Particle Systems and ODEs
Types of Animation

- Keyframing
- Procedural
- Physically-based
  - Particle Systems (today)
    - Smoke, water, fire, sparks, etc.
    - Usually heuristic as opposed to simulation, but not always
    - Mass-Spring Models (Cloth)
      - Thursday
  - Continuum Mechanics (fluids, etc.), finite elements
    - Not in this class
  - Rigid body simulation
    - Not in this class
Types of Animation: Physically-Based

- Assign physical properties to objects
  - Masses, forces, etc.
- Also procedural forces (like wind)
- Simulate physics by solving equations of motion
  - Rigid bodies, fluids, plastic deformation, etc.
- Realistic but difficult to control

\[ m \quad v_0 \quad g \]
Types of dynamics

- Point
Types of dynamics

- Point
- Rigid body
Types of dynamics

- Point
- Rigid body
- Deformable body (include clothes, fluids, smoke, etc.)
Today We Focus on Point Dynamics

• Lots of points!
• Particles systems
  – Borderline between procedural and physically-based
Particle Systems Overview

- **Emitters** generate tons of “particles”
  - Sprinkler, waterfall, chimney, gun muzzle, exhaust pipe, etc.
Particle Systems Overview

- **Emitters** generate tons of “particles”
- Describe the external **forces** with a force field
  - E.g., gravity, wind
Particle Systems Overview

- **Emitters** generate tons of “particles”
- Describe the external **forces** with a force field
- **Integrate** the laws of mechanics (ODEs)
  - Makes the particles move

Image Jeff Lander

http://www.particlesystems.org/
Particle Systems Overview

- **Emitters** generate tons of “particles”
- Describe the external **forces** with a force field
- **Integrate** the laws of mechanics (ODEs)
- In the simplest case, each particle is **independent**
Particle Systems Overview

- **Emitters** generate tons of “particles”
- Describe the external **forces** with a force field
- **Integrate** the laws of mechanics (ODEs)
- In the simplest case, each particle is **independent**
- If there is enough **randomness** (in particular at the emitter) you get nice effects
  - sand, dust, smoke, sparks, flame, water, …
Demo

- Simple “waterfall” built on top of Assignment 0
  - Each particle affected by gravity
  - Simple obstacle geometry (spheres)
  - Braindead rendering (screen-aligned transparent quads)
Demo

• Simple “waterfall” built on top of Assignment 0
  – Each particle affected by gravity
  – Simple obstacle geometry (spheres)
  – Braindead rendering (screen-aligned transparent quads)
 • Could add animated textures and some sounds

Note: No interaction between particles. It’s not a fluid simulation! But then again, it took under an hour to code up.
Real-Time Particles Demo

- 3DMark03 by Futuremark Corp.
  - Explosions, vapor trails, muzzle flashes are particles
Generalizations

- More advanced versions of behavior
  - flocks, crowds
- Forces between particles
  - Not independent any more

See http://www.red3d.com/cwr/boids/ for discussion on how to do flocking.

We’ll come back to this a little later.
Generalizations – Thursday

- Mass-spring and deformable surface dynamics
  - surface represented as a set of points
  - forces between neighbors keep the surface coherent
Generalizations

• It’s not all hacks: Smoothed Particle Hydrodynamics (SPH)
  – A family of “real” particle-based fluid simulation techniques.

  – Fluid flow is described by the Navier-Stokes Equations, a nonlinear partial differential equation (PDE)
    • SPH discretizes the fluid as small packets (particles!), and evaluates pressures and forces based on them.
These Stanford folks use SPH for resolving the small-scale spray and mist that would otherwise be too much for the grid solver to handle.

Another SPH Example

Predictive-Corrective Incompressible SPH. Barbara Solenthaler, Renato Pajarola. ACM Transactions on Graphics (SIGGRAPH), 2009
Meshless Techniques

- Most simulation techniques work on either regular grids or meshes constructed from triangles/tets
- PDEs defined on space are discretized on the grid.
Meshless Techniques

- Most simulation techniques work on either regular grids or meshes constructed from triangles/tets
- In contrast, so-called *meshless methods* do not require the underlying space to be discretized
  - Instead, represent things using points (particles!)
  - They can still be “well-founded”: SPH is an example.
  - Another example: Point-Based Animation of Elastic, Plastic and Melting Objects (Müller, Keiser, Nealen, Pauly, Gross, Alexa, SCA 2004)
Take-Home Message

• Particle-based methods can range from pure heuristics (hacks that happen to look good) to “real” simulation

• Basics are the same: Things always boil down to integrating ODEs!
  – Also in the case of grids/computational meshes
Questions?
What is a Particle System?

- Collection of many small simple pointlike things
  - Described by their current state: position, velocity, age, color, etc.
- Particle motion influenced by external force fields and internal forces between particles
- Particles created by generators or emitters
  - With some randomness
- Particles often have lifetimes
- Particles are often independent
- Treat as points for dynamics, but rendered as anything you want

Simple particle system: sprinkler

PL: linked list of particle = empty;
Simple particle system: sprinkler

PL: linked list of particle = empty;
spread=0.1; // how random the initial velocity is
colorSpread=0.1; // how random the colors are
Simple particle system: sprinkler

PL: linked list of particle = empty;
spread=0.1; //how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
Simple particle system: sprinkler

PL: linked list of particle = empty;
spread=0.1; //how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
  Generate k particles
    p=new particle();
    p->position=(0,0,0);
    p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
    p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
    PL->add(p);
Simple particle system: sprinkler

PL: linked list of particle = empty;
spread=0.1; //how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
  Generate k particles
    p=new particle();
    p->position=(0,0,0);
    p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
    p.color=(0,0,1)+colorSpread*(rnd(), rnd(),rnd());
    PL->add(p);
  For each particle p in PL
    p->position+=p->velocity*dt; //dt: time step
    p->velocity-=g*dt; //g: gravitation constant
    glColor(p.color);
    glVertex(p.position);
Simple particle system: sprinkler

PL: linked list of particle = empty;
spread=0.1; //how random the initial velocity is
colorSpread=0.1; //how random the colors are
For each time step
  Generate k particles
    p=new particle();
    p->position=(0,0,0);
    p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
    p.color=(0,0,1)+colorSpread*(rnd(), rnd(), rnd());
    PL->add(p);
  For each particle p in PL
    p->position+=p->velocity*dt; //dt: time step
    p->velocity-=g*dt; //g: gravitation constant
    glColor(p.color);
    glVertex(p.position);
Questions?
Ordinary Differential Equations

\[
\frac{dX(t)}{dt} = f(X(t), t)
\]

- Given a function \( f(X, t) \) compute \( X(t) \)
- Typically, initial value problems:
  - Given values \( X(t_0) = X_0 \)
  - Find values \( X(t) \) for \( t > t_0 \)

- We can use lots of standard tools
Point mass: 2nd order ODE

\[ \vec{F} = m\ddot{x} \quad \text{or} \quad \vec{F} = m \frac{d^2\vec{x}}{dt^2} \]

Position \( \vec{x} \) and force \( \vec{F} \) are vector quantities
– We know \( \vec{F} \) and \( m \), want to solve for \( \vec{x} \)

You’ve all seen this a million times before
Reduction to 1st Order

• Point mass: 2nd order ODE

\[ \vec{F} = m \ddot{x} \quad \text{or} \quad \vec{F} = m \frac{d^2 \vec{x}}{dt^2} \]

• Corresponds to system of first order ODEs

\[
\begin{cases}
\frac{d}{dt} \vec{x} = \vec{v} \\
\frac{d}{dt} \vec{v} = \vec{F} / m
\end{cases}
\]

2 unknowns (x, v) instead of just x
Reduction to 1st Order

\[
\begin{aligned}
\frac{d}{dt} \vec{x} &= \vec{v} \\
\frac{d}{dt} \vec{v} &= \vec{F} / m
\end{aligned}
\]

2 variables (\(x, v\)) instead of just one

• Why reduce?
Reduction to 1st Order

\[
\begin{align*}
\frac{d}{dt} \vec{x} &= \vec{v} \\
\frac{d}{dt} \vec{v} &= \vec{F} / m
\end{align*}
\]

2 variables \((\vec{x}, \vec{v})\) instead of just one

- Why reduce?
  - Numerical solvers grow more complicated with increasing order, can just write one 1st order solver and use it
  - Note that this doesn’t mean it would always be easy :-}
Notation

• Let’s stack the pair \((x, v)\) into a bigger state vector \(X\)

\[
X = \begin{pmatrix} \vec{x} \\ \vec{v} \end{pmatrix}
\]

For a particle in 3D, state vector \(X\) has 6 numbers

\[
\frac{d}{dt} X = f(X, t) = \begin{pmatrix} \vec{v} \\ \vec{F}(x, v)/m \end{pmatrix}
\]
Now, Many Particles

• We have N point masses
  – Let’s just stack all $x$s and $v$s in a big vector of length $6N$

\[
X = \begin{pmatrix}
  x_1 \\
  v_1 \\
  \vdots \\
  x_N \\
  v_N 
\end{pmatrix}
\quad
f(X, t) = \begin{pmatrix}
  v_1 \\
  F^1(X, t) \\
  \vdots \\
  v_N \\
  F^N(X, t)
\end{pmatrix}
\]
Now, Many Particles

- We have $N$ point masses
  - Let’s just stack all $x$s and $v$s in a big vector of length $6N$
  - $F_i$ denotes the force on particle $i$

- When particles don’t interact, $F_i$ only depends on $x_i$ and $v_i$.

\[
X = \begin{pmatrix}
x_1 \\
v_1 \\
\vdots \\
x_N \\
v_N
\end{pmatrix}
\]

\[
f(X, t) = \begin{pmatrix}
v_1 \\
F_1(X, t) \\
\vdots \\
v_N \\
F_N(X, t)
\end{pmatrix}
\]

$f$ gives $d/dt X$, remember!
Path through a Vector Field

- \( X(t) \): path in multidimensional phase space

\[
\frac{d}{dt} X = f(X, t)
\]

“When we are at state \( X \) at time \( t \), where will \( X \) be after an infinitely small time interval \( dt \)?”
Path through a Vector Field

- $X(t)$: path in multidimensional phase space

\[ \frac{d}{dt} X = f(X, t) \]

“When we are at state $X$ at time $t$, where will $X$ be after an infinitely small time interval $dt$?”

- $f = \frac{d}{dt} X$ is a vector that sits at each point in phase space, pointing the direction.
Questions?
Numerics of ODEs

• Numerical solution is called “integration of the ODE”
• Many techniques
  – Today, the simplest one
  – Thursday and next week we’ll look at some more advanced techniques
Intuitive Solution: Take Steps

- Current state \( X \)
- Examine \( f(X,t) \) at (or near) current state
- Take a step to new value of \( X \)

\[
\frac{d}{dt} X = f(X, t)
\]

\[\Rightarrow "dX = dt \cdot f(X, t)"\]

\( f = \frac{d}{dt} X \) is a vector that sits at each point in phase space, pointing the direction.
Euler’s Method

• Simplest and most intuitive
• Pick a step size $h$
• Given $X_0 = X(t_0)$, take step:

$$t_1 = t_0 + h$$

$$X_1 = X_0 + h f(X_0, t_0)$$

• Piecewise-linear approximation to the path
• Basically, just replace $dt$ by a small but finite number
Euler, Visually

\[ \frac{d}{dt} X = f(X, t) \]
Euler, Visually

\[ \frac{d}{dt} X = f(X, t) \]
Euler, Visually

\[ \frac{d}{dt} X = f(X, t) \]
\[ \frac{d}{dt} X = f(X, t) \]
Effect of Step Size

• Step size controls accuracy
• Smaller steps more closely follow curve
  – May need to take many small steps per frame
  – Properties of $f(X, t)$ determine this (more later)
Euler’s method: Inaccurate

- Moves along tangent; can leave solution curve, e.g.:
  \[ f(X, t) = \begin{pmatrix} -y \\ x \end{pmatrix} \]

- Exact solution is circle:
  \[ X(t) = \begin{pmatrix} r \cos(t+k) \\ r \sin(t+k) \end{pmatrix} \]
Euler’s method: Inaccurate

- Moves along tangent; can leave solution curve, e.g.:

  \[ f(X, t) = \begin{pmatrix} -y \\ x \end{pmatrix} \]

- Exact solution is circle:

  \[ X(t) = \begin{pmatrix} r \cos(t+k) \\ r \sin(t+k) \end{pmatrix} \]

- Euler spirals outward no matter how small \( h \) is
  - will just diverge more slowly
More Accurate Alternatives

• Midpoint, Trapezoid, Runge-Kutta
  – Also, “implicit methods” (next week)

More on this on Thursday

• Extremely valuable resource: SIGGRAPH 2001 course notes on physically based modeling
What is a Force?

• A force changes the motion of the system
  – Newton says: When there are no forces, motion continues uniformly in a straight line (good enough for us)

• Forces can depend on location, time, velocity
  – Gravity, spring, viscosity, wind, etc.

• For point masses, forces are vectors
  – Ie., to get total force, take vector sum of everything
Forces: Gravity on Earth

- Depends only on particle mass
- $f(X, t) = \text{constant}$
- Hack for smoke, etc: make gravity point up!
  - Well, you can call this buoyancy, too.

Gravity: $f^{(i)} = \begin{pmatrix} 0 \\ 0 \\ -m_i G \end{pmatrix}$
Forces: Gravity (N-body problem)

- Gravity depends on all other particles
- Opposite for pairs of particles
- Force in the direction of $p_i - p_j$ with magnitude inversely proportional to square distance

$$\| F_{ij} \| = \frac{G m_i m_j}{r^2}$$

where $G = 6.67 \times 10^{-11}$ Nm$^2$/kg$^2$

- Testing all pairs is $O(n^2)$!

Particles are not independent!
Real-Time Gravity Demo

- [Link to video]
An Aside on Gravity

• That was Brute Force
  – Meaning all $O(n^2)$ pairs of particles were considered when computing forces
  – Yes, computers are fast these days, but this gets prohibitively expensive soon. (The square in $n^2$ wins.)

• Hierarchical techniques approximate forces caused by many distant attractors by one force, yields $O(n)!$
  – This inspired very cool hierarchical illumination rendering algorithms in graphics (hierarchical radiosity, etc.)
Forces: Viscous Damping

\[ f^{(i)} = -d v^{(i)} \]

- Damping force on particle i determined its velocity
  - Opposes motion
  - E.g. wind resistance
- Removes energy, so system can settle
- Small amount of damping can stabilize solver
- Too much damping makes motion like in glue
Forces: Spatial Fields

- Externally specified force (or velocity) fields in space
- Force on particle $i$ depends only on its position
- Arbitrary functions
  - wind
  - attractors, repulsors
  - vortices
- Can depend on time
- Note: these add energy, may need damping
Example: Procedural Spatial Field


Plausible, controllable force fields – just advecting particles along the flow gives cool results!

And it’s simple, too!
Forces: Other Spatial Interaction

Spatial interaction: \( f^{(i)} = \sum_j f(x^{(i)}, x^{(j)}) \)

• E.g., approximate fluid using Lennard-Jones force:
  \[
  f(x^{(i)}, x^{(j)}) = \frac{k_1}{|x^{(i)} - x^{(j)}|^m} - \frac{k_2}{|x^{(i)} - x^{(j)}|^n}
  \]

• Repulsive + attractive force
• Again, \( O(N^2) \) to test all pairs
  – usually only local
  – Use buckets to optimize. Cf. 6.839

Particles are not independent!
Demo: Lennard-Jones

• Real-time particle system written in **CUDA**

• (May not run on external display)
More Eyecandy from NVIDIA

- Fluid flow solved using a regular grid solver
  - This gives a velocity field
- 0.5M smoke particles advected using the field
  - That means particle velocity is given by field
- Particles are for rendering, motion solved using other methods
- Link to video
That’s All for Today!

• Further reading
    • Extremely good, easy-to-read resource. Highly recommended!
  – William Reeves: Particle systems—a technique for modeling a class of fuzzy objects, Proc. SIGGRAPH 1983
    • The original paper on particle systems
  – particlesystems.org