6. Physically-Based Modelling

6.1 Overview

- **Non-physically based models**
  - Kinematic Models (positions controlled directly by animator)
    - Implicit surfaces
    - Spline (parametric) surfaces and subdivision surfaces
- **Physically-based models** (positions determined by dynamics)
  - Particle Systems
  - Mass-spring systems
  - Continuum mechanics
    - Finite-difference methods
    - Finite-element methods
  - Rigid Body models

6.2 Non-physically based 3D Models

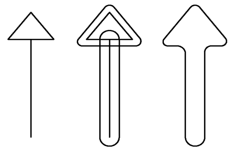
- Traditional 3D Models
  - Often use 3D surface models
  - Usually animated by hand to “look real” (eg. “Toy Story”)
  - rendered at regular intervals
- Typical applications
  - 3D games e.g. Doom, Quake, Unreal
    - Very basic geometry, BSP-trees for Visible surface preprocessing, depend on texture mapping
  - VRML 2.0
  - Professional Animation Packages (⇒ movie industry)
    - e.g. 3D Studio Max, Maya
  - Flight Simulators (for pilot training)
    - Very expensive special-purpose hardware systems

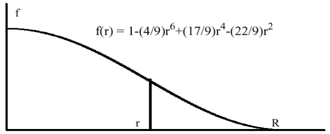
Implicit Surfaces

- Many “non-geometric” objects best defined by implicit surfaces
- An implicit surface is an iso-valued surface in a scalar field $f(x,y,z)$, i.e. all points satisfying $f(x,y,z) = k$
  - e.g. a sphere is the surface $x^2+y^2+z^2=r^2$
- ISs are usually rendered with ray tracing or by polygonization methods like *Marching Cubes* (see Volume Visualization notes)
- Usually have to solve for ray-object intersection with numerical techniques (i.e. root finding)
  - Sphere example is different - have an explicit ray-object intersection solution
- Can model “soft” objects as a particular contour of a "force field" surrounding a set of points whose individual fields are added
- Can model by pulling points around & adding and deleting points

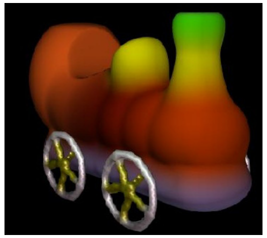
Example-"Blobby Objects"

- Model created by union of skeleton objects (points, lines, etc) with a density field fitted around them such that density is 1 at the skeleton surface and 0 at a distance of *R*. The model is defined by the *c*-isosurface of the density field (0<*c*<1).





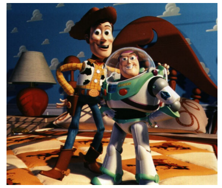
Bloomenthal and Wyvill "Interactive techniques for implicit modelling" [Computer Graphics, 24(2), 1990, pp. 109-116]



Bryan Wyvill "The great train rubbery" [SIGGRAPH 88 Electronic Theatre and Video Review, Issue 26]

6.3 Professional Modelling and Animation Packages

- Build "computerised puppets" usually by using spline surfaces and subdivision surfaces.
- Controlled by skilled modellers/animators, via large number of control points (parameters)
- e.g. Toy Story
  - Number of control points for Woody: 712
  - Control points for Woody's face: 212
  - Control points for Woody's mouth: 58
  - Control points for Sid's backpack: 128



- Large repertoire of "special effects" tools, e.g. particle systems for animating hair or modelling burning torches
- May interface to "motion capture" systems for "first hack" at control parameter sequences

Example – 3D Studio Max

- A high-quality modeller/renderer with animation controls
  - Construct world at t = 0, move time slider to next key-frame time, adjust model/camera as reqd, step to next key-frame time etc
  - System interpolates all parameters (in-betweening)
    - Can plug in different interpolators/path controllers
- Lots of fancy features
  - Inverse Kinematics system
  - Free-form deformations (FFDs) and other space warps
  - Rigid-body dynamics simulations (rudimentary??)
  - Particle systems (for rain, snow, water, explosions, crowds, ….)
- Motion-capture interface available

Controlling the Animation

- Hard-code it!
  - Need artists who are also good programmers!
- Scripts
  - Usually low-level description
  - e.g. POV, VRML2
- Manual control
  - Next slide
- Interactive control (real-time animations)
  - Slide after next

“Manual” control

- Usually hierarchical models
- Controlled by set of parameters (e.g. joint angles, facial muscles)
- Need parametric states of objects at each key frame
- Do “In-betweening” by smoothly interpolating parameters
  - Discontinuities a problem (e.g. bouncing ball)
  - May need to add extra key frames and/or velocity information
- Such methods are called *kinematic* methods
  - *Kinematics:* study of motion in terms of positions, velocities and accelerations)
  - *c.f. Dynamics*: study of motion in terms of forces, torques and their effect
- May have *Inverse Kinematics* (IK) system
  - Move one part of hierarchy, system moves the rest to match (subject to joint constraints, spring controls etc)

Interactive methods

- E.g. racing-car games -- user(s) explicitly control objects in scene
  - Need collision detection
  - Often incorporates other physically-based animation controls
- Ultimate in interactive methods is “Motion capture” ⇒ Have user directly controlling puppet
  - Video methods
    - Most common: add coloured sequins to different parts of body, illuminate brightly, filter to extract just sequin positions
    - Emerging: image processing of unadorned human
  - Mechanical gadgetry attached to user
    - e.g. “data glove”

6.4 Physically-based Modelling

- Goal: compute motions according to the laws of physics
  - Use dynamics (forces, torques) to determine kinematics
- Involves solving differential equations (DE)
- Collisions need to be detected, and interrupt DE solution
- Physically-based animation
  - Advantages
    - reduces/eliminates need for human animators
    - models can be reused in different applications
    - models can react appropriately to any user input
  - Disadvantages
    - mathematically & computationally more complex

Physically-based Modelling (cont’d)

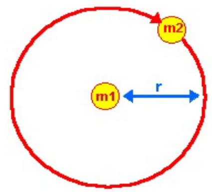
- Four main methods:
  - Particle systems
    - Fluids, mist, clouds, weather
      - Usually require large number of particles
    - Mass-spring systems
      - soft and/or flexible objects
  - Finite Difference Method (FDM)
    - Engineering simulations of fluid dynamics
  - Finite Element Method (FEM)
    - Engineering simulations of deformable solids
  - Rigid body methods
    - Solid objects, Mechanical constructions (robots, etc)

6.5 Particle Systems

- Have a set of particles, sometimes with masses {*mi*}
  - e.g. representing water drops/splashes
- May have forces (outside forces and forces between particles)
  - e.g. gravity, electromagnetic fields ⇒ particles can attract and repel each other
- Have known initial state (position, velocity, and acceleration/forces) for all particles
- Run a simulation loop. At each time step:
  - Compute force on each particle
  - Adjust position according to current velocity and velocity according to current acceleration (due to force)
  - Display (e.g. as a single textured object)

Particle Systems (cont'd)

Example: 2D orbiting mass: 2 "particles" with mass, velocity and acceleration.



$\mathbf{f} = g\frac{m_1 m_2}{r^2}$, if $m_2 \ll m_1$ we can consider particle 1 as stable and can compute the orbit of particle 2 by using $\mathbf{a}_{gravitation} = g\frac{m_1}{r^2} = \mathbf{a}_{centripital} = \frac{v^2}{r}$

Particle Systems (cont'd)

In general we might have millions of particles exerting forces on each other. The change of position of a particle at time *t* depends on the force acting on this particle, which depends on the position of all other particles at time *t*. Hence particle movements are computed by solving an ordinary differential equation (ODE):

$$\frac{dx}{dt} = v$$

$$\frac{d^2x}{dt^2} = \frac{f}{m} \Leftrightarrow \frac{dv}{dt} = a$$

$$a = f / m$$

Solution methods are explained in the next sections.

Mass-spring Systems

Special case of a particle system: only particles connected by a spring exert forces on each other.

- The basic idea
- A simple algorithm
- The problem
- The mid-point method
- A more-general formulation
- Other issues

The basic idea

- Have a set of point masses, $\{m_i\}$
  - e.g. representing sample points on a single elastic object
- Pairs of masses are connected by springs
  - Each spring has known rest length and spring constant (force per unit compression/expansion from rest length)
- May have other forces
  - e.g. gravity, viscous damping
- Have known initial state (position and velocity) for all masses
- Run a simulation loop. At each time step:
  - Compute force on each mass
  - Compute mass's acceleration **a** = **F**/m
  - Adjust position according to current velocity and velocity according to current acceleration
  - Display (e.g. as a single textured object)

A simple algorithm

- Particle mass m_i is connected to m_j by a spring with rest length r_{ij} and force constant k_{ij}
- Let $\mathbf{x}_i$, $\mathbf{x}_j$ be the position vectors of particles m_i and m_j resp.
- Spring force law (Hooke's law, **F** = -k**x**) on particle m_i due to particle m_j is

$$\mathbf{F}_{ij} = -k_{ij}\left(\left|\mathbf{x}_i - \mathbf{x}_j\right| - r_{ij}\right)\frac{\mathbf{x}_i - \mathbf{x}_j}{\left|\mathbf{x}_i - \mathbf{x}_j\right|}$$

- For each time step $t = t + \Delta t$ *and* each particle *i:*

$$\mathbf{a}_i = \sum_j \mathbf{F}_{ij} \Big/ m_i$$

$$\mathbf{v}_i = \mathbf{v}_i + \mathbf{a}_i \Delta t$$

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \Delta t$$

ODE Solvers

- The above simple minded d.e. solving method is called *Euler's Method*
- Errors accumulate steadily
- Can be unstable
  - e.g. imagine too long a time step
    - Spring goes from a compression of *d* to an expansion of >*d* in one time step
    - Problem blows up!
- Need to use very small step sizes to get tolerable results
  - Expensive and inefficient
- Need a better differential equation solving method

The mid-point method

- Can write **x**(t + Δt) as a Taylor expansion

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

- Euler's method takes 1st 2 terms on RHS. Improve by taking more.
- If take three, get mid-point method: $\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t(f(\mathbf{x} + \frac{\Delta t}{2} f(x))$

$$\text{where} \quad f(\mathbf{x}(t)) = \frac{d\mathbf{x}}{dt}$$

- In words:
  - compute Euler step to get first guess at **x**(t+Δ*t*)
  - Determine mid point **x**(t)+**x**(t+Δ*t*)
  - Evaluate **x'**=d**x**/dt (i.e. forces, accelerations etc) at this point
    - Represents an estimate of average rate of change over the whole step
  - Use that new estimate of rate of change to compute a new step from the starting point

A more-general formulation

- Can define *state* of a particle as **S**=(**x**, **v**) and then have d**S**/dt = (**v**, **a**)
- More generally can define state of whole system as a vector of all (**x**, **v**) pairs, or just as a big vector of floats, 6 per particle.
- Want to keep d.e. solver logic separate, so pass it:
  - A function that returns how many floats are in the state vector (i.e. 6 * n)
  - A function to return the current state vector (i.e. all **x** and **v** values in one 6*n vector)
  - A procedure to accept a new state vector
  - A function to return the derivative of the state vector (i.e. compute all forces and accelerations; return all (**v**, **a**) pairs as a 6*n vector)

Other issues

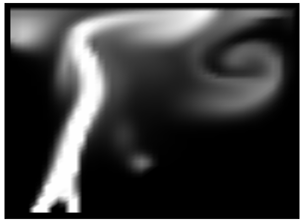
- Even mid-point method often not good enough
  - Use even higher-order methods, e.g. fourth order Runge-Kutta
- Need adaptive step sizes for best efficiency - use long time steps when things are moving slowly, short ones when changes are rapid.
- Easy to incorporate forces other than springs.
  - Usually include some viscous drag for numerical stability
- When collisions occur, have to stop system, back off to collision point, reverse appropriate velocities, restart
- Fundamental limitation of mass-spring systems: “stiffness”.
  - Can’t use very stiff springs to maintain “constraints” (e.g. to try to make a particle follow a particular path)
  - Reason: stiff springs cause things to happen very fast, so require very short time steps. Gets too expensive
- In theory can simulate anything. In practice can only simulate soggy stuff.

6.7 Finite Difference Method (FDM)

- A simple method to solve complex physical models described by partial differential equations.
  - partial differential equations are equations where the unknown variable is a function in one of several variables
  - Example: Heat equation: The change of the temperature distribution T(**x**) in an object is described by the formula

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \frac{\partial T}{\partial t}$$

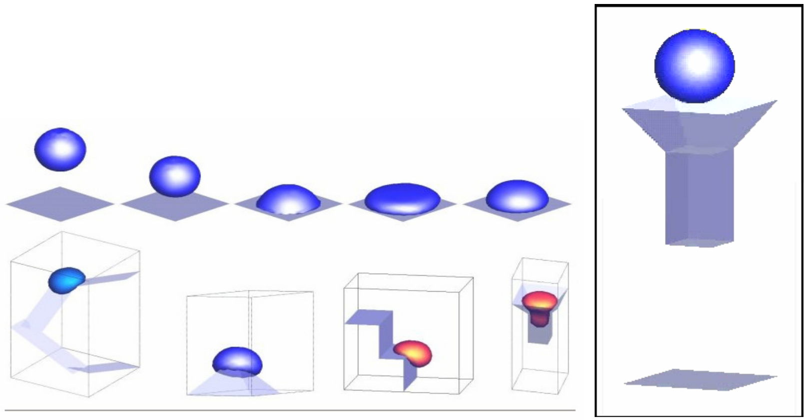
- Popular in computational fluid dynamics



Example

- Daniel Nixon’s Master thesis “A fluid based soft object model” (1998)
  - Goal was to model soft objects such as a cushion or a balloon filled with treacle.
  - Model consists of two components
    - an elastic surface modeled by a mass-spring system.
    - An incompressible fluid enclosed by the surface modelled by a finite difference method approximating the Navier-Stokes equation for fluid flow.
  - In contrast to non-physically based models volume of the object is maintained and surface tension is a natural feature of the model.

Example (continued)

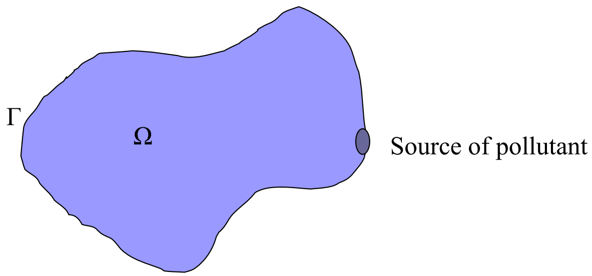


FDM: Solution steps

- Describe problem
  - Problem domain
  - Governing equation
  - Boundary conditions
- Discretize the solution domain
- Replace partial derivatives with finite differences
- Compute unknown variable for all time steps starting with initial values for time step t_0.

Example

- Compute the distribution of a pollutant in a lake Ω with boundary Γ



Problem Description

- Problem domain is Ω
- The governing equation is the diffusion equation, which describes the change in concentration $c(\mathbf{x})$ of a material in a fluid:

$$\frac{\partial c}{\partial t} = D\left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right) \qquad (1)$$

- In order to get a unique solution boundary conditions must be specified:

$$c(\mathbf{x}, t_0) = c_0 \quad \text{for } \mathbf{x} \in \Omega \qquad (2)$$

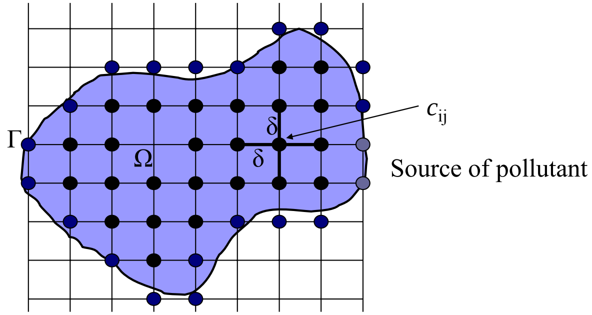
  - For the start time t_0 we need the concentration at all points of the domain.

$$\frac{\partial c}{\partial n} = \gamma_0 \quad \text{for } \mathbf{x} \in \Gamma \qquad (3)$$

  - For all other times we need the change of the concentration in normal direction at the boundary

Discretizing the Solution Domain

- Approximate the domain with a mesh of points. Want to compute concentration c_{ij}^n at all mesh points (*i,j*) and for all time steps t_n.



Discretizing the governing equation

- We discretize the governing equation (1) by replacing partial derivatives with finite differences.

For mesh point $(i, j) \equiv \mathbf{x}$ at time step t_n this gives

$$\frac{\partial c(\mathbf{x}, t_n)}{\partial t} = \frac{c_{i,j}^{n+1} - c_{i,j}^n}{\Delta t}$$

$$\frac{\partial^2 c(\mathbf{x}, t_n)}{\partial x^2} = \frac{\dfrac{c_{i+1,j}^n - c_{i,j}^n}{\delta} - \dfrac{c_{i,j}^n - c_{i-1,j}^n}{\delta}}{\delta} = \frac{c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n}{\delta^2}$$

$$\frac{\partial^2 c(\mathbf{x}, t_n)}{\partial y^2} = \frac{\dfrac{c_{i,j+1}^n - c_{i,j}^n}{\delta} - \dfrac{c_{i,j}^n - c_{i,j-1}^n}{\delta}}{\delta} = \frac{c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n}{\delta^2}$$

Discretizing the governing equation (cont'd)

The governing equation (1) becomes

$$\frac{c_{i,j}^{n+1} - c_{i,j}^n}{\Delta t} = D\left[\frac{c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n}{\delta^2} + \frac{c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n}{\delta^2}\right]$$

$$= D\frac{c_{i+1,j}^n + c_{i-1,j}^n + c_{i,j+1}^n + c_{i,j-1}^n - 4c_{i,j}^n}{\delta^2}$$

and therefore

$$c_{i,j}^{n+1} = c_{i,j}^n + \Delta t D\frac{c_{i+1,j}^n + c_{i-1,j}^n + c_{i,j+1}^n + c_{i,j-1}^n - 4c_{i,j}^n}{\delta^2} \qquad (4)$$

Note that if a boundary point $c_{i,j}$ does not have a neighbour the corresponding finite difference expression must be replaced with the boundary condition at that point.

Compute the solution

- Compute the pollutant concentration for all time steps and all mesh points by starting with the initial concentrations c_{ij}^0 (2) and by computing equation (4) for all time steps.
- The presented solution is relatively unstable. A more advanced method (*Crank-Nicolson* method) replaces spatial derivatives with expressions dependent on both c_{ij}^n and c_{ij}^{n+1}. The concentrations for a new time step are now computed by solving a linear system of equations which can be solved iteratively with the *Gauss-Seidel* method.

6.8 Finite Element Method (FEM)

- Became popular in the 1960's as a tool to solve problems in structural mechanics
- Now common in fields as diverse as medicine (bioengineering) and computer graphics.
- approximates domain with a (frequently irregular) mesh. The cells of the mesh are called *finite elements*.
- uses finite element interpolation functions to interpolate the unknown variable at the mesh vertices over the domain.
- the governing equation is converted to an integral equation, which, when computed over an element results in an equation dependent on the unknown variable at the element vertices. Combining the equations for all elements yields a linear system of equation, whose solution are the unknown variables at the mesh vertices.

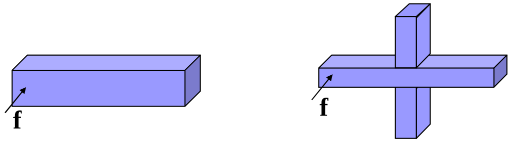
Example- Virtual Face

- Finite elements fitted to skull data.
- Uses theory of large deformation elasticity.
- Unknown variable is the displacement vector at each mesh point.
- Volume textures added for realism.

© 2001, Bioengineering Institute, University of Auckland

6.9 Rigid Body Modelling

- Objects totally inelastic (no deformation)
- Involves extremely complicated mathematics
- Have to consider force, and linear and angular velocity and acceleration, impulse and momentum
- Behaviour of an object is dependent on its centre of gravity and and its inertia tensor, which describes the distribution of mass in the body.



Example

- RobinOtte's Master thesis "Physically Based Modelling and Animation of Rigid Body Systems" (1999)