MIT EECS 6.837 Computer Graphics

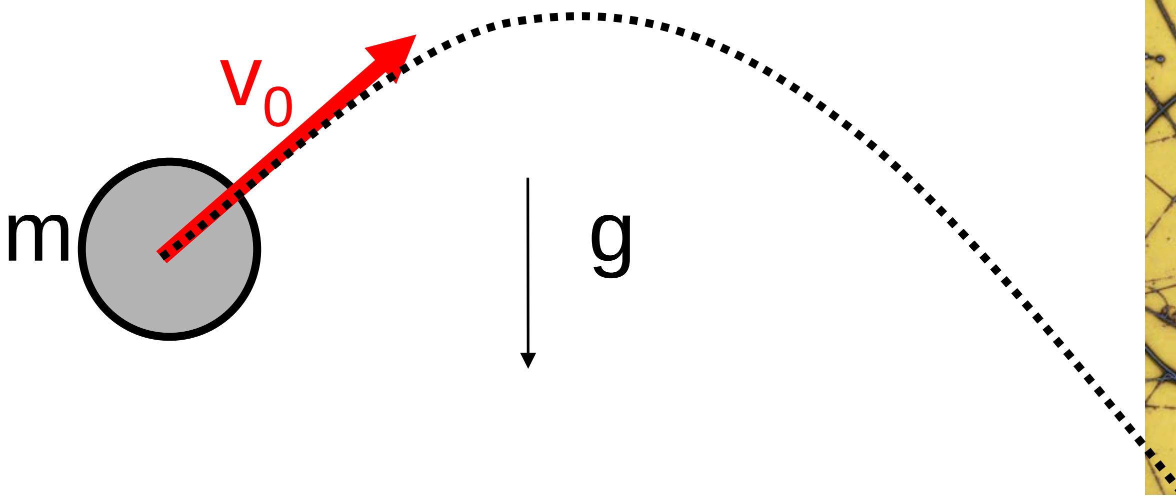
Particle Systems and ODEs

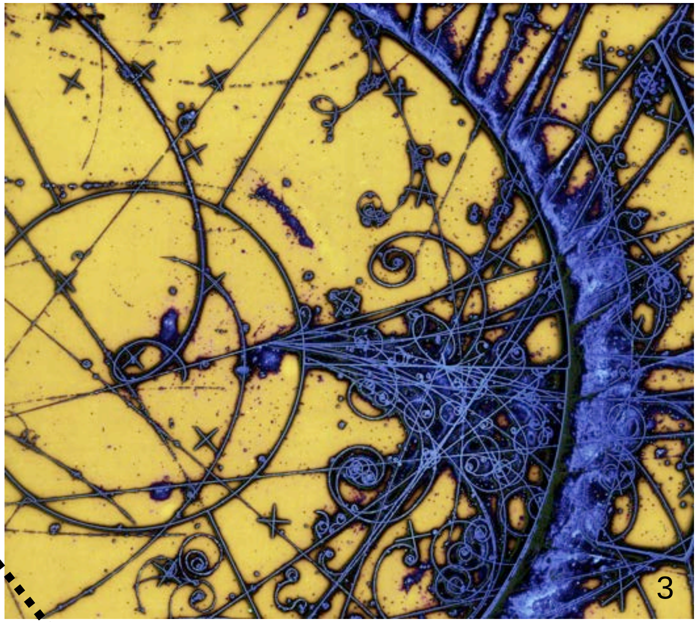
Types of Animation

- 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) **NEXT CLASS**
  - Continuum Mechanics (fluids, etc.), finite elements
    - Not in this class
  - Rigid body simulation
    - Not in this class

Types of Animation: Physically-Based

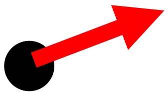
- Assign physical properties to objects
  - Masses, forces, etc.
- Also procedural forces (like wind)
- Simulate physics by solving equations of motion
  - Rigid bodies, fluids, plastic deformation, etc.
- Realistic but difficult to control





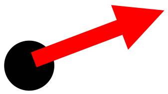
Types of Dynamics

- Point



Types of Dynamics

- Point

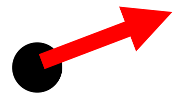


- Rigid body

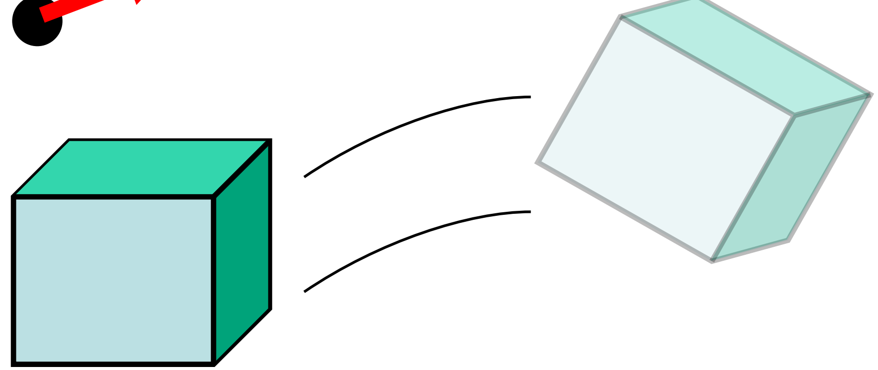
Animation by Mark Carlson

Types of Dynamics

- Point



- Rigid body

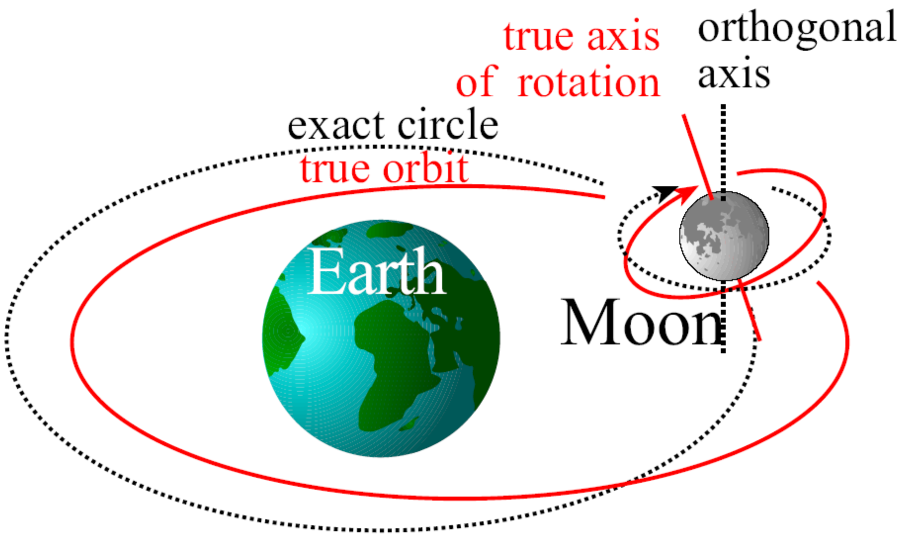


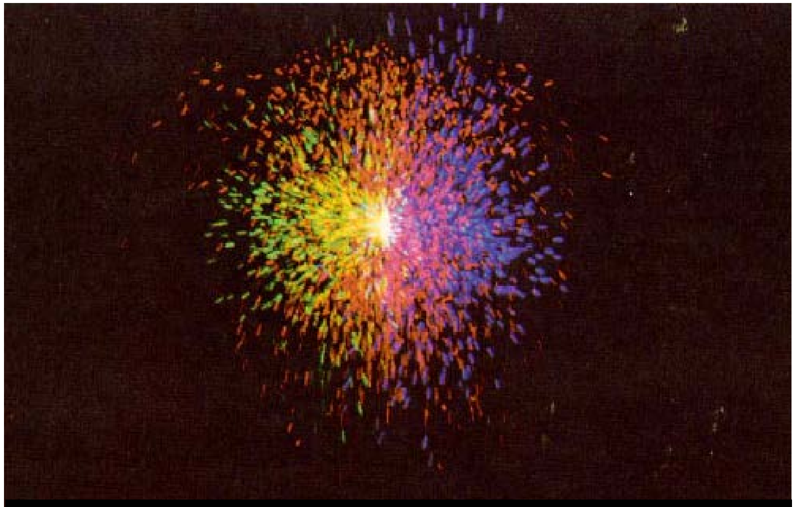
- Deformable body
  (include clothes, fluids, smoke, etc.)

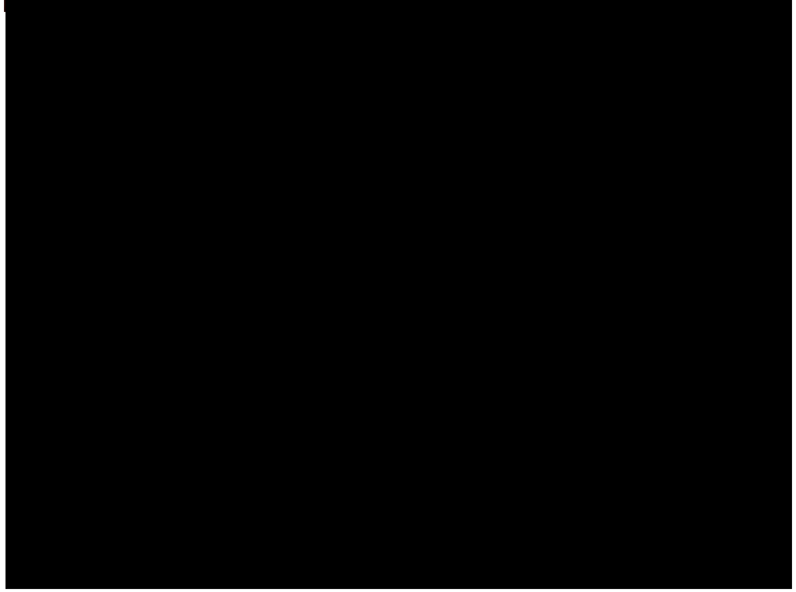
Mark Carlson

Today We Focus on Point Dynamics

- Lots of points!
- Particles systems
  - Borderline between procedural and physically-based

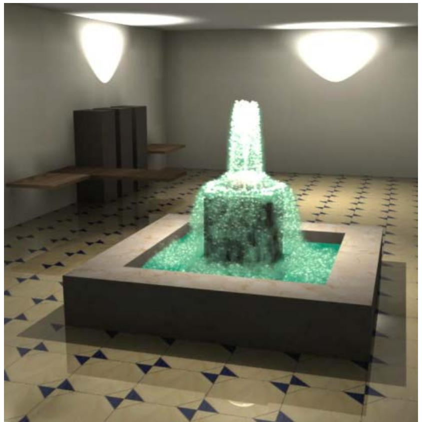


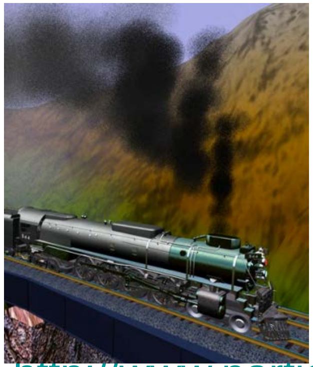




Particle Systems Overview

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

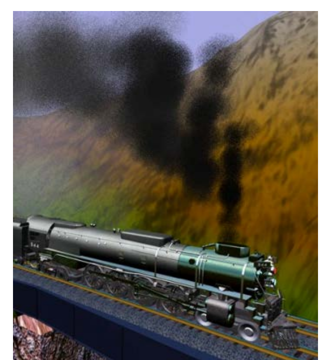


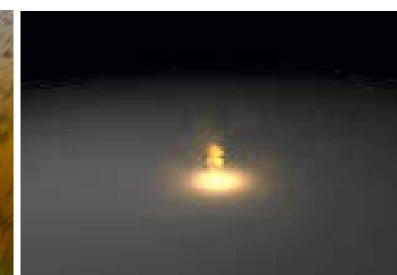


http://www.particlesystems.org/

Particle Systems Overview

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

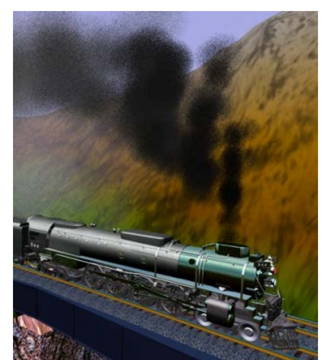
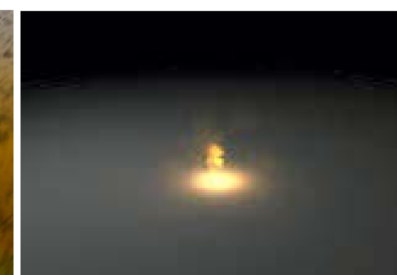




http://www.particlesystems.org/

Particle Systems Overview

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

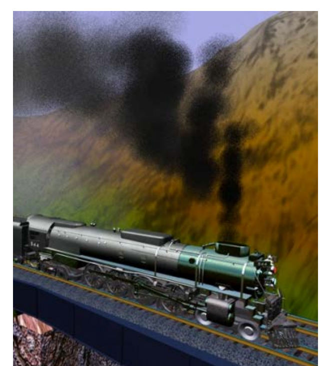




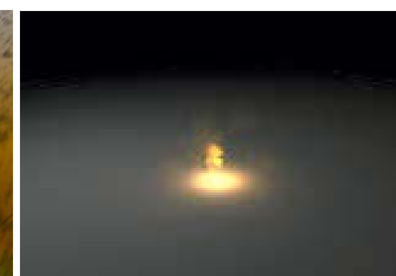
http://www.particlesystems.org/

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**

Image Jeff Lander



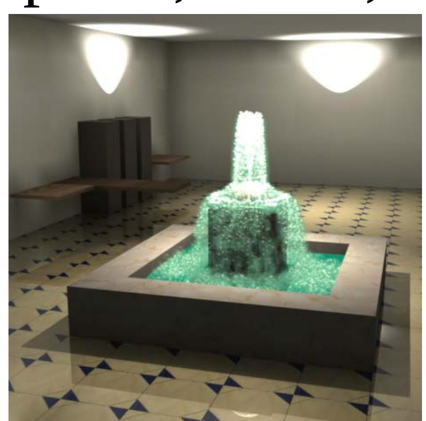


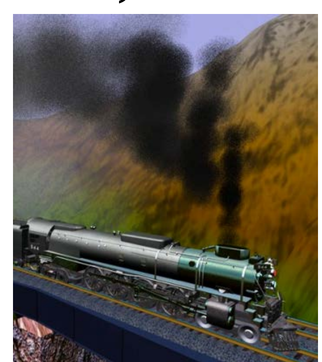
http://www.particlesystems.org/

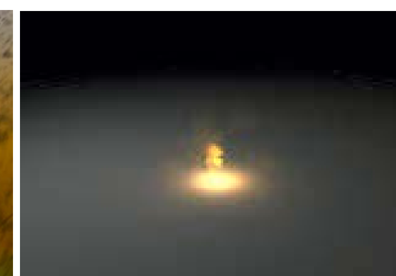
Particle Systems Overview

- **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**
- If there is enough **randomness** (in particular at the emitter) you get nice effects
  - sand, dust, smoke, sparks, flame, water, …

Image Jeff Lander







http://www.particlesystems.org/

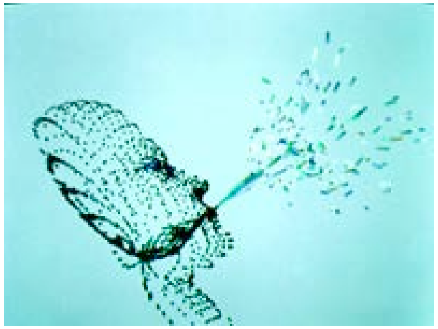
Sprinkler

- http://www.youtube.com/watch?v=rhvH12nC6_Q

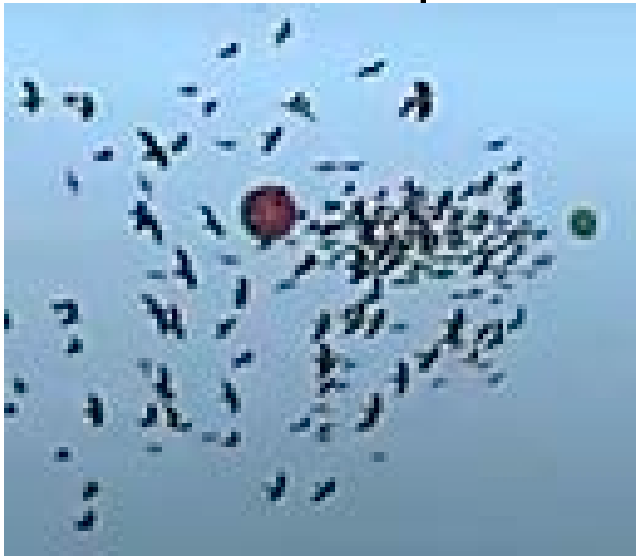
Fire

- http://www.youtube.com/watch?v=6hG00etwRBU

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.

http://www.blendernation.com/2008/01/05/simulating-flocks-herds-and-swarms-using-experimental-blender-boids-particles/

Generalizations – Next Class

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

Image Witkin & Baraff

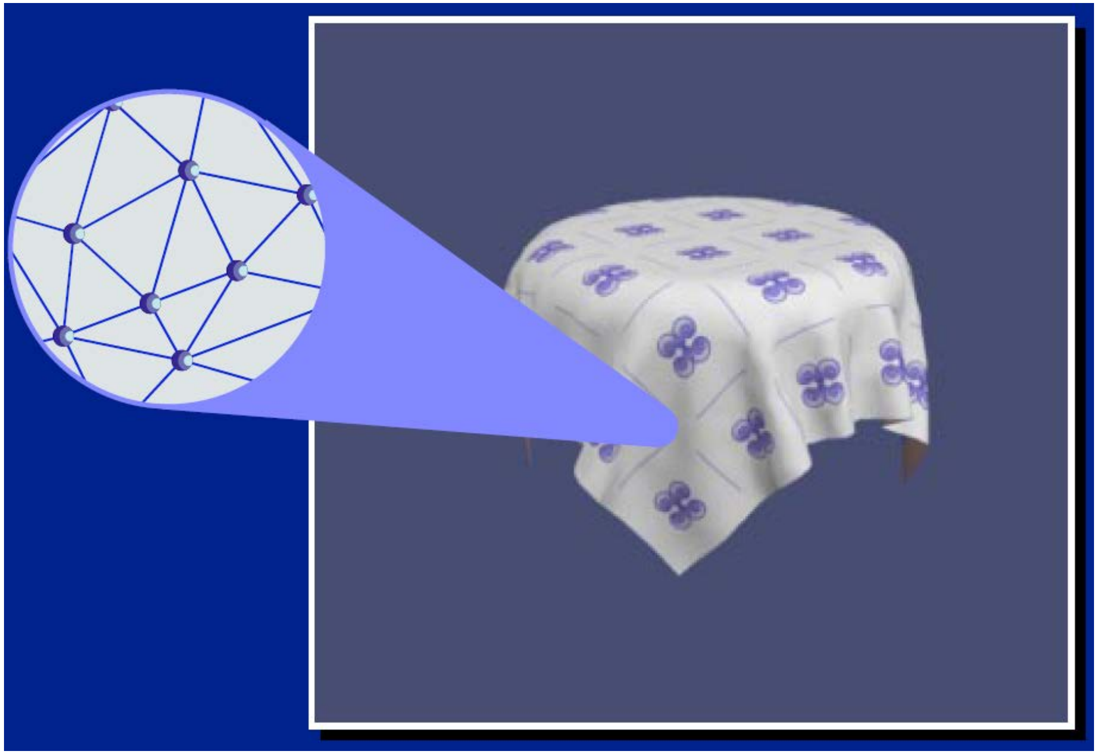
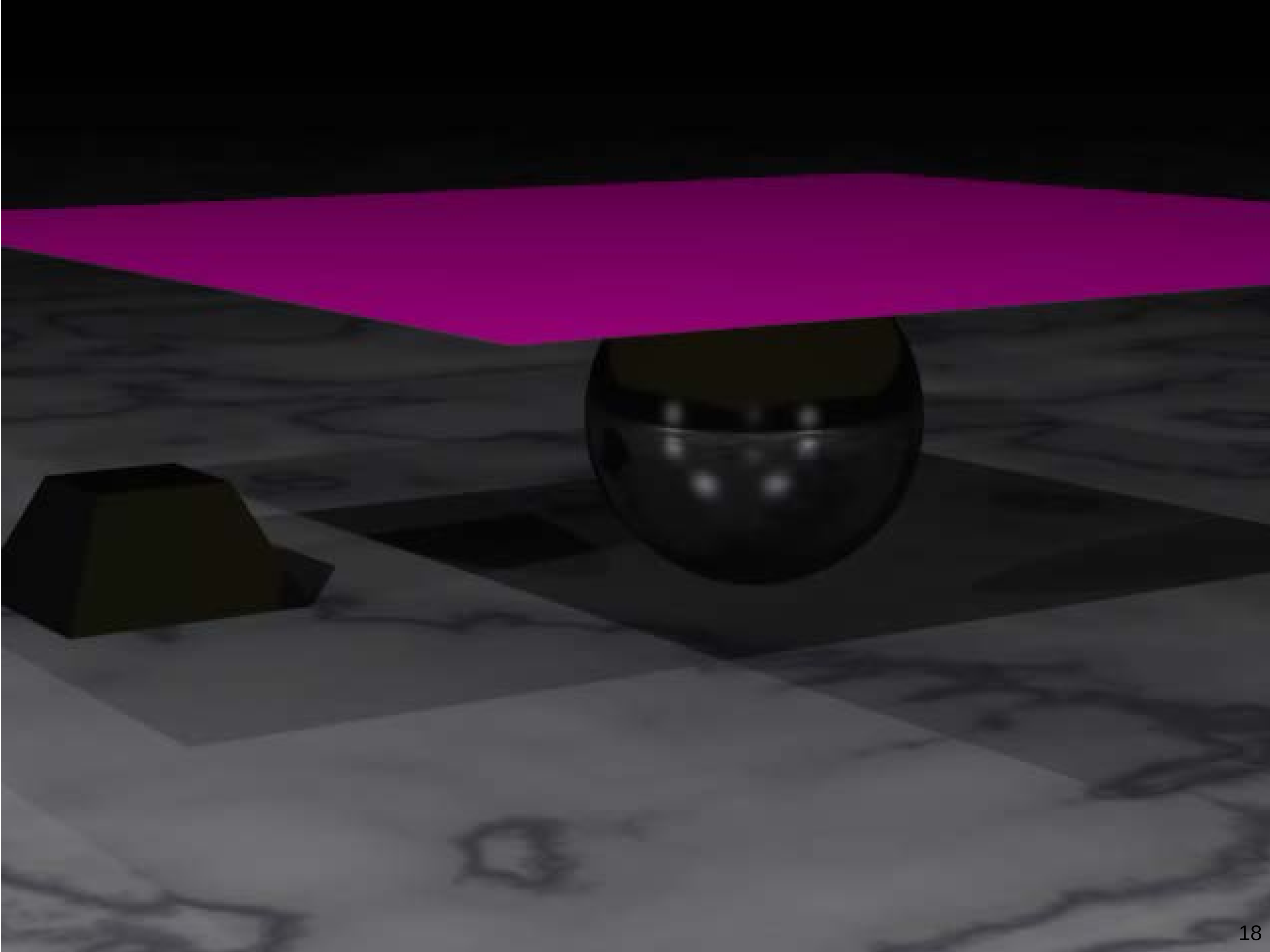


Image Michael Kass

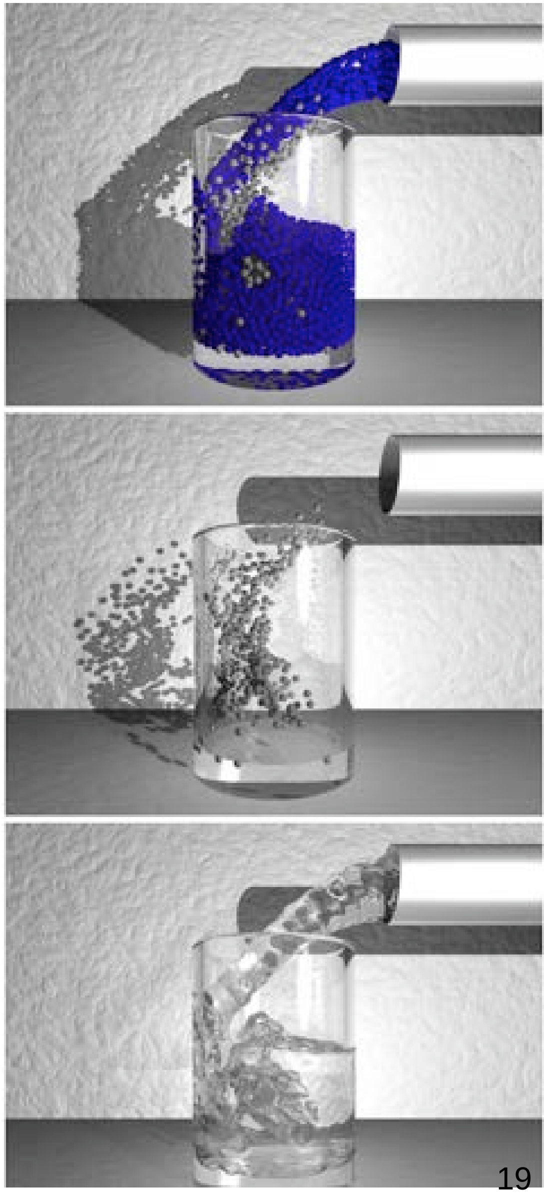
Cloth Video

Selle, A., Su, J., Irving, G. and Fedkiw, R., "Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction," IEEE TVCG 15, 339-350 (2009).

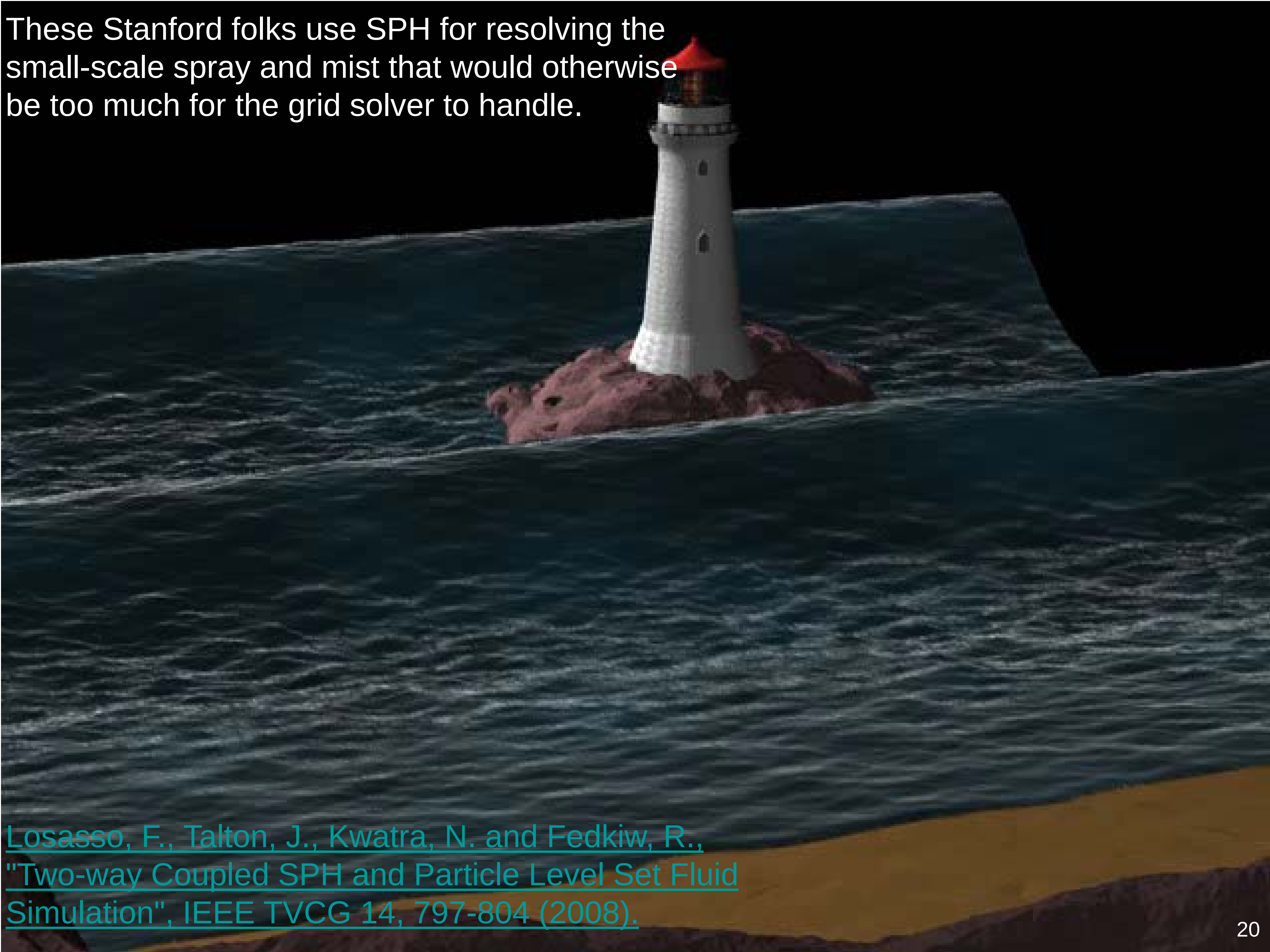


Generalizations

Müller et al. 2005



- 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.

[Losasso, F., Talton, J., Kwatra, N. and Fedkiw, R., "Two-way Coupled SPH and Particle Level Set Fluid Simulation", IEEE TVCG 14, 797-804 (2008).]()

Real-time particles in games

- http://www.youtube.com/watch?v=6DicVajK2xQ

EA Fight Night 4 Physics Trailer

Take-Home Message

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

Andrew Selle et al.

- Basics are the same:
  Things always boil down to integrating ODEs!
  – Also in the case of grids/computational meshes

Questions?

http://www.cs.columbia.edu/cg/ESIC/esic.html

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

http://en.wikipedia.org/wiki/Image:Particle_Emitter.jpg

Simple Particle System: Sprinkler

```
PL: linked list of particle = empty;
```

Image Jeff Lander

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
```

Image Jeff Lander

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
```

Image Jeff Lander

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);
```

Image Jeff Lander

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);
```

Image Jeff Lander

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);
```

Image Jeff Lander

Demo with Processing

- http://processing.org/learning/topics/simpleparticlesystem.html

This example is for Processing version 1.5+. If you have a previous version, use the examples included with your software. If you see any errors or have suggestions, please let us know.

Simple Particle System by Daniel Shiffman.

Particles are generated each cycle through draw(), fall with gravity and fade out over time A ParticleSystem object manages a variable size (ArrayList) list of particles.

```
ParticleSystem ps;

void setup() {
  size(640, 360);
  colorMode(RGB, 255, 255, 255, 100);
  ps = new ParticleSystem(1, new PVector(width/2,height/2,0));
  smooth();
}

void draw() {
  background(0);
  ps.run();
  ps.addParticle(mouseX,mouseY);
}
```

Questions?

Path forward

- Basic particle systems are simple hacks
- Extend to physical simulations, e.g., clothes
- For this, we need to understand numerical integration
- This lecture: point particles
- Next lecture: mass-spring and clothes

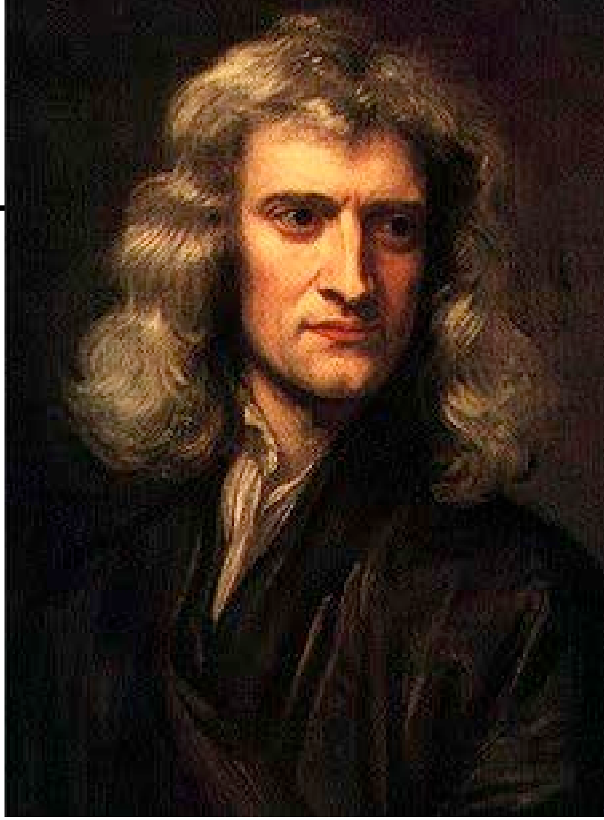
Ordinary Differential Equations

$$\frac{d\,\mathbf{X}(t)}{dt} = f(\mathbf{X}(t), t)$$

- Given a function $f(\mathbf{X},t)$ compute $\mathbf{X}(t)$
- Typically, *initial value problems*:
  - Given values $\mathbf{X}(t_0)=\mathbf{X}_0$
  - Find values $\mathbf{X}(t)$ for $t > t_0$

- We can use lots of standard tools

Newtonian Mechanics

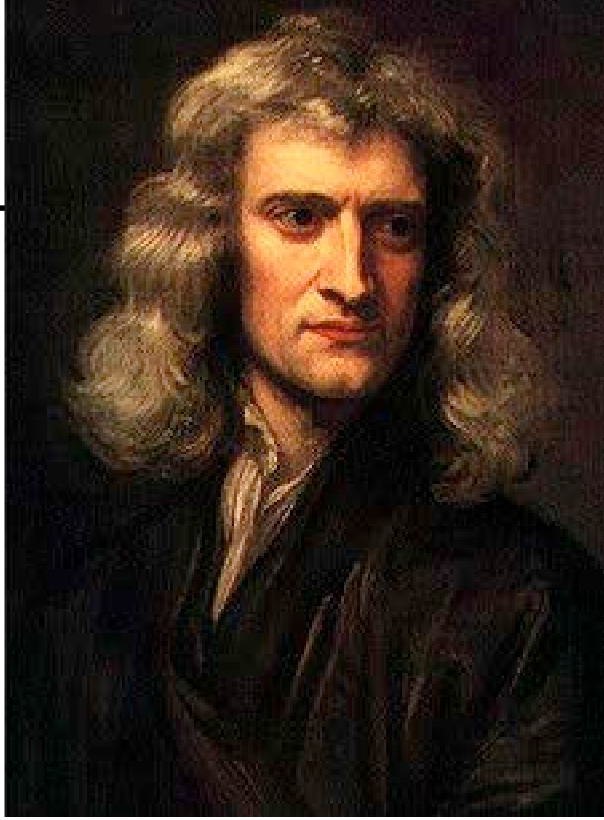


- Point mass: 2nd order ODE

$$\vec{F} = m\vec{a} \quad \text{or} \quad \vec{\boldsymbol{F}} = m\frac{d^2\vec{\boldsymbol{x}}}{dt^2}$$

- Position ***x*** and force ***F*** are vector quantities
  - We know ***F*** and *m*, want to solve for ***x***
- You have all seen this a million times before

Reduction to 1st Order



- Point mass: 2nd order ODE

$$\vec{F} = m\vec{a} \quad \text{or} \quad \vec{\boldsymbol{F}} = m\frac{d^2\vec{\boldsymbol{x}}}{dt^2}$$

- Corresponds to system of first order ODEs

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

2 unknowns (**x**, **v**) instead of just **x**

Reduction to 1st Order

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

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

- Why reduce?

Reduction to 1st Order

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

2 variables (**x**, **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**

$$\boldsymbol{X} = \begin{pmatrix} \vec{\boldsymbol{x}} \\ \vec{\boldsymbol{v}} \end{pmatrix}$$

For a particle in 3D, state vector **X** has 6 numbers

$$\frac{d}{dt}\boldsymbol{X} = f(\boldsymbol{X}, t) = \begin{pmatrix} \vec{\boldsymbol{v}} \\ \vec{\boldsymbol{F}}(x, v)/m \end{pmatrix}$$

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

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{v}_N \end{pmatrix} \qquad f(\boldsymbol{X}, t) = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{F}^1(\boldsymbol{X}, t) \\ \vdots \\ \boldsymbol{v}_N \\ \boldsymbol{F}^N(\boldsymbol{X}, t) \end{pmatrix}$$

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
  - $\mathbf{F}^i$ denotes the force on particle *i*
    - When particles don't interact, $\mathbf{F}^i$ only depends on $\mathbf{x}_i$ and $\mathbf{v}_i$.

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{v}_N \end{pmatrix} \qquad f(\boldsymbol{X}, t) = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{F}^1(\boldsymbol{X}, t) \\ \vdots \\ \boldsymbol{v}_N \\ \boldsymbol{F}^N(\boldsymbol{X}, t) \end{pmatrix}$$

***f* gives d/dt X, remember!**

Path through a Vector Field

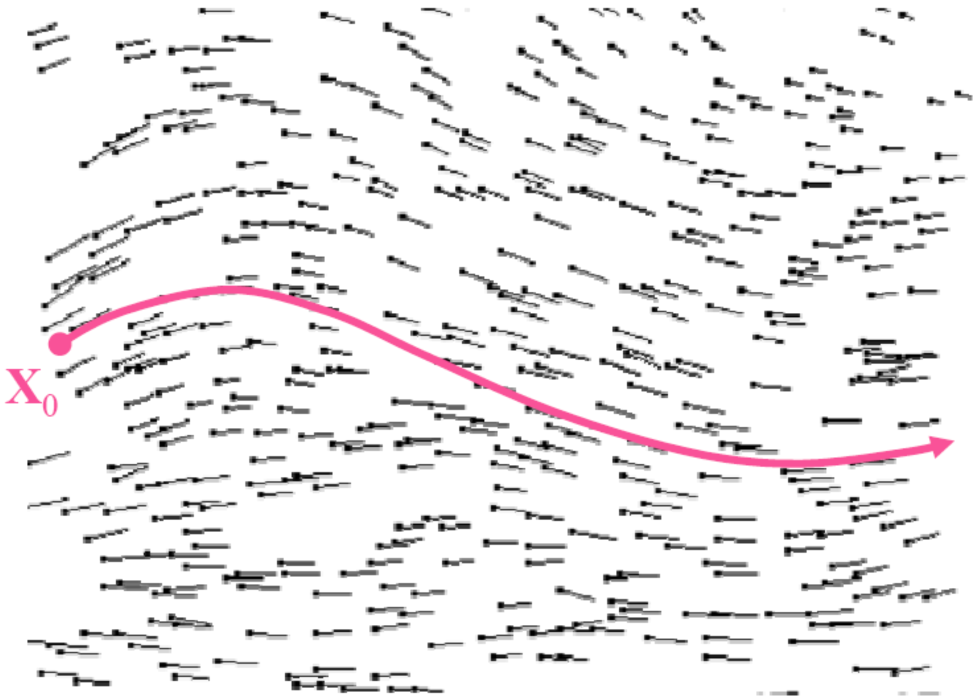
- ***X***(*t*): path in multidimensional <u>phase space</u>

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

"When we are at state **X** at time *t*, where will **X** be after an infinitely small time interval d*t* ?"

Path through a Vector Field

- ***X***(t): path in multidimensional <u>phase space</u>



$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

"When we are at state **X** at time *t*, where will **X** be after an infinitely small time interval d*t* ?"

- f=d/dt ***X*** is a vector that sits at each point in phase space, pointing the direction.

Questions?

http://vimeo.com/14597952

Numerics of ODEs

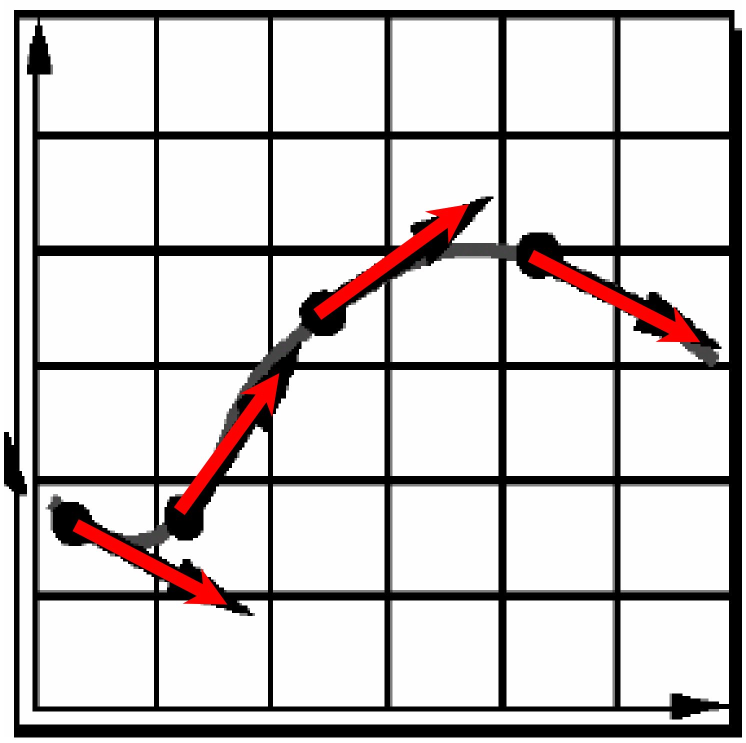
- Numerical solution is called “integration of the ODE”
- Many techniques
  - Today, the simplest one
  - Thursday and next week we’ll look at some more advanced techniques

Intuitive Solution: Take Steps

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$

$$\Rightarrow \text{“}\mathrm{d}\boldsymbol{X} = \mathrm{d}t\, f(\boldsymbol{X}, t)\text{”}$$



f = d/d*t* ***X*** is a vector that sits at each point in phase space, pointing the direction.

Euler's Method

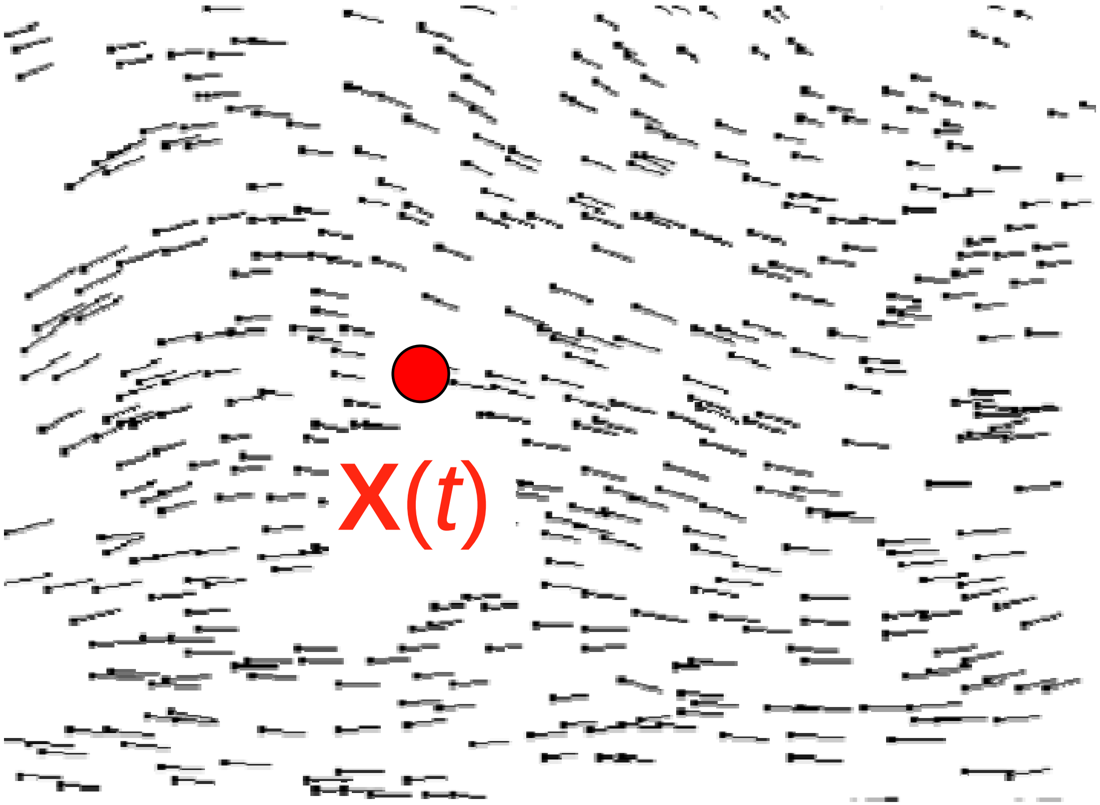
- Simplest and most intuitive
- Pick a **step size** h
- Given $\mathbf{X}_0 = \mathbf{X}(t_0)$, take step:

$$t_1 = t_0 + h$$
$$\mathbf{X}_1 = \mathbf{X}_0 + h\,f(\mathbf{X}_0, t_0)$$

- Piecewise-linear approximation to the path
- **Basically, just replace d*t* by a small but finite number**

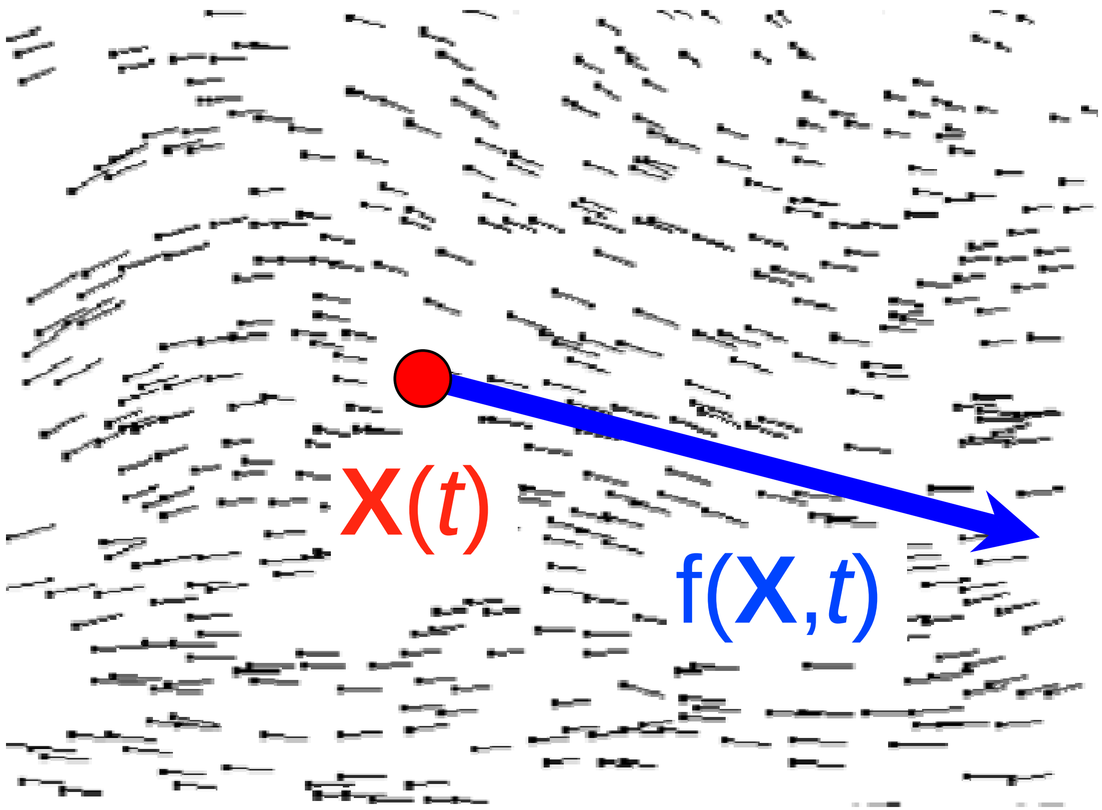
Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$



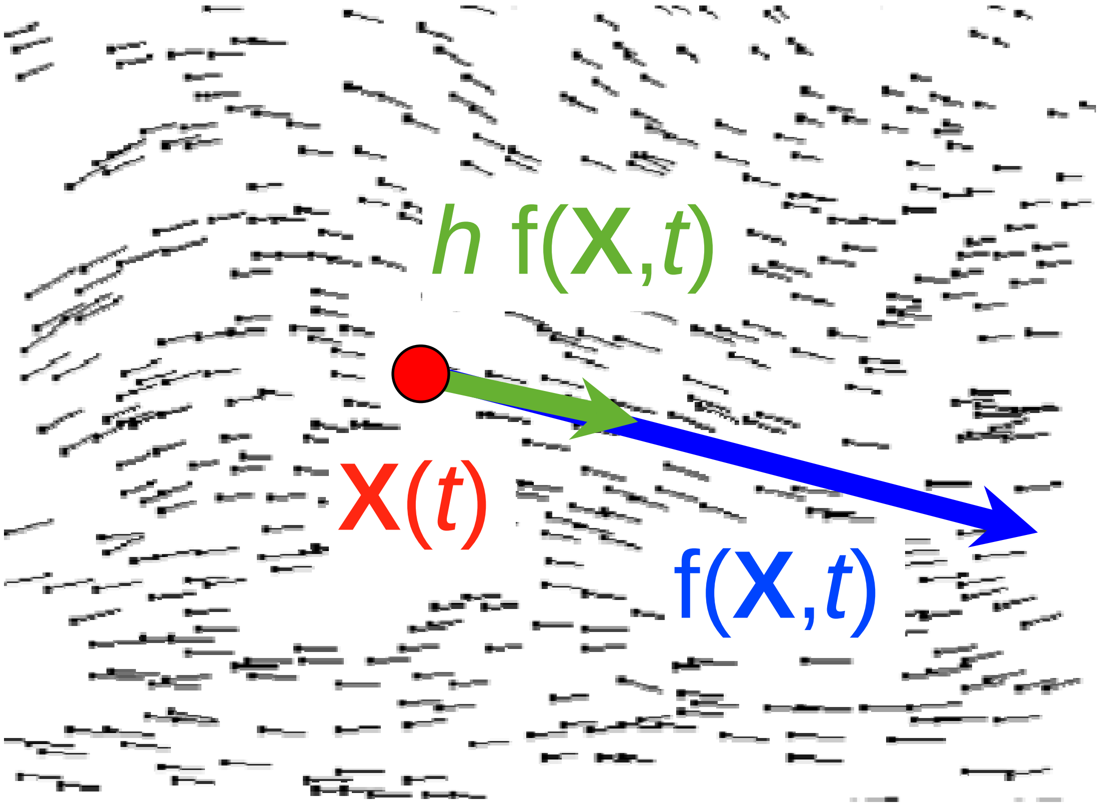
Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$



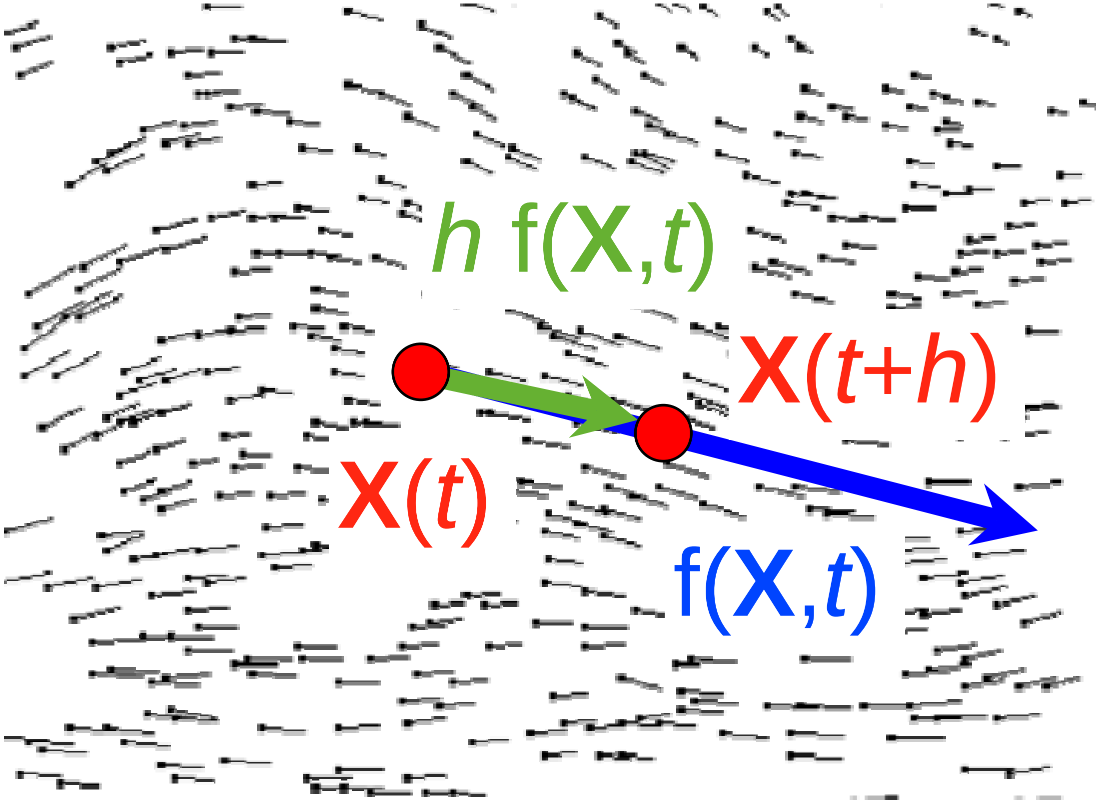
Euler, Visually

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$



Euler, Visually

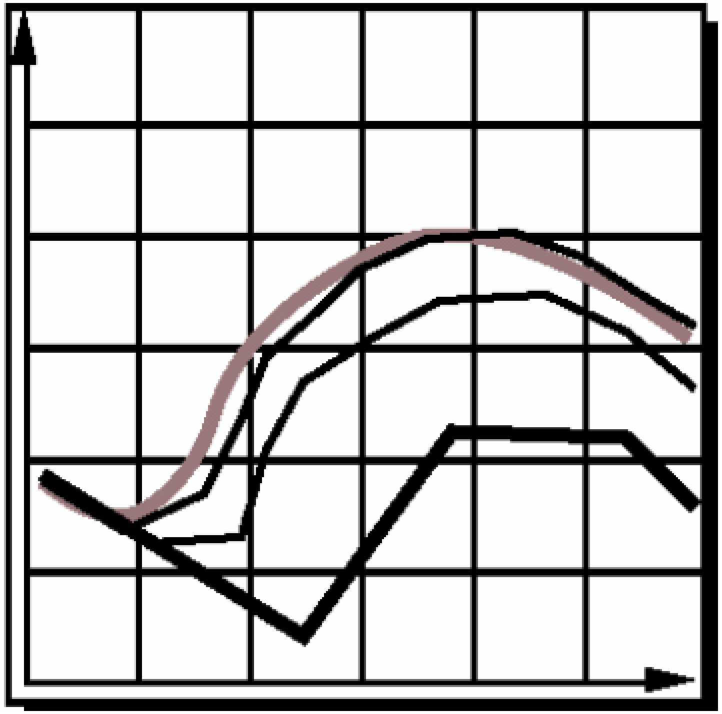
$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{X} = f(\boldsymbol{X}, t)$$



Questions?

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 $f(\mathbf{X}, t)$ determine this (more later)



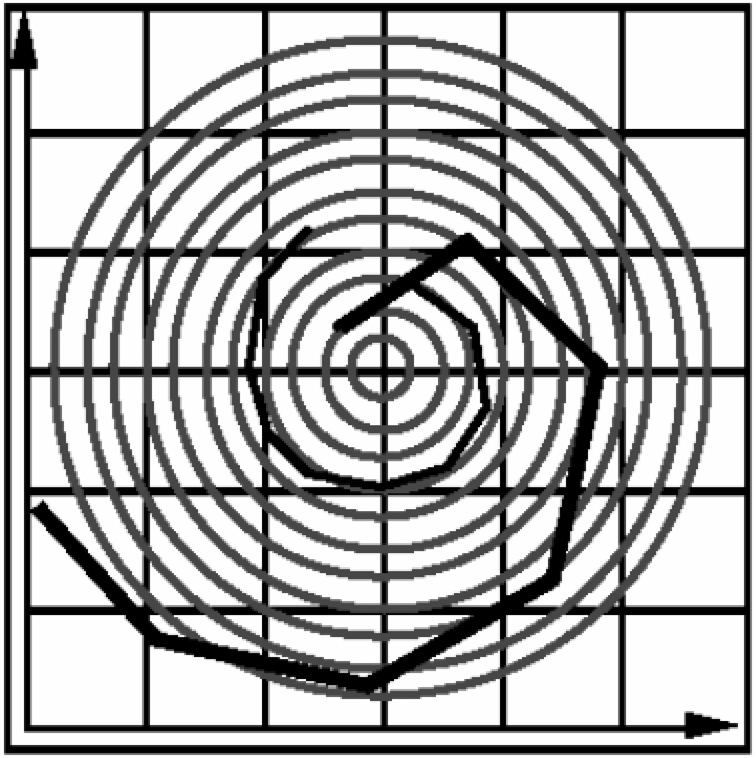
Euler's method: Inaccurate

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

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

- Exact solution is circle:

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



Euler's method: Inaccurate

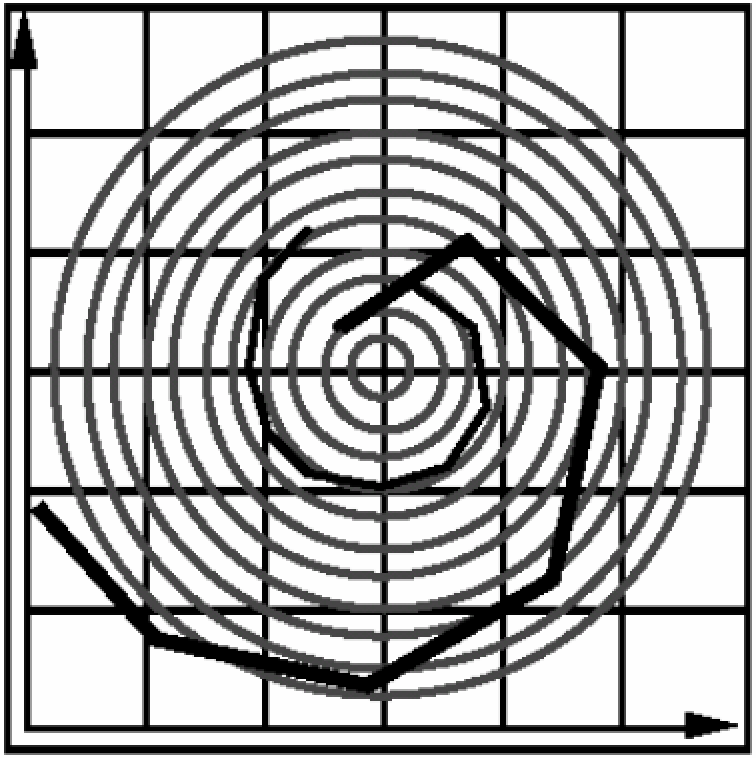
- Moves along tangent; can leave solution curve, e.g.:

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

- Exact solution is circle:

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

- Euler spirals outward
  no matter how small *h* is
  - will just diverge more slowly



More Accurate Alternatives

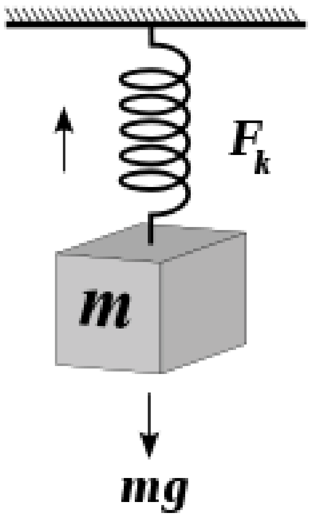
- Midpoint, Trapezoid, Runge-Kutta
  - Also, “implicit methods” (next week)

More on this during next class

- Extremely valuable resource: [SIGGRAPH 2001 course notes on physically based modeling]

What is a Force?

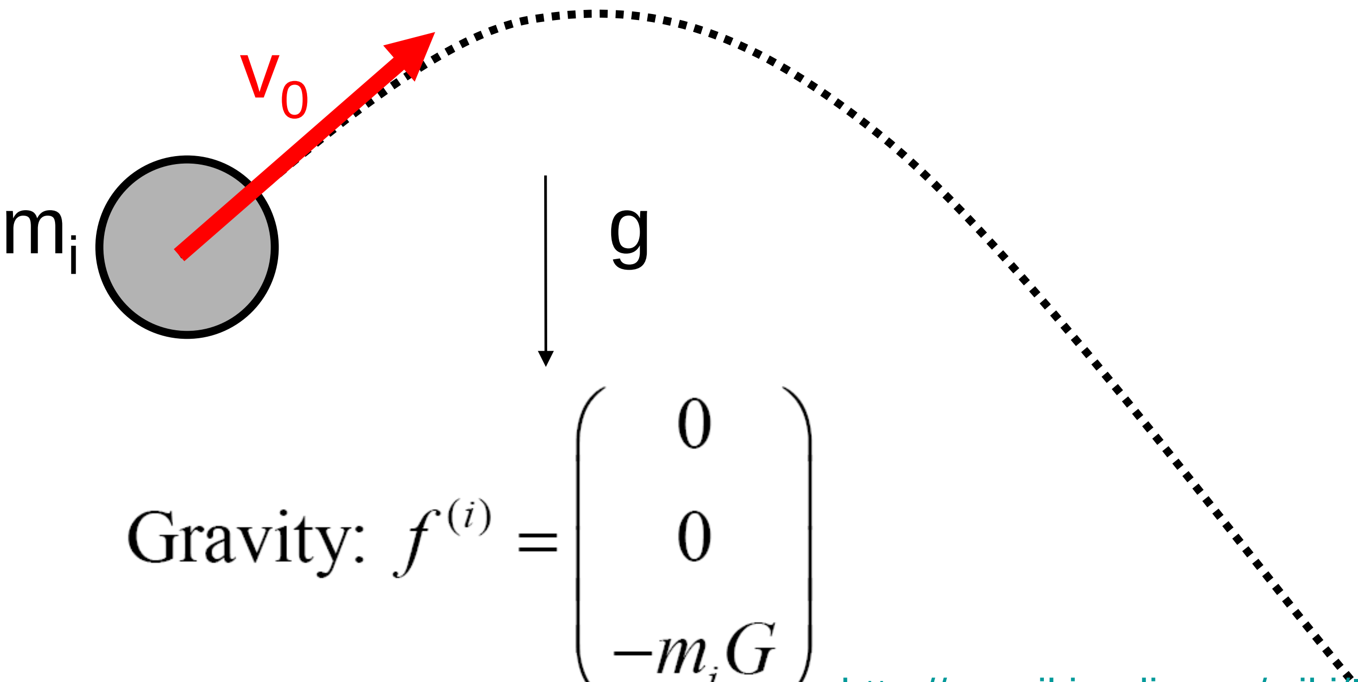
- A force changes the motion of the system
  - Newton says: When there are no forces, motion continues uniformly in a straight line (good enough for us)
- Forces can depend on location, time, velocity
  - Gravity, spring, viscosity, wind, etc.
- For point masses, forces are vectors
  - Ie., to get total force, take vector sum of everything



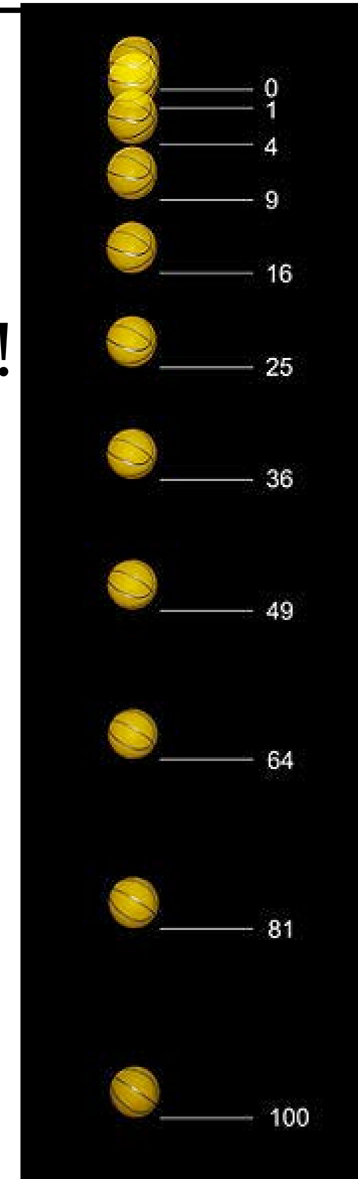
Wikipedia

Forces: Gravity on Earth

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



$$\text{Gravity: } f^{(i)} = \begin{pmatrix} 0 \\ 0 \\ -m_i G \end{pmatrix}$$



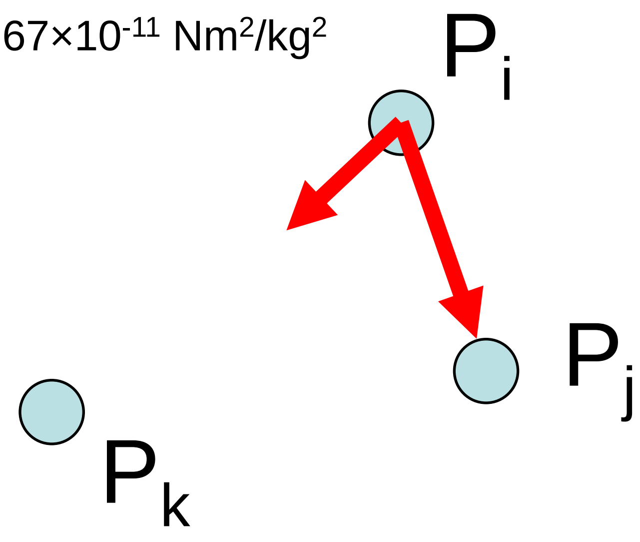
http://en.wikipedia.org/wiki/Image:Falling_ball.jpg

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

$$\|\boldsymbol{F}_{ij}\| = \frac{G\, m_i\, m_j}{r^2}$$

where $G=6.67\times10^{-11}$ Nm^2/kg^2



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

Particles are not independent!

Real-Time Gravity Demo

NVIDIA

http://www.youtube.com/watch?v=uhTuJZiAG64

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)!
  - "Fast Multipole Method", Greengard and Rokhlin, J Comput Phys 73, p. 325 (1987)
  - This inspired very cool hierarchical illumination rendering algorithms in graphics (hierarchical radiosity, etc.)

Forces: Viscous Damping

$$f^{(i)} = -dv^{(i)}$$

- 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

Bridson et al.

Processing demo

- http://processing.org/learning/topics/smokeparticlesystem.html

Smoke Particle System

This example is for Processing version 1.5+. If you have a previous version, use the examples included with your software. If you see any errors or have suggestions, »please let us know.

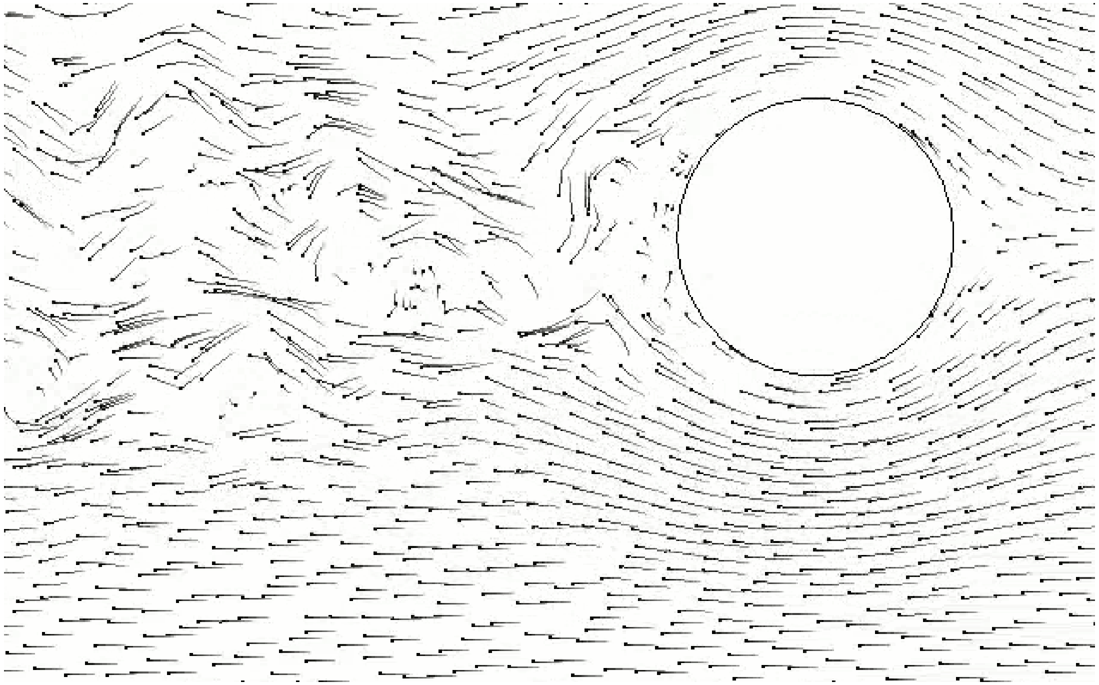
Smoke Particle System by Daniel Shiffman.

A basic smoke effect using a particle system. Each particle is rendered as an alpha masked image.

```
ParticleSystem ps;
Random generator;

void setup() {

  size(640, 200);
  colorMode(RGB, 255, 255, 255, 100);

  // Using a Java random number generator for Gaussian random numbers
  generator = new Random();

  // Create an alpha masked image to be applied as the particle's texture
  PImage msk = loadImage("texture.gif");
  PImage img = new PImage(msk.width,msk.height);
  for (int i = 0; i < img.pixels.length; i++) img.pixels[i] = color(255);
  img.mask(msk);
  ps = new ParticleSystem(0, new PVector(width/2,height-20 ),img);

  smooth();
}
```

Example: Procedural Spatial Field

- Curl noise for procedural fluid flow, R. Bridson, J. Hourihan, and M. Nordenstam, Proc. ACM SIGGRAPH 2007.



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

And it's simple, too!

Curl-Noise for Procedural Fluid Flow

Robert Bridson

Jim Hourihan

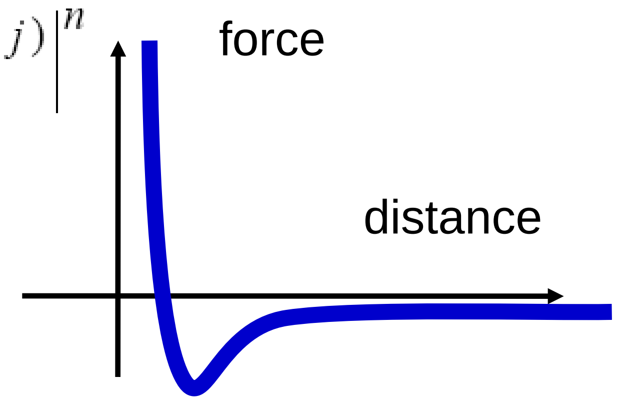
Markus Nordenstam

Forces: Other Spatial Interaction

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

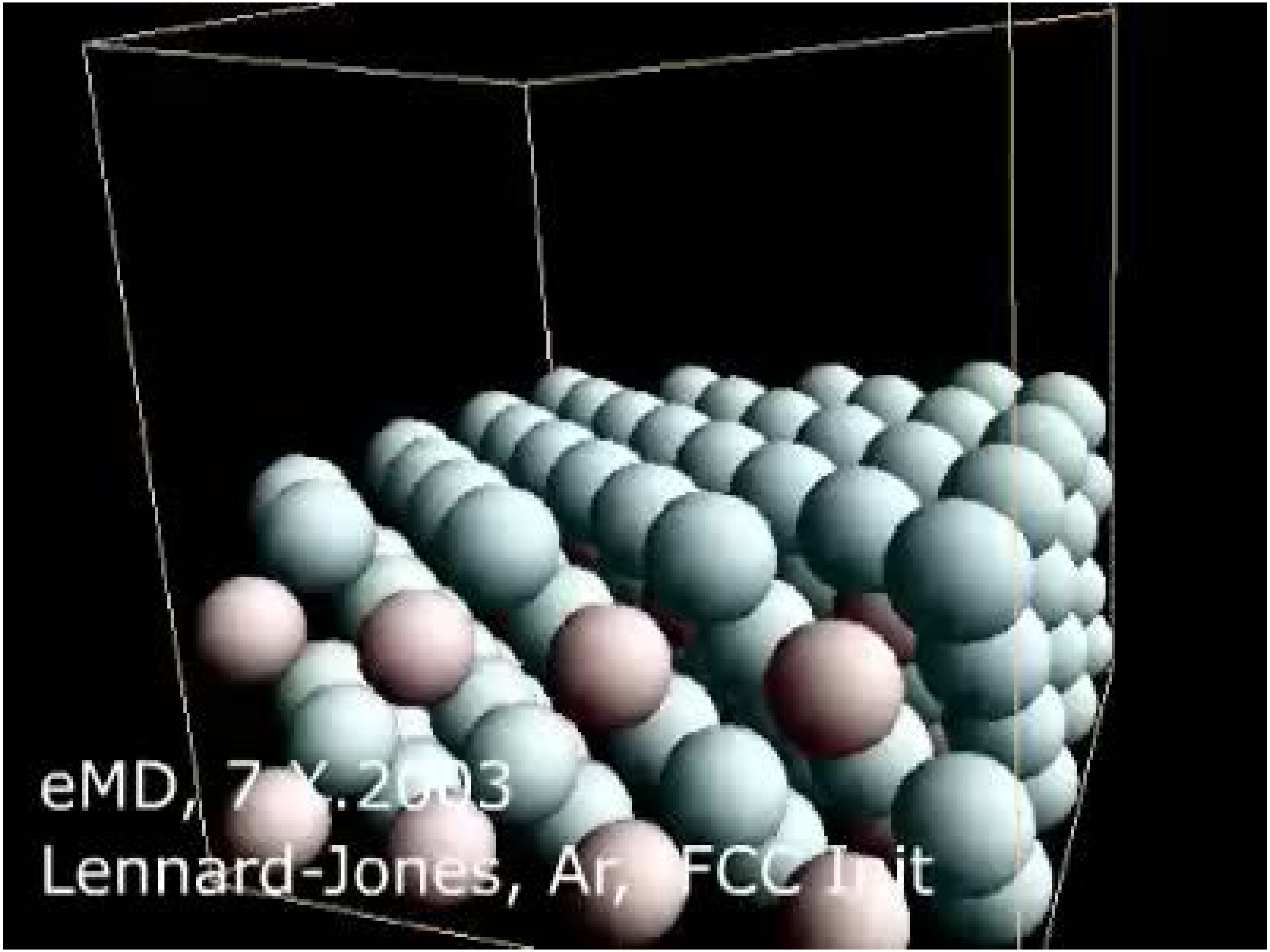
- E.g., approximate fluid using Lennard-Jones force:

$$f(x^{(i)}, x^{(j)}) = \frac{k_1}{\left|x^{(i)} - x^{(j)}\right|^m} - \frac{k_2}{\left|x^{(i)} - x^{(j)}\right|^n}$$



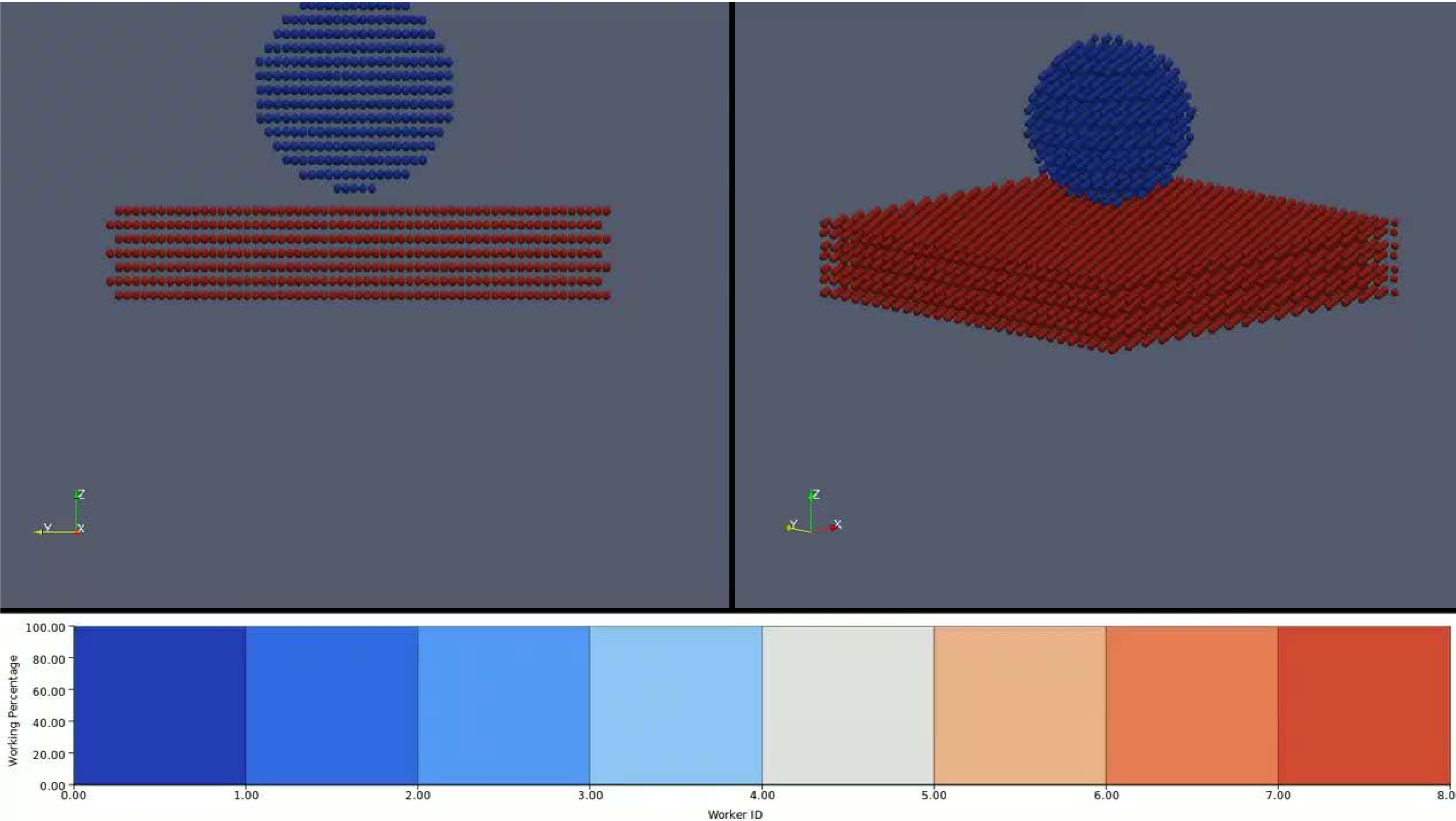
- Repulsive + attractive force
- Again, $O(N^2)$ to test all pairs
  - usually only local
  - Use buckets to optimize. Cf. 6.839

Particles are not independent!



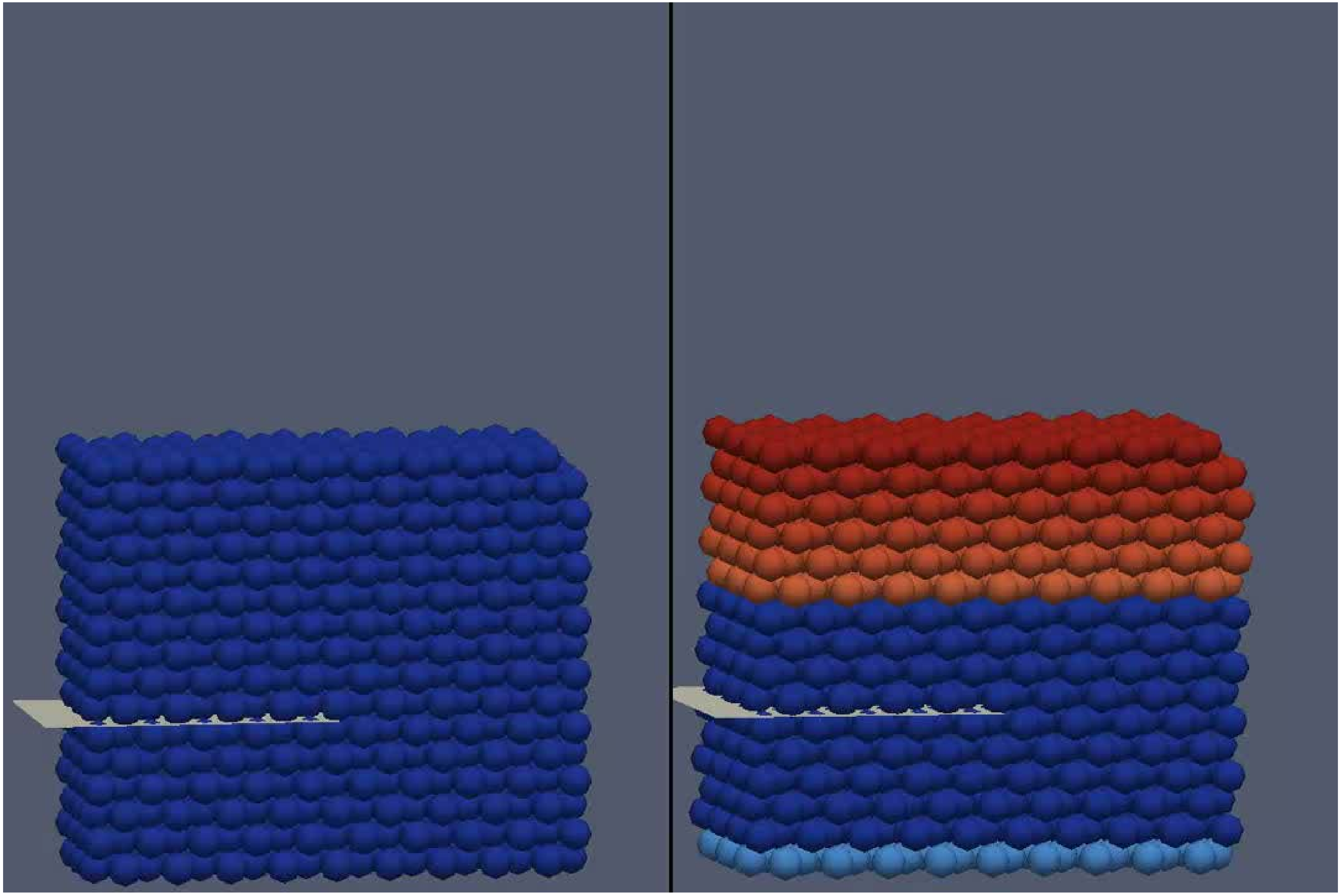
http://www.youtube.com/watch?v=nl7maklgYnI&feature=related

Lennard-Jones forces



http://www.youtube.com/watch?v=XfjYlKxKIWQ&feature=autoplay&list=PL0605C44C6E8D5EDB&lf=autoplay&playnext=2

Questions?

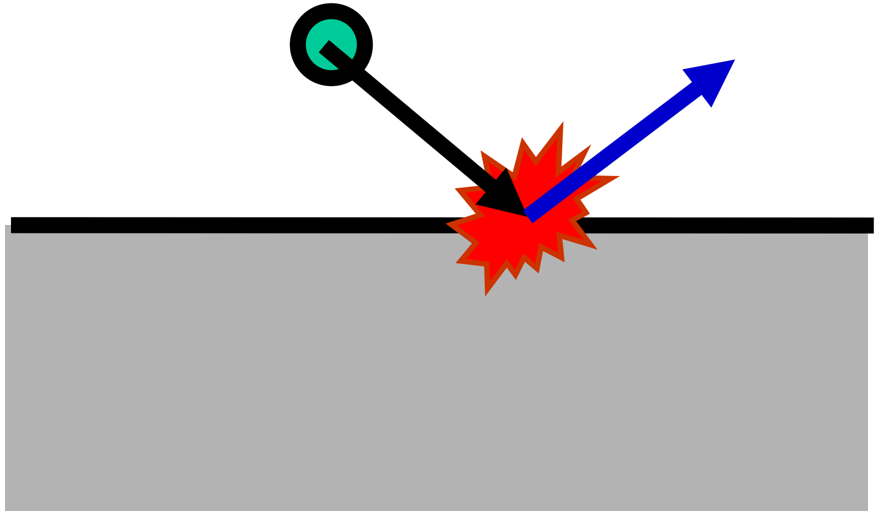


http://www.youtube.com/watch?v=dHWCT7RPjPo

Collisions

- Detection
- Response

- Covered later

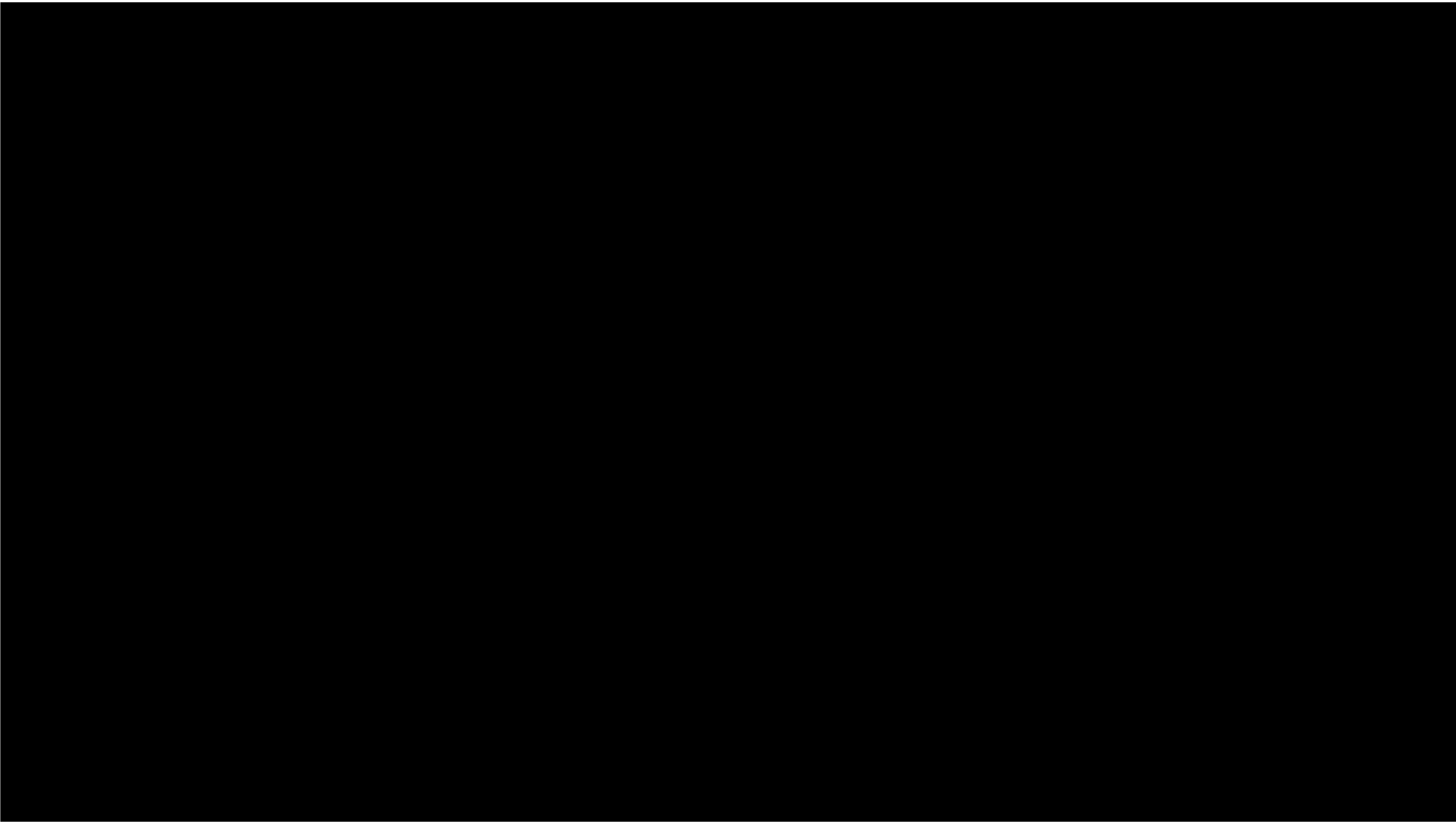


More Eyecandy from NVIDIA

- 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]

NVIDIA

More Eyecandy from NVIDIA



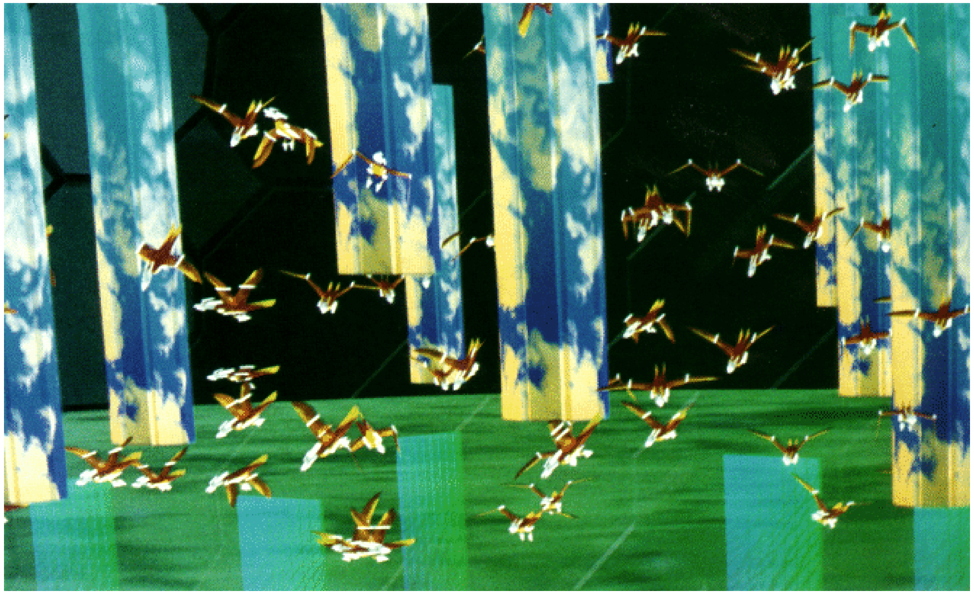
More Advanced “Forces”

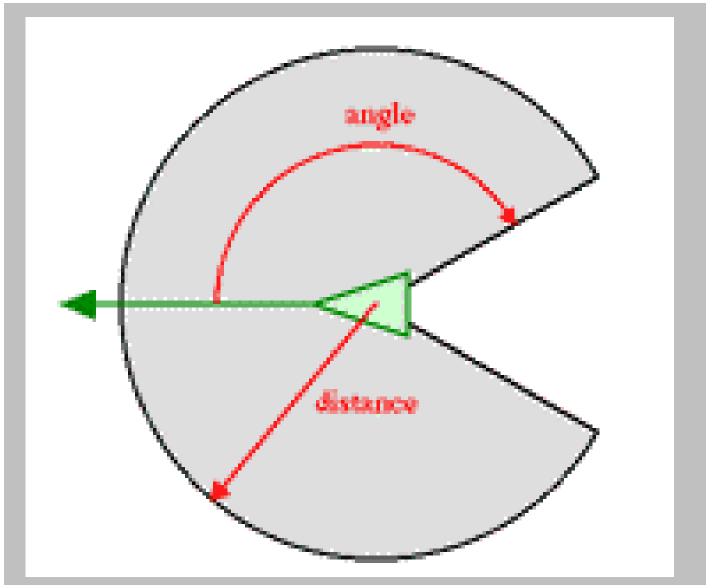
- Flocking birds, fish shoals
  - http://www.red3d.com/cwr/boids/
- Crowds (www.massivesoftware.com)

Battle of Helm’s Deep, LOTR

Flocks ("Boids")

- From Craig Reynolds
- Each bird modeled as a complex particle ("boid")
- A set of forces control its behavior
- Based on location of other birds and control forces

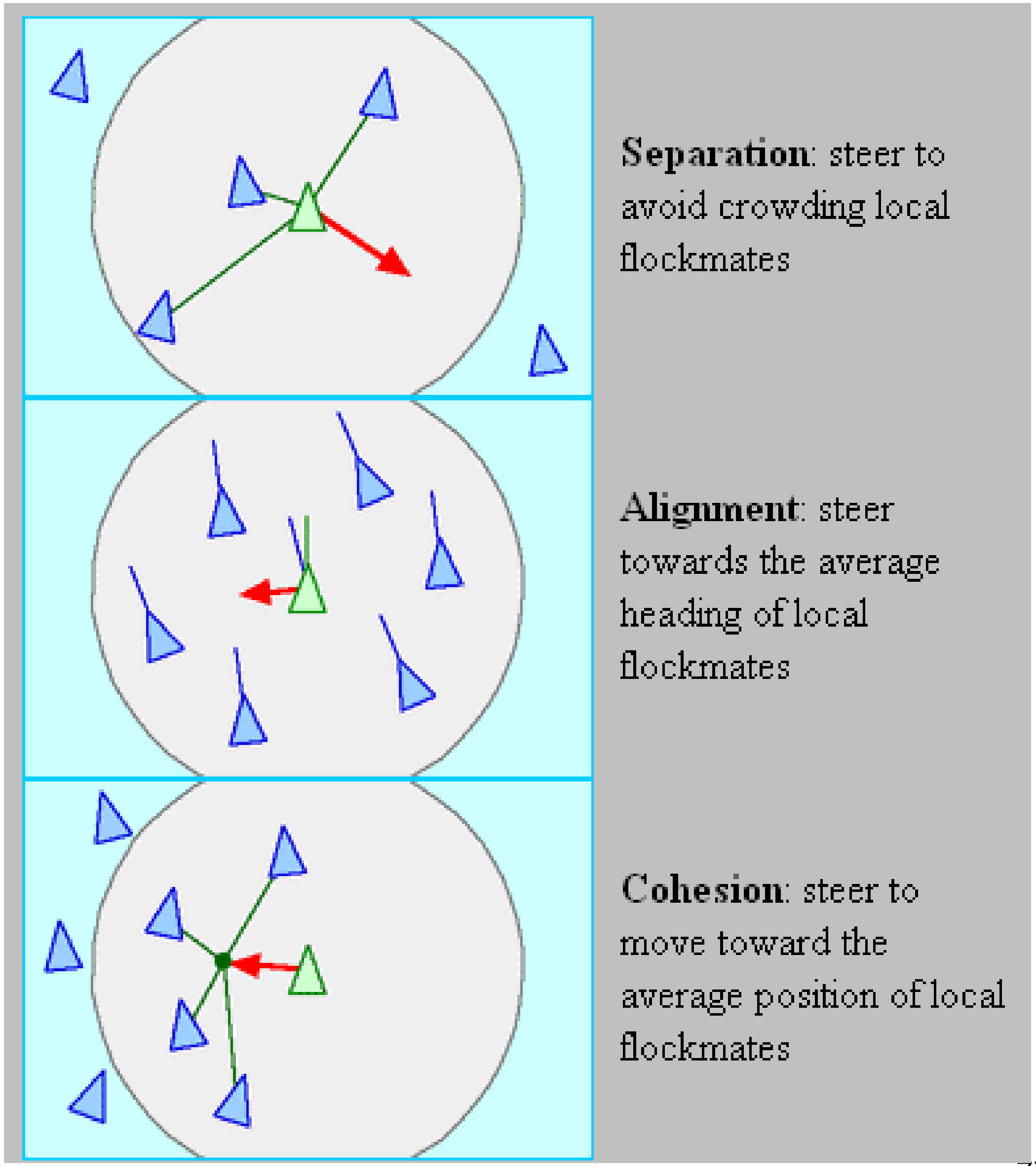




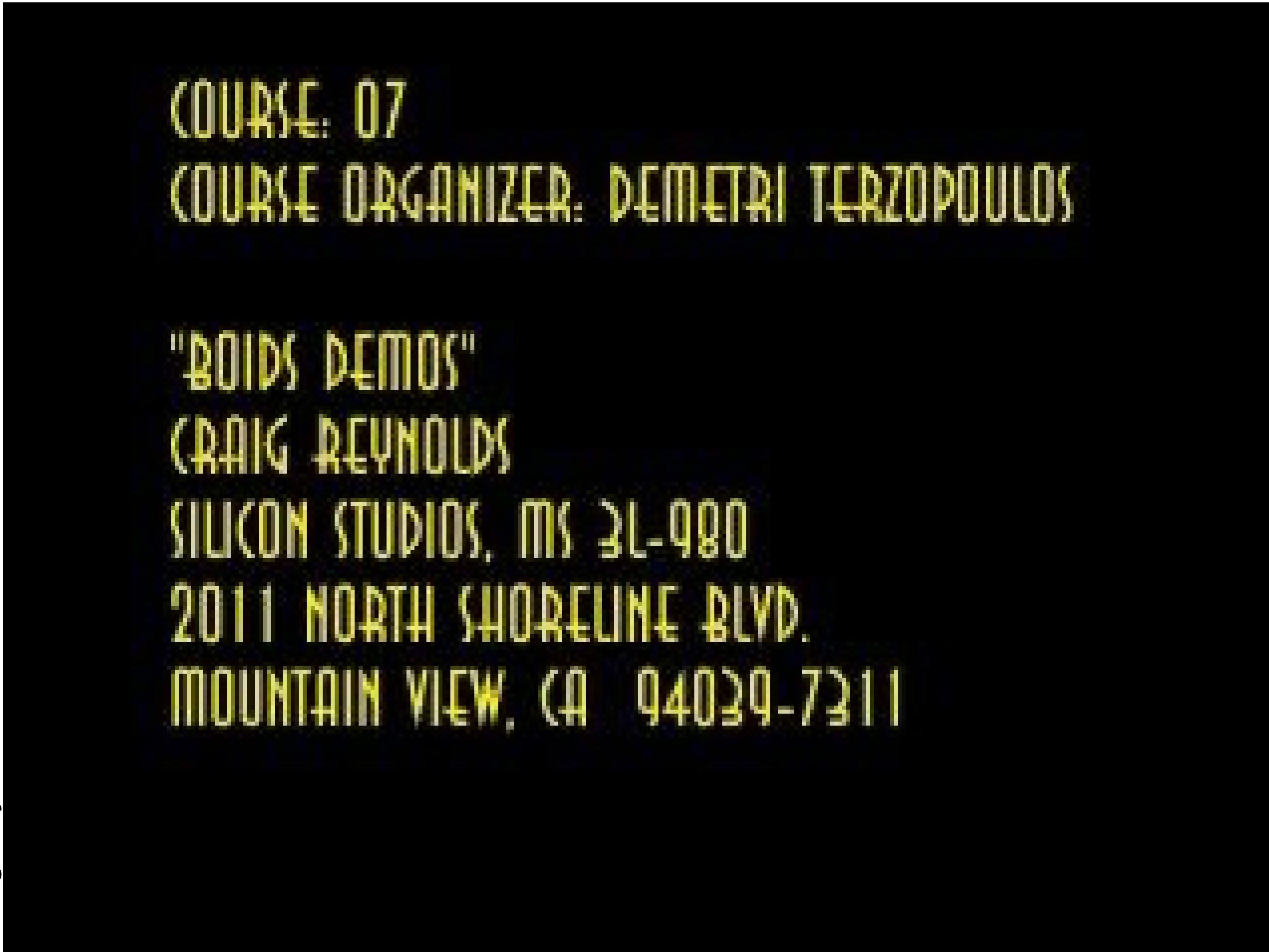
a boid's neighborhood

Flocks ("Boids")

- ("Boid" was an abbreviation of "birdoid". His rules applied equally to simulated flocking birds, and shoaling fish.)



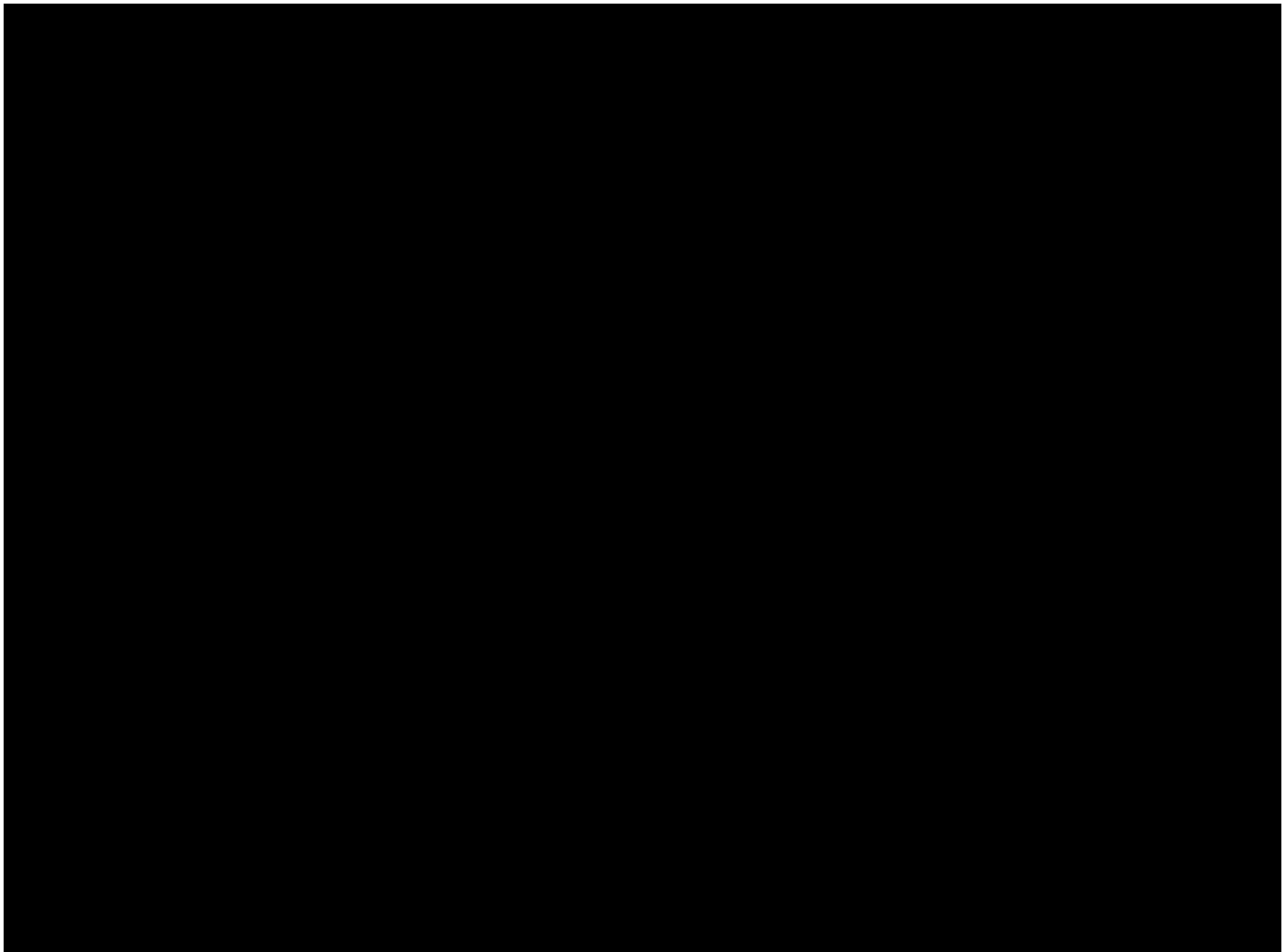
Flocks ("Boids")



Craig Reynolds

Predator-Prey

- http://www.youtube.com/watch?v=rN8DzlgMt3M



Massive software

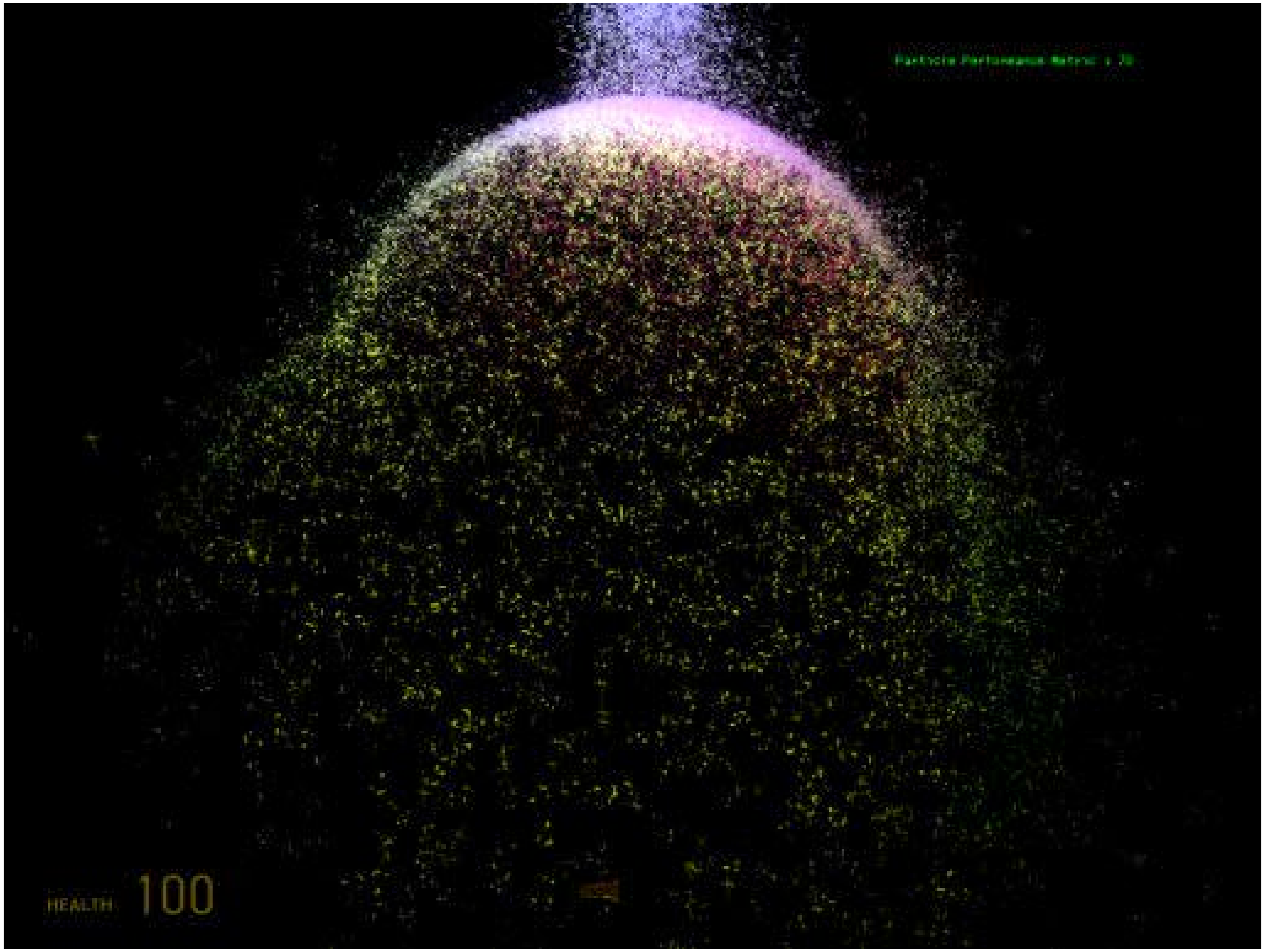
- http://www.massivesoftware.com/
- Used for battle scenes in the Lord of The Rings

Processing demo

- http://processing.org/learning/topics/flocking.html

Battle of the Helm's deep, LOTR

Questions?



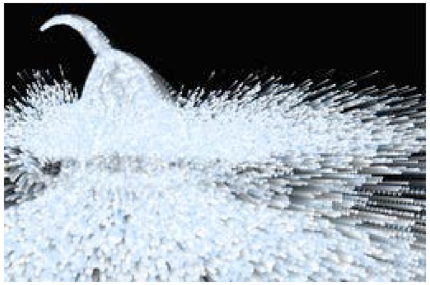
Valve Software

Where do particles come from?

- Often created by generators or **emitters**
  - Can be attached to objects in the model
- Given rate of creation: particles/second
  - record t_{last} of last particle created

$$n = \lfloor (t - t_{last})\, rate \rfloor$$

  - create n particles.
    update t_{last} if $n > 0$
- Create with (random) distribution of initial $\boldsymbol{x}$ and $\boldsymbol{v}$
  - if creating $n > 1$ particles at once, spread out on path

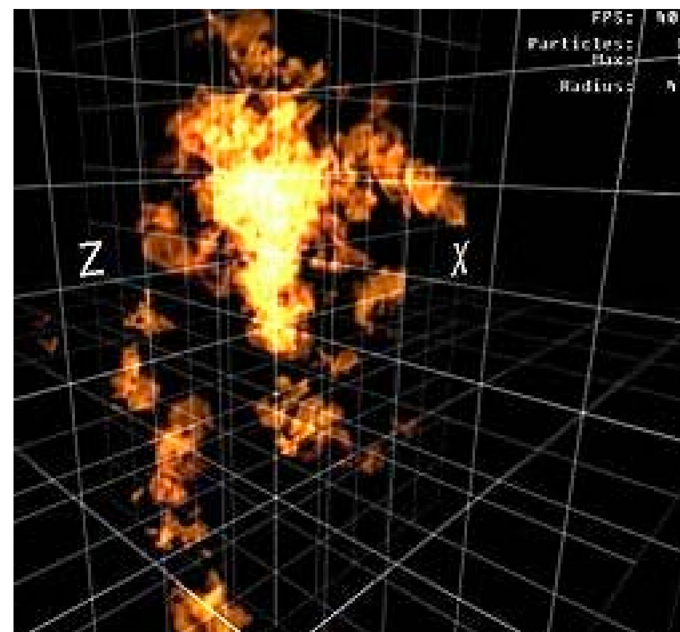


http://www.particlesystems.org/

Particle Controls

MAX PAYNE 2

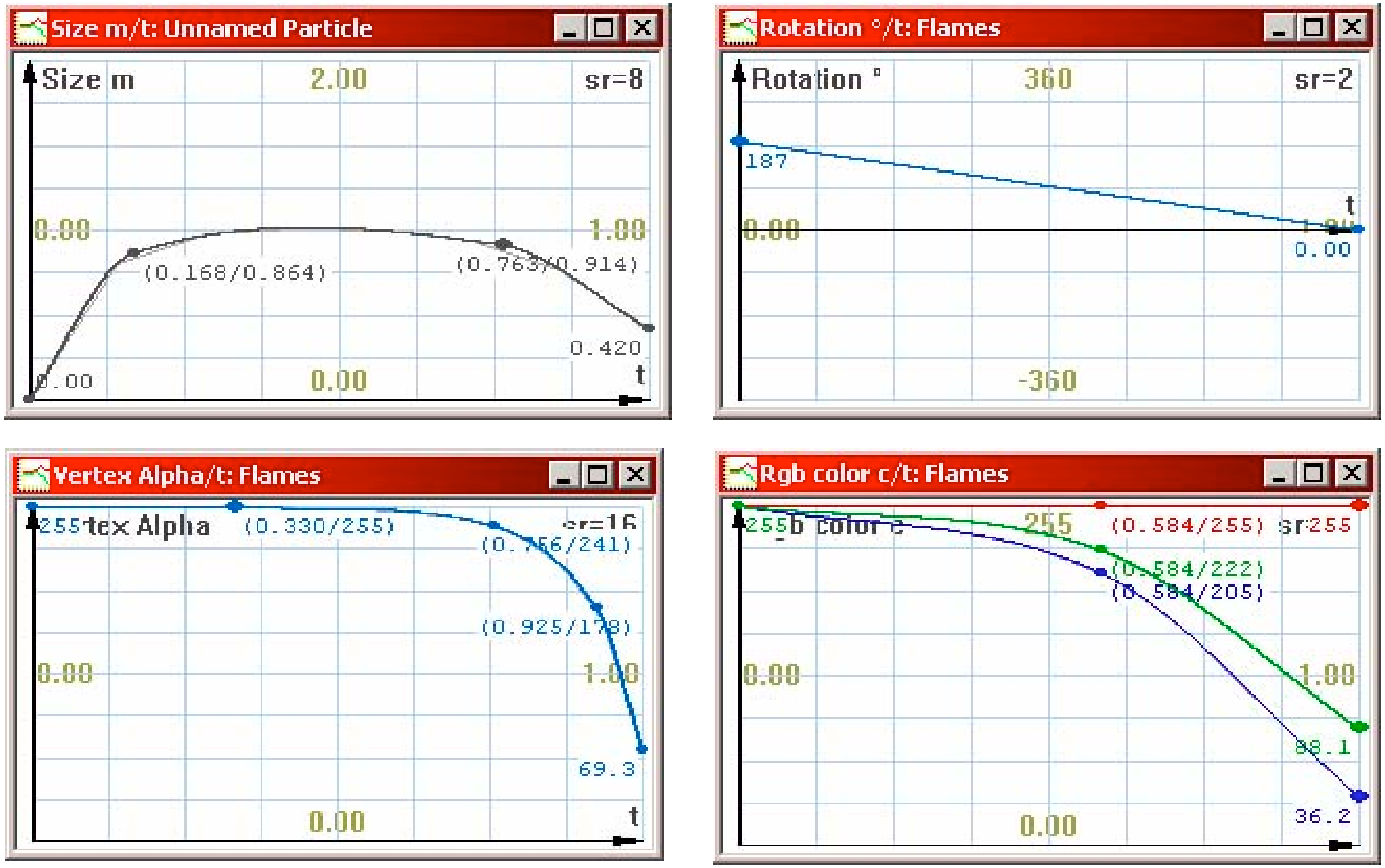
- In production tools, all these variables are time-varying and controllable by the user (artist)
  - Emission rate, color, velocity distribution, direction spread, textures, etc. etc.
    - All as a function of time!
  - Example: ParticleFX (Max Payne Particle Editor)
    - Custom editor software
    - You can **download it** (for Windows) and easily create your own particle systems. Comes with examples!
    - This is what we used for all the particles in the game!



Emitter Controls

MAX PAYNE 2

- Again, reuse splines!

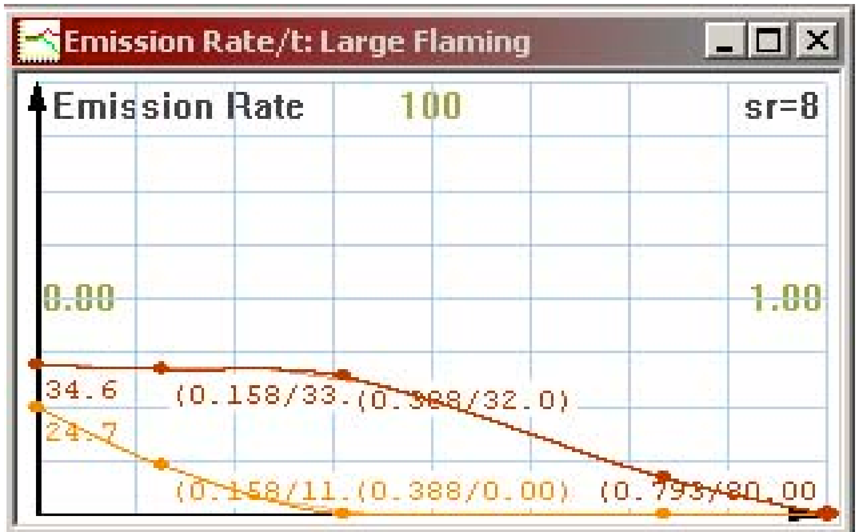


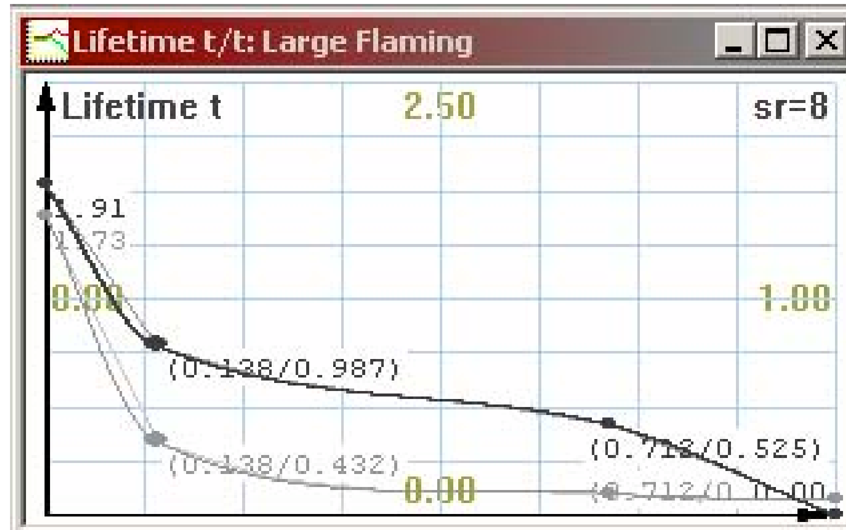
Controls from ParticleFX by Remedy Entertainment

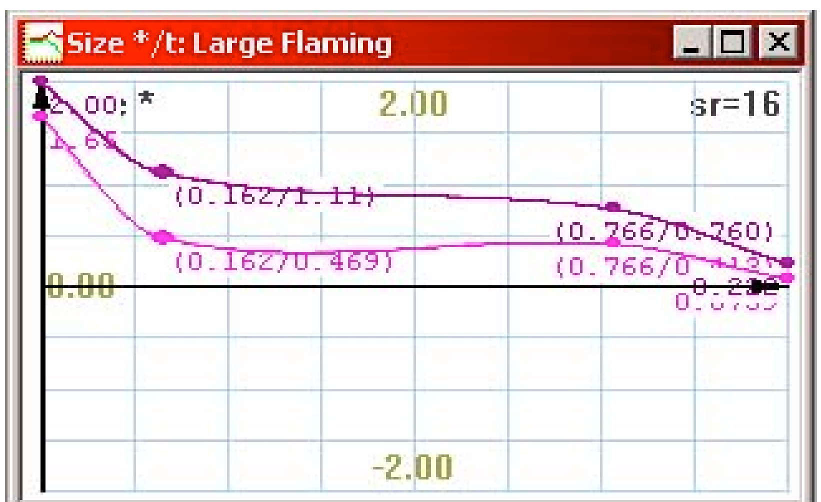
Emitter Controls

MAX PAYNE 2

- Again, reuse splines!

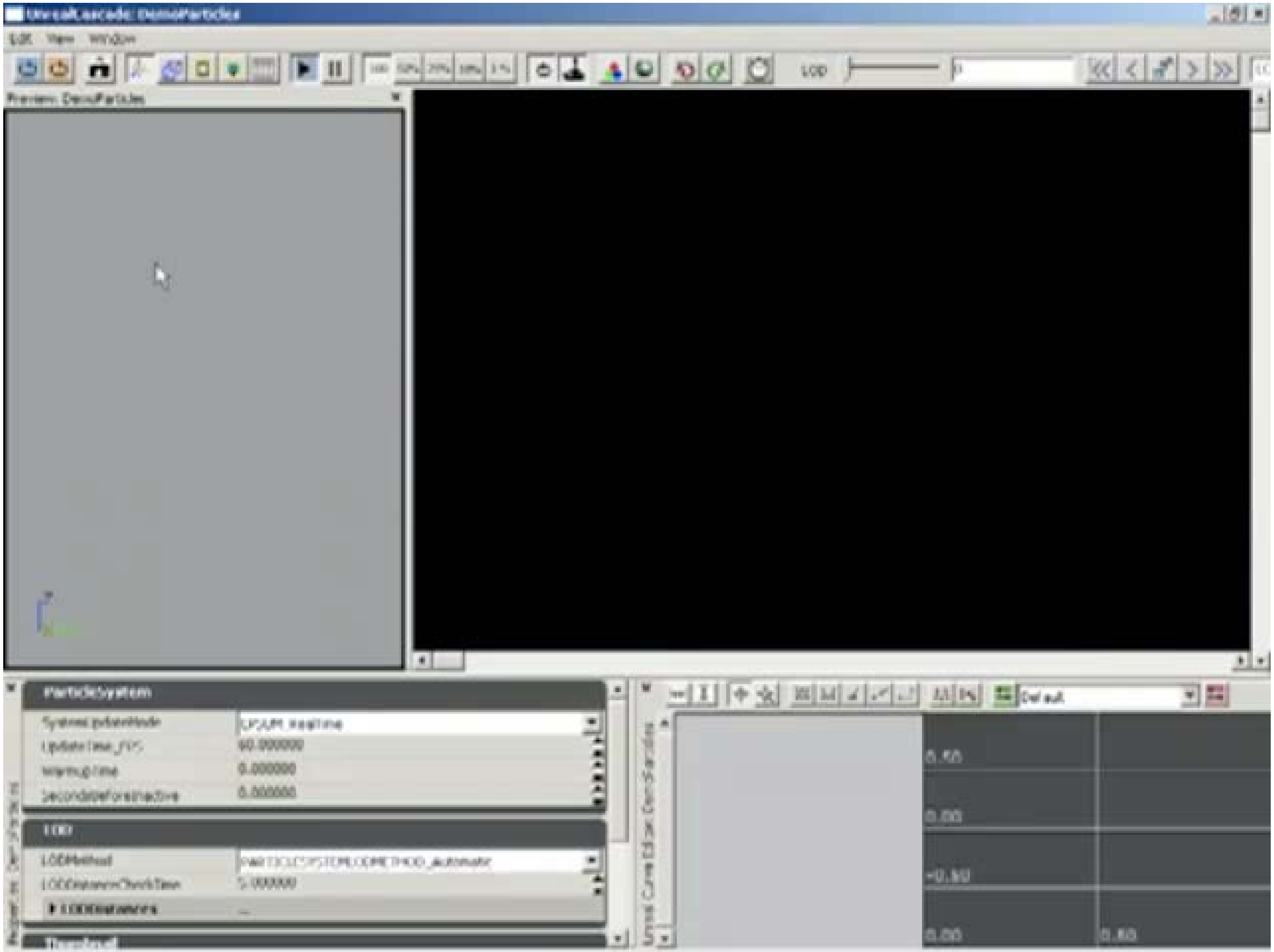






Controls from ParticleFX by Remedy Entertainment

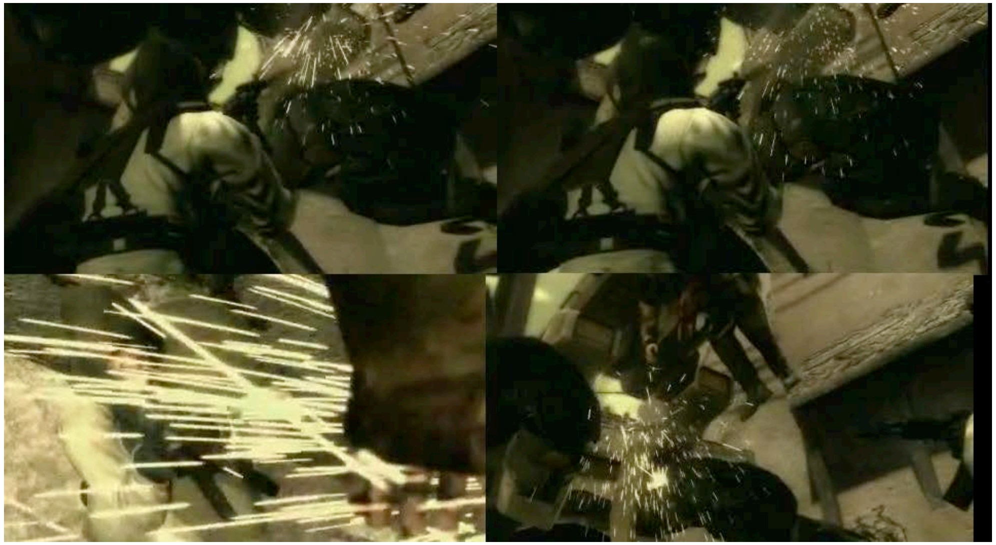
Unreal Engine



Unreal Engine

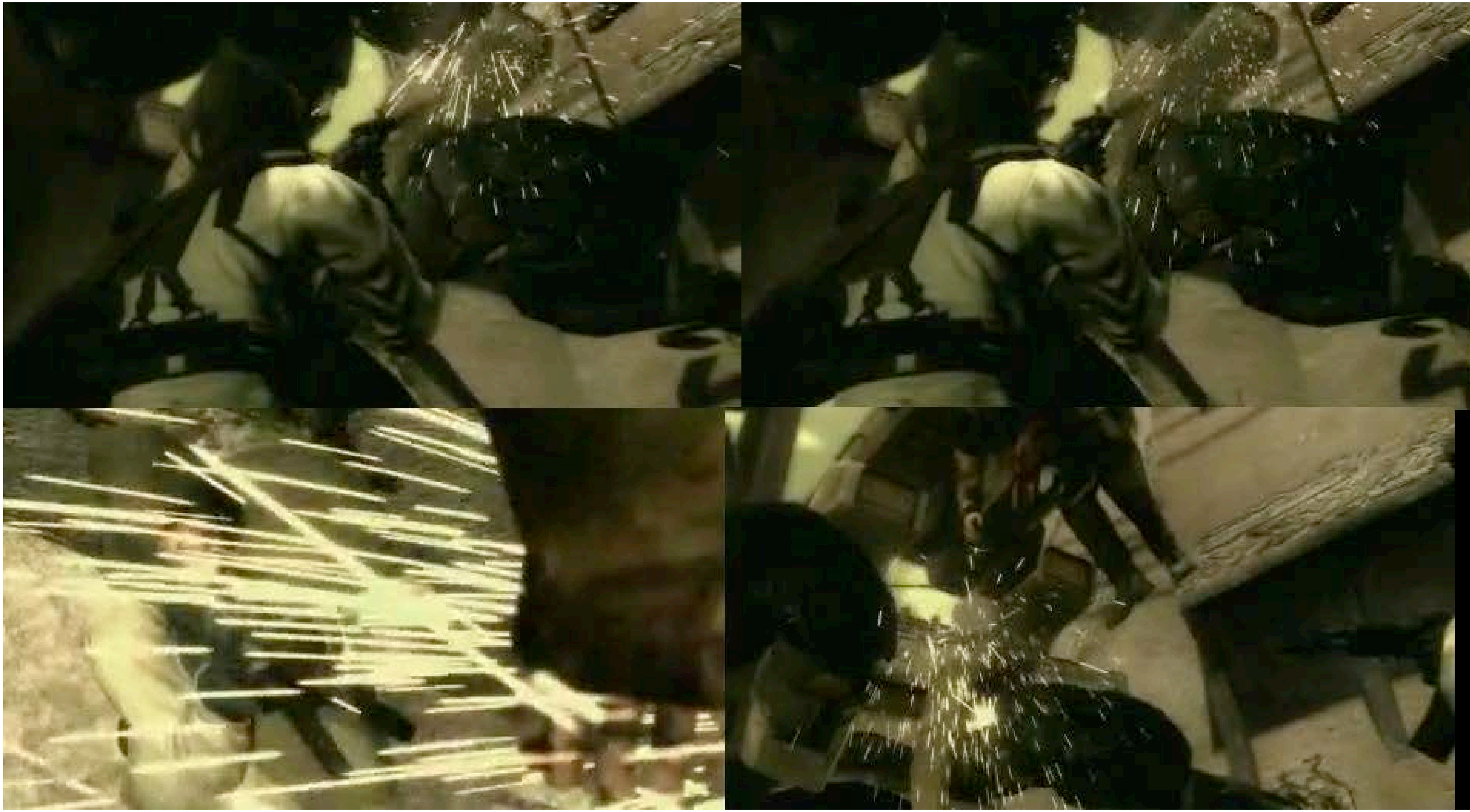
Rendering and Motion Blur

- Often not shaded (just emission, think sparks)
  - But realistic non-emissive particles needs shadows, etc.
- Most often, particles don’t contribute to the z-buffer, i.e., they do not fully occlude stuff that’s behind
  - Rendered with z testing on (particles get occluded by solid stuff)
- Draw a line for motion blur
  - ($\mathbf{x}$, $\mathbf{x}+\mathbf{v}\,dt$)
  - Or an elongated quad with texture



Metal Gear Solid by Konami

Rendering and Motion Blur



Metal Gear Solid by Konami

Rendering and Motion Blur

Futuremark Corp., used with permission

- Often use texture maps (fire, clouds, smoke puffs)
  - Called “billboards” or “sprites”
  - Always parallel to image plane

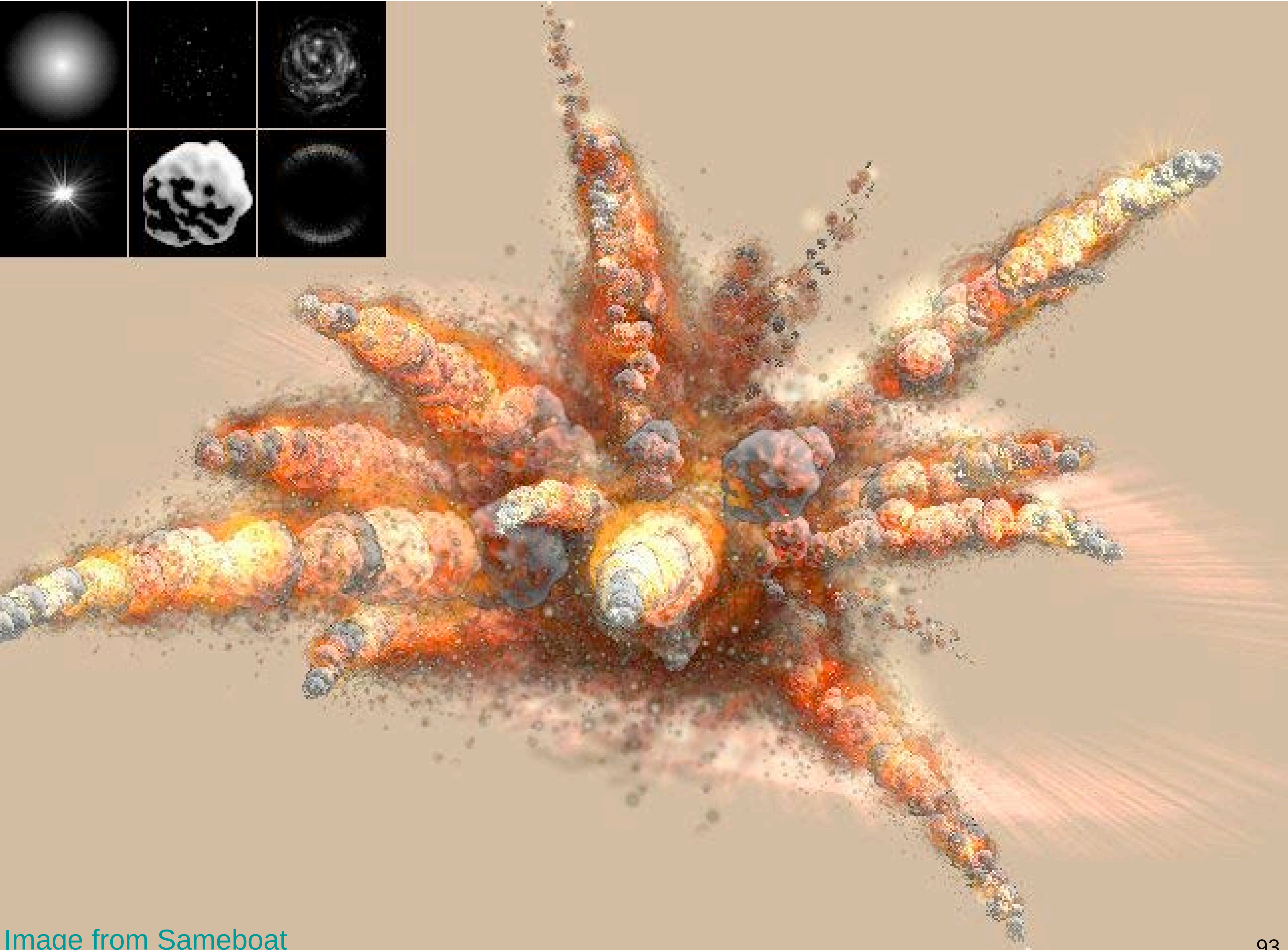


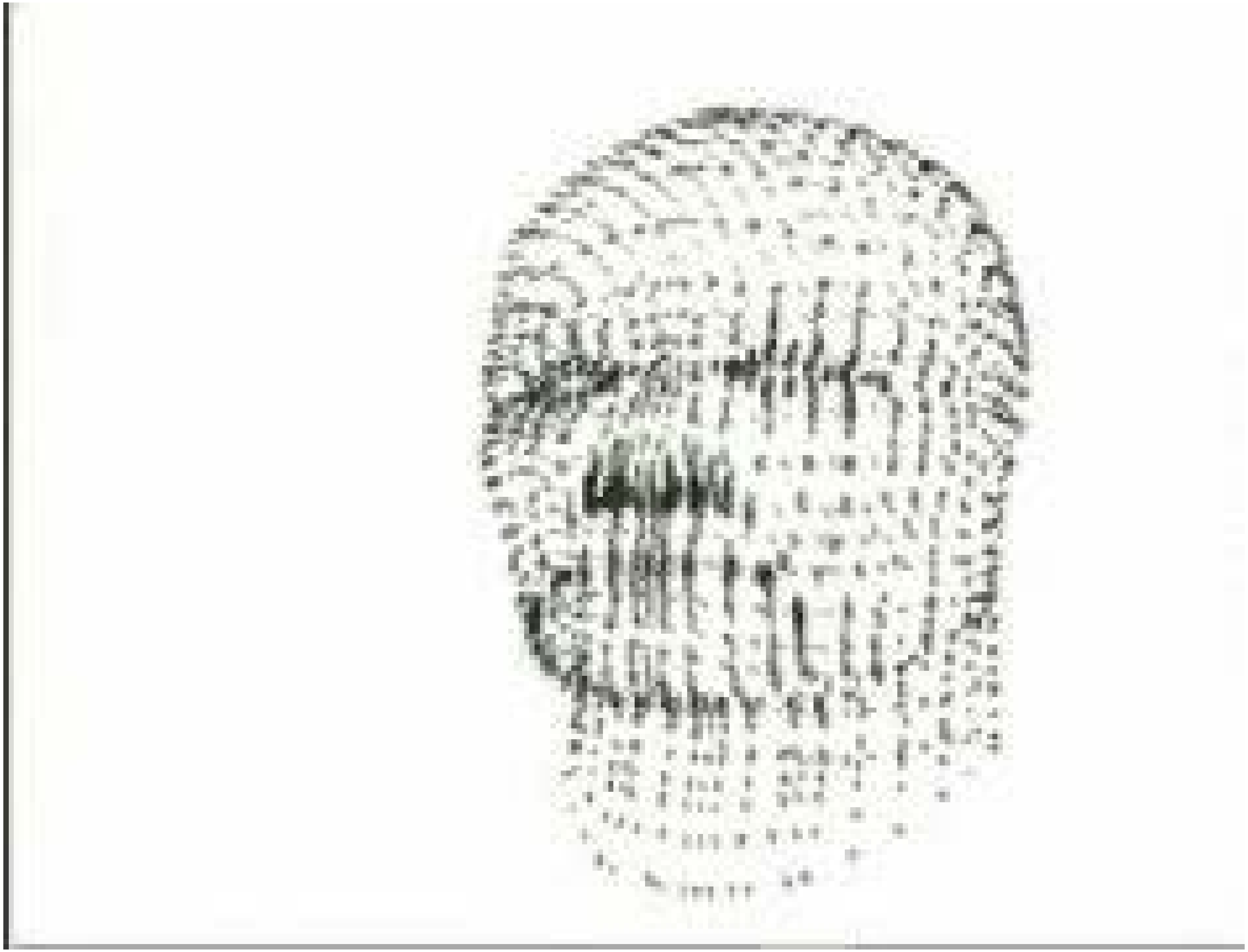
Image from Sameboat

Star Trek 2 – The Wrath of Khan

- One of the earliest particle systems (from 1982)
- Also, fractal landscapes
- Described in [Reeves, 1983]

Paramount Pictures

Questions?



Early particle fun by Karl Sims

That's All for Today!

- Further reading
  - Witkin, Baraff, Kass: Physically-based Modeling Course Notes, SIGGRAPH 2001
    - **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