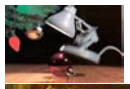
Animation

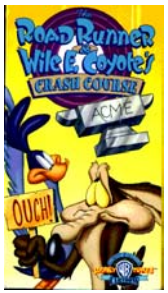




Computer Animation

Animation Methods
Keyframing
Interpolation
Kinematics
Inverse Kinematics

Slides courtesy of Leonard McMillan and Jovan Popovic



Administrative

Assignment 6 is due Monday 28 at 5pm
- Ray tracing
- Shading
- Shadow, reflection, refraction
- Supersampling

Office hours
- Durand Thursday 4-6pm in 4-035
- Ngan Thursday 4-7 in W20-575
- Teller Friday 3-5 in W20-575
- Ngan Monday 10-12 NE43-256
- Durand Monday 2-4pm in NE43-254

Final project
- Deadline for team building: Friday Nov 1
- Brainstorming session on Thursday

Conventional Animation

Draw each frame of the animation
- great control
- tedious

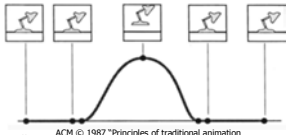
Reduce burden with cel animation
- layer
- keyframe
- inbetween
- cel panoramas (Disney's Pinocchio)
- ...

ACM © 1997 "Multiperspective panoramas for cel animation"

Computer-Assisted Animation

Keyframing
- automate the inbetweening
- good control
- less tedious
- creating a good animation still requires considerable skill and talent

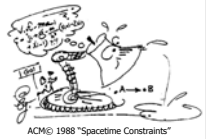
Procedural animation
- describes the motion algorithmically
- express animation as a function of small number of parameteres
- Example: a clock with second, minute and hour hands
  - hands should rotate together
  - express the clock motions in terms of a "seconds" variable
  - the clock is animated by varying the seconds parameter
- Example 2: A bouncing ball
  - Abs(sin($\omega t+\theta_0$))*e^{-kt}



ACM © 1987 "Principles of traditional animation applied to 3D computer animation"

Computer-Assisted Animation

Physically Based Animation
- Assign physical properties to objects (masses, forces, inertial properties)
- Simulate physics by solving equations
- Realistic but difficult to control



ACM© 1988 "Spacetime Constraints"

Motion Capture
- Captures style, subtle nuances and realism
- You must observe someone do something




Overview

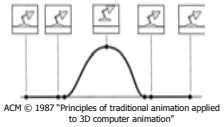
Keyframing and interpolation

Interpolation of rotations, quaternions

Kimematrics, articulation

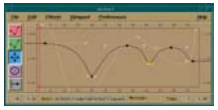
Physisally-based simulation & particles

Keyframing



ACM © 1987 "Principles of traditional animation applied to 3D computer animation"

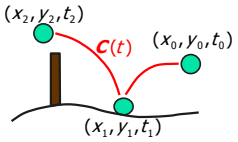
Describe motion of objects as a function of time from a set of key object positions. In short, compute the inbetween frames.



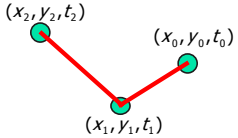
Interpolating Positions

Given positions: $(x_i, y_i, t_i),\ i = 0,\ldots,n$

find curve $\boldsymbol{C}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$ such that $\boldsymbol{C}(t_i) = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$



Linear Interpolation

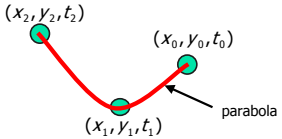


Simple problem: linear interpolation between first two points assuming t_0=0 and t_1=1: $x(t) = x_0(1-t) + x_1 t$

The x-coordinate for the complete curve in the figure:

$$x(t) = \begin{cases} \dfrac{t_1 - t}{t_1 - t_0} x_0 + \dfrac{t - t_0}{t_1 - t_0} x_1, & t \in [t_0, t_1) \\ \dfrac{t_2 - t}{t_2 - t_1} x_1 + \dfrac{t - t_1}{t_2 - t_1} x_2, & t \in [t_1, t_2] \end{cases}$$

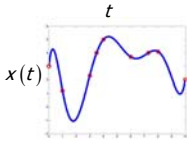
Polynomial Interpolation



An n-degree polynomial can interpolate any n+1 points. The Lagrange formula gives the n+1 coefficients of an n-degree polynomial that interpolates n+1 points. The resulting interpolating polynomials are called Lagrange polynomials. On the previous slide, we saw the Lagrange formula for n = 1.

Spline Interpolation

Lagrange polynomials of small degree are fine but high degree polynomials are too wiggly.



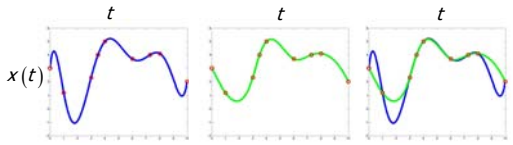
8-degree polynomial

How many n-degree polynomials interpolate n+1 points?

Spline Interpolation

Lagrange polynomials of small degree are fine but high degree polynomials are too wiggly.

Spline (piecewise cubic polynomial) interpolation produces nicer interpolation.



8-degree polynomial | spline | spline vs. polynomial

Spline Interpolation

A cubic polynomial between each pair of points:

$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

Four parameters (degrees of freedom) for each spline segment.

Number of parameters:

n+1 points ▫ n cubic polynomials ▫ 4n degrees of freedom

Number of constraints:

- interpolation constraints
  n+1 points ▫ 2 + 2 (n-1) = 2n interpolation constraints
- continuous velocity
  n+1 points ▫ n-1 velocity constraints (one for each interior point)
- continuous acceleration
  n+1 points ▫ n-1 acceleration constraints (one for each interior point)

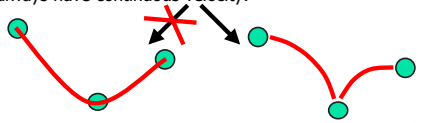
Interpolation of Positions

Solve an optimization to set remaining degrees of freedom:

$$\max_{\text{dof}} \ \text{quality}(\boldsymbol{C}(t)) \quad \text{subject to constraints} \quad \equiv \quad \min_{\text{dof}} \ \text{badness}(\boldsymbol{C}(t)) \quad \text{subject to constraints}$$

$$\text{badness}(\boldsymbol{C}(t)) = -\text{quality}(\boldsymbol{C}(t))$$

We want to support general constraints: not just smooth velocity and acceleration. For example, a bouncing ball does not always have continuous velocity:

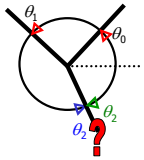


Interpolating Angles

Given angles (θ_i, t_i), $i = 0,\ldots,n$

find curve $\theta(t)$ such that $\theta(t_i) = \theta_i$

Angle interpolation is ambiguous. Different angle measurements will produce different motion:



Keyframing

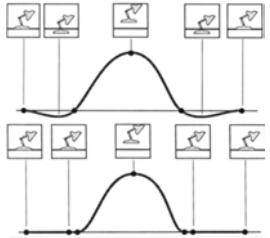
Given keyframes $K_i = (x_i, y_i, \theta_i, t_i)$, $i = 0,\ldots,n$

find curve $\boldsymbol{K}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix}$ such that $\boldsymbol{K}(t_i) = \boldsymbol{K}_i$

Interpolate each parameter separately

Interpolating Key Frames

Interpolation is not fool proof. The splines may undershoot and cause interpenetration. The animator must also keep an eye out for these types of side-effects.



Traditional Animation Principles

The in-betweening, was once a job for apprentice animators. We described the automatic interpolation techniques that accomplish these tasks automatically. However, the animator still has to draw the key frames. This is an art form and precisely why the experienced animators were spared the in-betweening work even before automatic techniques.

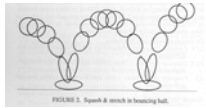
The classical paper on animation by John Lasseter from Pixar surveys some the standard animation techniques:

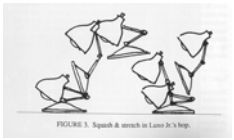
"*Principles of Traditional Animation Applied to 3D Computer Graphics,* "**SIGGRAPH'87**, pp. 35-44.

Squash and stretch

Squash: flatten an object or character by pressure or by its own power

Stretch: used to increase the sense of speed and emphasize the squash by contrast



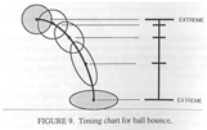


Timing

Timing affects weight:

- Light object move quickly
- Heavier objects move slower

Timing completely changes the interpretation of the motion. Because the timing is critical, the animators used the draw a time scale next to the keyframe to indicate how to generate the in-between frames.

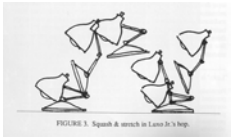


Anticipation

An action breaks down into:

- Anticipation
- Action
- Reaction

Anatomical motivation: a muscle must extend before it can contract. Prepares audience for action so they know what to expect. Directs audience's attention. Amount of anticipation can affect perception of speed and weight.



Overview

Keyframing and interpolation

Interpolation of rotations, quaternions

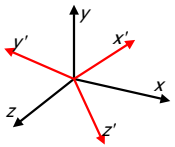
Kimematrics, articulation

Physisally-based simulation & particles

Interpolating Orientations in 3-D

Rotation matrices

Given rotation matrices M_i and time t_i, find M(t) such that $M(t_i)=M_i$.



$$M = \begin{bmatrix} | & | & | \\ \boldsymbol{x}' & \boldsymbol{y}' & \boldsymbol{z}' \\ | & | & | \end{bmatrix}$$

Flawed Solution

Linearly interpolate each entry independently

Example: M_0 is identity and M_1 is 90-deg rotation around x-axis

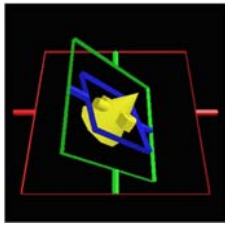
$$\text{Interpolate}\left(\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},\begin{bmatrix}1&0&0\\0&0&1\\0&-1&0\end{bmatrix}\right)=\begin{bmatrix}1&0&0\\0&0.5&0.5\\0&-0.5&0.5\end{bmatrix}$$

Is the result a rotation matrix?

The result R is not a rotation matrix. For example, check that RR^T does not equal identity. In short, this interpolation does not preserve the rigidity (angles and lengths) of the transformation.

Euler Angles

An Euler angle is a rotation about a single axis. Any orientation can be described composing three rotation around each coordinate axis. We can visualize the action of the Euler angles: each loop is a rotation around one coordinate axis.

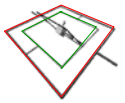


Interpolating Euler Angles

Natural orientation representation: three angles for three degrees of freedom

Unnatural interpolation: A rotation of 90-degrees first around the z-axis and then around the y-axis has the effect of a 120-degree rotation around the axis (1, 1, 1). But rotation of 30-degrees around the z- and y-axis does not have the effect of a 40-degree rotation around the axis (1, 1, 1).

Gimbal lock: two or more axis align resulting in a loss of rotation degrees of freedom. For example, if the green loop in previous slide aligns with the red loop then both the rotation around the blue loop and the rotation around the red loop produces identical rotation.



Quaternion Interpolation

Linear int**erp**olation (lerp) of quaternion representation of orientations gives us something better:

$$\text{lerp}(\mathbf{q}_0,\mathbf{q}_1,t)=\mathbf{q}(t)=\mathbf{q}_0(1-t)+\mathbf{q}_1 t$$

Quaternion Introduction

- Due to Hamilton (1843); also Shoemake, Siggraph '85
- By analogy: 1-, 2-, 3-DOF rotations as constrained points on 1, 2, 3-spheres

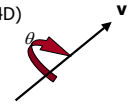
Quaternions

Quaternions are unit vectors on 3-sphere (in 4D)

Right-hand rotation of θ radians about **v** is

q = {cos(θ/2); **v** sin(θ/2)}

- Often noted (s, **v**)
- What if we use –**v** ?



What is the quaternion of Identity rotation?

Is there exactly one quaternion per rotation?

What is the inverse $\mathbf{q}^{-1}$ of quaternion **q** {a, b, c, d}?

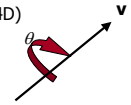
Quaternions

Quaternions are unit vectors on 3-sphere (in 4D)

Right-hand rotation of θ radians about **v** is

q = {cos(θ/2); **v** sin(θ/2)}

- Often noted (s, **v**)
- Also rotation of –θ around -**v**



What is the quaternion of Identity rotation?

- **q**i = {1; 0; 0; 0}

Is there exactly one quaternion per rotation?

- No, **q**={cos(θ/2); **v** sin(θ/2)} is the same rotation as -**q**={cos((θ+2π)/2); **v** sin((θ+2π)/2)}
- Antipodal on the quaternion sphere

What is the inverse $\mathbf{q}^{-1}$ of quaternion **q** {a, b, c, d}?

- {a, -b, -c, -d}

Quaternion Algebra

Two general quaternions are multiplied by a special rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1s_2 - \left(\vec{V}_1 \bullet \vec{V}_2\right), s_1\vec{V}_2 + s_2\vec{V}_1 + \vec{V}_1 \times \vec{V}_2\right)$$

Sanity check : {cos(α/2); **v** sin(α/2)} {cos(β/2); **v** sin(β/2)}

Quaternion Algebra

Two general quaternions are multiplied by a special rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1s_2 - \left(\vec{V}_1 \bullet \vec{V}_2\right), s_1\vec{V}_2 + s_2\vec{V}_1 + \vec{V}_1 \times \vec{V}_2\right)$$

Sanity check : {cos(α/2); **v** sin(α/2)} {cos(β/2); **v** sin(β/2)}

{cos(α/2)cos(β/2) - sin(α/2)**v**. sin(β/2)} **v**,
cos(β/2) sin(α/2) **v** + cos(α/2)sin(β/2) **v** +**v** × **v**}

{cos(α/2)cos(β/2) - sin(α/2) sin(β/2),
v(cos(β/2) sin(α/2) + cos(α/2) sin(β/2))}

{cos((α+β)/2), **v** sin((α+β)/2) }

Quaternion Algebra

Two general quaternions are multiplied by a special rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1s_2 - \left(\vec{V}_1 \bullet \vec{V}_2\right), s_1\vec{V}_2 + s_2\vec{V}_1 + \vec{V}_1 \times \vec{V}_2\right)$$

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**} $\mathbf{q}^{-1}$ = {-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,zs²)+c²**p**+(-csy, csx,0)+(-csy, csx, 0)+(s²x, s²y, 0)}
= {0, (c²x-2csy-s²x, c²y+2csx-s²y, zs²+sc²)}
= {0, x cos(θ)-y sin(θ), x sin(θ)+y cos(θ), z }

Quaternion Algebra

Two general quaternions are multiplied by a special rule:

$$\mathbf{q}_1\mathbf{q}_2 = \left(s_1s_2 - \left(\vec{V}_1 \bullet \vec{V}_2\right), s_1\vec{V}_2 + s_2\vec{V}_1 + \vec{V}_1 \times \vec{V}_2\right)$$

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

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

Quaternions are associative:

- (**q**1 **q**2) **q**3 = **q**1 (**q**2 **q**3)

But not commutative:

- **q**1 **q**2 ≠ **q**2 **q**1

Quaternions

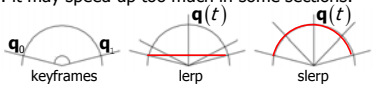
Can also be defined like complex numbers
a+bi+cj+dk
Multiplication rules

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

…

Quaternion Interpolation

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.

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

where $\omega = \cos^{-1}(\mathbf{q}_0 \square \mathbf{q}_1)$

Quaternion Interpolation

Higher-order interpolations: must stay on sphere

See Shoemake paper for:

- Matrix equivalent of composition
- Details of higher-order interpolation
- More of underlying theory

Problems

- No notion of favored direction (e.g. up for camera)
- No notion of multiple rotations, needs more key points

Fun:Julia Sets in Quaternion space

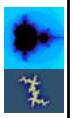
Mandelbrot set: $Z_{n+1}=Z_n^2+Z_0$

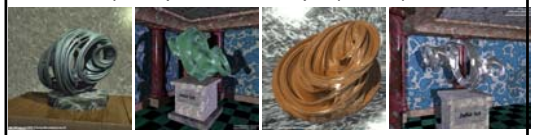
Julia set $Z_{n+1}=Z_n^2+C$

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

Do the same with Quaternions!

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





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

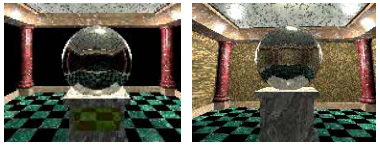
Fun:Julia Sets in Quaternion space

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



Overview

Keyframing and interpolation

Interpolation of rotations, quaternions

Kimematrics, articulation

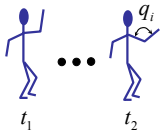
Physisally-based simulation & particles

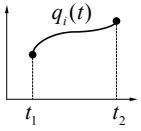
Articulated Models

Articulated models:

- rigid parts
- connected by joints

They can be animated by specifying the joint angles (or other display parameters) as functions of time.





Forward Kinematics

Describes the positions of the body parts as a function of the joint angles.

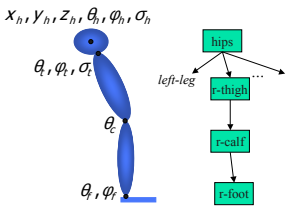
1 DOF: knee

2 DOF: wrist

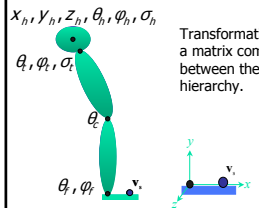
3 DOF: arm

Skeleton Hierarchy

Each bone transformation described relative to the parent in the hierarchy:



Forward Kinematics



Transformation matrix for a sensor/effecter $\mathbf{v}_s$ is a matrix composition of all joint transformation between the sensor/effecter and the root of the hierarchy.

$$v_w = T(x_h, y_h, z_h) R(\theta_h, \varphi_h, \sigma_h) TR(\theta_t, \varphi_t, \sigma_t) TR(\theta_c) TR(\theta_f, \varphi_f) v_s$$

$$v_w = \mathbf{S}\underbrace{\left(x_h, y_h, z_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \theta_r, \phi_f\right)}_{\mathbf{p}} v_s = \mathbf{S}(\mathbf{p}) v_s$$

Inverse Kinematics

Forward Kinematics

- Given the skeleton parameters (position of the root and the joint angles) $\boldsymbol{p}$ and the position of the sensor/effecter in local coordinates v_s, what is the position of the sensor in the world coordinates v_w.
- Not too hard, we can solve it by evaluating $\mathbf{S}(\mathbf{p})v_s$

Inverse Kinematics

- Given the the position of the sensor/effecter in local coordinates v_s and the position of the sensor in the world coordinates v_w, what are the skeleton parameters $\boldsymbol{p}$.
- Much harder requires solving the inverse of the non-linear function $\mathbf{S}(\mathbf{p})$
- We can solve it by root-finding $\mathbf{p}$? such that $\mathbf{S}(\mathbf{p})v_s - v_w = 0$
- We can solve it by optimization $\underset{\mathbf{p}}{\text{minimize}} \left(\mathbf{S}(\mathbf{p})v_s - v_w\right)^2$

Kinematics vs. Dynamics

Kinematics

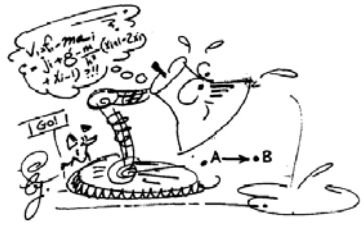
Describes the positions of the body parts as a function of the joint angles.

Dynamics

Describes the positions of the body parts as a function of the applied forces.

Next

Dynamics



ACM© 1988 "Spacetime Constraints"

Overview

Keyframing and interpolation

Interpolation of rotations, quaternions

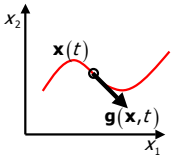
Kimematrics, articulation

Physisally-based simulation & particles

Particle

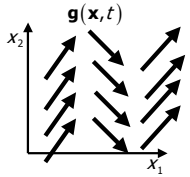
A single particle in 2-D moving in a flow field

- 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:

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

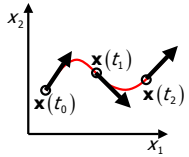
The position of the particle is computed by integrating the differential equation:

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

For most interesting cases, this integral cannot be computed analytically.

Numeric Integration

Instead we compute the particle's position by numeric integration: starting at some initial point $\mathbf{x}(t_0)$ we step along the vector field to compute the position at each subsequent time instant. This type of a problem is called an **initial value problem**.



Euler's Method

Euler's method is the simplest solution to an initial value problem. Euler's method starts from the initial value and takes 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?

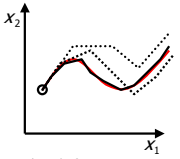
Let's look at a Taylor series expansion of function $\mathbf{x}(t)$:

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

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, the Euler's method is inaccurate and unstable requiring very small steps.



Other methods:

- Midpoint (2nd order Runge-Kutta)
- Higher order Runge-Kutta (4th order, 6th 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:

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

The mass m of a particle 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 yields an ordinary differential equation of second 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 same numeric differentiation solvers for first-order differential equations. If we define a new phase space vector $\mathbf{y}$, which consists of particle's position $\mathbf{x}$ and velocity $\mathbf{v}$, then we can construct a new first-order differential equation whose solution will also solve the second-order differential equation.

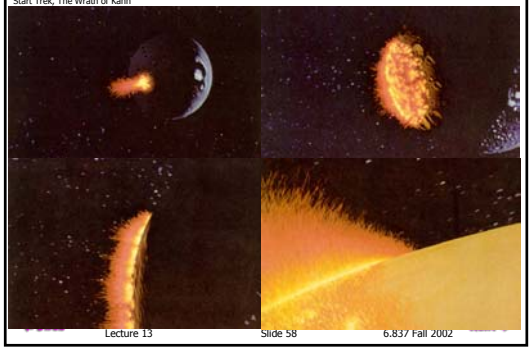
$$\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}, \quad \frac{d\mathbf{y}}{dt} = \begin{bmatrix} d\mathbf{x}/dt \\ d\mathbf{v}/dt \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{f}/m \end{bmatrix}$$

Particle Animation

```
AnimateParticles(n, y0, t0, tf)
{
        y = y0
        t = t0
        DrawParticles(n, y)
        while(t != tf) {
                f = ComputeForces(y, t)
                dydt = AssembleDerivative(y, f)
                {y, t} = ODESolverStep(6n, y, dy/dt)
                DrawParticles(n, y)
        }
}
```


Particle Animation [Reeves et al. 1983]

Start Trek, The Wrath of Kahn



Particle Modeling [Reeves et al. 1983]

Other physical animation

Rigid body
Collision detection and response
Deformable bodies
Fluid simulation (water, smoke, fire)
Cloth simulation
Aging of materials

[Dorsey 1996]

Next time

Final project brainstorming