Implicit Integration
Collision Detection

Philippe Halsman: Dali Atomicus

This image is in the public domain. Source: [Wikimedia Commons](https://commons.wikimedia.org/wiki/File:DaliAtomicus.jpg).
Midterm

- Tuesday, October 16\textsuperscript{th} 2:30pm – 4:00pm
- In class
- Two-pages of notes (double sided) allowed
Plan

• Implementing Particle Systems
• Implicit Integration
• Collision detection and response
  – Point-object and object-object detection
  – Only point-object response
ODEs and Numerical Integration

\[ \frac{dX(t)}{dt} = f(X(t), t) \]

- Given a function \( f(X, t) \) compute \( X(t) \)
- Typically, initial value problems:
  - Given values \( X(t_0) = X_0 \)
  - Find values \( X(t) \) for \( t > t_0 \)

- We can use lots of standard tools
ODE: Path Through a Vector Field

• $X(t)$: path in multidimensional phase space

\[
\frac{d}{dt} X = f(X, t)
\]

“When we are at state $X$ at time $t$, where will $X$ be after an infinitely small time interval $dt$?”

• $f=d/dt X$ is a vector that sits at each point in phase space, pointing the direction.
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 do not interact, $F^i$ only depends on $x_i$ and $v_i$. 
  
\[
X = \begin{pmatrix}
  x_1 \\
  v_1 \\
  \vdots \\
  x_N \\
  v_N 
\end{pmatrix} \quad f(X, t) = \begin{pmatrix}
  v_1 \\
  F^1(X, t) \\
  \vdots \\
  v_N \\
  F^N(X, t) 
\end{pmatrix}
\]

$f$ gives $d/dt X$, remember!
• It pays off to abstract (as usual)
  – It’s easy to design your “Particle System” and “Time Stepper” to be unaware of each other

• Basic idea
  – “Particle system” and “Time Stepper” communicate via floating-point vectors $X$ and a function that computes $f(X,t)$
    • “Time Stepper” does not need to know anything else!
Basic idea

- “Particle System” tells “Time Stepper” how many dimensions (N) the phase space has.
- “Particle System” has a function to write its state to an N-vector of floating point numbers (and read state from it).
- “Particle System” has a function that evaluates \( f(\mathbf{X}, t) \), given a state vector \( \mathbf{X} \) and time \( t \).

- “Time Stepper” takes a “Particle System” as input and advances its state.
class ParticleSystem
{
    virtual int getDimension();
    virtual setDimension(int n);
    virtual float* getStatePositions();
    virtual setStatePositions(float* positions);
    virtual float* getStateVelocities();
    virtual setStateVelocities(float* velocities);
    virtual float* getForces(float* positions, float* velocities);
    virtual setMasses(float* masses);
    virtual float* getMasses();

    float* m_currentState
};
Time Stepper Class

class TimeStepper
{
    virtual takeStep(ParticleSystem* ps, float h)
}

Forward Euler Implementation

class ForwardEuler : TimeStepper
{
    void takeStep(ParticleSystem* ps, float h)
    {
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        newPositions = positions + h*velocities
        newVelocities = velocities + h*accelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
    }
}
class MidPoint : TimeStepper
{
    void takeStep(ParticleSystem* ps, float h)
    {
        velocities = ps->getStateVelocities()
        positions = ps->getStatePositions()
        forces = ps->getForces(positions, velocities)
        masses = ps->getMasses()
        accelerations = forces / masses
        midPositions = positions + 0.5*h*velocities
        midVelocities = velocities + 0.5*h*accelerations
        midForces = ps->getForces(midPositions, midVelocities)
        midAccelerations = midForces / masses
        newPositions = positions + 0.5*h*midVelocities
        newVelocities = velocities + 0.5*h*midAccelerations
        ps->setStatePositions(newPositions)
        ps->setStateVelocities(newVelocities)
    }
}
Particle System Simulation

```java
ps = new MassSpringSystem(particleCount, masses, springs, externalForces)
stepper = new ForwardEuler()
time = 0
while time < 1000
    stepper->takeStep(ps, 0.0001)
time = time + 0.0001
// render
```
Ps = new MassSpringSystem(particleCount, masses, springs, externalForces)
stepper = new MidPoint()
time = 0
while time < 1000
    stepper->takeStep(ps, 0.0001)
time = time + 0.0001
// render
Computing Forces

- When computing the forces, initialize the force vector to zero, then sum over all forces for each particle
  - Gravity is a constant acceleration
  - Springs connect two particles, affects both
  - $\frac{dv_i}{dt} = F_i(X, t)$ is the vector sum of all forces on particle $i$
  - For 2nd order $F^i=m_ia_i$ system, $dx_i/dt$ is just the current $v_i$

$$f(X, t) = \begin{pmatrix} v_1 \\ F^1(X, t) \\ \vdots \\ v_N \\ F^N(X, t) \end{pmatrix}$$
Questions?

Image removed due to copyright restrictions.
Euler Has a Speed Limit!

- $h > 1/k$: oscillate. $h > 2/k$: explode!

Image removed due to copyright restrictions -- please see slide 5 on "Implicit Methods" from Online Siggraph '97 Course notes, available at [http://www.cs.cmu.edu/~baraff/sigcourse/](http://www.cs.cmu.edu/~baraff/sigcourse/).
Integrator Comparison

- **Midpoint:**
  - \(\frac{1}{2}\) 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
Midpoint Speed Limit

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

© David Baraff and Andrew Witkin. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.
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 \textit{stiff systems}.

• In the general case, $k$ values are eigenvalues of the local Jacobian!

From the siggraph PBM notes

© David Baraff and Andrew Witkin. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.
Explicit Integration

• So far, we have seen explicit Euler
  \[ X(t+h) = X(t) + h X'(t) \]

• We also saw midpoint and trapezoid methods
  – They took small Euler steps, re-evaluated \( X' \) there, and used some combination of these to step away from the original \( X(t) \).
  – Yields higher accuracy, but not impervious to stiffness (twice the speed limit of Euler)
Implicit Integration

• So far, we have seen explicit Euler
  \[ X(t+h) = X(t) + h \, X'(t) \]

• Implicit Euler uses the derivative at the destination!
  \[ X(t+h) = X(t) + h \, X'(t+h) \]
  – It is implicit because we do not have \( X'(t+h) \),
    it depends on where we go (HUH?)

  – aka backward Euler
Difference with Trapezoid

- **Trapezoid**
  - take “fake” Euler step
  - read derivative at “fake” destination

- **Implicit Euler**
  - take derivative at the real destination
  - harder because the derivative depends on the destination and the destination depends on the derivative
Implicit Integration

• Implicit Euler uses the derivative at the destination!
  – \( X(t+h) = X(t) + h \, X'(t+h) \)
  – It is implicit because we do not have \( X'(t+h) \), it depends on where we go (HUH?)
  – Two situations
    • \( X' \) is known analytically and everything is closed form (\textit{doesn’t happen in practice})
    • We need some form of iterative non-linear solver.
Simple Closed Form Case

- Remember our model problem: $x' = -kx$
  - Exact solution was a decaying exponential $x_0 e^{-kt}$

- Explicit Euler: $x(t+h) = (1-hk) x(t)$
  - Here we got the bounds on $h$ to avoid oscillation/explosion
Simple Closed Form Case

- Remember our model problem: $x' = -kx$
  - Exact solution was a decaying exponential $x_0 e^{-kt}$

- Explicit Euler: $x(t+h) = (1-hk) x(t)$

- Implicit Euler: $x(t+h) = x(t) + h x'(t+h)$
Simple Closed Form Case

- Remember our model problem: \( x' = -kx \)
  - Exact solution was a decaying exponential \( x_0 e^{-kt} \)

- Explicit Euler: \( x(t+h) = (1-hk) x(t) \)

- Implicit Euler: \( x(t+h) = x(t) + h x'(t+h) \)
  \[
  x(t+h) = x(t) - hk x(t+h) \\
  x(t+h) + hk x(t+h) = x(t) \\
  x(t+h) = x(t) / (1+hk)
  \]
  - It is a hyperbola!
Implicit Euler is unconditionally stable!

- Explicit Euler: \( x(t+h) = (1-hk) \cdot x(t) \)

- Implicit Euler: \( x(t+h) = x(t) + h \cdot x'(t+h) \)
  \[
  x(t+h) = x(t) - h \cdot k \cdot x(t+h) \\
  \quad = x(t) / (1+hk)
  \]
  - It is a hyperbola!  
    \( 1/(1+hk) < 1, \) when \( h,k > 0 \)
Implicit vs. Explicit

Image removed due to copyright restrictions -- please see slide 12 on "Implicit Methods" from Online Siggraph '97 Course notes, available at http://www.cs.cmu.edu/~baraff/sigcourse/.

From the Siggraph PBM notes
Implicit vs. Explicit

Questions?

Image removed due to copyright restrictions -- please see slide 12 on "Implicit Methods" from Online Siggraph '97 Course notes, available at http://www.cs.cmu.edu/~baraff/sigcourse/.
Implicit Euler, Visually

\[ X_{i+1} = X_i + h f( X_{i+1}, t+h ) \]
\[ X_{i+1} - h f( X_{i+1}, t+h ) = X_i \]
Implicit Euler, Visually

\[ X_{i+1} = X_i + hf(X_{i+1}, t+h) \]
\[ X_{i+1} - hf(X_{i+1}, t+h) = X_i \]

What is the location \( X_{i+1} = X(t+h) \) such that the derivative there, multiplied by \(-h\), points back to \( X_i = X(t) \) where we are starting from?
Implicit Euler in 1D

- To simplify, consider only 1D time-invariant systems
  - This means \( x' = f(x, t) = f(x) \) is independent of \( t \)
  - Our spring equations satisfy this already

- \( x(t+h) = x(t) + dx = x(t) + h f(x(t+h)) \)
- \( f \) can be approximated it by 1\(^{st}\) order Taylor:
  \( f(x+dx) = f(x) + dx f'(x) + O(dx^2) \)
- \( x(t+h) = x(t) + h \left[ f(x) + dx f'(x) \right] \)
- \( dx = h \left[ f(x) + dx f'(x) \right] \)
- \( dx = hf(x) / [1 - hf'(x)] \)
- Pretty much Newton solution
Newton’s Method (1D)

- Iterative method for solving non-linear equations
  \[ f(x) = 0 \]
- Start from initial guess \( x_0 \), then iterate
Newton’s Method (1D)

- Iterative method for solving non-linear equations
  \[ f(x) = 0 \]

- Start from initial guess \( x_0 \), then iterate
  \[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

- Also called *Newton-Raphson iteration*
Newton’s Method (1D)

• Iterative method for solving non-linear equations

\[ f(x) = 0 \]

• Start from initial guess \( x_0 \), then iterate

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

\[ \iff f'(x_i)(x_{i+1} - x_i) = -f(x_i) \]

one step
Newton, Visually

This image is in the public domain. Source: [Wikimedia Commons](https://commons.wikimedia.org/wiki/File:Newton_method.png)."
Newton, Visually

Let’s approximate $f$ by its tangent at point $(x_n, f(x_n))$.

This image is in the public domain. Source: [Wikimedia Commons](https://commons.wikimedia.org/wiki/Newton%27s_method#/media/File:Newton_method.png).
Newton, Visually

Let’s approximate $f$ by its tangent at point $(x_n, f(x_n))$

Then we’ll see where the tangent line crosses zero and take that as next guess

We are here
Newton, Visually

Image courtesy of Ralf Pfeifer 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/.
Newton, Visually

Questions?

Image courtesy of Ralf Pfeifer 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/.
Implicit Euler and Large Systems

• To simplify, consider only time-invariant systems
  – This means $X' = f(X,t) = f(X)$ is independent of $t$
  – Our spring equations satisfy this already

• Implicit Euler with $N$-$D$ phase space:
  $X_{i+1} = X_i + h f(X_{i+1})$
To simplify, consider only time-invariant systems
- This means \( X' = f(X,t) = f(X) \) is independent of \( t \)
- Our spring equations satisfy this already

Implicit Euler with \( N-D \) phase space:
\[
X_{i+1} = X_i + h f(X_{i+1})
\]

Non-linear equation,
unknown \( X_{i+1} \) on both the LHS and the RHS
Newton’s Method – N Dimensions

- 1D: \[ f'(x_i)(x_{i+1} - x_i) = -f(x_i) \]

- Now locations \( X_i, X_{i+1} \) and \( F \) are N-D
- N-D Newton step is just like 1D:

\[
J_F(X_i)(X_{i+1} - X_i) = -F(X_i)
\]

NxN Jacobian matrix replaces unknown N-D step from \( f' \) current to next guess
Newton’s Method – N Dimensions

• Now locations \( X_i, X_{i+1} \) and \( F \) are N-D

• Newton solution of \( F(X_{i+1}) = 0 \) is just like 1D:

\[
J_F(X_i)(X_{i+1} - X_i) = -F(X_i)
\]

\( J_F(X_i) \) is an \( N \times N \) Jacobian matrix

• Must solve a linear system at each step of Newton iteration
  – Note that also Jacobian changes for each step
Newton’s Method – N Dimensions

• Now locations $X_i$, $X_{i+1}$ and $F$ are N-D
• Newton solution of $F(X_{i+1}) = 0$ is just like 1D:

$$J_F(X_i)(X_{i+1} - X_i) = -F(X_i)$$

NxN Jacobian matrix unknown N-D step from current to next guess

• Must solve a linear system at each step of Newton iteration
  – Note that also Jacobian changes for each step
Implicit Euler – N Dimensions

• Implicit Euler with $N$-$D$ phase space:
  \[ X_{i+1} = X_i + hf(X_{i+1}) \]

• Let’s rewrite this as $F(Y) = 0$, with

  \[ F(Y) = Y - X_i - hf(Y) \]
Implicit Euler – N Dimensions

- Implicit Euler with $N$-$D$ phase space:
  \[ X_{i+1} = X_i + hf(X_{i+1}) \]

- Let’s rewrite this as $F(Y) = 0$, with
  \[ F(Y) = Y - X_i - hf(Y) \]

- Then the $Y$ that solves $F(Y)=0$ is $X_{i+1}$
Implicit Euler – N Dimensions

\[ F(Y) = Y - X_i - h f(Y) \]

- \( Y \) is variable \( X_i \) is fixed

- Then iterate
  - Initial guess \( Y_0 = X_i \) (or result of explicit method)
  - For each step, solve \( J_F(Y_i) \Delta Y = -F(Y_i) \)
  - Then set \( Y_{i+1} = Y_i + \Delta Y \)
What is the Jacobian?

\[ F(Y) = Y - X_i - hf(Y) \]

• Simple partial differentiation...

\[ J_F(Y) = \left[ \frac{\partial F}{\partial Y} \right] = I - hJ_f(Y) \]

• Where

\[ J_f(Y) = \left[ \frac{\partial f}{\partial Y} \right] \]

The Jacobian of the Force function f
Putting It All Together

• Iterate until convergence
  
  – Initial guess $Y_0 = X_i$ (or result of explicit method)
  
  – For each step, solve
    
    $$\left(I - h J_f(Y_i)\right) \Delta Y = -F(Y_i)$$
    
  – Then set $Y_{i+1} = Y_i + \Delta Y$
Implicit Euler with Newton, Visually
Implicit Euler with Newton, Visually

What is the location $X_{i+1} = X(t+h)$ such that the derivative there, multiplied by $-h$, points back to $X_i = X(t)$ where we are starting from?

$X_i = Y_0$

$-hf(X,t)$

$Y = X_{i+1}$

Image by MIT OpenCourseWare.
One-Step Cheat

• Often, the 1\textsuperscript{st} Newton step may suffice
  – People often implement Implicit Euler using only one step.
  – This amounts to solving the system

\[
\left( I - h \frac{\partial f}{\partial X} \right) \Delta X = hf(X)
\]

where the Jacobian and $f$ are evaluated at $X_i$, and we are using $X_i$ as an initial guess.
• Often, the 1st Newton step may suffice
  – People often implement Implicit Euler using only one step.
  – This amounts to solving the system

\[
\left( I - h \frac{\partial f}{\partial X} \right) \Delta X = hf(X)
\]

where the Jacobian and \( f \) are evaluated at \( X_i \), and we are using \( X_i \) as an initial guess.
Good News

• The Jacobian matrix $J_f$ is usually sparse
  – Only few non-zero entries per row
  – E.g. the derivative of a spring force only depends on the adjacent masses’ positions

• Makes the system cheaper to solve
  – Don’t invert the Jacobian!
  – Use iterative matrix solvers like conjugate gradient, perhaps with preconditioning, etc.

$$\left( I - J_f(Y_i) \right) \Delta Y = -F(Y_i)$$
Implicit Euler Pros & Cons

- **Pro:** Stability!

- **Cons:**
  - Need to solve a linear system at each step
  - Stability comes at the cost of “numerical viscosity”, but then again, you do not have to worry about explosions.
    - Recall exp vs. hyperbola

- **Note that accuracy is not improved**
  - error still $O(h)$
  - There are lots and lots of implicit methods out there!
Reference

• Large steps in cloth simulation
• David Baraff  Andrew Witkin
• http://portal.acm.org/citation.cfm?id=280821
Animation removed due to copyright restrictions.
Simulating Knitted Cloth at the Yarn Level
Jonathan Kaldor, Doug L. James, and Steve Marschner

Animation removed due to copyright restrictions.
Efficient Simulation of Inextensible Cloth
Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, Eitan Grinspun

Animation removed due to copyright restrictions.
Questions?
Collisions

- Detection
- Response
- Overshooting problem
  (when we enter the solid)
Detecting Collisions

- Easy with implicit equations of surfaces:

\[ H(x,y,z) = 0 \quad \text{on the surface} \]
\[ H(x,y,z) < 0 \quad \text{inside surface} \]

- 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

\[ \mathbf{v} = \mathbf{v}_n + \mathbf{v}_t \]

- 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 $\varepsilon$
- When $\varepsilon = 1$, mirror reflection
  - $\varepsilon < 1$
  - $\varepsilon = 1$
Collisions – Overshooting

• Usually, we detect collision when it is too late: we are already inside
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
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
Questions?

- **Pong:** $\varepsilon =$?
  - [http://www.youtube.com/watch?v=sWY0Q_lMFfw](http://www.youtube.com/watch?v=sWY0Q_lMFfw)
  - [http://www.xnet.se/javaTest/jPong/jPong.html](http://www.xnet.se/javaTest/jPong/jPong.html)

Animation removed due to copyright restrictions.

This image is in the public domain. Source: [Wikimedia Commons](http://en.wikipedia.org/wiki/Pong).

Image courtesy of [Chris Rand](http://en.wikipedia.org/wiki/Pong) 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/](http://ocw.mit.edu/help/faq-fair-use/).
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 \( O(\text{static} \times \text{dynamic} + \text{dynamic}^2) \)
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.

© Oscar Meruvia-Pastor, Daniel Rypl. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.
Sphere-Sphere Collision Test

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

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

Root encloses whole object
Examples of Hierarchy

boolean intersect(node1, node2)
   // no overlap? ==> no intersection!
   if (!overlap(node1->sphere, node2->sphere))
      return false

   // recurse down the larger of the two nodes
   if (node1->radius()>node2->radius())
      for each child c of node1
         if intersect(c, node2) return true
   else
      for each child c of node2
         if intersect(c, node1) return true

   // no intersection in the subtrees? ==> no intersection!
   return false
boolean intersect(node1, node2)
    if (!overlap(node1->sphere, node2->sphere))
        return false
    if (node1->radius() > node2->radius())
        for each child c of node1
            if intersect(c, node2) return true
    else
        for each child c of node2
            if intersect(c, node1) return true
    return false
boolean intersect(node1, node2)
    if (!overlap(node1->sphere, node2->sphere))
        return false
    if (node1->radius() > node2->radius())
        for each child c of node1
            if intersect(c, node2) return true
    else
        for each child c of node2
            if intersect(c, node1) return true
    return false
boolean intersect(node1, node2)
    if (!overlap(node1->sphere, node2->sphere))
        return false
    if (node1->radius() > node2->radius())
        for each child c of node1
            if intersect(c, node2) return true
    else
        for each child c of node2
            if intersect(c, node1) return true
    return false
boolean intersect(node1, node2)
  if (!overlap(node1->sphere, node2->sphere))
    return false
  if (node1->radius()>node2->radius())
    for each child c of node1
      if intersect(c, node2) return true
  else
    for each child c f node2
      if intersect(c, node1) return true
  return false
boolean intersect(node1, node2)
    if (!overlap(node1->sphere, node2->sphere))
        return false
    if (node1->radius() > node2->radius())
        for each child c of node1
            if intersect(c, node2) return true
    else
        for each child c of node2
            if intersect(c, node1) return true
    return false
boolean intersect(node1, node2)
  if (!overlap(node1->sphere, node2->sphere))
    return false
  if (node1->radius()>node2->radius())
    for each child c of node1
      if intersect(c, node2) return true
    else
      for each child c f node2
        if intersect(c, node1) return true
  return false
Pseudocode (with leaf case)

```java
boolean intersect(node1, node2)
    if (!overlap(node1->sphere, node2->sphere)
        return false
    // if there is nowhere to go, test everything
    if (node1->isLeaf() && node2->isLeaf())
        perform full test between all primitives within nodes
    // otherwise go down the tree in the non-leaf path
    if ( !node2->isLeaf() && !node1->isLeaf() )
        // pick the larger node to subdivide, then recurse
    else
        // recurse down the node that is not a leaf
    return false
```
Other Options

- Axis Aligned Bounding Boxes
  - “R-Trees”

- Oriented bounding boxes

- Binary space partitioning trees; kd-trees
Questions?

- http://www.youtube.com/watch?v=b_cGXtc-nMg
- http://www.youtube.com/watch?v=nFd9BIcpHX4&feature=related
- http://www.youtube.com/watch?v=2SXixK7yCGU
Hierarchy Construction

• Top down
  – Divide and conquer

• Bottom up
  – Cluster nearby objects

• Incremental
  – Add objects one by one, binary-tree style.
Bounding Sphere of a Set of Points

- Trivial given center $C$
  - radius $= \max_i ||C-P_i||$
Bounding Sphere of a Set of Points

- Using axis-aligned bounding box
  - \( \text{center} = \frac{(x_{\text{min}} + x_{\text{max}})}{2}, \frac{(y_{\text{min}} + y_{\text{max}})}{2}, \frac{(z_{\text{min}}, z_{\text{max}})}{2} \)
  - Better than the average of the vertices because does not suffer from non-uniform tessellation
• Using axis-aligned bounding box
  
  - center=
    \[((x_{\text{min}} + x_{\text{max}})/2, (y_{\text{min}} + y_{\text{max}})/2, (z_{\text{min}}, z_{\text{max}})/2)\]
  
  - Better than the average of the vertices because does not suffer from non-uniform tessellation
Top-Down Construction

- Take longest scene dimension
- Cut in two in the middle
  - assign each object or triangle to one side
  - build sphere around it
Top-Down Construction - Recurse

- Take longest scene dimension
- Cut in two in the middle
  - assign each object or triangle to one side
  - build sphere/box around it
Top-Down Construction - Recurse

- Take longest scene dimension
- Cut in two in the middle
  - assign each object or triangle to one side
  - build sphere/box around it

Questions?
An image of the book, “Real Time Collision Detection” by Christer Ericson, has been removed due to copyright restrictions.
The Cloth Collision Problem

- A cloth has many points of contact
- Stays in contact
- Requires
  - Efficient collision detection
  - Efficient numerical treatment (stability)

© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.
Robust Treatment of Simultaneous Collisions
David Harmon, Etienne Vouga, Rasmus Tamstorf, Eitan Grinspun

Animation removed due to copyright restrictions.
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
That’s All for Today!