Computer Animation III

Quaternions

Dynamics

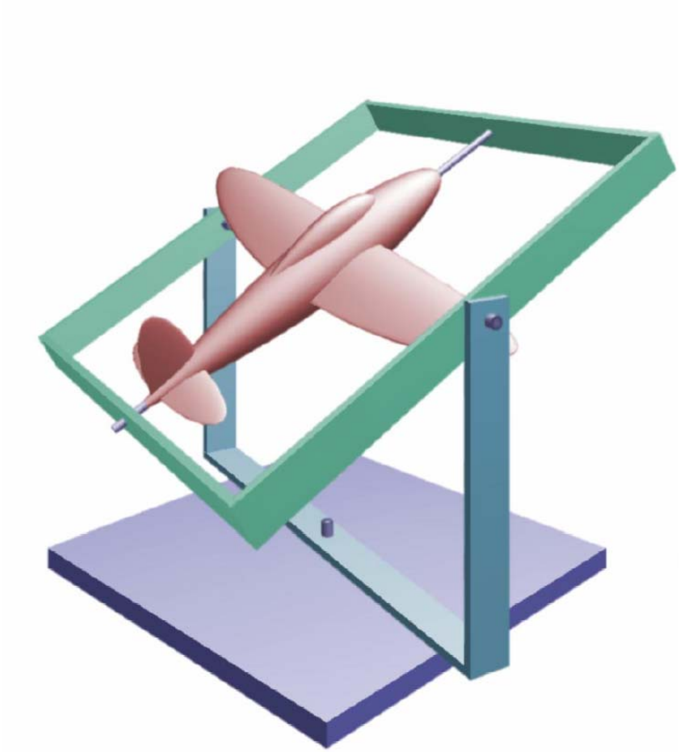
Some slides courtesy of
Leonard McMillan and
Jovan Popovic

Recap: Euler angles

3 angles along 3 axis

Poor interpolation, lock

But used in flight simulation, etc. because natural

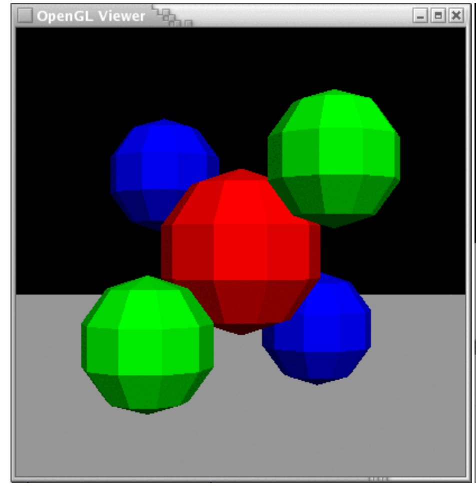


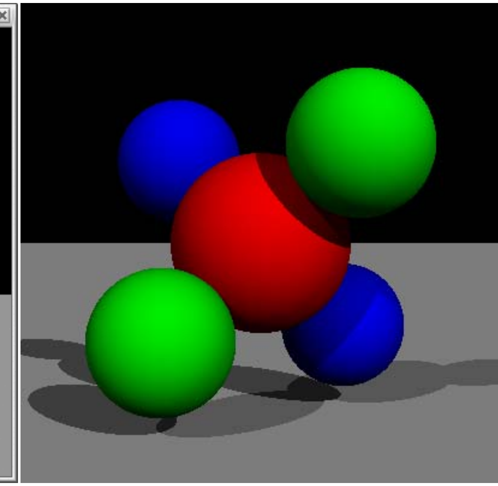
http://www.fho-emden.de/~hoffmann/gimbal09082002.pdf

Assignment 5: OpenGL

Interactive previsualization

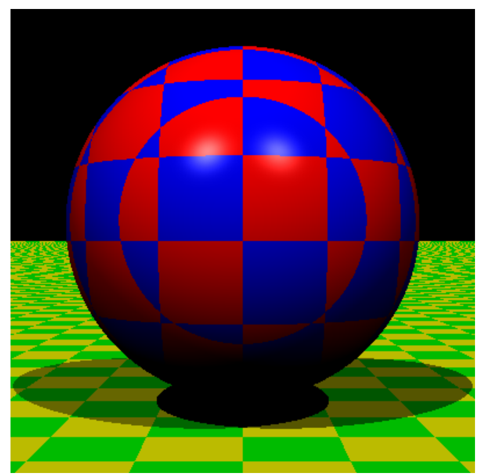
- OpenGL API
- Graphics hardware
- Jusr send rendering commands
- State machine

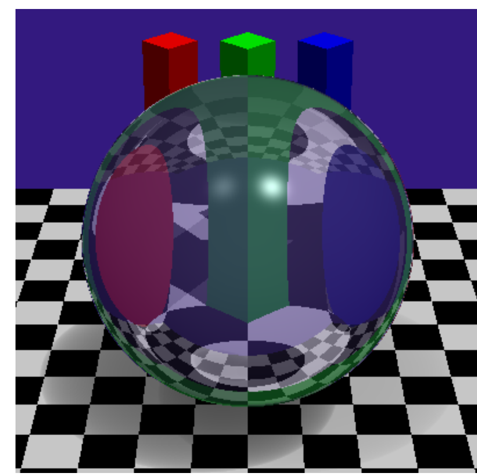




Solid textures

- New Material subclass
- Owns two Material*
- Chooses between them
- “Shader tree”





Final project

First brainstorming session on Thursday

Groups of three

Proposal due Monday 10/27

- A couple of pages
- Goals
- Progression

Appointment with staff

Final project

Goal-based

- Simulate a visual effect
- Natural phenomena
- Small animation
- Game
- Reconstruct an existing scene

Technique-based

- Monte-Carlo Rendering
- Radiosity
- Fluid dynamics

Overview

Interpolation of rotations, quaternions

- Euler angles
- Quaternions

Dynamics

- Particles
- Rigid body
- Deformable objects

Quaternion principle

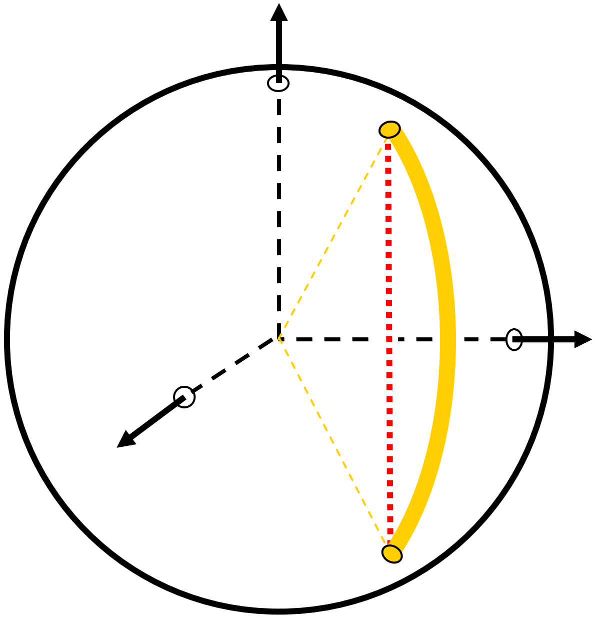
A quaternion = point on unit 3-sphere in 4D = orientation.

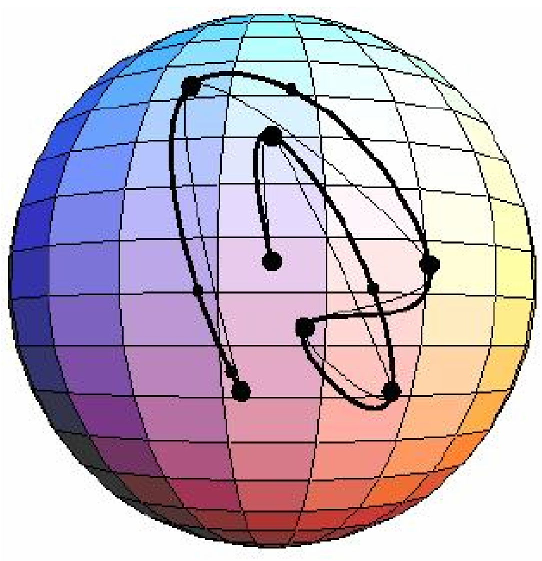
We can apply it to a point, to a vector, to a ray

We can convert it to a matrix

We can interpolate in 4D and project back onto sphere

- How do we interpolate?
- How do we project?

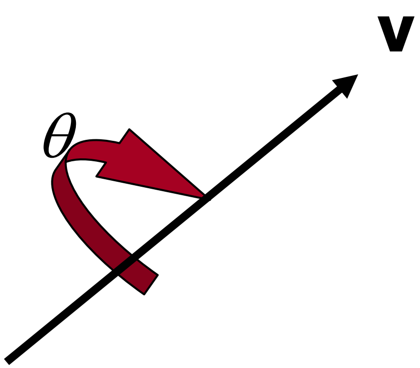




Quaternion recap 1 (wake up)

4D representation of orientation

$$\mathbf{q} = \{\cos(\theta/2);\ \mathbf{v}\sin(\theta/2)\}$$



Inverse is $\mathbf{q}^{-1} = (s, -\mathbf{v})$

Multiplication rule

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1 s_2 - \left(\vec{v}_1 \bullet \vec{v}_2\right), s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2\right)$$

- Consistent with rotation composition

How do we apply rotations?

How do we interpolate?

Quaternion Algebra

Two general quaternions are multiplied by a special rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1 s_2 - \left(\vec{v}_1 \bullet \vec{v}_2\right), s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2\right)$$

Sanity check : {cos($\alpha/2$); **v** sin($\alpha/2$)} {cos($\beta/2$); **v** sin($\beta/2$)}

{cos($\alpha/2$)cos($\beta/2$) - sin($\alpha/2$)**v**. sin($\beta/2$)} **v**,
cos($\beta/2$) sin($\alpha/2$) **v** + cos($\alpha/2$)sin($\beta/2$) **v** +**v** $\times$ **v**}

{cos($\alpha/2$)cos($\beta/2$) - sin($\alpha/2$) sin($\beta/2$),
v(cos($\beta/2$) sin($\alpha/2$) + cos($\alpha/2$) sin($\beta/2$))}

{cos($(\alpha+\beta)/2$), **v** sin($(\alpha+\beta)/2$) }

Quaternion Algebra

Two general quaternions are multiplied by a special rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1 s_2 - \left(\vec{V}_1 \bullet \vec{V}_2\right), s_1\vec{V}_2 + s_2\vec{V}_1 + \vec{V}_1 \times \vec{V}_2\right)$$

To rotate 3D point/vector **p** by **q**, compute

- **q** {0; **p**} $\mathbf{q}^{-1}$

p= (x,y,z) **q**={ cos(θ/2), 0,0,sin(θ/2) } = {c, 0,0,s}

q {0,**p**}= {c, 0, 0, s} {0, x, y, z}

= {c.0- zs, c**p**+0(0,0,s)+ (0,0,s) × **p**}

= {-zs, c **p** + (-sy,sx,0) }

q {0,**p**} **q** $^{-1}$ = {-zs, c **p** + (-sy,sx,0) } {c, 0,0,-s}

= {-zsc-(c**p**+(-sy,sx,0)).(0,0,-s),

-zs(0,0,-s)+c(c**p**+(-sy, sx,0))+ (c **p** + (-sy,sx,0)) x (0,0,-s) }

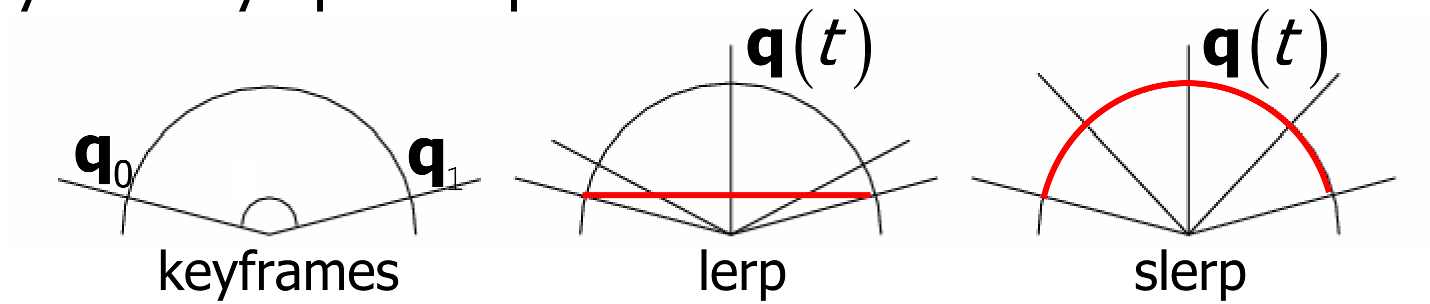
= {0, $(0,0,zs^2)+c^2\mathbf{p}+(-csy, csx,0)+(-csy, csx, 0)+(s^2x, s^2y, 0)$}

= {0, $(c^2x-2csy-s^2x, c^2y+2csx-s^2y, zs^{2+}sc^2)$}

= {0, x cos(θ)-y sin(θ), x sin(θ)+y cos(θ), z }

Quaternion Interpolation (velocity)

The only problem with **l**inear int**erp**olation (lerp) of quaternions is that it interpolates the straight line (the secant) between the two quaternions and not their spherical distance. As a result, the interpolated motion does not have smooth velocity: it may speed up too much in some sections:



Spherical **l**inear int**erp**olation (slerp) removes this problem by interpolating along the arc lines instead of the secant lines.

$$\text{slerp}\left(\mathbf{q}_0, \mathbf{q}_1, t\right) = \mathbf{q}(t) = \frac{\mathbf{q}_0 \sin\left(\left(1-t\right)\omega\right) + \mathbf{q}_1 \sin\left(t\omega\right)}{\sin\left(\omega\right)},$$

$$\text{where } \omega = \cos^{-1}\left(\mathbf{q}_0 \cdot \mathbf{q}_1\right)$$

Back
Next

Quaternions

Can also be defined like complex numbers

a+bi+cj+dk

Multiplication rules

- $i^2=j^2=k^2=-1$
- ij=k=-ji
- jk=i=-kj
- ki=j=-ik

…

Fun:Julia Sets in Quaternion space

Mandelbrot set: $Z_{n+1}=Z_n^2+Z_0$

Julia set $Z_{n+1}=Z_n^2+C$

http://aleph0.clarku.edu/~djoyce/julia/explorer.html

Do the same with Quaternions!

Rendered by Skal (Pascal Massimino) http://skal.planet-d.net/

Images removed due to copyright considerations.

See also http://www.chaospro.de/gallery/gallery.php?cat=Anim

Fun:Julia Sets in Quaternion space

Julia set $Z_{n+1}=Z_n^2+C$

Do the same with Quaternions!

Rendered by Skal (Pascal Massimino) http://skal.planet-d.net/

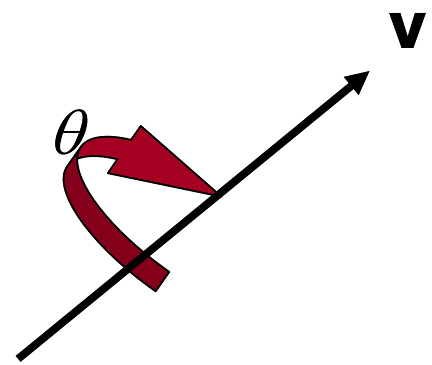
This is 4D, so we need the time dimension as well

Images removed due to copyright considerations.

Recap: quaternions

3 angles represented in 4D

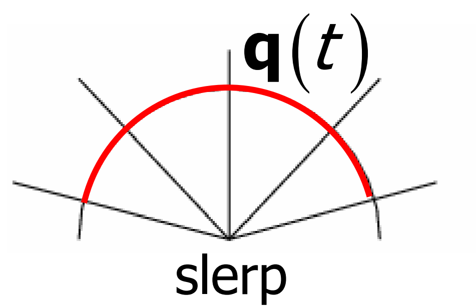
$$\mathbf{q} = \{\cos(\theta/2);\ \mathbf{v}\sin(\theta/2)\}$$

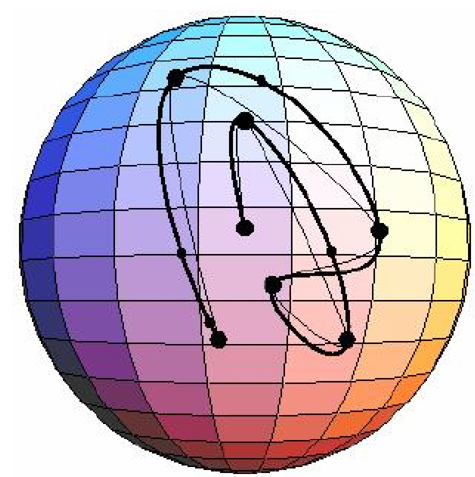


Weird multiplication rules

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1 s_2 - \left(\vec{v}_1 \bullet \vec{v}_2\right), s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2\right)$$

Good interpolation using slerp





Back Next

Overview

Interpolation of rotations, quaternions

- Euler angles
- Quaternions

Dynamics

- Particles
- Rigid body
- Deformable objects

Break: movie time

Pixar *For the Bird*

Now

Dynamics

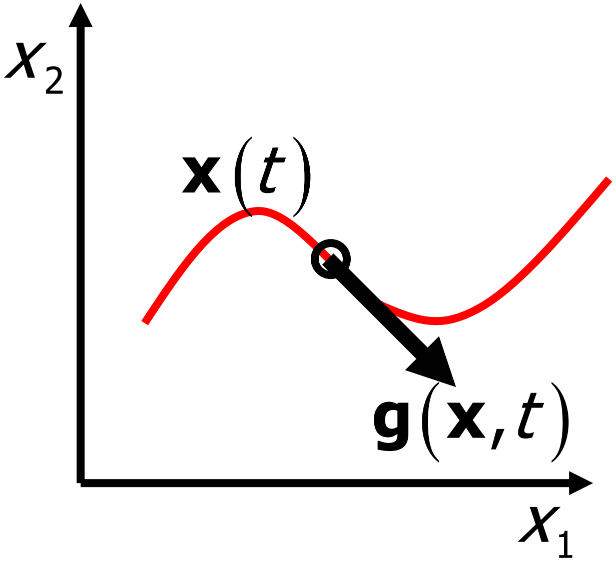
Particle

A single particle in 2-D moving in a flow field

- Position $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

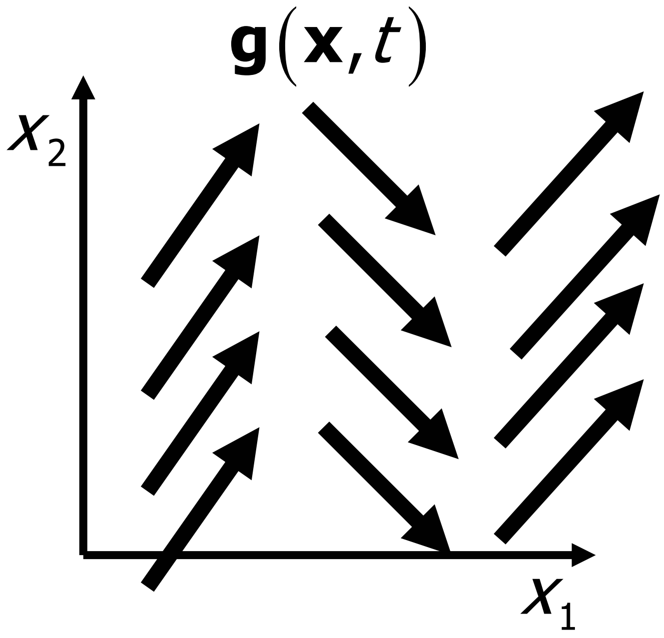
- Velocity $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \mathbf{v} = \frac{d\mathbf{x}}{dt}$

- The flow field function dictates particle velocity $\mathbf{v} = \mathbf{g}(\mathbf{x}, t)$



Vector Field

The flow field $\mathbf{g}(\mathbf{x},t)$ is a vector field that defines a vector for any particle position $\mathbf{x}$ at any time t.



How would a particle move in this vector field?

Differential Equations

The equation $\mathbf{v} = \mathbf{g}(\mathbf{x}, t)$ is a first order differential equation:

$$\frac{d\mathbf{x}}{dt} = \mathbf{g}(\mathbf{x}, t)$$

Position is computed by integrating the differential equation:

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{g}(\mathbf{x}, t)\, dt$$

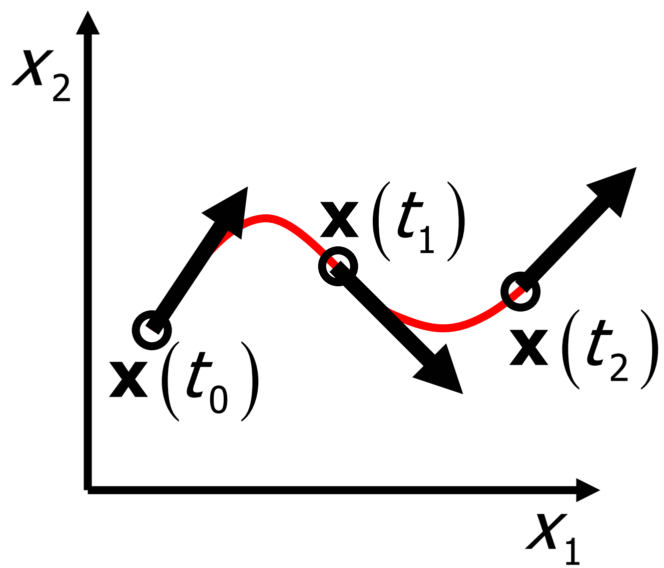
Usually, no analytical solution

Numeric Integration

Instead we use numeric integration:

- Start at initial point $\mathbf{x}(t_0)$
- Step along vector field to compute the position at each time

This is called an **initial value problem**.



Euler's Method

Simplest solution to an initial value problem.

- Starts from initial value
- Take small time steps along the flow:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\,\mathbf{g}(\mathbf{x},t)$$

Why does this work?

Consider Taylor series expansion of $\mathbf{x}(t)$:

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \frac{d\mathbf{x}}{dt} + \frac{\Delta t^2}{2}\frac{d^2\mathbf{x}}{dt^2} + \cdots$$

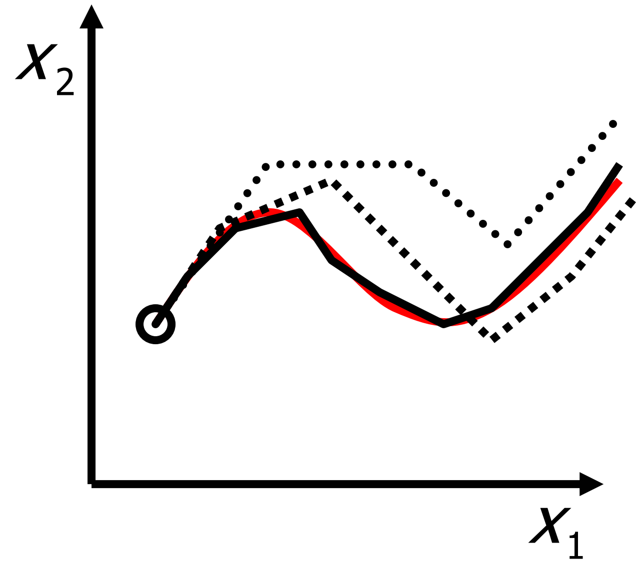
Disregarding higher-order terms and replacing the first derivative with the flow field function yields the equation for the Euler's method.

Other Methods

Euler's method is the simplest numerical method.
The error is proportional to Δt^2.

For most cases, it is inaccurate and unstable

- It requires very small steps.



Other methods:

- Midpoint (2^{nd} order Runge-Kutta)
- Higher order Runge-Kutta (4^{th} order, 6^{th} order)
- Adams
- Adaptive Stepsize

Particle in a Force Field

What is a motion of a particle in a force field?

The particle moves according to Newton's Law:

$$\frac{d^2\mathbf{x}}{dt^2} = \frac{\mathbf{f}}{m} \quad (\mathbf{f} = ma)$$

The mass m describes the particle's inertial properties:

Heavier particles are easier to move than lighter particles.

In general, the force field $\mathbf{f}(\mathbf{x}, \mathbf{v}, t)$ may depend on the time t and particle's position $\mathbf{x}$ and velocity $\mathbf{v}$.

Second-Order Differential Equations

Newton's Law => ordinary differential equation of 2nd order:

$$\frac{d^2\mathbf{x}(t)}{dt^2} = \frac{\mathbf{f}(\mathbf{x},\mathbf{v},t)}{m}$$

A clever trick allows us to reuse the numeric solvers for 1st-order differential equations.

Define new phase vector **y**:

- Concatenate position **x** and velocity **v**,

Then construct a new 1st-order differential equation whose solution will also solve the 2nd-order differential equation.

$$\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}, \quad \frac{d\mathbf{y}}{dt} = \begin{bmatrix} d\mathbf{x}/dt \\ d\mathbf{v}/dt \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{f}/m \end{bmatrix}$$

Particle Animation

AnimateParticles(n, $\mathbf{y}_0$, t_0, t_f)

{

  $\mathbf{y} = \mathbf{y}_0$

  $t = t_0$

  DrawParticles(n, $\mathbf{y}$)

  while(t != t_f) {

    $\mathbf{f}$ = ComputeForces($\mathbf{y}$, t)

    d$\mathbf{y}$dt = AssembleDerivative($\mathbf{y}$, $\mathbf{f}$)

    {$\mathbf{y}$, t} = ODESolverStep(6n, $\mathbf{y}$, d$\mathbf{y}$/dt)

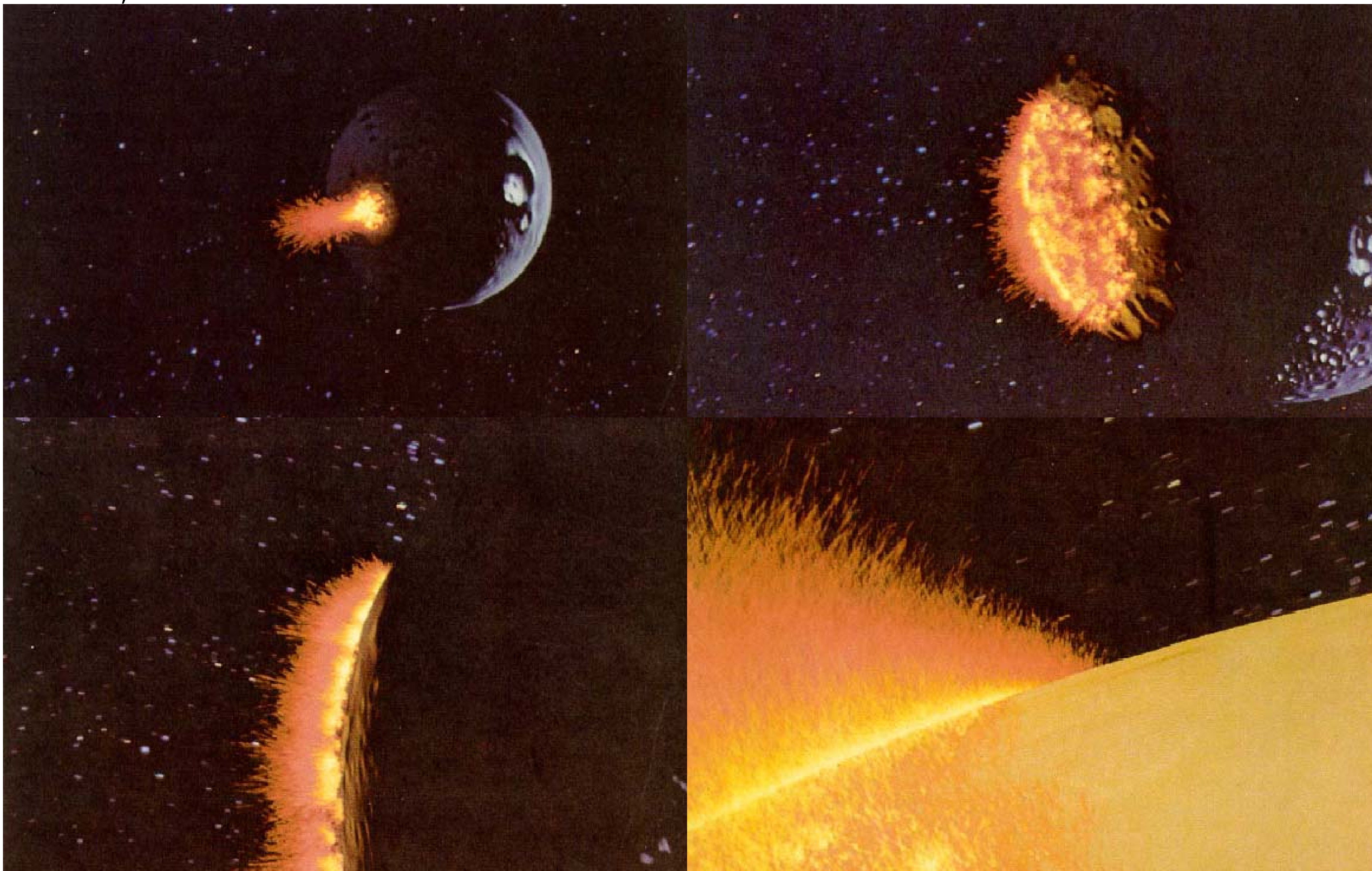
    DrawParticles(n, $\mathbf{y}$)

  }

}

Particle Animation [Reeves et al. 1983]

Start Trek, The Wrath of Kahn



Overview

Interpolation of rotations, quaternions

- Euler angles
- Quaternions

Dynamics

- Particles
- Rigid body
- Deformable objects

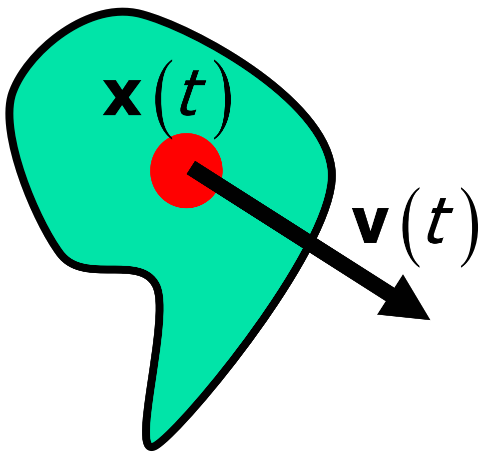
Rigid-Body Dynamics

Could use particles for all points

But rigid body does not deform

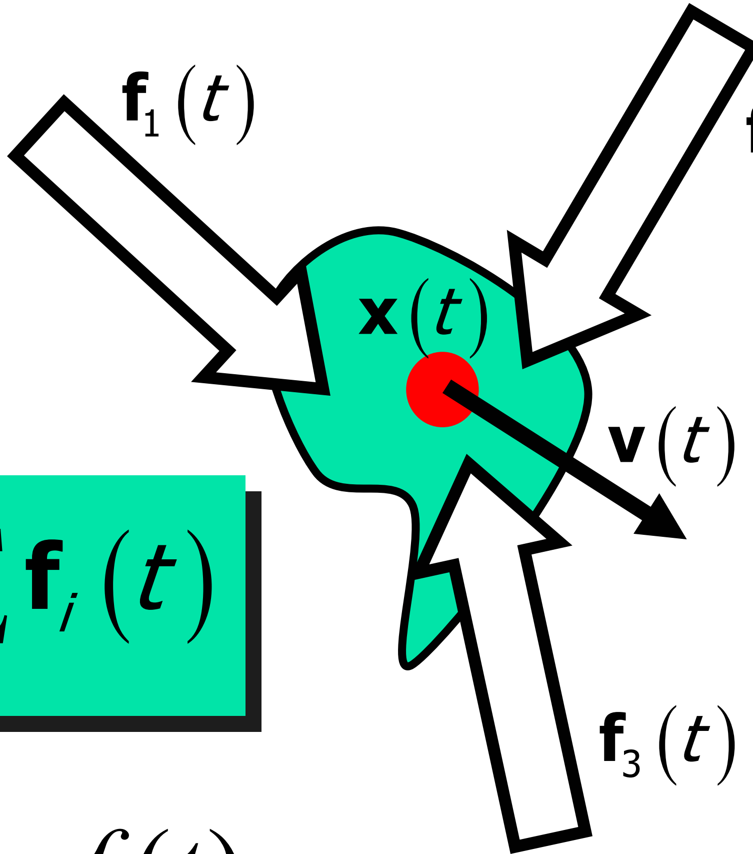
Few degrees of freedom

Start with only one particle at center of mass



$$\mathbf{y}(t) = \begin{bmatrix} \mathbf{x}(t) \\ ? \\ \mathbf{v}(t) \\ ? \end{bmatrix}$$

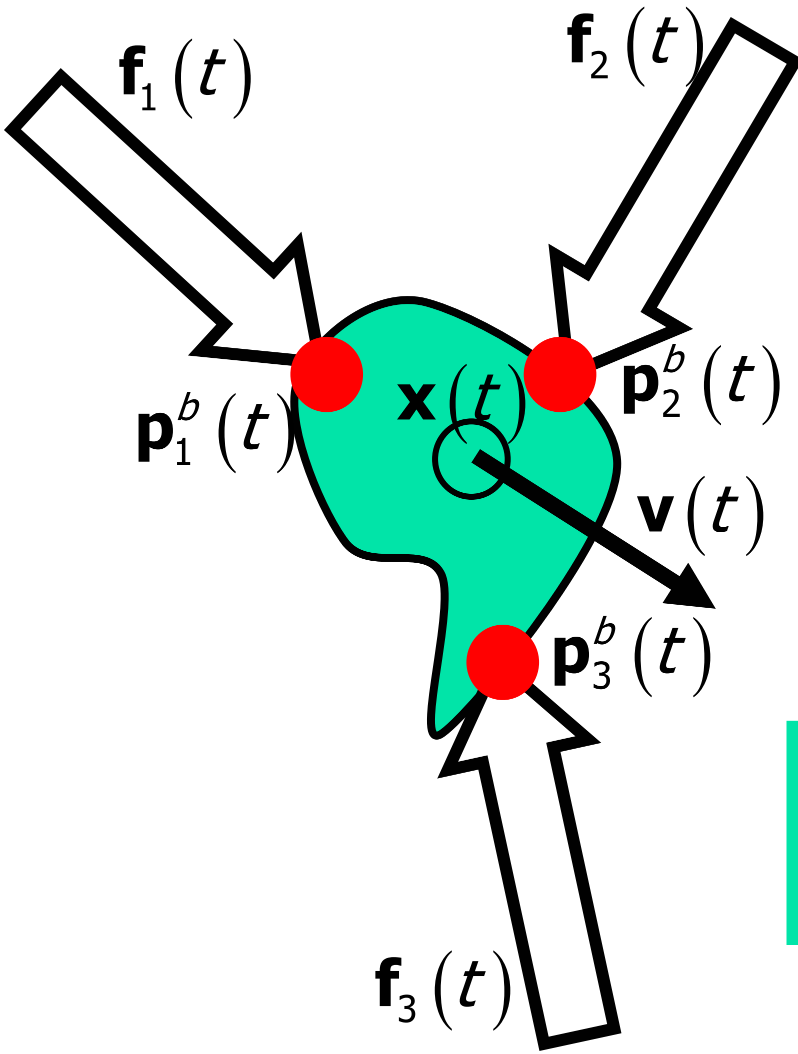
Net Force



$$\mathbf{f}(t) = \sum_i \mathbf{f}_i(t)$$

$$\frac{dMv(t)}{dt} = f(t)$$

Net Torque



$$\boldsymbol{\tau}(t) = \sum_i \left(\mathbf{p}_i^b - \mathbf{x}(t)\right) \times \mathbf{f}_i(t)$$

Rigid-Body Equation of Motion

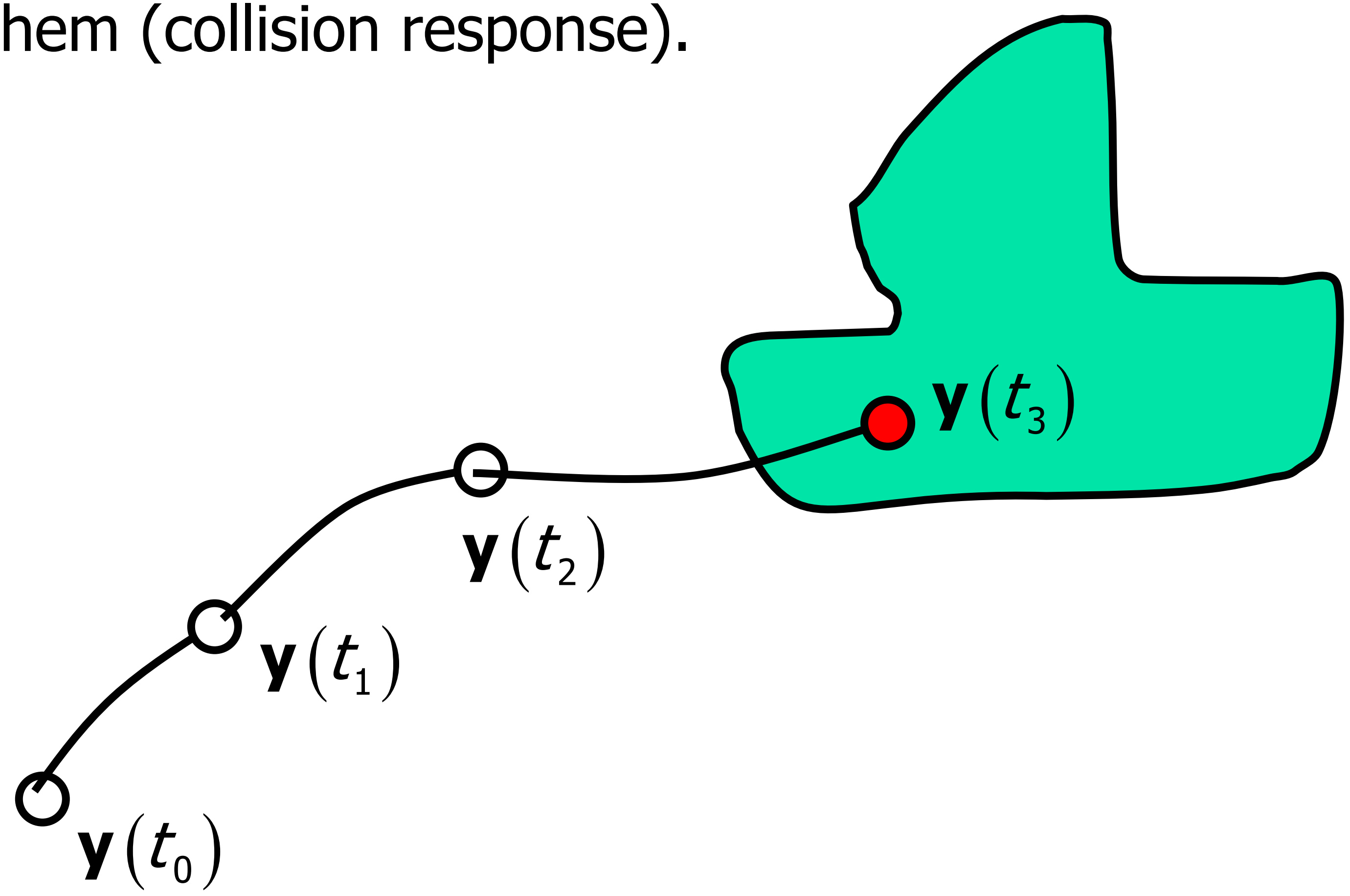
$$\frac{d}{dt}\mathbf{y}(t) = \frac{d}{dt}\begin{bmatrix} \mathbf{x}(t) \\ \mathbf{R}(t) \\ M\mathbf{v}(t) \\ \mathbf{I}(t)\boldsymbol{\omega}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(t) \\ \omega(t)\times\mathbf{R}(t) \\ \mathbf{f}(t) \\ \boldsymbol{\tau}(t) \end{bmatrix}$$

$M\mathbf{v}(t) \rightarrow$ linear momentum

$\mathbf{I}(t)\boldsymbol{\omega}(t) \rightarrow$ angular momentum

Simulations with Collisions

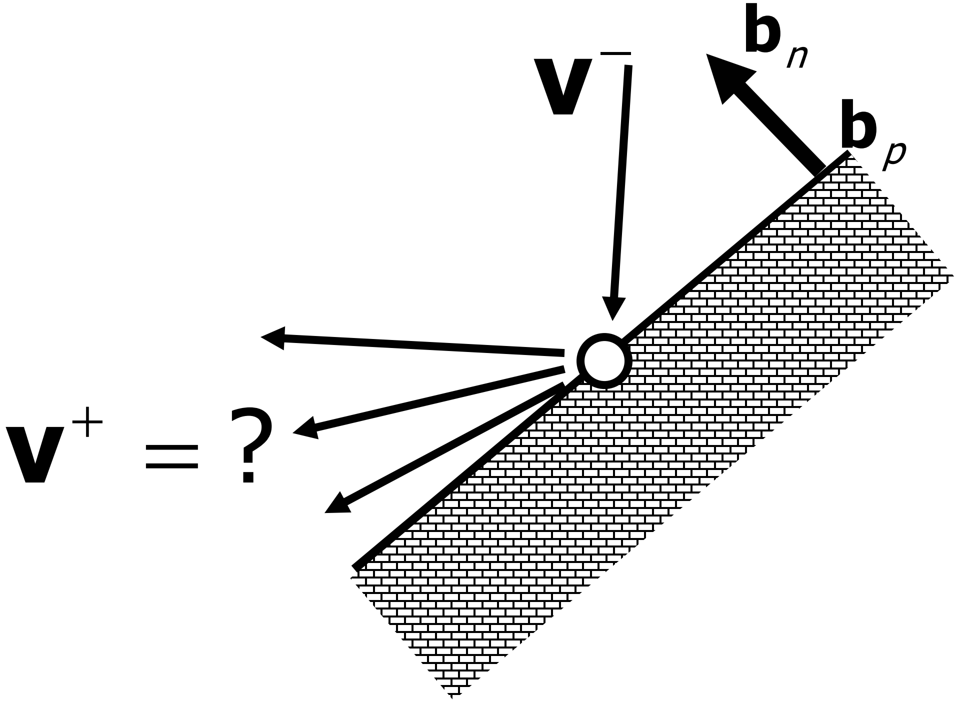
Simulating motions with collisions requires that we detect them (collision detection) and fix them (collision response).



Collision Response

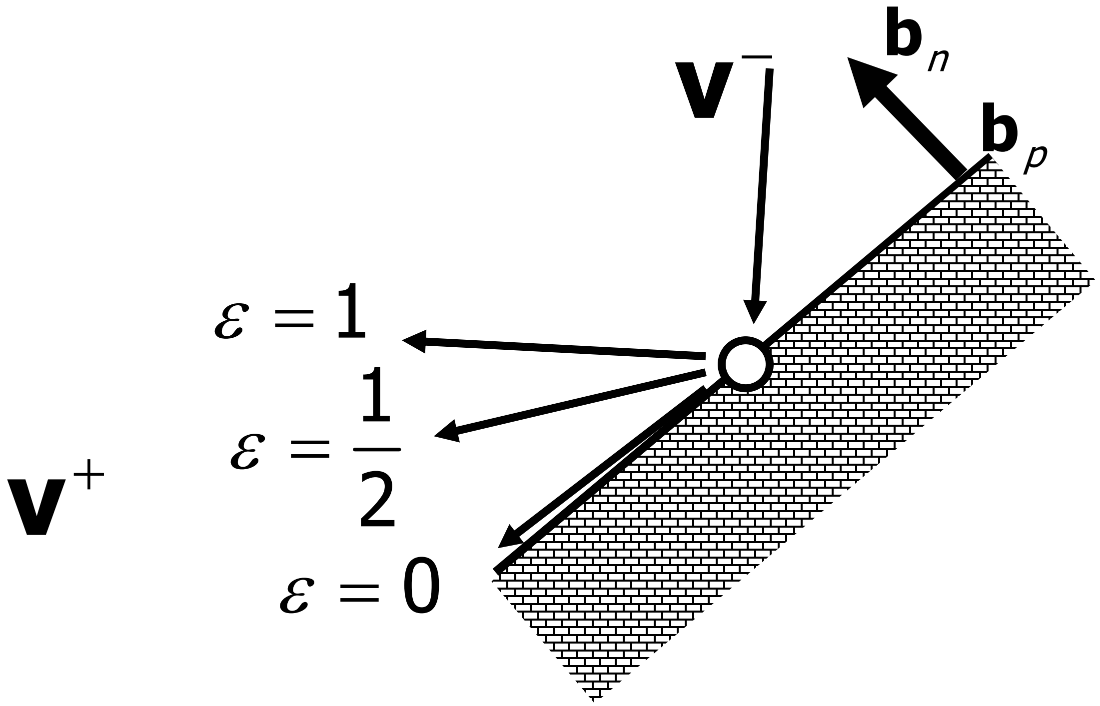
The mechanics of collisions are complicated

Simple model: assume that the two bodies exchange collision *impulse* instantaneously.



Frictionless Collision Model

$$\mathbf{b}_n \cdot \mathbf{v}^+ = -\varepsilon\left(\mathbf{b}_n \cdot \mathbf{v}^-\right)$$



Overview

Interpolation of rotations, quaternions

- Euler angles
- Quaternions

Dynamics

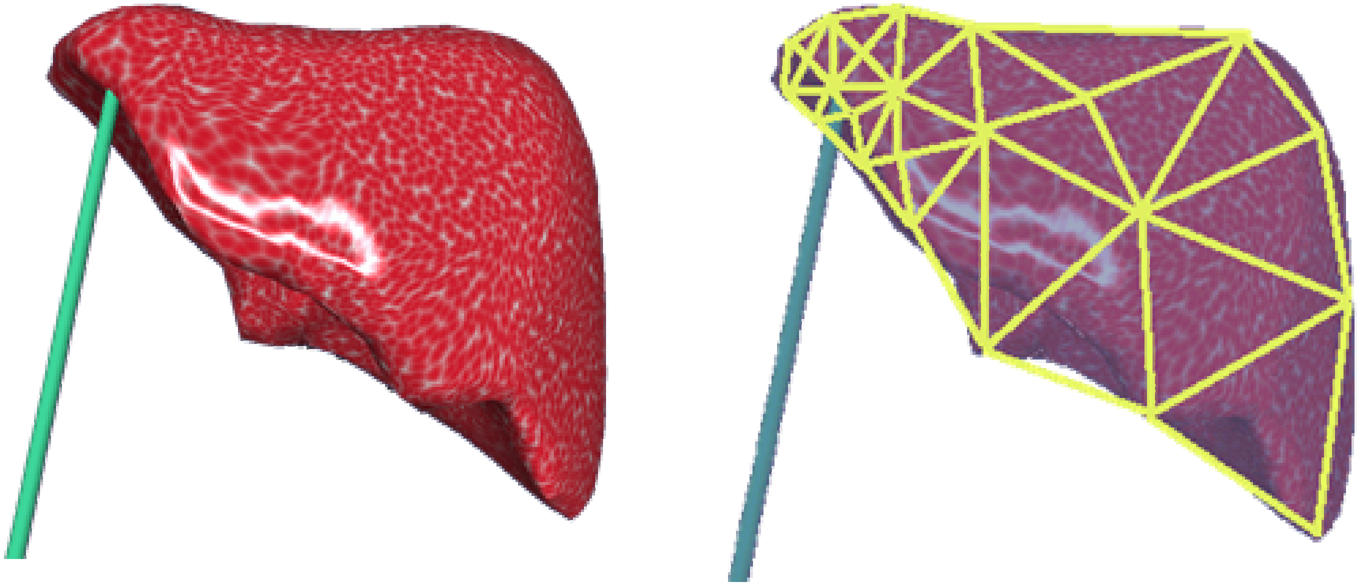
- Particles
- Rigid body
- Deformable objects

Deformable models

Shape deforms due to contact

Discretize the problem

Animation runs with smaller time steps than rendering (between 1/10,000s and 1/100s)



Back Next

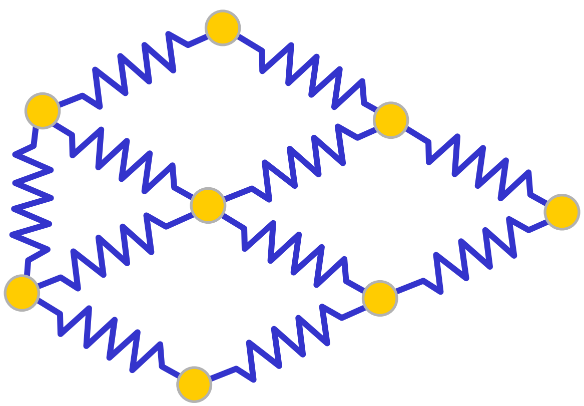
Mass-Spring system

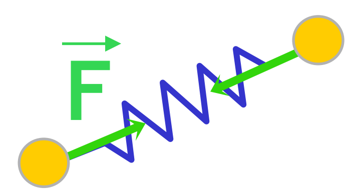
Network of masses and springs

Express forces

Integrate

Deformation of springs simulates deformation of objects





[Dorsey 1996]

Explicit Finite Elements

Discretize the problem

Solve locally

Simpler but less stable than implicit

Object

Finite Elements

Independent matricial systems

Back Next

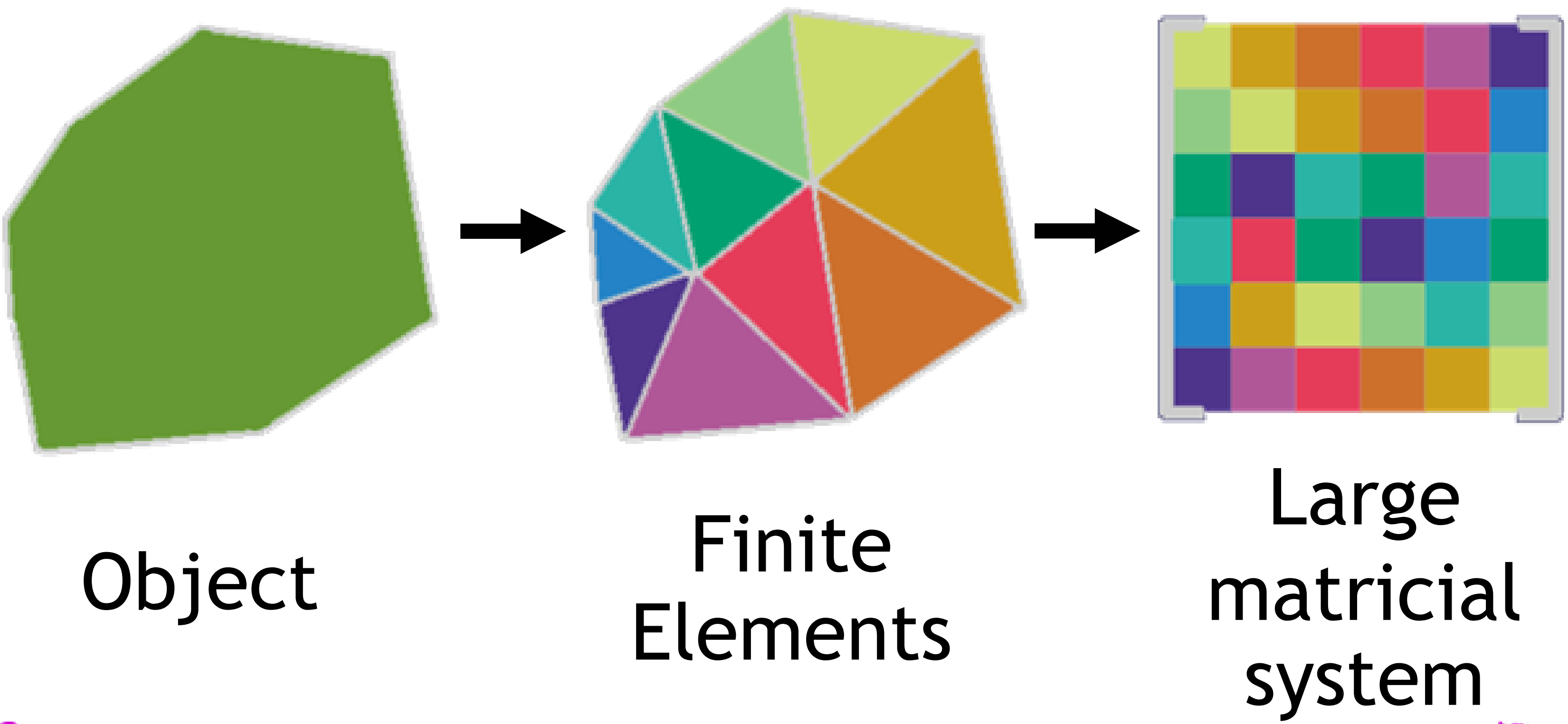
Implicit Finite Elements

Discretize the problem

Express the interrelationship

Solve a big system

More principled than mass-spring



Back
Next

Formally: Finite Elements

We are trying to solve a continuous problem

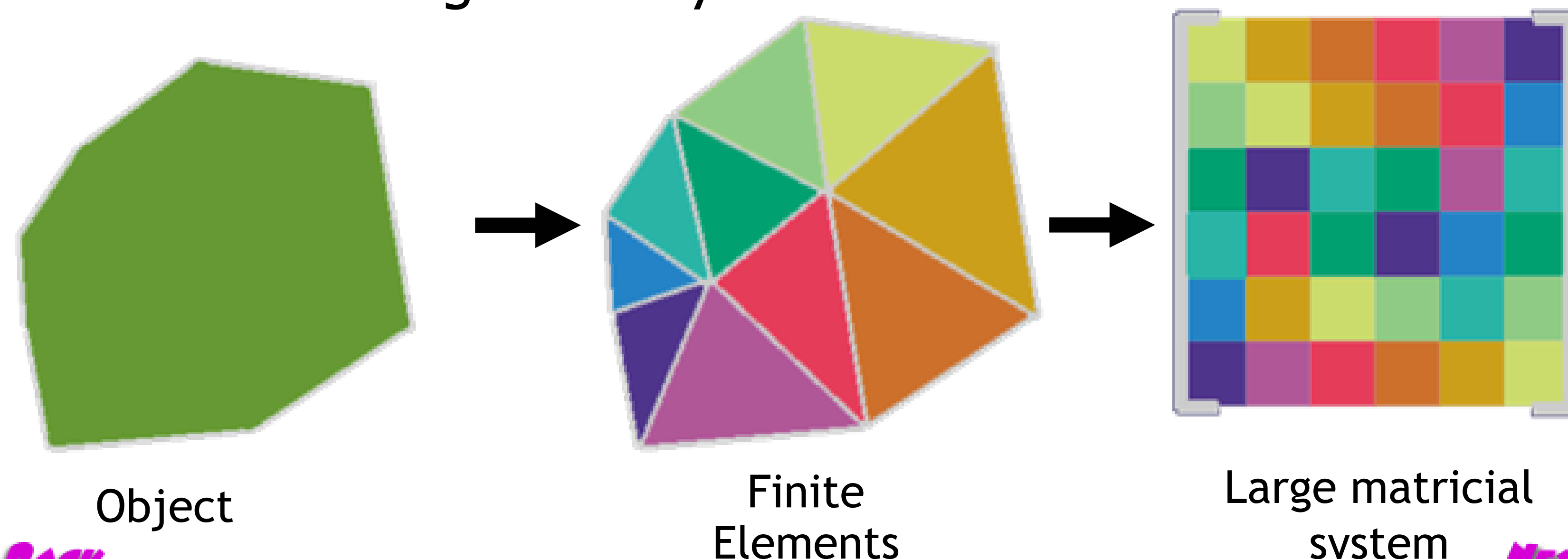
- Deformation of all points of the object
- Infinite space of functions

We project to a finite set of basis functions

- E.g. piecewise linear, piecewise constant

We project the equations governing the problem

This results in a big linear system



Back
Next

Cloth animation

Discretize cloth

Write physical equations

Integrate

Collision detection

Image removed due to copyright considerations.

Fluid simulation

Discretize volume of fluid

- Exchanges and velocity at voxel boundary

Write Navier Stokes equations

- Incompressible, etc.

Numerical integration

- Finite elements, finite differences

Challenges:

- Robust integration, stability
- Speed
- Realistic surface

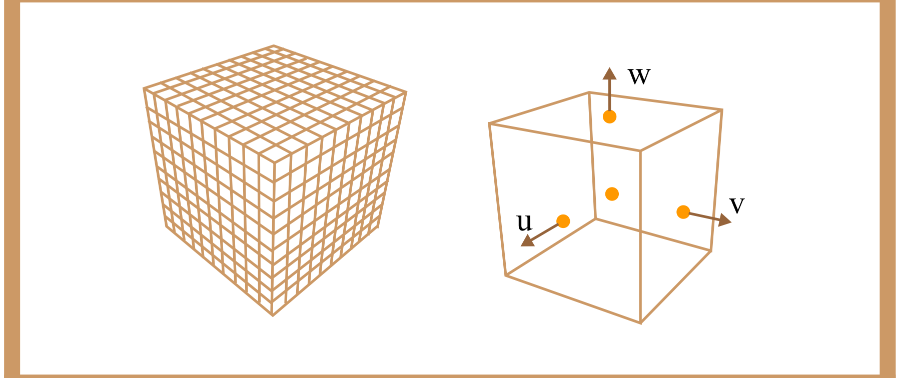


Image adapted from:
Fedkiw, Ronald. Jos Stam, and Henrik Wann Jensen. “Visual Simulation of Smoke.” Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. pp.15-22, August 2001.

Image removed due to copyright considerations.

How do they animate movies?

Keyframing mostly

Articulated figures, inverse kinematics

Skinning

- Complex deformable skin
- Muscle, skin motion

Images removed due to copyright considerations.

Hierarchical controls

- Smile control, eye blinking, etc.
- Keyframes for these higher-level controls

A huge time is spent building the 3D models, its skeleton and its controls

Physical simulation for secondary motion

- Hair, cloths, water
- Particle systems for "fuzzy" objects

Final project

First brainstorming session on Thursday

Groups of three

Large programming content

Proposal due Monday 10/27

- A couple of pages
- Goals
- Progression

Appointment with staff

Final project

Goal-based

- Render some class of object (leaves, flowers, CDs)
- Natural phenomena (plants, terrains, water)
- Weathering
- Small animation of articulated body, explosion, etc.
- Visualization (explanatory, scientific)
- Game
- Reconstruct an existing scene

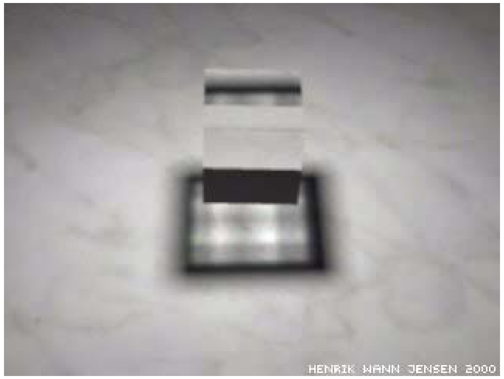
Technique-based

- Monte-Carlo Rendering
- Radiosity
- Finite elements/differences (fluid, cloth, deformable objects)
- Display acceleration
- Model simplification
- Geometry processing

Based on your ray tracer

Global illumination

- Distribution ray tracing (depth of field, motion blur, soft shadows)
- Monte-Carlo rendering
- Caustics



Appearance modeling

- General BRDFS
- Subsurface scattering