TIEA311
Tietokonegrafiikan perusteet
kevät 2018

(“Principles of Computer Graphics” – Spring 2018)

Copyright and Fair Use Notice:

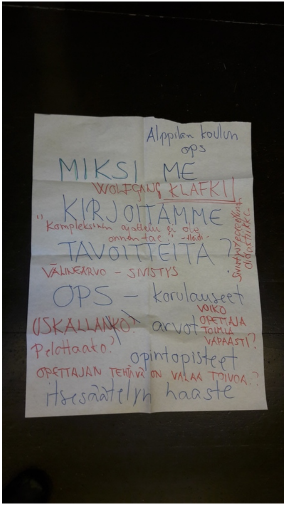
TIEA311 Tietokonegrafiikan perusteet – kevät 2018 (“Principles of Computer Graphics” – Spring 2018)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.

Frontpage of the local course version, held during Spring 2018 at the Faculty of Information technology, University of Jyväskylä: `http://users.jyu.fi/~nieminen/tgp18/`

TIEA311

Your teacher is once again many hours more qualified, with new knowledge:



Alppilan koulun ops

MIKSI ME

WOLFGANG KLAFKI

KIRJOITAMME

"Kompleksinen ajattelu ei ole onnen tae" -Heidi-

TAVOITTEITA ?

VÄLINEARVO - SIVISTYS

Sivistysteoreettinen didaktiikka

OPS - korulauseet

VOIKO OPETTAJA TOIMIA VAPAASTI?

USKALLANKO? arvot

Pelottaato?

opintopisteet

OPETTAJAN TEHTÄVÄ ON VALAA TOIVOA.?

itsesäätelyn haaste

TIEA311 - Today in Jyväskylä

Facing the fact that our original course material from MIT is a full-semester course whereas we only have one half, we need to cut stock a bit. On this lecture, we'll see "teasers" of what we skip, with ideas of where to fit similar material in our curriculum:

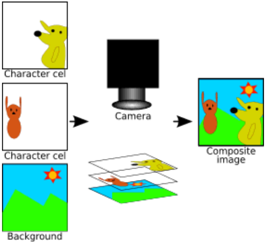
- ▶ While we cover animation from the original "Lecture 6", we skip **skinning**, and the skinning part of "Assignment 2".
- → This topic is covered in the follow-up course "Realtime Rendering" – skinning can be implemented in vertex shaders, which is also a topic of the follow-up course; benefits from quaternions, a piece of math suitable for the follow-up, too.
- ▶ We skip the original Lectures "7–9" about **physical models** and the practical "Assignment 3" that deals with those.
- → Maybe we could revive our own course about "physical models in computer animations" in the (near-ish?) future. . .

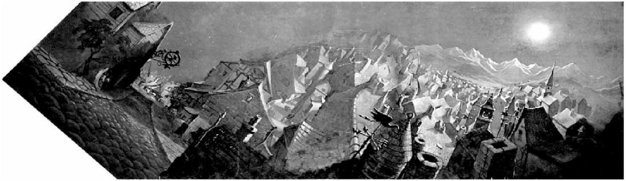
Basics of Computer Animation Skinning/Enveloping

Many slides courtesy of Jovan Popovic, Ronen Barzel, and Jaakko Lehtinen

Traditional Animation

- Draw each frame by hand
  - great control, but tedious
- Reduce burden with cel animation
  - Layer, keyframe, inbetween, …
  - Example: Cel panoramas (Disney's Pinocchio)





From ACM © 1997 "Multiperspective panoramas for cel animation."

Traditional Animation Principles

- The in-betweening, was once a job for apprentice animators. Splines accomplish these tasks automatically. However, the animator still has to draw the keyframes. 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.
- See also The Illusion of Life: Disney Animation, by Frank Thomas and Ollie Johnston.

Example: 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

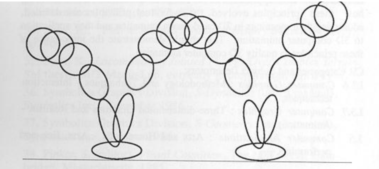


FIGURE 2. Squash & stretch in bouncing ball.

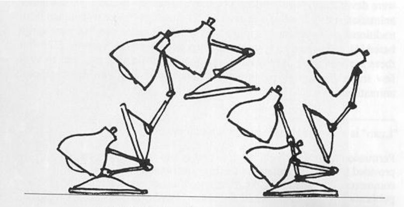


FIGURE 3. Squash & stretch in Luxo Jr.'s hop.

Image adapted from: Lasseter, John. "Principles of Traditional Animation applied to 3D Computer Animation." ACM SIGGRAPH Computer Graphics 21, no. 4 (July 1987): 35-44.

Example: Timing

- Timing affects weight:
  - Light object move quickly
  - Heavier objects move slower

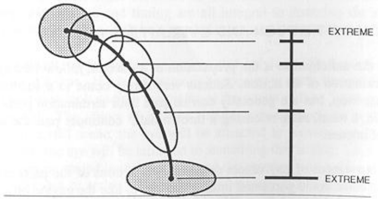
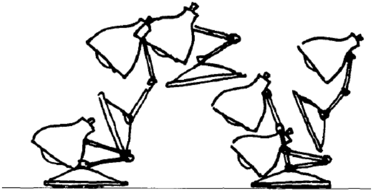


FIGURE 9. Timing chart for ball bounce.

- Timing completely changes the interpretation of the motion.

Computer Animation

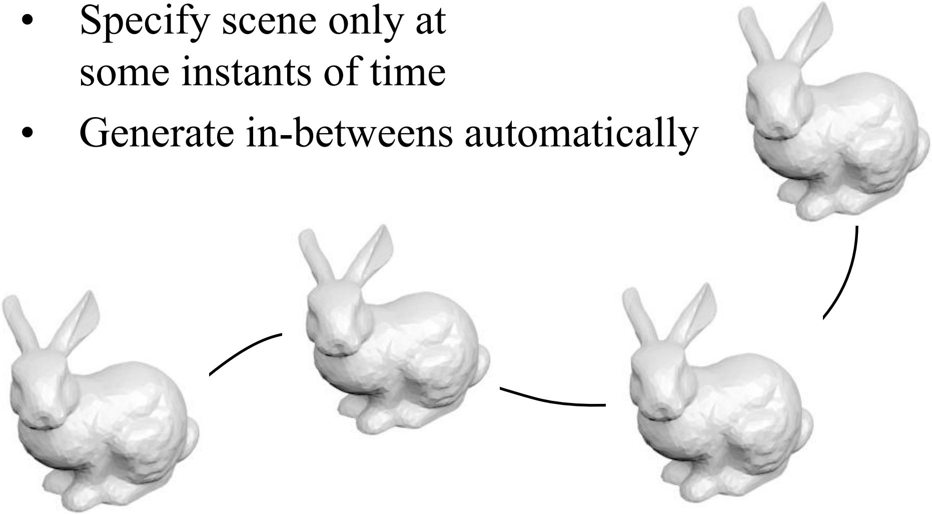
- How do we describe and generate motion of objects in the scene?



- Two very different contexts:
  - Production (offline)
    - Can be hardcoded, entire sequence know beforehand
  - Interactive (e.g. games, simulators)
    - Needs to react to user interaction, sequence not known

Types of Animation: Keyframing

- Specify scene only at some instants of time
- Generate in-betweens automatically

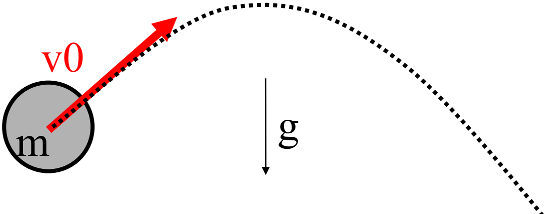


Types of Animation: Procedural

- Describes the motion algorithmically
- Express animation as a function of small number of parameters
- Example
  - a clock/watch with second, minute and hour hands
  - express the clock motions in terms of a "seconds" variable
    - the clock is animated by changing this variable
- Another example: Grass in the wind, tree canopies, etc.

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



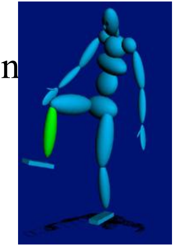
Another Example

- Physically-Based Character Animation
  - Specify keyframes, solve for physically valid motion that interpolates them by “spacetime optimization”

- Anthony C. Fang and Nancy S. Pollard, 2003. Efficient Synthesis of Physically Valid Human Motion, ACM Transactions on Graphics 22(3) 417-426, Proc. SIGGRAPH 2003.http://graphics.cs.cmu.edu/nsp/projects/spacetime/spacetime.html

Because we are Lazy...

- Animation is (usually) specified using some form of low-dimensional **controls** as opposed to remodeling the actual geometry for each frame.
  - Example: The joint angles (bone transformations) in a hierarchical character determine the pose
  - Example: A rigid motion is represented by changing the object-to-world transformation (rotation and translation).



Courtesy Robert C. Duvall, Duke University. License CC BY-NC-SA.

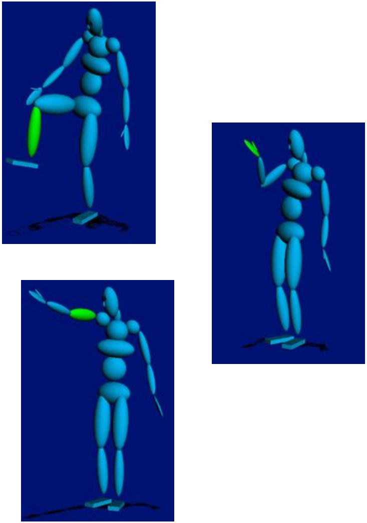
"Blendshapes" are keyframes that are just snapshots of the entire geometry.

Building 3D models and their animation controls is a major component of every animation pipeline.

Building the controls is called “rigging”.

Articulated Character Models

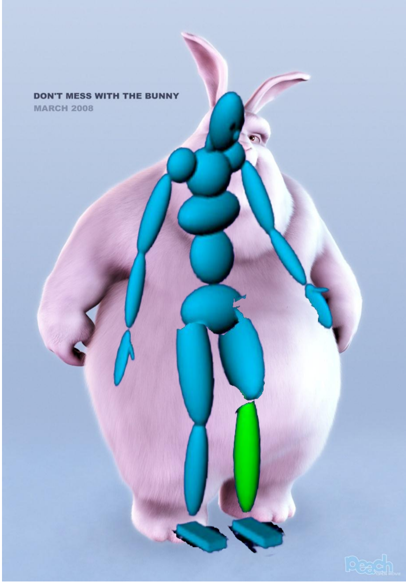
- Forward kinematics describes the positions of the body parts as a function of joint angles
  - Body parts are usually called “bones”
  - Angles are the low-dimensional control.
- Inverse kinematics specifies constraint locations for bones and solves for joint angles.



Courtesy Robert C. Duvall, Duke University. License CC BY-NC-SA.

Skinning Characters

- Embed a skeleton into a detailed character mesh
- Animate "bones"
  - Change the joint angles over time
  - Keyframing, procedural, etc.
- Bind skin vertices to bones
  - Animate skeleton, skin will move with it



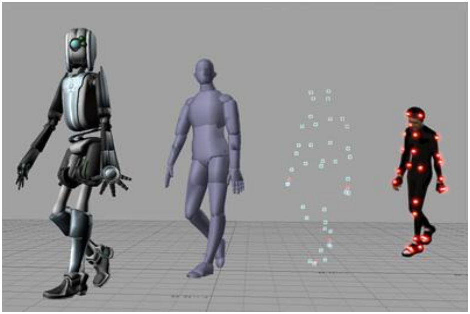
Motion Capture

- Usually uses optical markers and multiple high-speed cameras
- Triangulate to get marker 3D position
  - (Again, structure from motion and projective geometry, i.e., homogeneous coordinates)
- Captures style, subtle nuances and realism
- But need ability to record someone

Courtesy Robert C. Duvall, Duke University. License CC BY-NC-SA.

Motion Capture

- Motion capture records 3D marker positions
  - But character is controlled using animation controls that affect bone transformations!
- Marker positions must be translated into character controls ("retargeting")

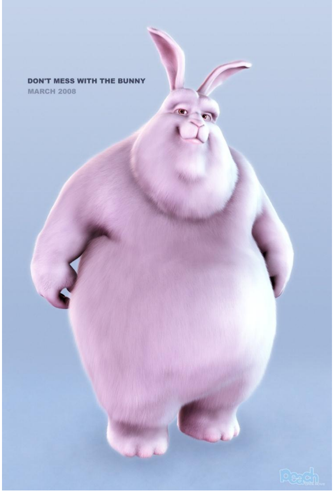


This image is in the public domain. Source: Wikimedia Commons.

Skinning/Enveloping

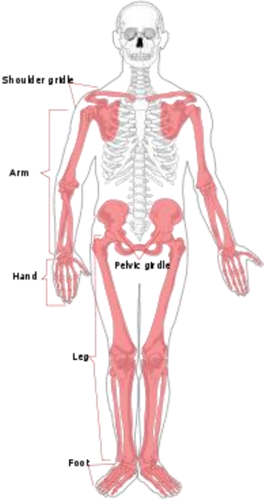
Skinning

- We know how to animate a bone hierarchy
  - Change the joint angles, i.e., bone transformations, over time (keyframing)
- Embed a skeleton into a detailed character mesh
- Bind skin vertices to bones
  - Animate skeleton, skin will move with it
  - But how?



Skinning/Enveloping

- Need to infer how skin deforms from bone transformations.
- Most popular technique: Skeletal Subspace Deformation (SSD), or simply Skinning
  - Other aliases
    - vertex blending
    - matrix palette skinning
    - linear blend skinning



This image is in the public domain. Source: Wikimedia Commons.

TIEA311 - Today in Jyväskylä

Facing the fact that our original course material from MIT is a full-semester course whereas we only have one half, we need to cut stock a bit. On this lecture, we'll see "teasers" of what we skip, with ideas of where to fit similar material in our curriculum:

- ▶ While we cover animation from the original "Lecture 6", we skip **skinning**, and the skinning part of "Assignment 2".
- → This topic is covered in the follow-up course "Realtime Rendering" – skinning can be implemented in vertex shaders, which is also a topic of the follow-up course; benefits from quaternions, a piece of math suitable for the follow-up, too.
- ▶ We skip the original Lectures "7–9" about **physical models** and the practical "Assignment 3" that deals with those.
- → Maybe we could revive our own course about "physical models in computer animations" in the (near-ish?) future. . .

MIT EECS 6.837 Computer Graphics

Particle Systems and ODEs

Image removed due to copyright restrictions.

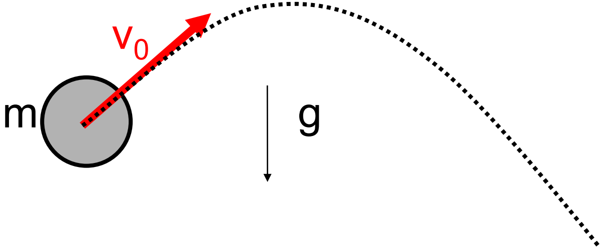
MIT EECS 6.837 – Matusik

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

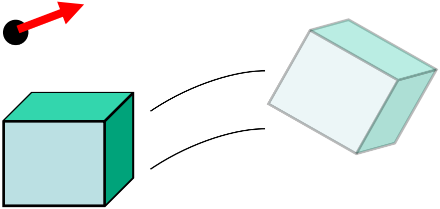
Types of Animation: Physically-Based

- 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
- Rigid body

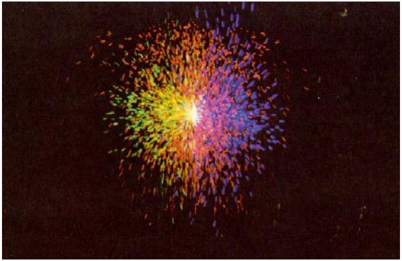


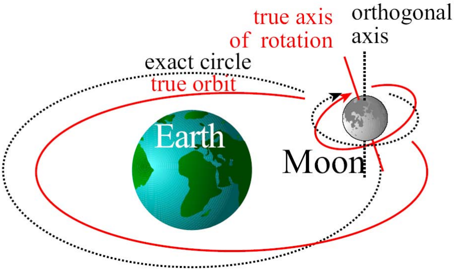
- Deformable body (include clothes, fluids, smoke, etc.)

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"
- 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, …

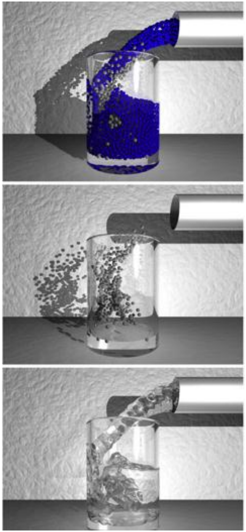
Images of particle systems removed due to copyright restrictions.

http://www.particlesystems.org/

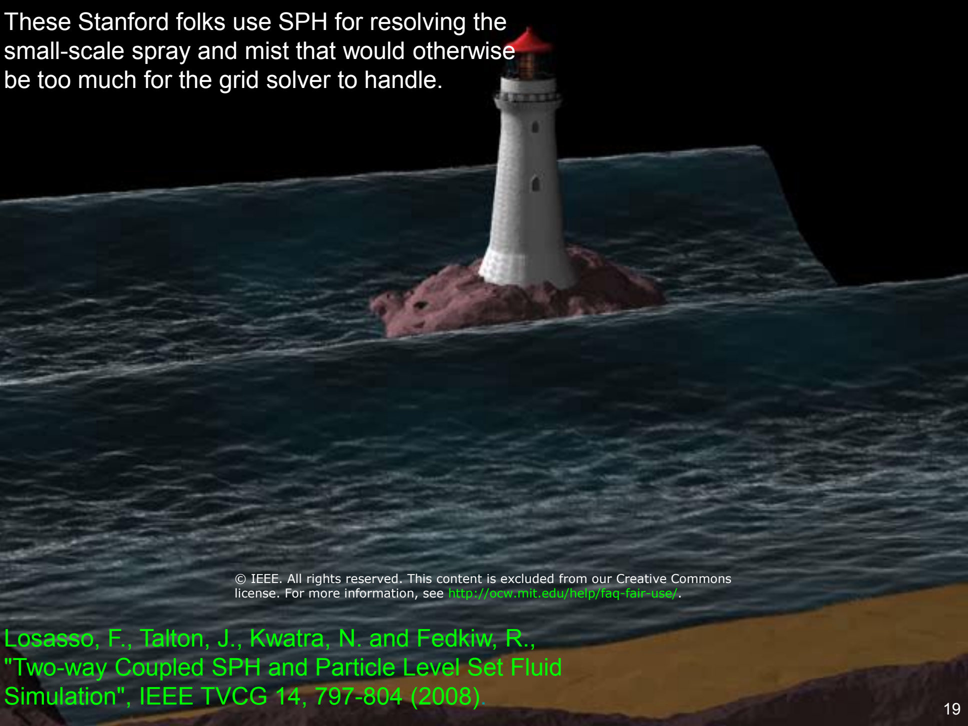
Generalizations

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).

Take-Home Message

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

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

Andrew Selle et al.

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

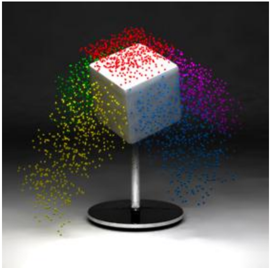


Image courtesy of Halixi72 on Wikimedia Commons. License: CC-BY-SA. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.

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 by Jeff Lander removed due to copyright restrictions.

Image Jeff Lander

Demo with Processing

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

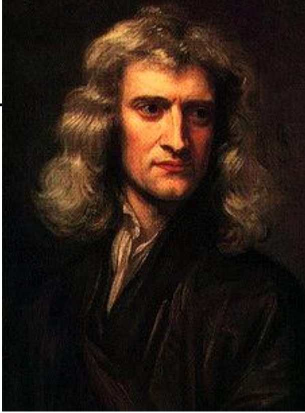
Ordinary Differential Equations

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

- 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



This image is in the public domain.
Source: Wikimedia Commons.

- 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

Now, Many Particles

- We have N point masses
  - Let’s just stack all **x**s and **v**s in a big vector of length 6N
  - **F**^{i} denotes the force on particle *i*
    - When particles don’t interact, **F**^{i} only depends on **x**$_i$ and **v**$_i$.

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

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

Path through a Vector Field

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

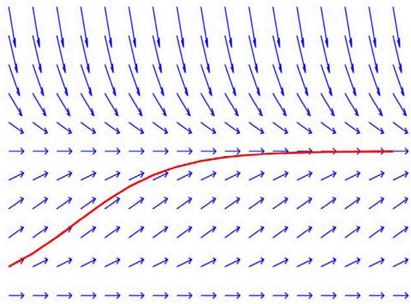


Image by MIT OpenCourseWare.

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

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

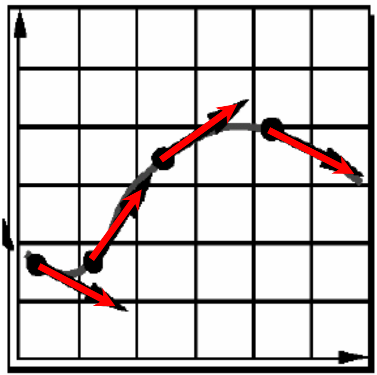
- *f*=d/d*t* ***X*** is a vector that sits at each point in phase space, pointing the direction.

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'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

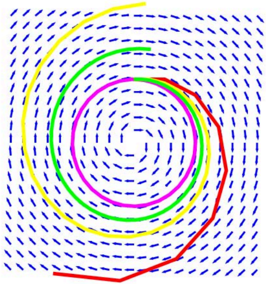


Image by MIT OpenCourseWare.

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

Processing demo

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

Massive software

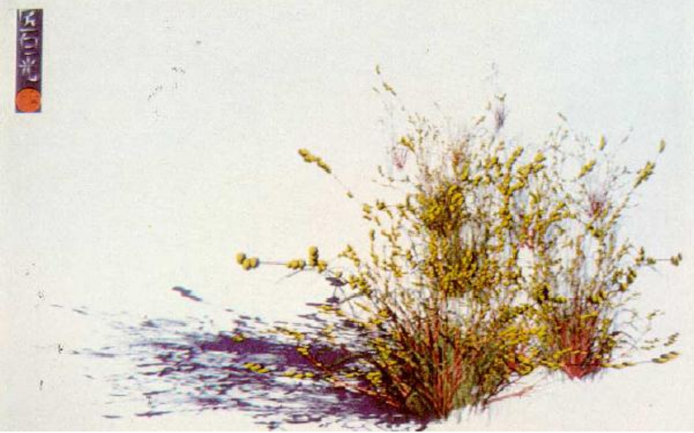
- http://www.massivesoftware.com/
- Used for battle scenes in the Lord of The Rings

Processing demo

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

Particle Modeling [Reeves 1983]

- The grass is made of particles
  - The entire lifetime of the particle is drawn at once.
  - This can be done procedurally on the GPU these days!



William Reeves

MIT EECS 6.837 Computer Graphics

Particle Systems and ODE Solvers II, Mass-Spring Modeling

With slides from Jaakko Lehtinen and others

Image removed due to copyright restrictions.

Picture: A. Selle et al.

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

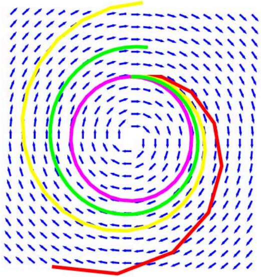
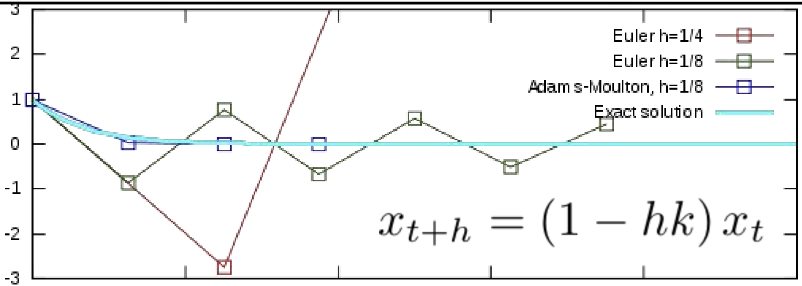


Image by MIT OpenCourseWare.

Euler’s Method: Not Always Stable



Wikipedia user Berland

This image is in the public domain. Source: Wikimedia

- Limited step size!
  - When $0 \leq (1 - hk) < 1 \Leftrightarrow h < 1/k$ things are fine, the solution decays
  - When $-1 \leq (1 - hk) \leq 0 \Leftrightarrow 1/k \leq h \leq 2/k$ we get oscillation
  - When $(1 - hk) < -1 \Leftrightarrow h > 2/k$ things explode

Analysis: Taylor Series

- Expand exact solution **X***(t)*

$$\mathbf{X}(t_0+h)=\mathbf{X}(t_0)+h\left(\tfrac{d}{dt}\mathbf{X}(t)\right)\Big|_{t_0}+\tfrac{h^2}{2!}\left(\tfrac{d^2}{dt^2}\mathbf{X}(t)\right)\Big|_{t_0}+\tfrac{h^3}{3!}\left(\cdots\right)+\cdots$$

- Euler's method approximates:

$$\mathbf{X}(t_0+h)=\mathbf{X}_0+h\,f(\mathbf{X}_0,t_0)\qquad\ldots+O(h^2)\,\text{error}$$

$$h\to h/2\;\Rightarrow\;\; error\to error/4\,\text{per step}\times\text{twice as many steps}$$
$$\to error/2$$

- First-order method: Accuracy varies with *h*
- To get 100x better accuracy need 100x more steps

Can We Do Better?

- Problem: *f* varies along our Euler step
- Idea 1: look at *f* at the arrival of the step and compensate for variation

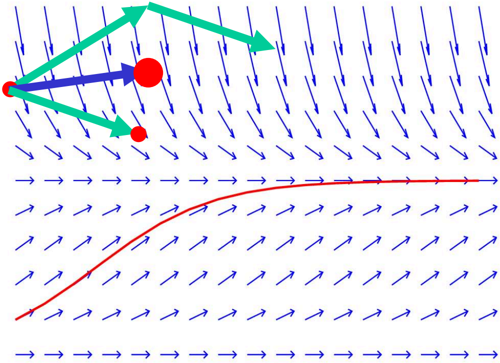


Image by MIT OpenCourseWare.

2nd Order Methods

- This translates to...

$$\boxed{\begin{aligned} f_0 &= f(\mathbf{X}_0, t_0) \\ f_1 &= f(\mathbf{X}_0 + h f_0, t_0 + h) \end{aligned}}$$

- and we get

$$\boxed{\mathbf{X}(t_0 + h) = \mathbf{X}_0 + \tfrac{h}{2}(f_0 + f_1)} + O(h^3)$$

- This is the *trapezoid method*
  - Analysis omitted (see 6.839)
- Note: What we mean by "2nd order" is that the error goes down with h^2 *,* not h – the equation is still 1st order!

Can We Do Better?

- Problem: *f* has varied along our Euler step
- Idea 2: look at *f* after a smaller step, use that value for a full step from initial position

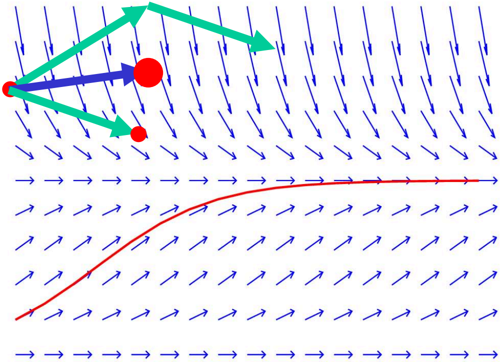


Image by MIT OpenCourseWare.

2nd Order Methods Cont'd

- This translates to...

$$\boxed{\begin{aligned} f_0 &= f(\mathbf{X}_0, t_0) \\ f_m &= f(\mathbf{X}_0 + \tfrac{h}{2} f_0, t_0 + \tfrac{h}{2}) \end{aligned}}$$

- and we get $\boxed{\mathbf{X}(t_0 + h) = \mathbf{X}_0 + h\, f_m} + O(h^3)$

- This is the *midpoint method*
  - Analysis omitted again,
    but it's not very complicated, see here.

Comparison

- Midpoint:
  - ½ Euler step
  - evaluate f_m
  - full step using f_m
- Trapezoid:
  - Euler step (a)
  - evaluate f_1
  - full step using f_1 (b)
  - average (a) and (b)
- Not exactly same resulı, but same order of accuracy

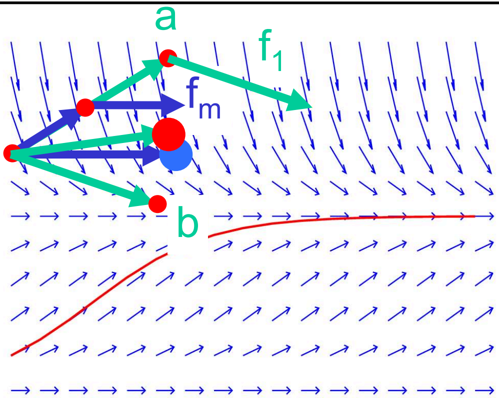


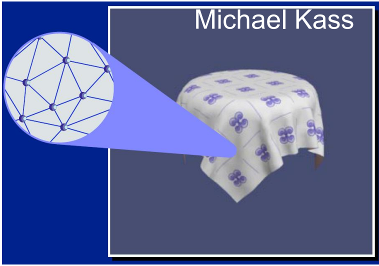
Image by MIT OpenCourseWare.

Can We Do Even Better?

- You bet!
- You will implement Runge-Kutta for assignment 3
- Again, see Witkin, Baraff, Kass: Physically-based Modeling Course Notes, SIGGRAPH 2001
- See eg http://www.youtube.com/watch?v=HbE3L5CIdQg

Mass-Spring Modeling

- Beyond pointlike objects: strings, cloth, hair, etc.
- Interaction between particles
  - Create a network of spring forces that link pairs of particles



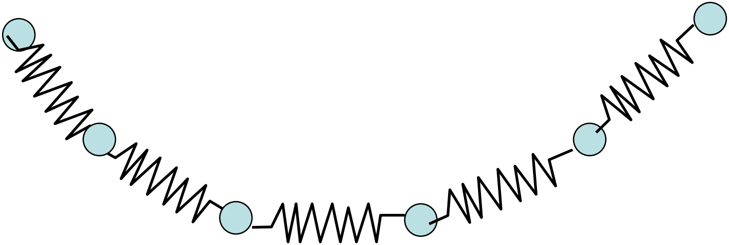
- First, slightly hacky version of cloth simulation
- Then, some motivation/intuition for *implicit integration* (NEXT LECTURE)

Springs

Image courtesy of Jean-Jacques MILAN on Wikimedia Commons. License: CC-BY-SA. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.

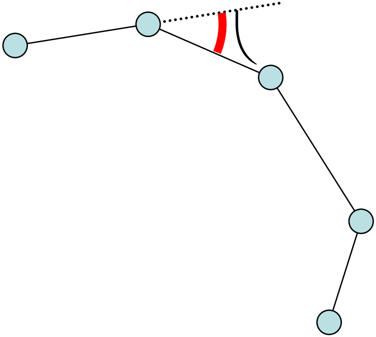
How Would You Simulate a String?

- Springs link the particles
- Springs try to keep their rest lengths and preserve the length of the string
- Not exactly preserved though, and we get oscillation
  - Rubber band approximation



Hair

- Linear set of particles
- Length-preserving **structural** springs like before
- **Deformation** forces proportional to the angle between segments
- **External** forces



Springs for Cloth

- Network of masses and springs
- **Structural** springs:
  - link (i j) and (i+1, j); and (i, j) and (i, j +1)
- **Deformation:**
  - Shear springs
    - (i j) and (i+1, j+1)
  - Flexion springs
    - (i,j) and (i+2,j); (i,j) and (i,j+2)
- See Provot's Graphics Interface '95 paper for details

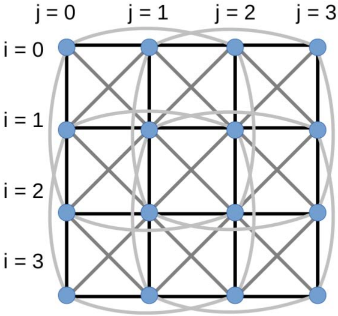


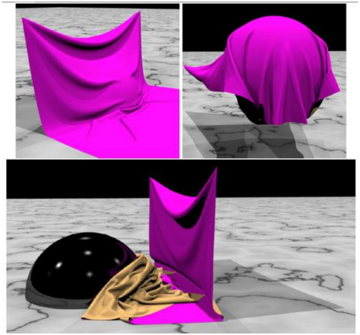
Image by MIT OpenCourseWare.

Provot 95

Collisions

Robert Bridson, Ronald Fedkiw & John Anderson
Robust Treatment of Collisions, Contact
and Friction for Cloth Animation
SIGGRAPH 2002

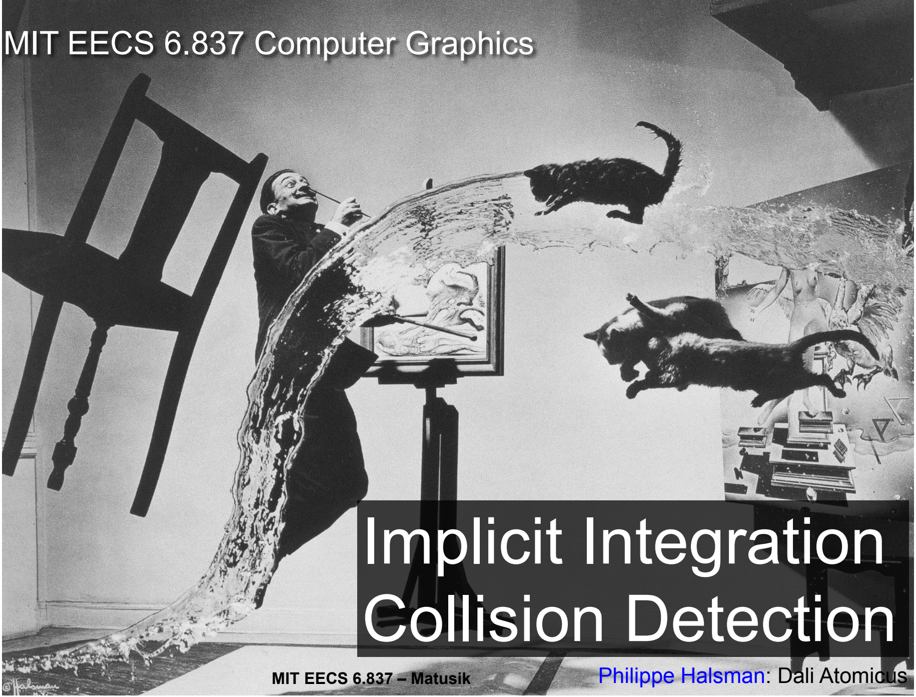
- Cloth has many points of contact
- Need efficient collision detection and stable treatment



Cool Cloth/Hair Demos

- Robert Bridson, Ronald Fedkiw & John Anderson: Robust Treatment of Collisions, Contact and Friction for Cloth Animation SIGGRAPH 2002
- 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).
- Selle, A., Lentine, M. and Fedkiw, R., "A Mass Spring Model for Hair Simulation", SIGGRAPH 2008, ACM TOG 27, 64.1-64.11 (2008).

MIT EECS 6.837 Computer Graphics



Implicit Integration
Collision Detection

Philippe Halsman: Dali Atomicus

This image is in the public domain. Source: Wikimedia Commons.

Plan

- Implementing Particle Systems
- Implicit Integration
- Collision detection and response
  - Point-object and object-object detection
  - Only point-object response

ODEs and Numerical Integration

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

- 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

Integrator Comparison

- Midpoint:
  - ½ Euler step
  - evaluate f_m
  - full step using f_m
- Trapezoid:
  - Euler step (a)
  - evaluate f_1
  - full step using f_1 (b)
  - average (a) and (b)
- Better than Euler but still a speed limit

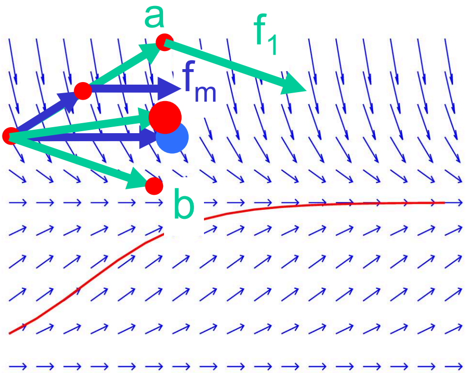


Image by MIT OpenCourseWare.

Midpoint Speed Limit

- $x' = -kx$
- First half Euler step: $x_m = x - 0.5\ hkx = x(1 - 0.5\ hk)$
- Read derivative at x_m: $f_m = -kx_m = -k(1 - 0.5\ hk)x$
- Apply derivative at origin:
  $x(t+h) = x + hf_m = x - hk(1 - 0.5hk)x = x(1 - hk + 0.5\ h^2k^2)$
- Looks a lot like Taylor...
- We want $0 < x(t+h)/x(t) < 1$

  $-hk + 0.5\ h^2k^2 < 0$

  $hk(-1 + 0.5\ hk) < 0$

  For positive values of h & k => $h < 2/k$
- Twice the speed limit of Euler

Stiffness

- In more complex systems, step size is limited by the largest *k*.
  - One stiff spring can ruin things for everyone else!
- Systems that have some big *k* values are called *stiff systems*.
- In the general case, *k* values are eigenvalues of the local Jacobian!

From the siggraph PBM notes

Simple Closed Form Case

Implicit Euler is unconditionally stable!

- Explicit Euler: $x(t+h) = (1\text{-}hk)\, x(t)$

- Implicit Euler: $x(t+h) = x(\mathrm{t}) + h\, x'(t+h)$

$$x(t+h) = x(t) - h\, k\, x(\mathrm{t+h})$$

$$= x(t) \,/\, \boxed{(1+hk)}$$

  - It is a hyperbola!

$1/(1+hk) < 1$,
when h,k > 0

Newton's Method – N Dimensions

- Now locations $\boldsymbol{X}_i$, $\boldsymbol{X}_{i+1}$ and F are N-D
- Newton solution of $F(\boldsymbol{X}_{i+1}) = 0$ is just like 1D:

$$J_F(\boldsymbol{X}_i)(\boldsymbol{X}_{i+1} - \boldsymbol{X}_i) = -F(\boldsymbol{X}_i)$$

NxN Jacobian matrix

unknown N-D step from current to next guess

$$J_F(\boldsymbol{X}_i) = \left[\frac{\partial F}{\partial X}\right]_{\boldsymbol{X}_i}$$

- Must solve a linear system at each step of Newton iteration
  – Note that also Jacobian changes for each step

Detecting Collisions

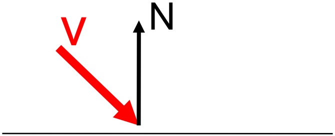
- Easy with implicit equations of surfaces:

  $H(x,y,z) = 0$ on the surface
  $H(x,y,z) < 0$ inside surface

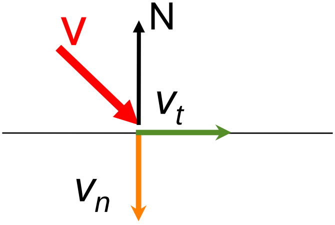
- So just compute H and you know that you are inside if it is negative

- More complex with other surface definitions like meshes
  - A mesh is not necessarily even closed, what is inside?

Collision Response for Particles



Collision Response for Particles



$$v = v_n + v_t$$

normal component
tangential component

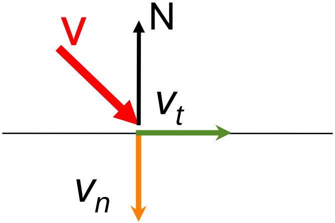
Collision Response for Particles

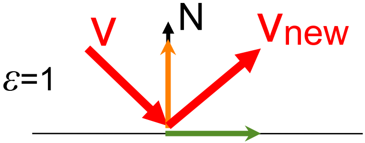
- Tangential velocity v_t *often* unchanged
- Normal velocity v_n reflects:

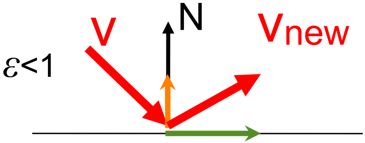
$$v = v_t + v_n$$

$$v \leftarrow v_t - \varepsilon v_n$$

- Coefficient of restitution ε
- When ε = *1*, mirror reflection

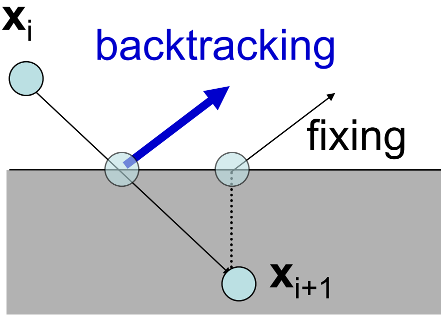






Collisions – Overshooting

- Usually, we detect collision when it is too late: we are already inside
- Solution: Back up
  - Compute intersection point
  - Ray-object intersection!
  - Compute response there
  - Advance for remaining fractional time step
- Other solution: Quick and dirty hack
  - Just project back to object closest point



Collision Detection in Big Scenes

- Imagine we have *n* objects. Can we test all pairwise intersections?
  - Quadratic cost $O(n^2)$!

- Simple optimization: separate static objects
  - But still O(static × dynamic+ dynamic^2)

Hierarchical Collision Detection

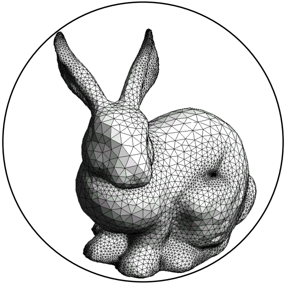
- Use simpler conservative proxies (e.g. bounding spheres)

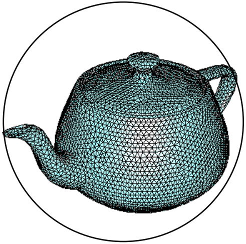
- Recursive (hierarchical) test
  - Spend time only for parts of the scene that are close

- Many different versions, we will cover only one

Bounding Spheres

- Place spheres around objects
- If spheres do not intersect, neither do the objects!
- Sphere-sphere collision test is easy.

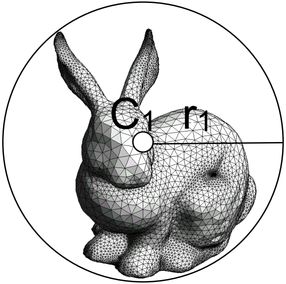


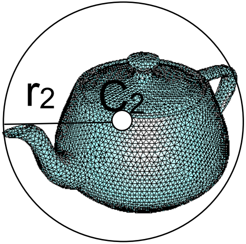


Courtesy of Patrick Laug. Used with permission.

Sphere-Sphere Collision Test

- Two spheres, centers C_1 and C_2, radii r_1 and r_2
- Intersect only if $||C_1C_2||<r_1+r_2$

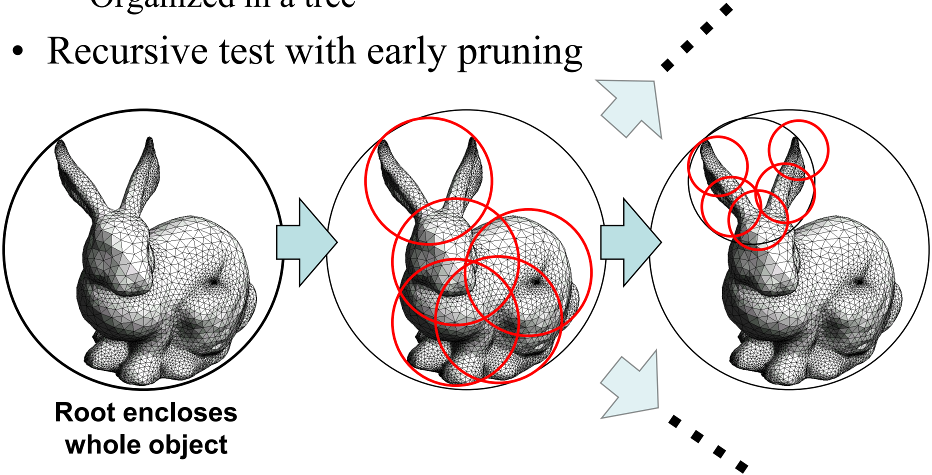




Courtesy of Patrick Laug. Used with permission.

Hierarchical Collision Test

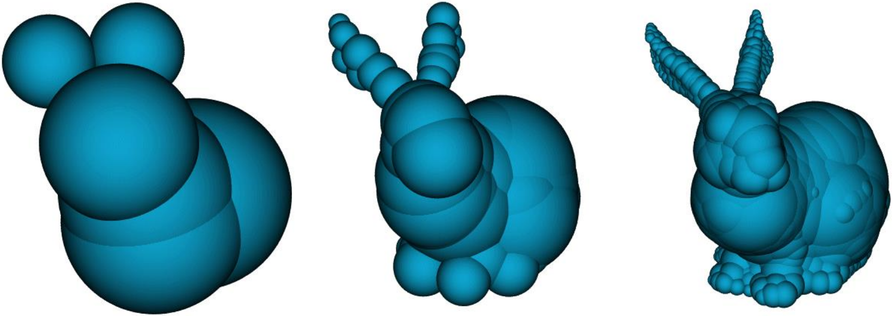
- Hierarchy of bounding spheres
  - Organized in a tree
- Recursive test with early pruning



Root encloses whole object

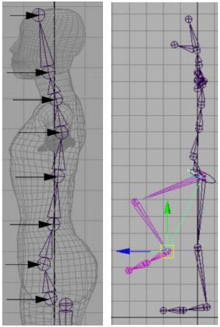
Examples of Hierarchy

- http://isg.cs.tcd.ie/spheretree/



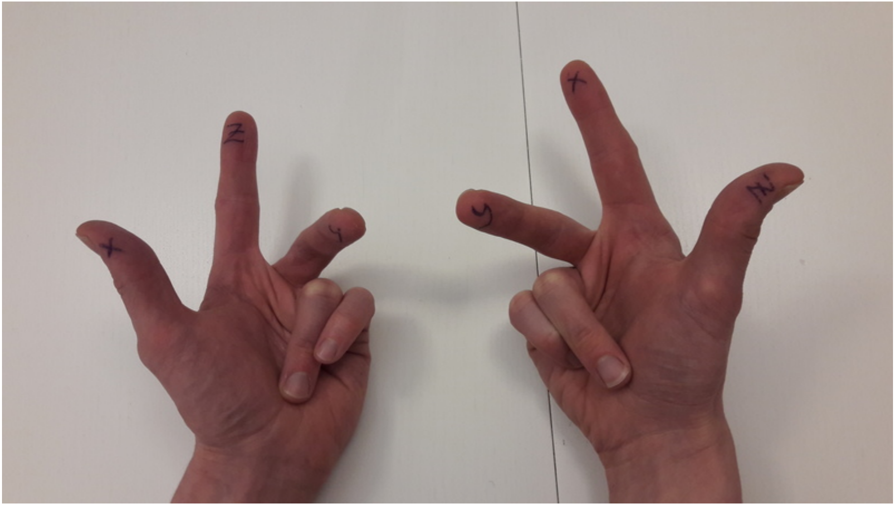
How Do They Animate Movies?

- Keyframing mostly
- Articulated figures, inverse kinematics
- Skinning
  - Complex deformable skin, muscle, skin motion
- 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 (rigging)
- Physical simulation for secondary motion
  - Hair, cloths, water
  - Particle systems for “fuzzy” objects



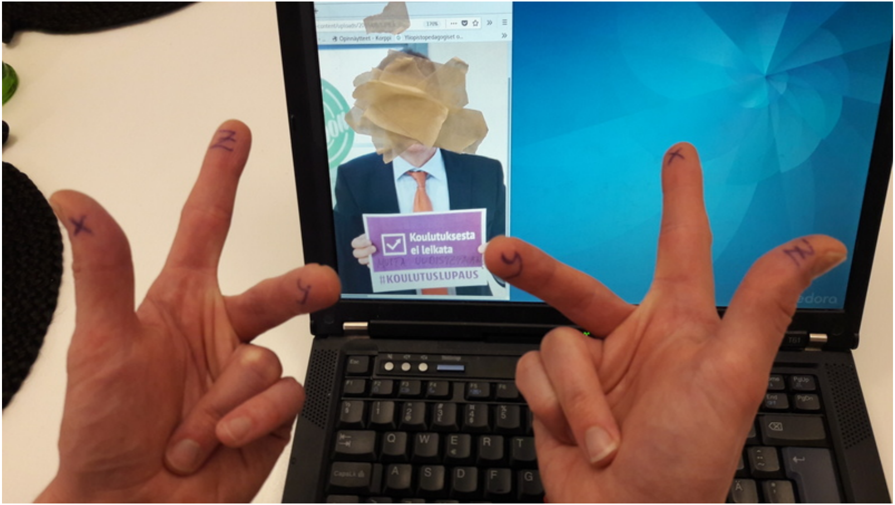
Images from the Maya tutorial

Your nearly-qualified teacher also observed he has two right hands – only a matter of symbols and level of abstraction.



TIEA311

Y-axes pointing out a message shown in a random image found in the Internet.



TIEA311 - Today in Jyväskylä

Now: Break until tomorrow morning. Sleep if you have time.

Tomorrow (if Visual Studio allows):

- Assignment 2 and 4 live.
- C++ static member functions (i.e., “static methods”)
- C++ object instantiation using constructors, operator overloading, temporary objects, pass-by-value vs. pass-by-reference
- C++ (and C) pass-by-pointer
- C++ pointer types and inheritance
- Dots, asterisks, ampersands, and arrows in C++ (and C)