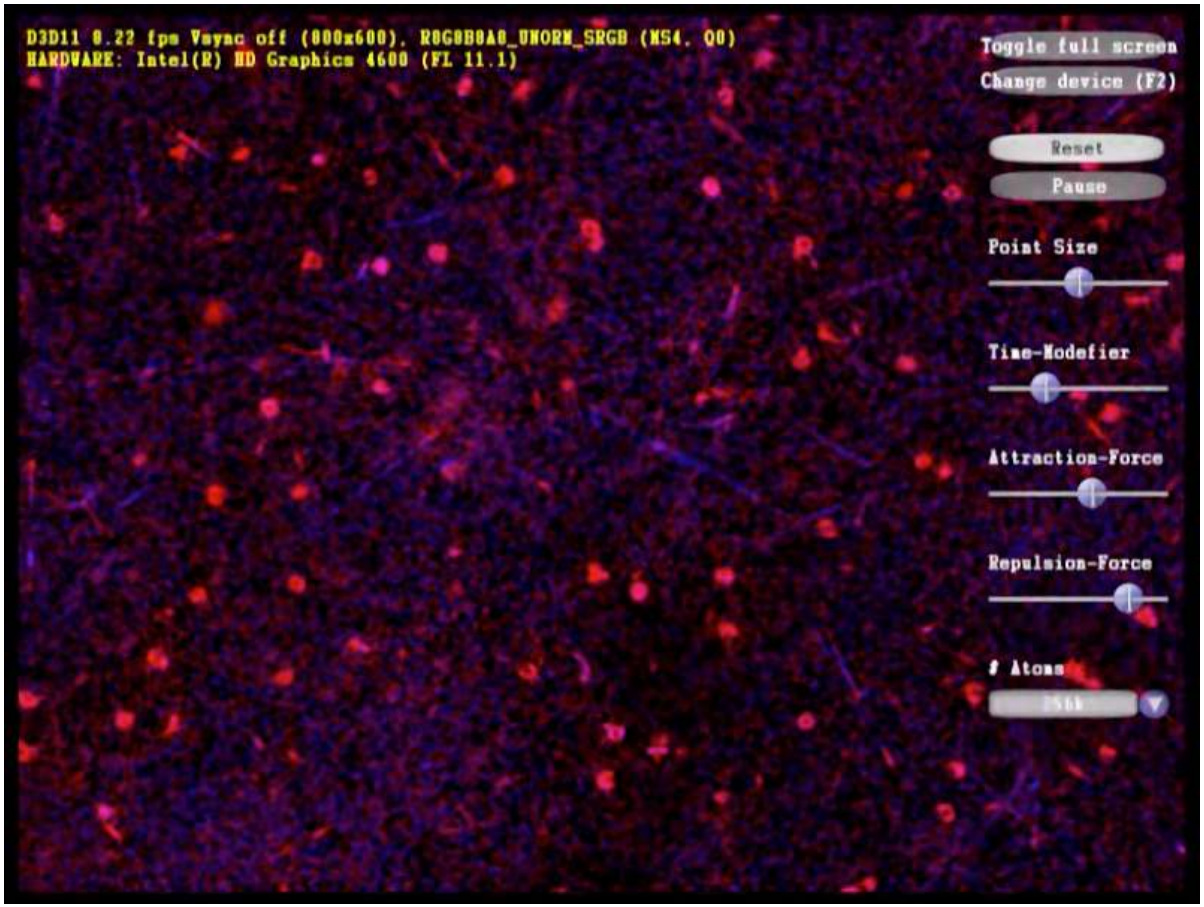
$$\mathbf{F} = \varepsilon \cdot \left(c \left(\frac{\sigma}{d} \right)^m - \left(\frac{\sigma}{d} \right)^n \right) \cdot \frac{\mathbf{x}_1 - \mathbf{x}_2}{\|\mathbf{x}_1 - \mathbf{x}_2\|}$$

where $d = \|\mathbf{x}_1 - \mathbf{x}_2\|$

and ε, c, m, n are arbitrary constants (for our purposes)



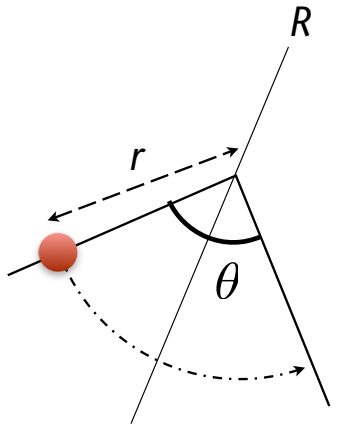
Lennard-Jones Demo (256,000 particles, forces between , 50 FPS)



Student project – anyone up for implementing it on the GPU and/or improving it?

Non-Physical Effects on Particles

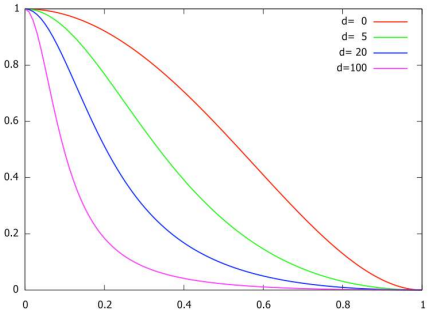
- Vortex: rotate particle about axis R with angle $\theta = a \cdot f(r)$, where a = "force" of the vortex, r = distance particle$\leftrightarrow$axis, and



or

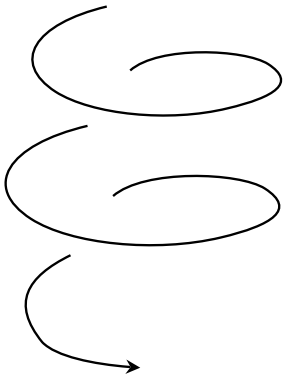
$$f(r) = \frac{1}{r^{\alpha}}$$

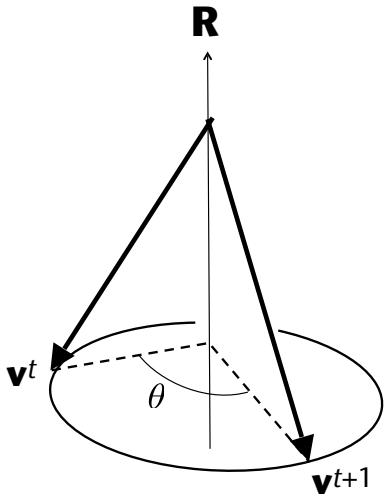
$$f(r) = \begin{cases} \frac{r^4 - 2r^2 + 1}{1 + dr^2} & , \ r \leq 1 \\ 0 & , \ r > 1 \end{cases}$$



- Extensions:
  - Take mass of particle into account
  - Use B-spline as axis of the vortex (e.g., for a tornado)
  - Animate the axis of the vortex

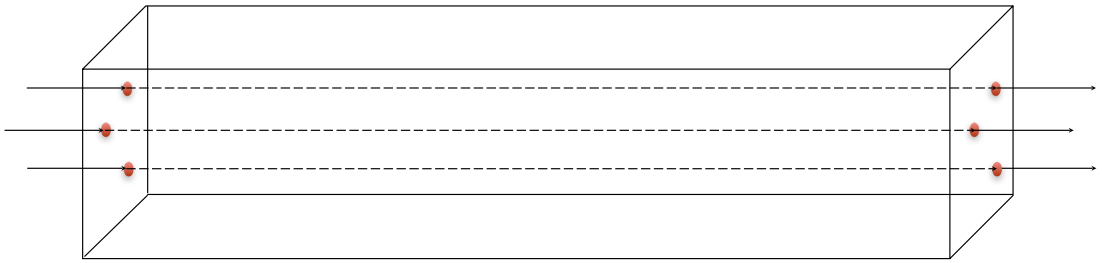
- Simple idea to make each particle move on its own individual spiraling path (e.g., for snow or smoke):
  - Rotate velocity **v** (instead of the position) about an angle $\theta = \sigma \cdot \Delta t$
  - Again, σ can/should vary slightly and randomly, dito the rotation axis
  - Also, σ and R can be animated over time





Positional Constraints on Particles

- Rather rare, e.g. “tunneling”



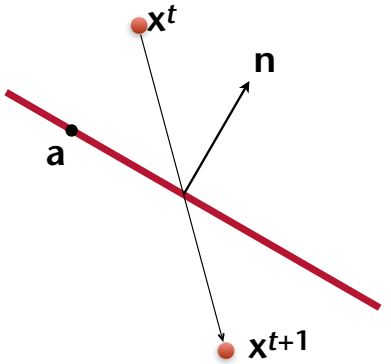
  - Mostly done for collision handling

Collisions

- Most important kind of geometric constraint
- First, consider a collision with a plane

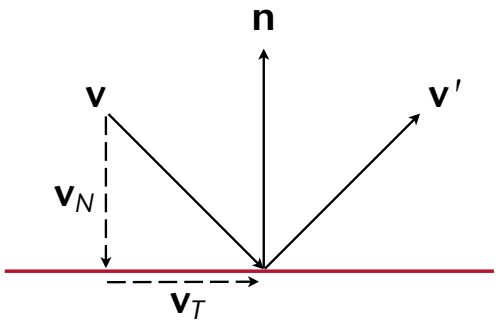
1. Collision check:

$$(\mathbf{x}^t - \mathbf{a})\,\mathbf{n} > 0 \;\wedge\; (\mathbf{x}^{t+1} - \mathbf{a})\,\mathbf{n} < 0$$



2. Collision handling: reflect **v** around the normal

$$\mathbf{v}_N = (\mathbf{v}\cdot\mathbf{n})\,\mathbf{n}$$
$$\mathbf{v}_T = \mathbf{v} - \mathbf{v}_N$$
$$\mathbf{v}' = \mathbf{v}_T - \mathbf{v}_N = \mathbf{v} - 2(\mathbf{v}\cdot\mathbf{n})\,\mathbf{n}$$

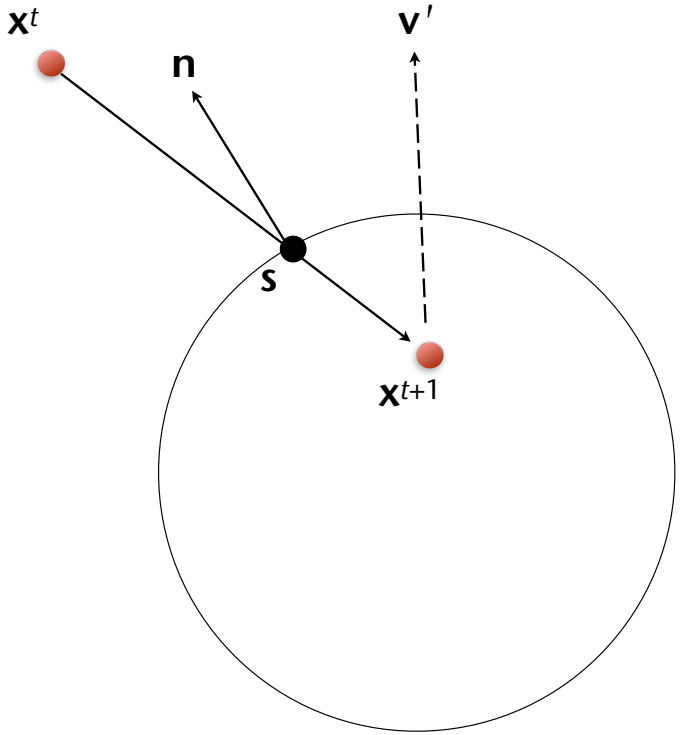


- Extension to friction and elastic/inelastic impact:

$$\mathbf{v}' = (1-\mu)\,\mathbf{v}_T - \varepsilon\mathbf{v}_N$$

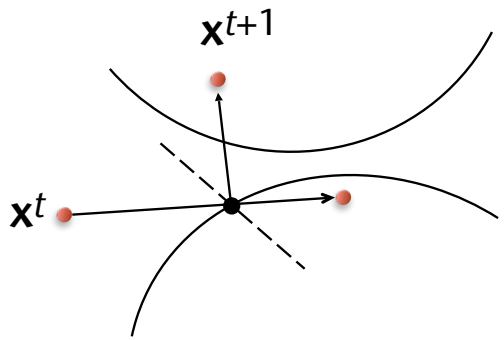
with μ = friction parameter and ε = resilience (Federung / Elastizität)

- Collision with a sphere:
  - Compute exact intersection of $\mathbf{x}^t\mathbf{x}^{t+1}$ with sphere $\longrightarrow \mathbf{s}$ (if any)
  - Determine normal **n** at point **s**
  - Then reflect **v**, as before
- Conclusion: collision detection for particles = "point inside geometry test", or, more precisely: intersection test between line segment and geometry

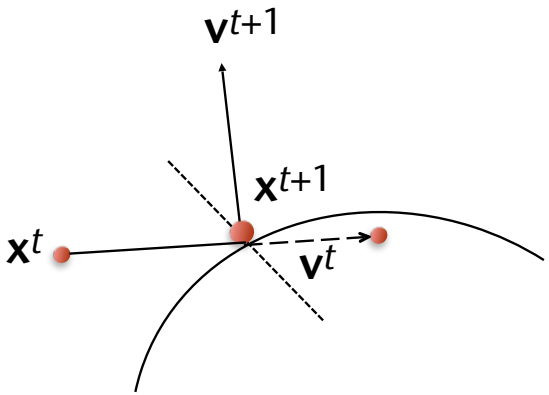


(For intersection tests with polyhedra and terrain: see "Computer Graphics 1". For implicit surfaces: see "Advanced Computer Graphics")

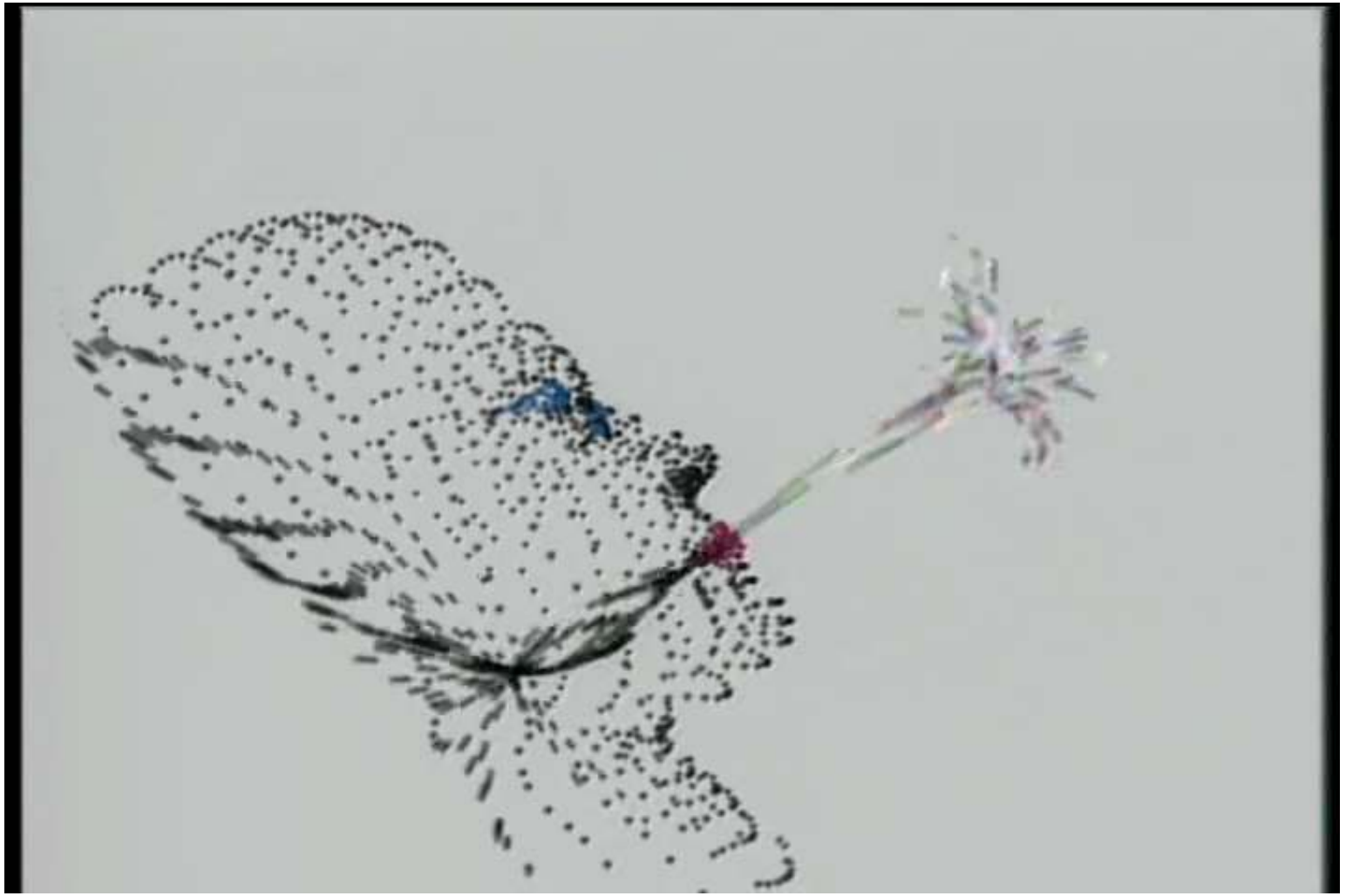
- Challenge: always create a *consistent* system after the collision handling!
- Problem: "double collisions" at narrow places
  - Example:



  - Better handling:



  - There are more ways to handle these kinds of situations ...

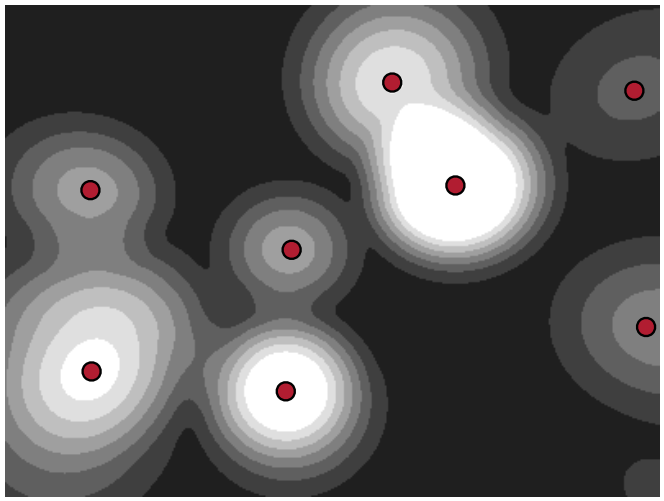


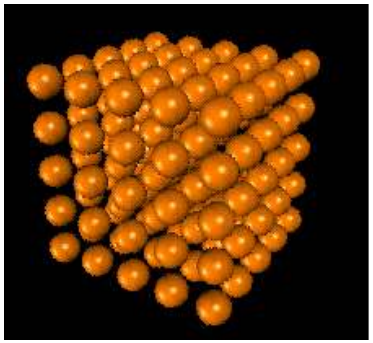
Karl Sims: Particle Dreams

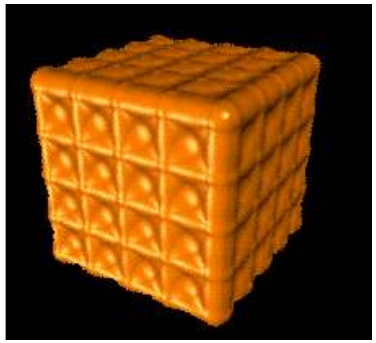
https://www.youtube.com/watch?v=t2an3xMuiew

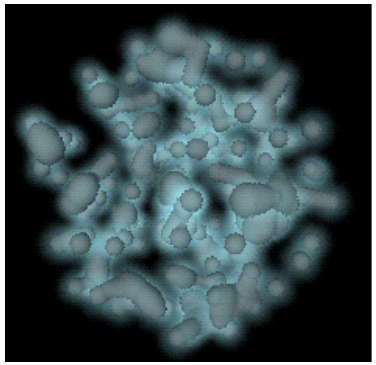
Rendering Particles as "Blobby Objects"

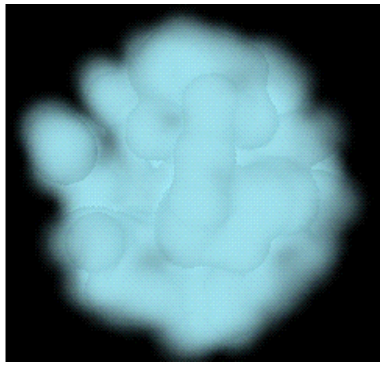
- Regard particles as metaballs
  - See "Advanced CG": *metaballs* = potential fields that blend together to form (implicit) surfaces
  - Render using ray-casting
  - Either: find the root of the potential fct along ray
  - Or: accumulate the "densities" along the ray and interpret this as opacity or as luminance



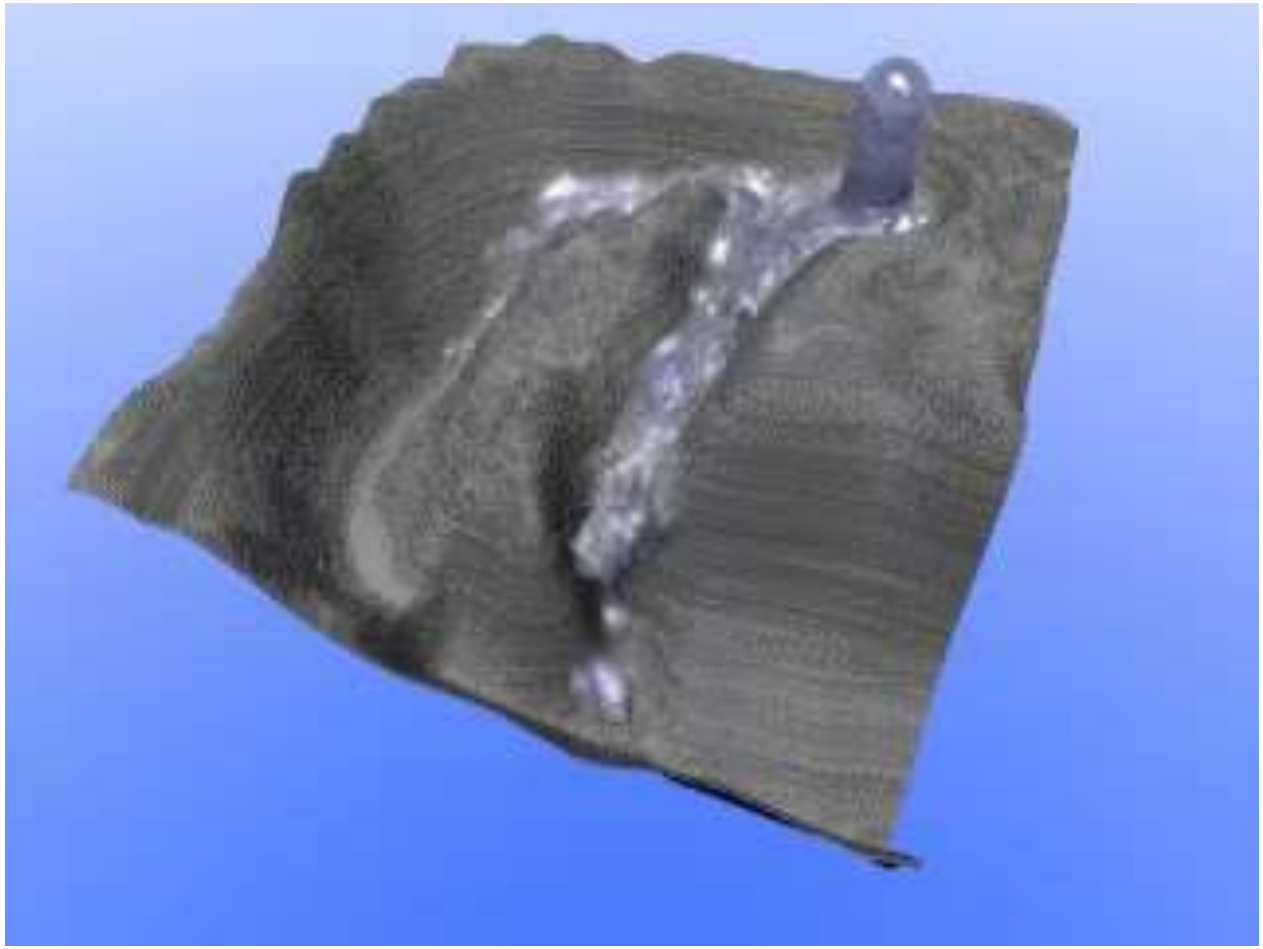






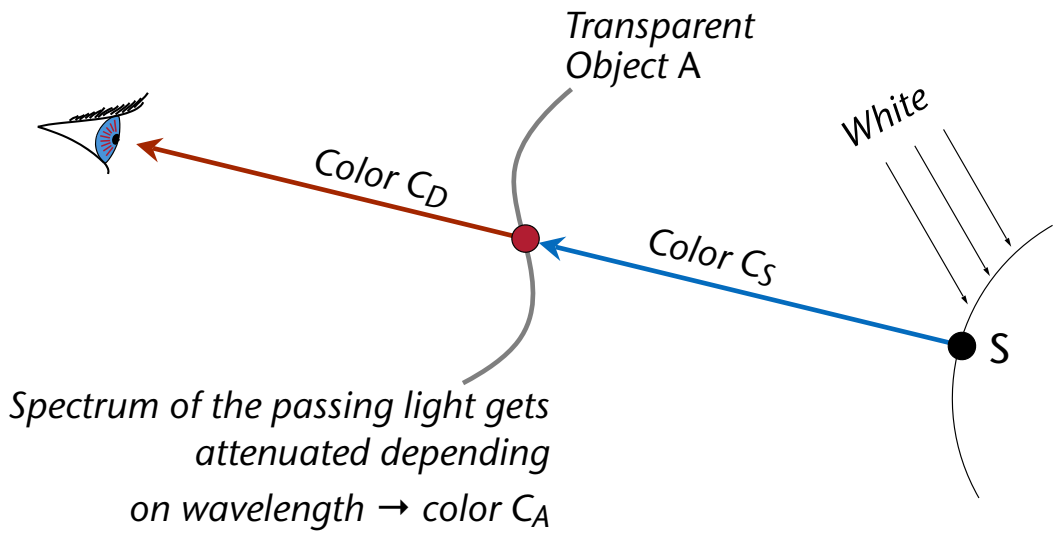


Example



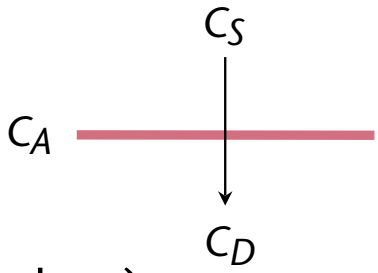
Rendering Transparent Particles

- For example: water, fire, smoke, etc., can be modeled using particle systems
- Transparency ≈ material that lets light pass partially
- Often, some wavelengths are attenuated differently → colored transparency
  - Extreme case: color filter in photography



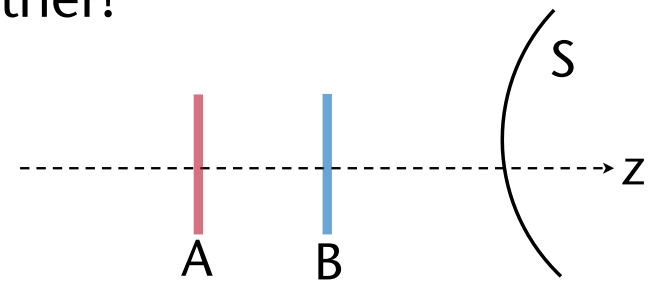
- Approximation: alpha blending
  - $\alpha \in [0, 1]$ = opacity (= opposite of transparency)
    - $\alpha = 0 \longrightarrow$ completely transparent, $\alpha = 1 \longrightarrow$ completely opaque
  - Outgoing color:

$$C_D = \alpha C_A + (1 - \alpha) C_S$$



  - Practical implementation: $\alpha = 4^{th}$ component in color vectors C = (r, g, b, α)
- During rendering, the graphics card performs these fragment operations:
  1. Read color from frame buffer $\rightarrow C_S$
  2. Compute C_D by above equation
  3. Write C_D into framebuffer

- Problem: several transparent objects behind each other!
  - Assume Z-buffer is off, i.e., no Z-test is done



1. Rendering first *A* then *B* results in:

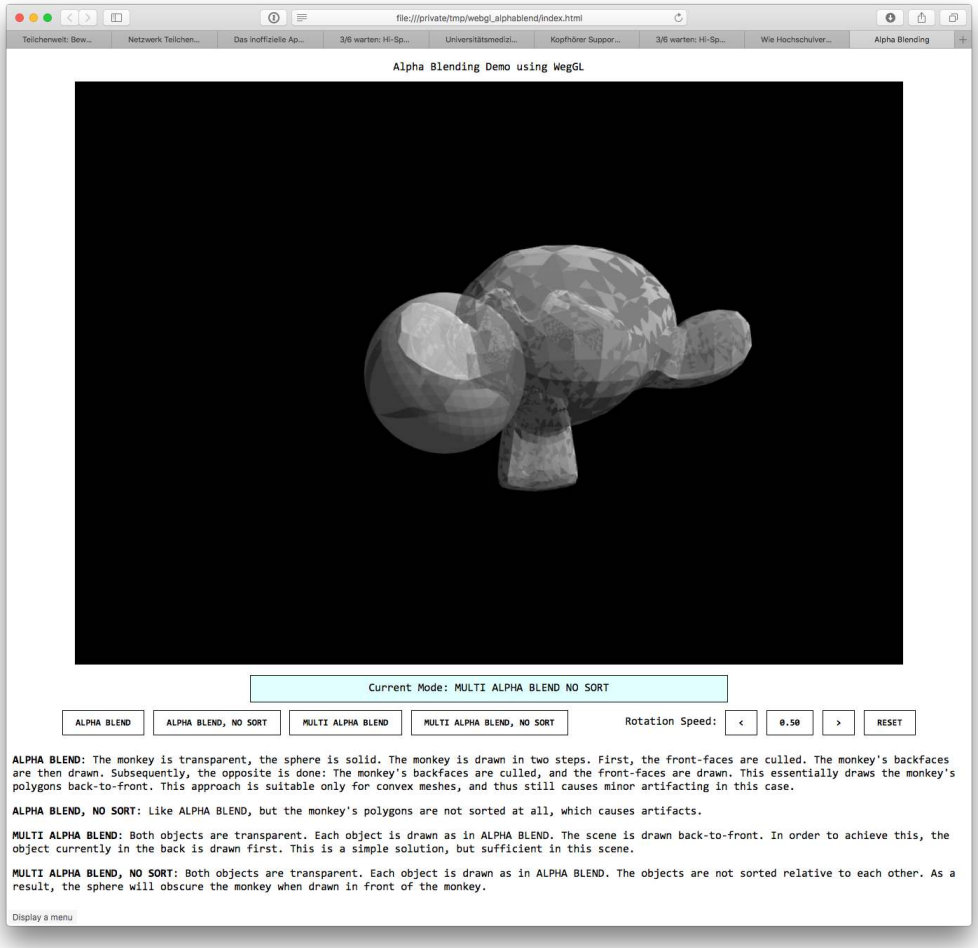
$$C'_D = \alpha_A C_A + (1 - \alpha_A)\, C_S$$
$$C_D = \alpha_B C_B + (1 - \alpha_B)\, C'_D$$
$$= \alpha_B C_B + (1 - \alpha_B)\, \alpha_A C_A + (1 - \alpha_B)\,(1 - \alpha_A)\, C_S$$

2. Rendering first *B* then *A* results in:

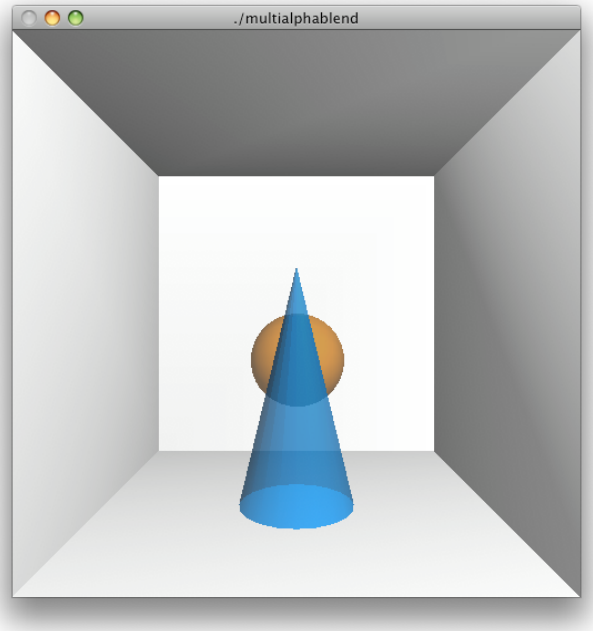
$$C'_D = \alpha_B C_B + (1 - \alpha_B)\, C_S$$
$$C_D = (1 - \alpha_A)\, \alpha_B C_B + \alpha_A C_A + (1 - \alpha_B)\,(1 - \alpha_A)\, C_S$$

- Conclusion: you must render transparent polygons/particles from back to front! (and the Z-buffer can be switched off)

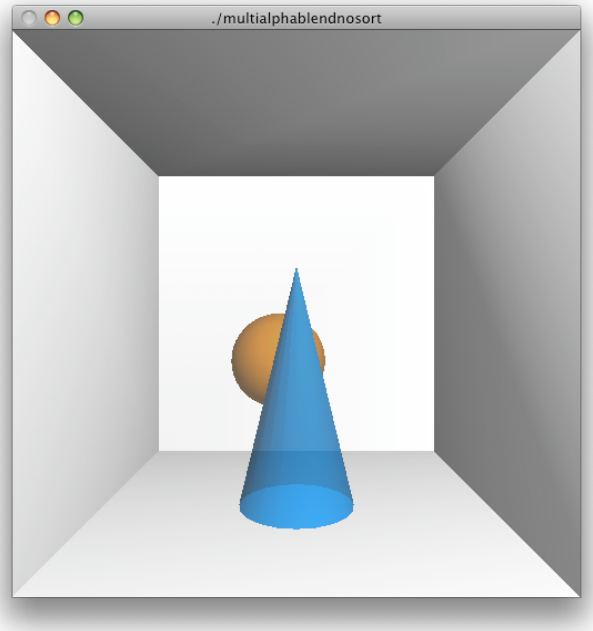
Demo


file:///private/tmp/webgl_alphablend/index.html
Alpha Blending
Alpha Blending Demo using WegGL
Current Mode: MULTI ALPHA BLEND NO SORT
ALPHA BLEND
ALPHA BLEND, NO SORT
MULTI ALPHA BLEND
MULTI ALPHA BLEND, NO SORT
Rotation Speed:
<
0.50
>
RESET
ALPHA BLEND: The monkey is transparent, the sphere is solid. The monkey is drawn in two steps. First, the front-faces are culled. The monkey's backfaces are then drawn. Subsequently, the opposite is done: The monkey's backfaces are culled, and the front-faces are drawn. This essentially draws the monkey's polygons back-to-front. This approach is suitable only for convex meshes, and thus still causes minor artifacting in this case.
ALPHA BLEND, NO SORT: Like ALPHA BLEND, but the monkey's polygons are not sorted at all, which causes artifacts.
MULTI ALPHA BLEND: Both objects are transparent. Each object is drawn as in ALPHA BLEND. The scene is drawn back-to-front. In order to achieve this, the object currently in the back is drawn first. This is a simple solution, but sufficient in this scene.
MULTI ALPHA BLEND, NO SORT: Both objects are transparent. Each object is drawn as in ALPHA BLEND. The objects are not sorted relative to each other. As a result, the sphere will obscure the monkey when drawn in front of the monkey.
Display a menu

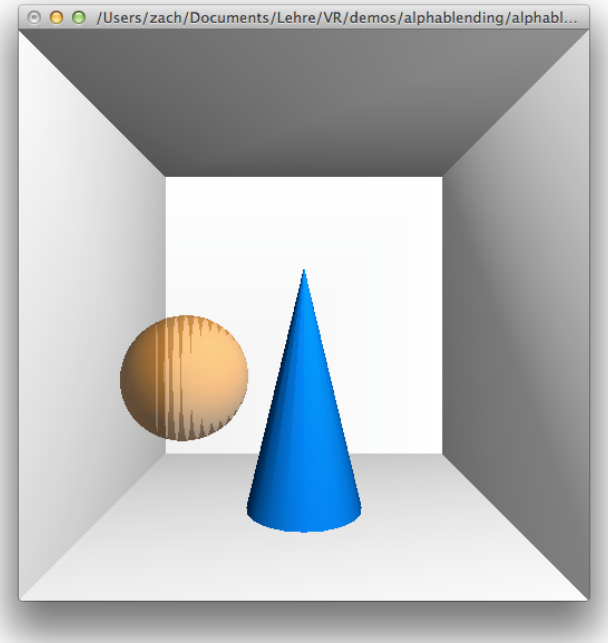
More Examples (1 is correct, 2 with artifacts)



Alpha-blending
with sorting
(two transp. objects)



Alpha-blending
without sorting
(two transp. objects)



Alpha-blending
without sorting
(one transp. object)

Rendering Particles as Transparent Objects

- Render a small disc for each particle (splat, sprite)
- Often with transparency that decreases toward the rim
- Orient splat always towards viewpoint (billboard)

Complex Flames & Fire

- Goals:
  1. Flames that look convincing
  2. Complete control over the flames
- The model:
  1. Represent individual flame (elements) by parametric curves → “spine” of a flame
  2. Regard the control points of the spine as particles
  3. Create surface around the spine where the burning happens
  4. Sample space in the proximity of the surface by "fire" particles
  5. Render these particles (either volumetrically, or with alpha-blending)
- Controls for animators:
  - Average length of spines
  - Lifespan of spine particles
  - Intensity of fire (=number of fire particles; particle sources, wind, etc
  - Color and size of fire particles

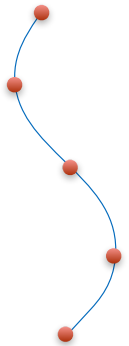
- Creation of the *spines*:
  - Create a spine particle P in the first frame
  - Simulate P: let it move upwards (buoyancy) and sideways (wind)

$$\mathbf{x}_P^{t+1} = \mathbf{x}_P^t + w(\mathbf{x}_P, t) + b(T_P) + d(T_P)$$

where

w = wind field
b = buoyancy
d = diffusion = noise;
T_P = temperature of particle = age

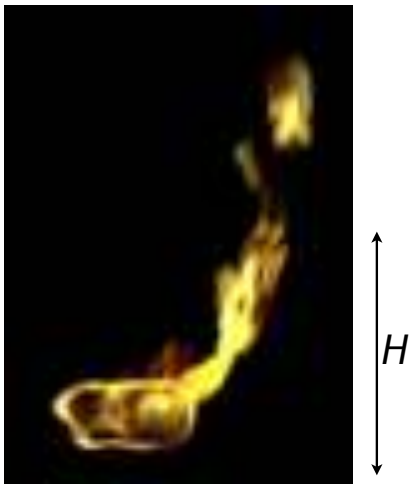
(simplification here: particles don't have a mass)



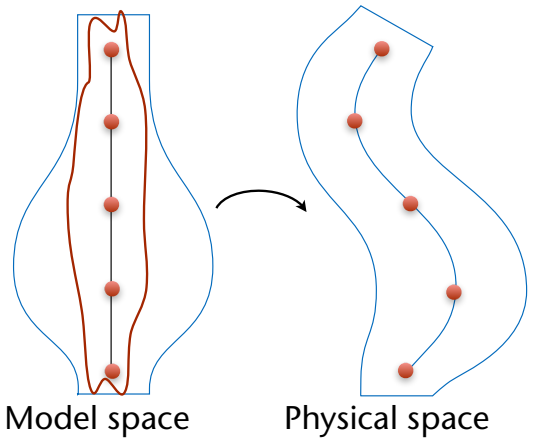
  - In subsequent frames: create more particles; until max. number per flame is reached
  - Connect all spine particles by a curve

- At top of flames: break flame apart
  - Top part of spine is separated from rest at a random point in time, if height > *H*
  - Lifespan after the split:

$$\alpha^3 \text{ sec}, \text{ with random } \alpha \in [0, 1]$$



- The profile of a flame:
  - Rotationally symmetric surface around the spine (= generalized cylinder)
  - Can be drawn by artists

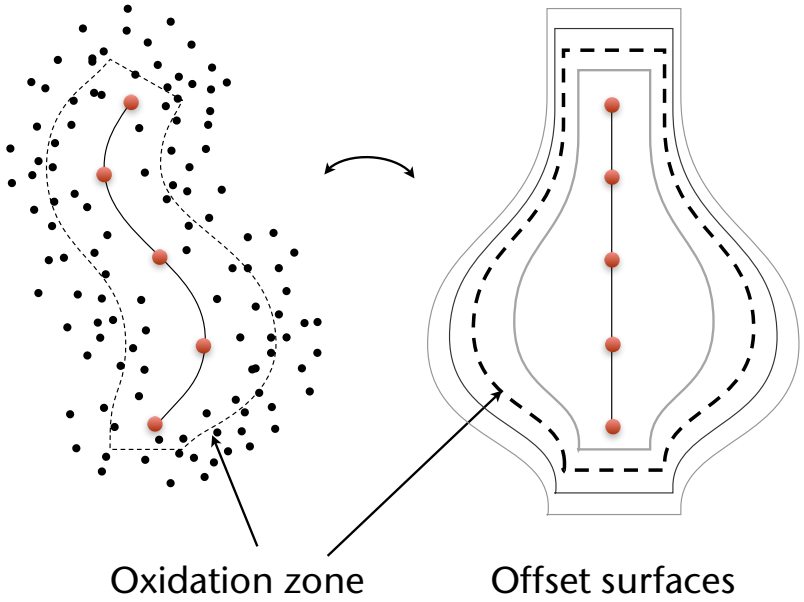


Rendering Flames

- Idea: create lots of small "fire" particles (burning stuff) in the oxidation zone (close to the profile surface); i.e., sample space according to this density function

$$D(\mathbf{x}) = \frac{1}{1 + ||\mathbf{x} - \mathbf{x}'||}$$

where **x**' = point on the profile surface closest to **x**



1. Create random **x**
2. Transform into model space
3. Compute **x**'
4. If $D(\mathbf{x})$ > random number ⟶ keep fire particle **x**
5. Texture profile surface with reference photo of real flame ⟶ basis color for **x**

- Brightness of a fire particle at position **x**:

$$E(\mathbf{x}) = k\frac{D(\mathbf{x})}{n}$$

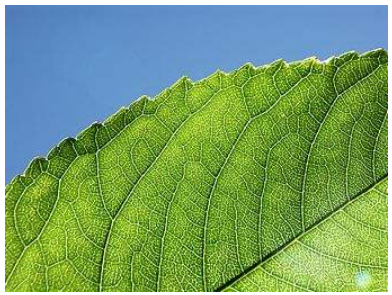
  where k = factor for animator's control, n = number of samples
- Rule of thumb: ca. 10 samples per pixel, ca 10,000 samples per flame
- Discard samples on the inside of obstacles
- Smoke: render fire particles with height > "smoke height" in grey/black

Examples

Arnauld Lamorlette and Nick Foster, PDI/DreamWorks

Procedural Modeling of Plants with Particles

- Idea: use particles to simulate the transportation of water inside a leaf
  - Paths of particles constitute the vessels/"arteries"in the leaf
- Axioms:
  1. Nature always tries to minimize the total length of all arteries → particles will try to merge
  2. No water gets lost or gets added within the arteries → if 2 particles merge their paths, the resulting artery must have twice the cross-sectional area
  3. All arteries/paths emanate from the stem of the leaf

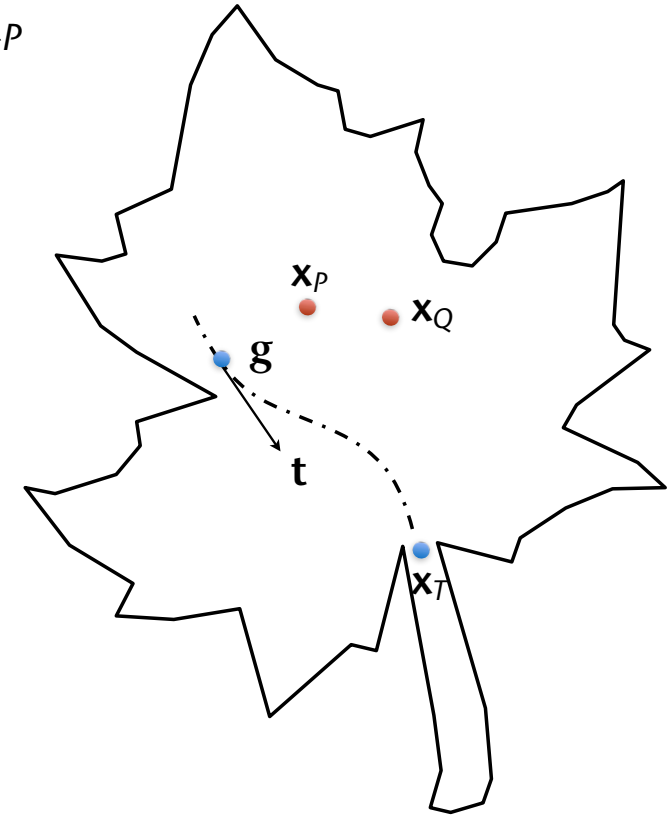


Overview of the Algorithm

```
Initialize particles randomly on surface/rim of the leaf
loop until no particle is left:
   move each particle closer towards its nearest neighbor
       or towards an existig path,
       and in the direction of the stem
   if particle has reached the stem:
       kill it
   if two particles are "close enogh" to each other:
       merge both particles
```

On the Motion/Simulation of Particles

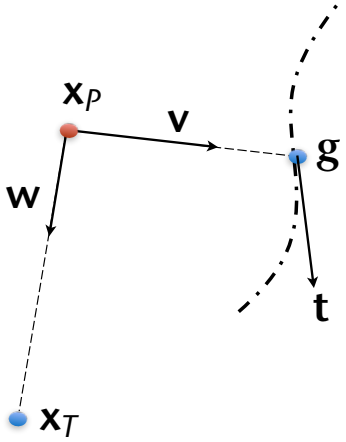
- Let $\mathbf{x}_P$ = current position of particle P
  $\mathbf{x}_T$ = target position (stem of leaf)
  $\mathbf{g}$ = point on an existing path closest to $\mathbf{x}_P$
  $\mathbf{t}$ = tangent in $\mathbf{g}$ (normalized)
  $\mathbf{x}_Q$ = particle closest to P



- If $\|\mathbf{x}_P - \mathbf{g}\| < \|\mathbf{x}_P - \mathbf{x}_Q\|$:
  - Let:

$$\mathbf{v} = \frac{\mathbf{g} - \mathbf{x}_P}{\|\mathbf{g} - \mathbf{x}_P\|}$$

$$\mathbf{w} = \frac{\mathbf{x}_T - \mathbf{x}_P}{\|\mathbf{x}_T - \mathbf{x}_P\|}$$



  - New position:

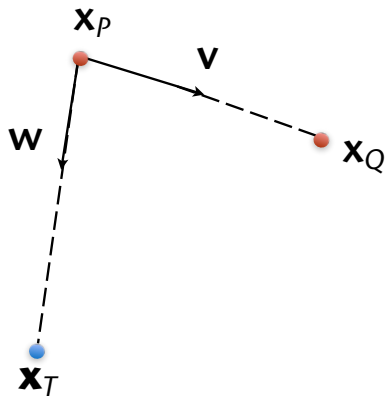
$$\mathbf{x}'_P = \mathbf{x}_P + \alpha\mathbf{w} + (1-\alpha)\left(\beta\mathbf{v} + (1-\beta)\,\mathbf{t}\right)$$

with

$$\beta = \beta\left(\|\mathbf{x}_P - \mathbf{g}\|\right)$$

- If function β is (approximately) linear, this will yield particle paths, that are tangential to existing paths, and perpendicular to them when further away

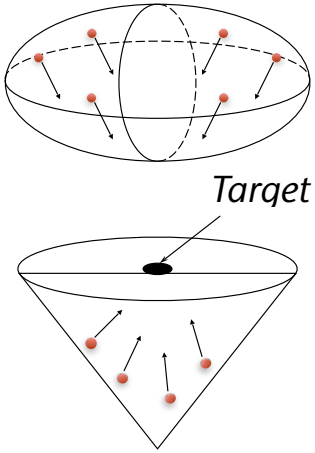
- Else (i.e., $||\mathbf{x}_P - \mathbf{x}_Q|| < ||\mathbf{x}_P - \mathbf{g}||$) :
  - Let $$\mathbf{v} = \frac{\mathbf{x}_Q - \mathbf{x}_P}{||\mathbf{x}_Q - \mathbf{x}_P||}$$
  - New position:
$$\mathbf{x}'_P = \mathbf{x}_P + \gamma\mathbf{v} + (1-\gamma)\mathbf{w}$$

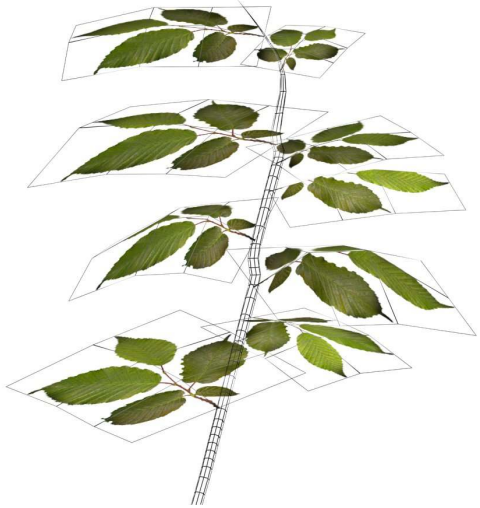


- About the thickness of the arteries:
  - Size of particle = size of cross-sectional area of artery
  - At beginning: each particle has unit size
  - In case of merging: add sizes
  - In case of particle hitting existing path: add size of particle from there on until the stem (target position)

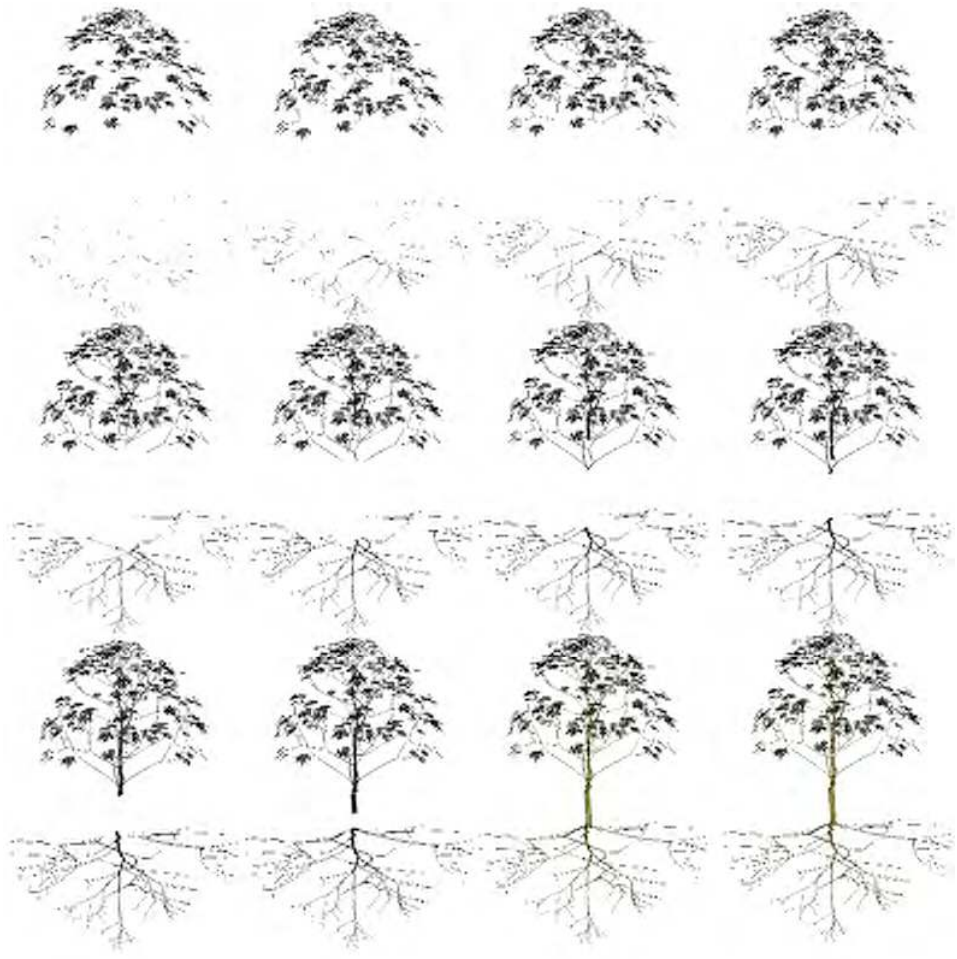
Modeling of Trees

- Works exactly the same
- Input from the animator: geometry of tree crown (= particle source)
  - Create particles within the volume by stochastic process
- Create geometry of branches & twigs by sweeping a disk along the path
- Place leaf primitives at end of twigs



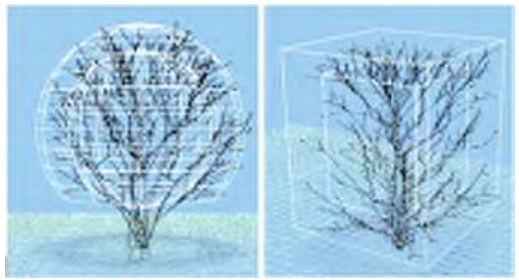


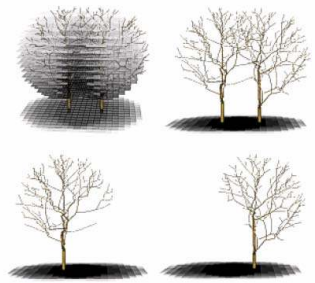
Example of the procedural modeling process



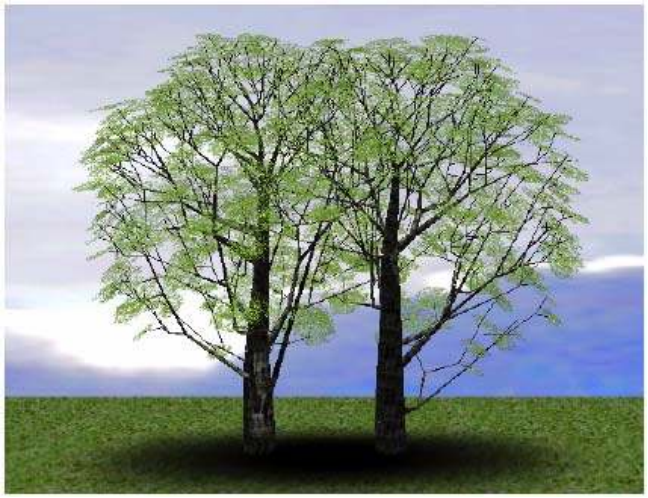
Incorporation of Lighting Conditions

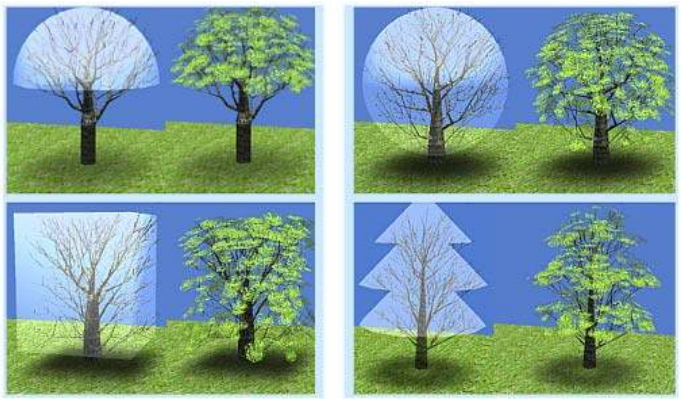
- Observation: regions with less light irradiation have less branches/leaves
- Can be modeled relatively easy:
  - Put tree inside 3D grid
  - Approximate the (not yet existing) foliage by a spherical or cubical shell
  - Compute light irradiation for each grid node by casting a ray outward
  - During particle creation: modify probability of creation according to irradiation (obtained by trilinear interpolation of grid nodes)

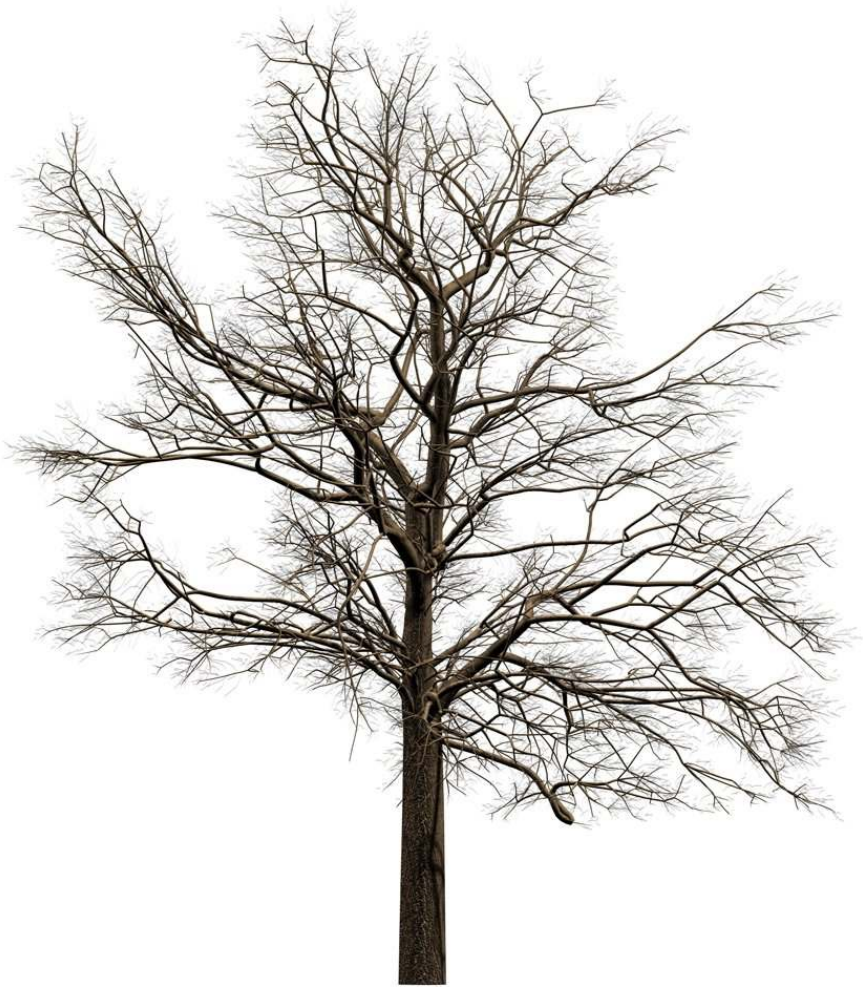




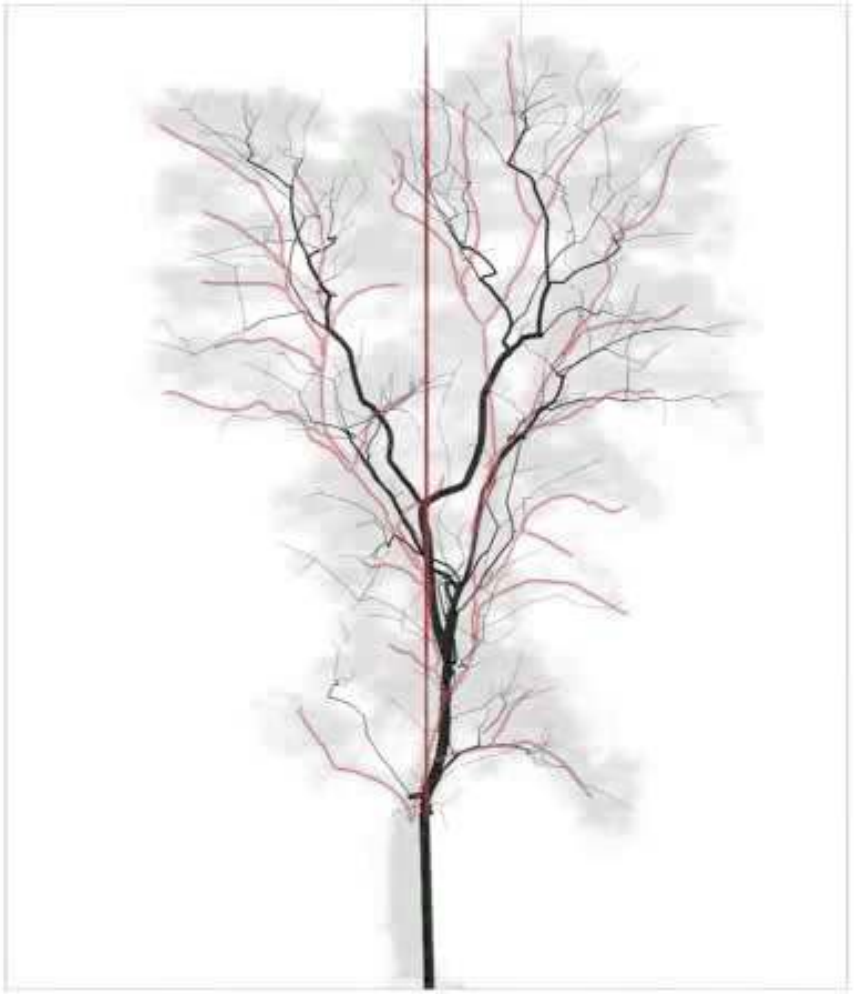
Examples







Videos



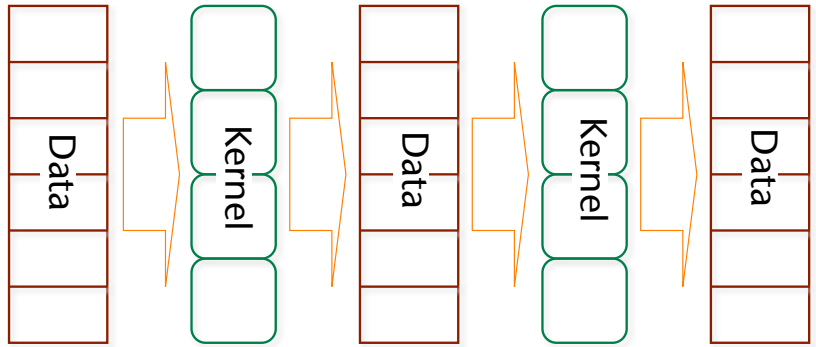
Vintage Video

The Adventures of André and Wally B. (Pixar, 1984)

Massively-Parallel Simulation on Stream Architectures

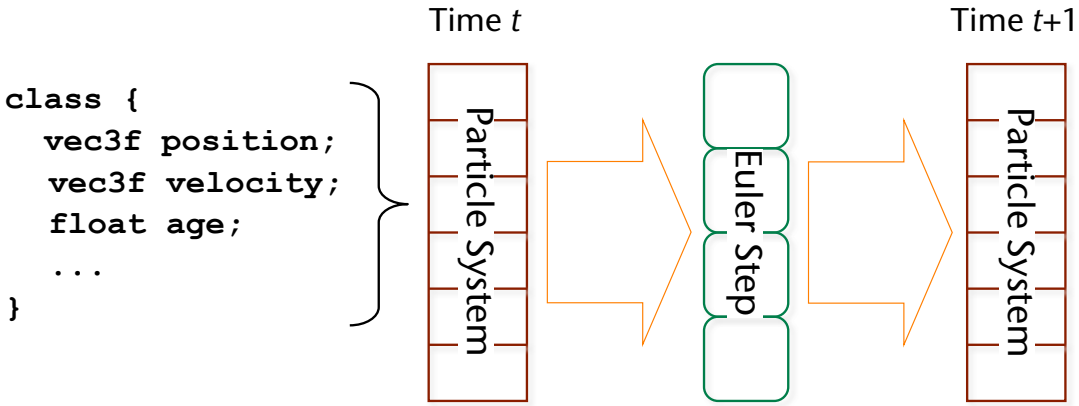
- Background on streaming architectures (and GPUs):
  - Stream Programming Model = *"Arrays of data passing through computation kernels."*
  - Stream = ordered, homogenous set of data of arbitrary type
  - Kernel = program to be performed on *each* element of the input stream
- Sample stream program:

```
{
      stream A, B, C;
      ...
      kernelfunc1( input: A, output: B);
      kernelfunc2( input: B, output: C);
    ...
}
```



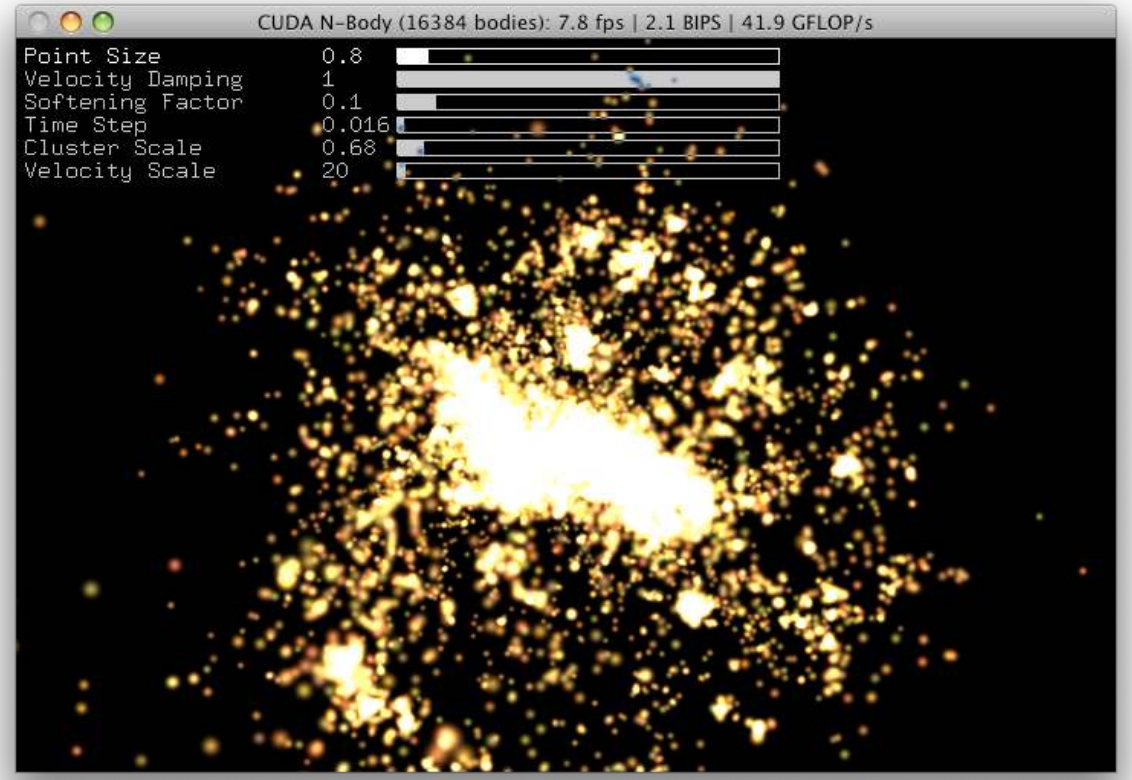
- Today's GPU's are streaming architectures, i.e., massively-parallel , general purpose computing architectures

- Today's GPU's have — at least conceptually — 1000's of processors
- Each processor (kernel) can read several (a few) elements from the input stream, but it can/should write only one output element!
- Particle Simulation on GPU's:



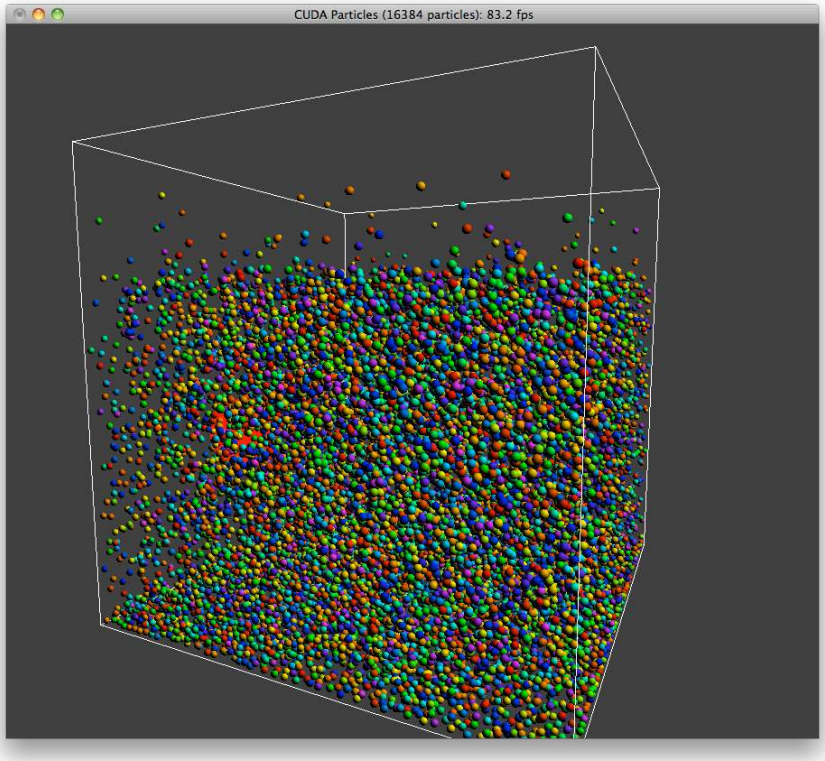
- For more ⟶ "Massively-Parallel Algorithms" course

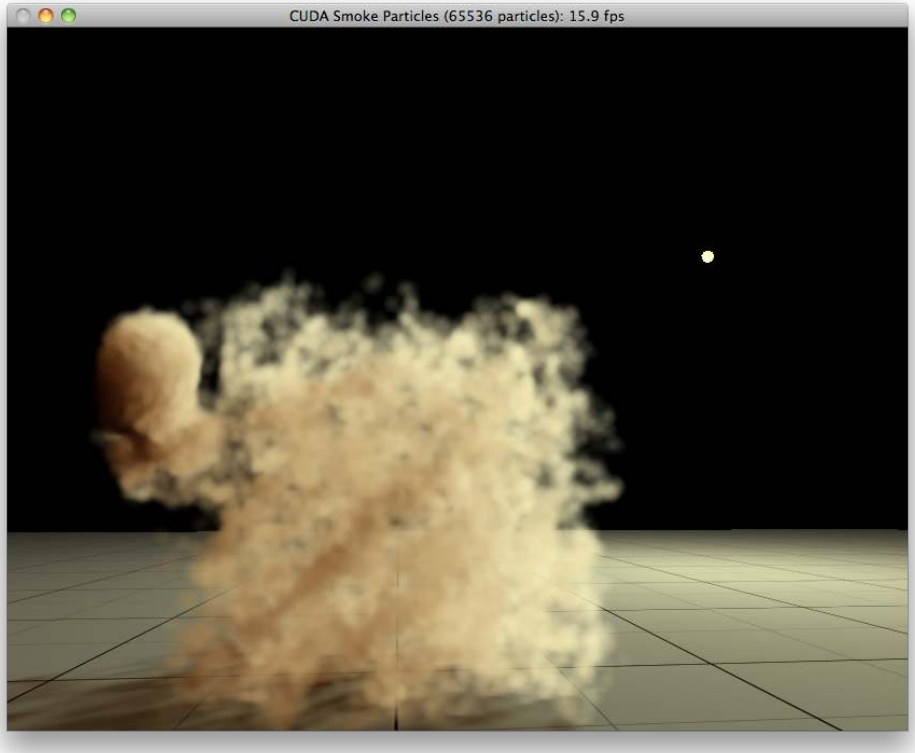
Demos



N-body simulation

http://www.nvidia.com/cuda


CUDA Particles (16384 particles): 83.2 fps


CUDA Smoke Particles (65536 particles): 15.9 fps