CS-C3100 Computer Graphics
Jaakko Lehtinen

Implicit Integration
Collision Detection

Good pointers to extra credit in Assignment 4)

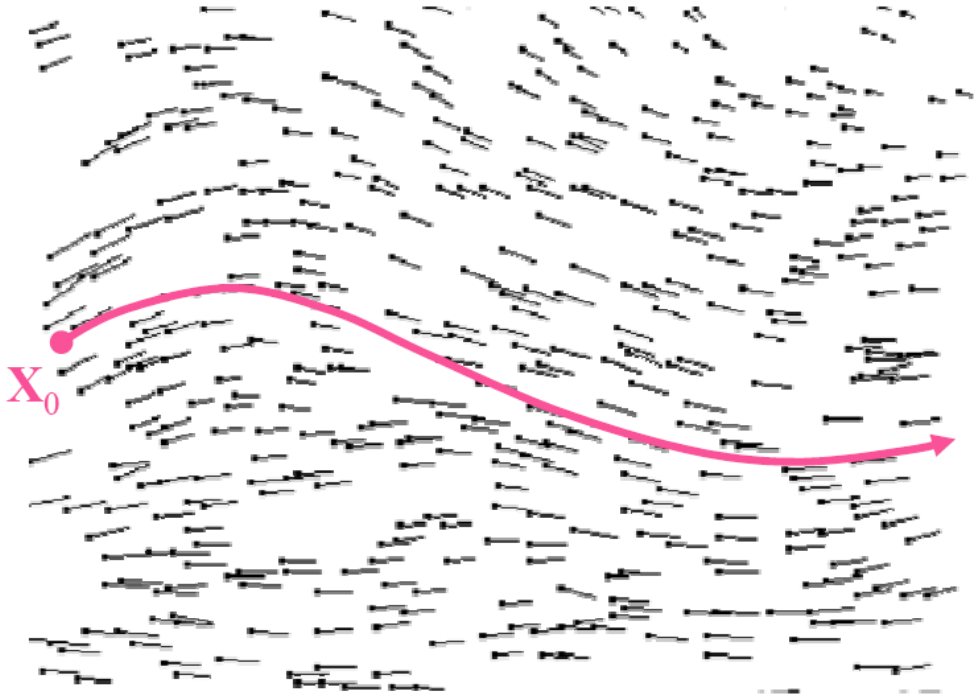
Philippe Halsman: Dali Atomicus

In These Slides

- Implicit Integration
- Collision detection and response
  - Point-object and object-object detection
  - Only point-object response

ODE: Path Through a Vector Field

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



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

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

- $f = \mathrm{d}/\mathrm{d}t\ \mathbf{X}$ is a vector that sits at each point in phase space, pointing the direction.

Implicit integration

- So far, we have seen **explicit** Euler
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t})$

- 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

Implicit integration

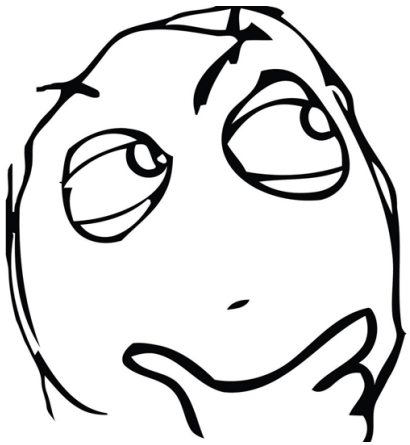
- So far, we have seen **explicit** Euler
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t})$

- Implicit Euler uses the derivative at the destination!
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t+h})$
  - It's implicit because we don't have $\mathbf{X}'(t+h)$, it depends on where we go

Implicit integration

- So far, we have seen **explicit** Euler
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t})$

- Implicit Euler uses the derivative at the destination!
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t+h})$
  - It's implicit because we don't have $\mathbf{X}'(t+h)$, it depends on where we go



Implicit integration

- So far, we have seen **explicit** Euler
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t})$

- Implicit Euler uses the derivative at the destination!
  - $\mathbf{X}(t+h) = \mathbf{X}(t) + h\,\mathbf{X}'(\mathbf{t+h})$
  - It's implicit because we don't have $\mathbf{X}'(t+h)$, it depends on where we go (HUH?)
  - Two situations
    - $\mathbf{X}'$ is known analytically and everything is closed form (*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: $x(t+h) = x(\mathrm{t}) + h\, x'(t+h)$

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: $x(t+h) = x(t) + h\, x'(t+h)$

$$x(t+h) = x(t) - hk\, x(t+h)$$

$$= x(t) / (1+hk)$$

  - It's a hyperbola!

Simple Closed Form Case

Implicit Euler is unconditionally stable!

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

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

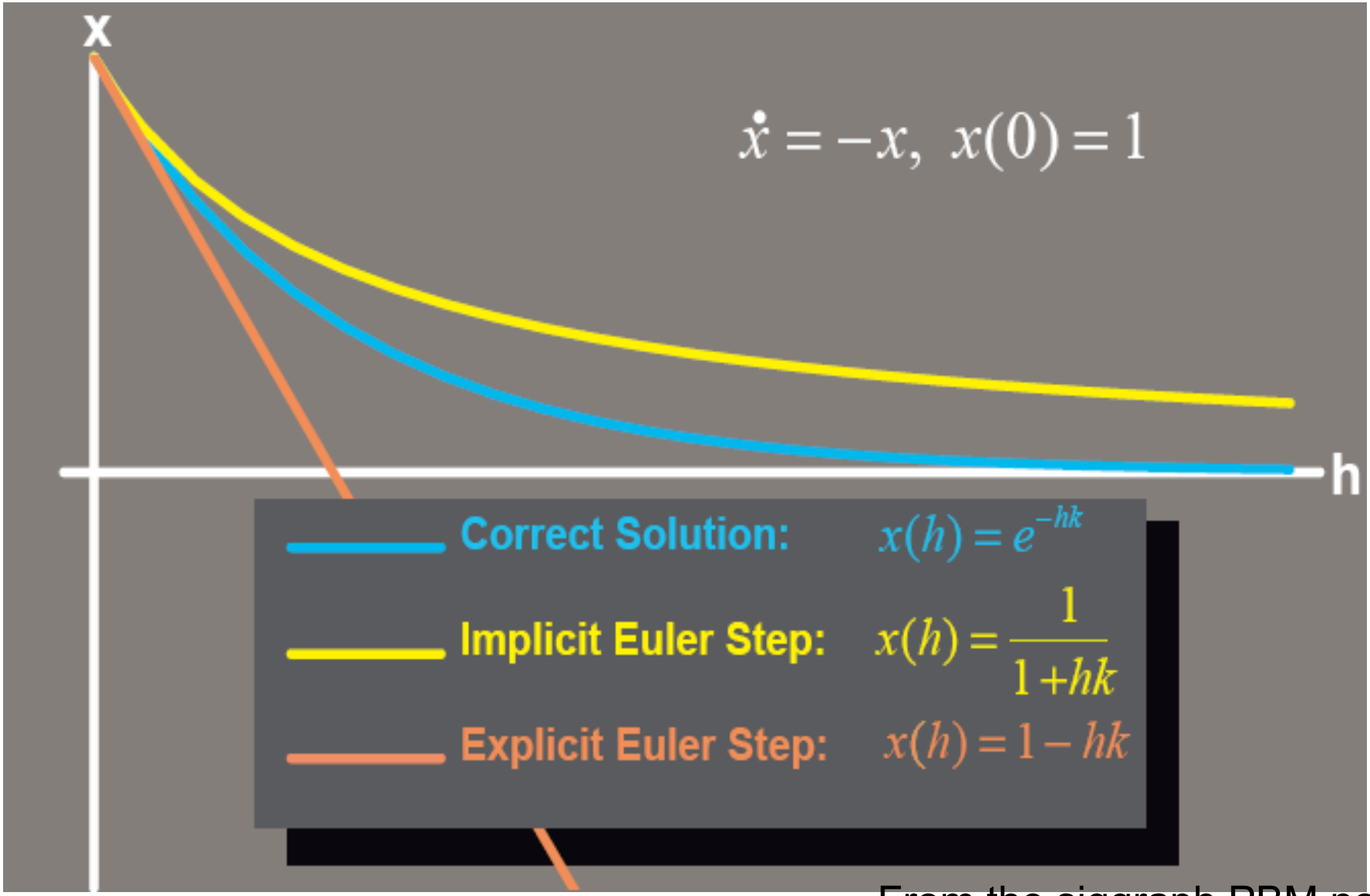
$$x(t+h) = x(t) - h\, k\, x(t+h)$$

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

  - It's a hyperbola!

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

Implicit vs. Explicit

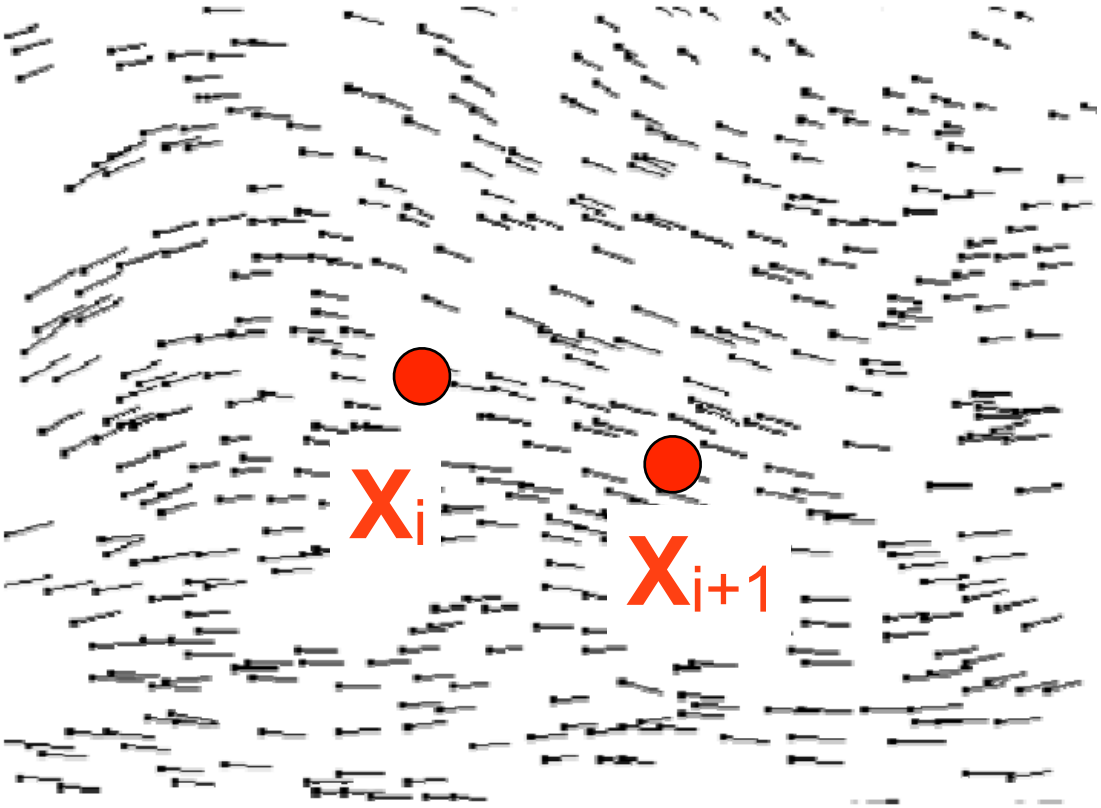


From the siggraph PBM notes

Implicit Euler, Visually

$$\mathbf{X}_{i+1} = \mathbf{X}_i + h\, f(\, \mathbf{X}_{i+1},\, \mathbf{t+h}\,)$$

$$\mathbf{X}_{i+1} - h\, f(\, \mathbf{X}_{i+1},\, \mathbf{t+h}\,) = \mathbf{X}_i$$

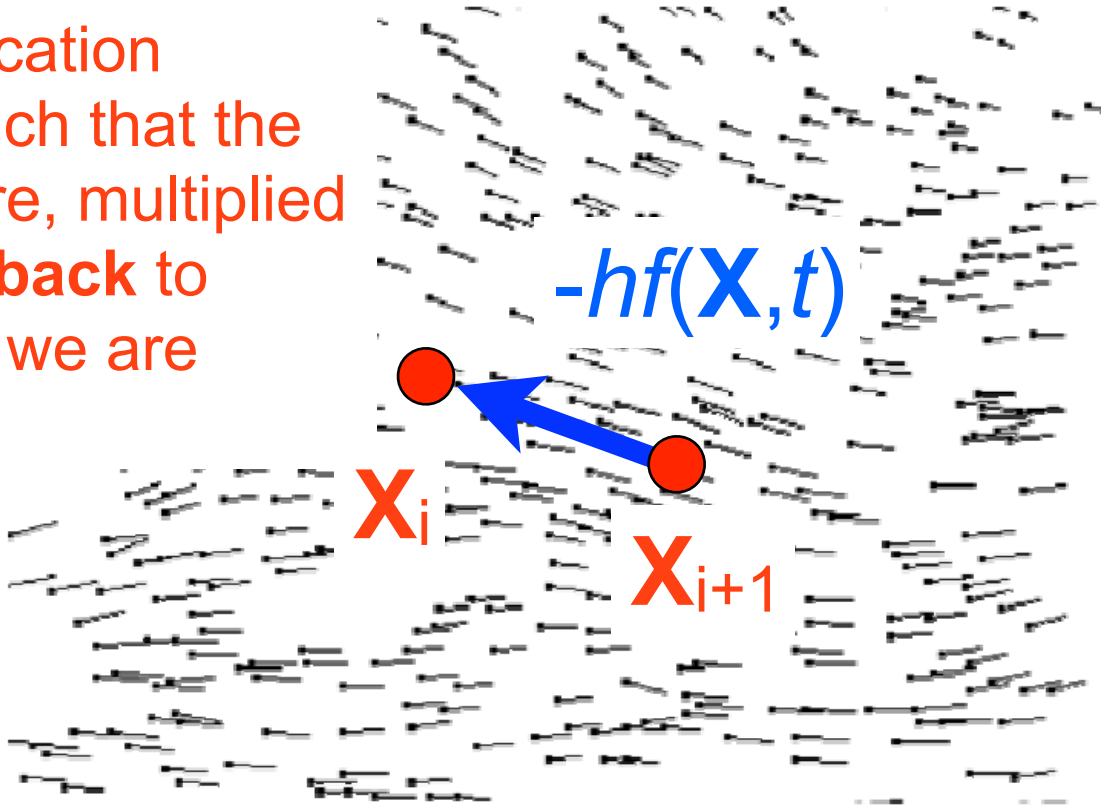


Implicit Euler, Visually

$$\mathbf{X}_{i+1} = \mathbf{X}_i + h\, f(\, \mathbf{X}_{i+1},\, \mathbf{t+h}\,)$$

$$\mathbf{X}_{i+1} - h\, f(\, \mathbf{X}_{i+1},\, \mathbf{t+h}\,) = \mathbf{X}_i$$

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



Implicit Euler and Large Systems

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

- Implicit Euler with N-D phase space:
  $\mathbf{X}_{i+1} = \mathbf{X}_i + h\, f(\mathbf{X}_{i+1})$

Implicit Euler and Large Systems

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

- Implicit Euler with N-D phase space:
  $\mathbf{X}_{i+1} = \mathbf{X}_i + h\, f(\mathbf{X}_{i+1})$

- Non-linear equation,
  unknown $\mathbf{X}_{i+1}$ on both the LHS and the RHS
  - **Do you know methods for solving such equations?**

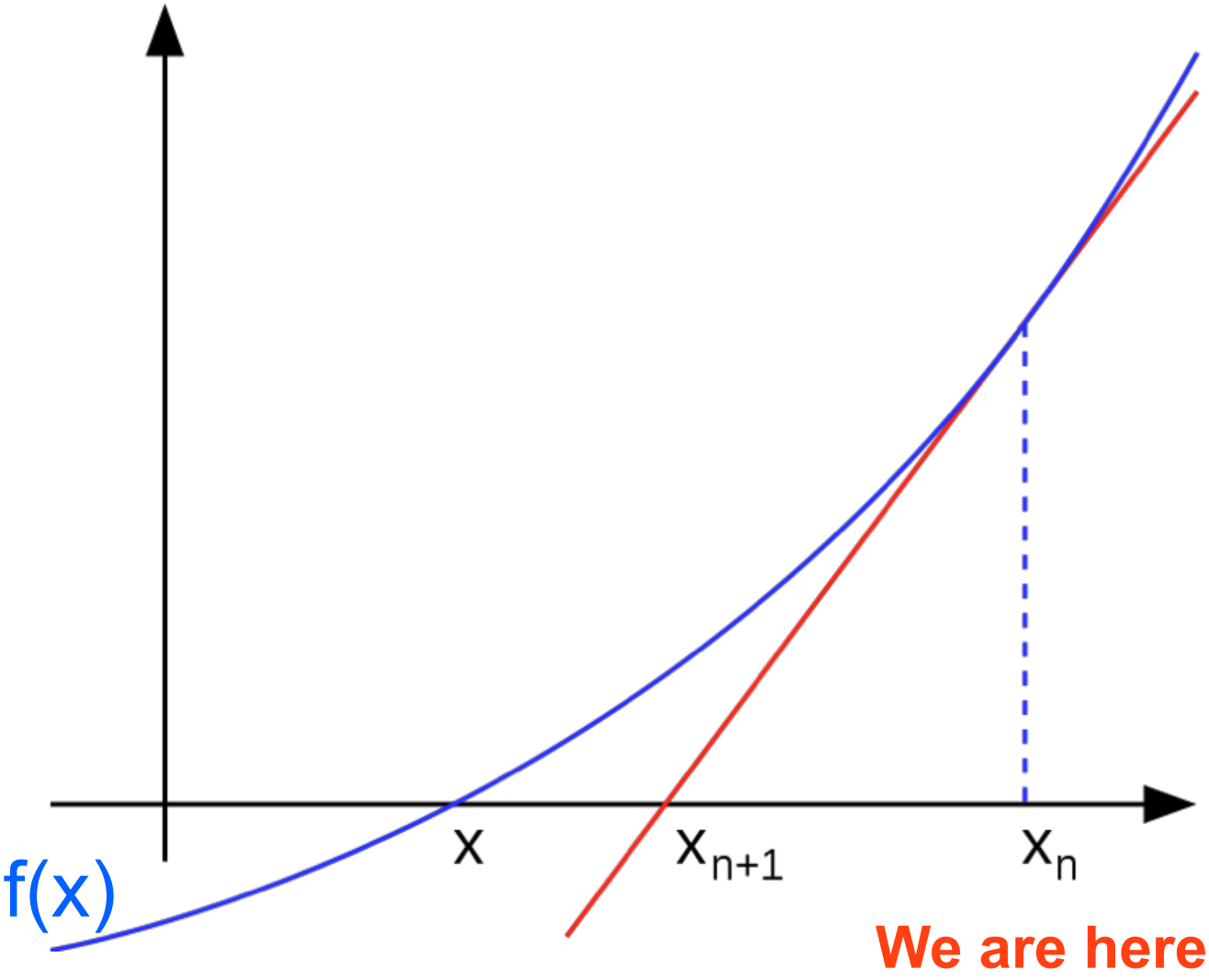
Newton's Method (1D)

- Iterative method for solving non-linear equations

$$F(x) = 0$$

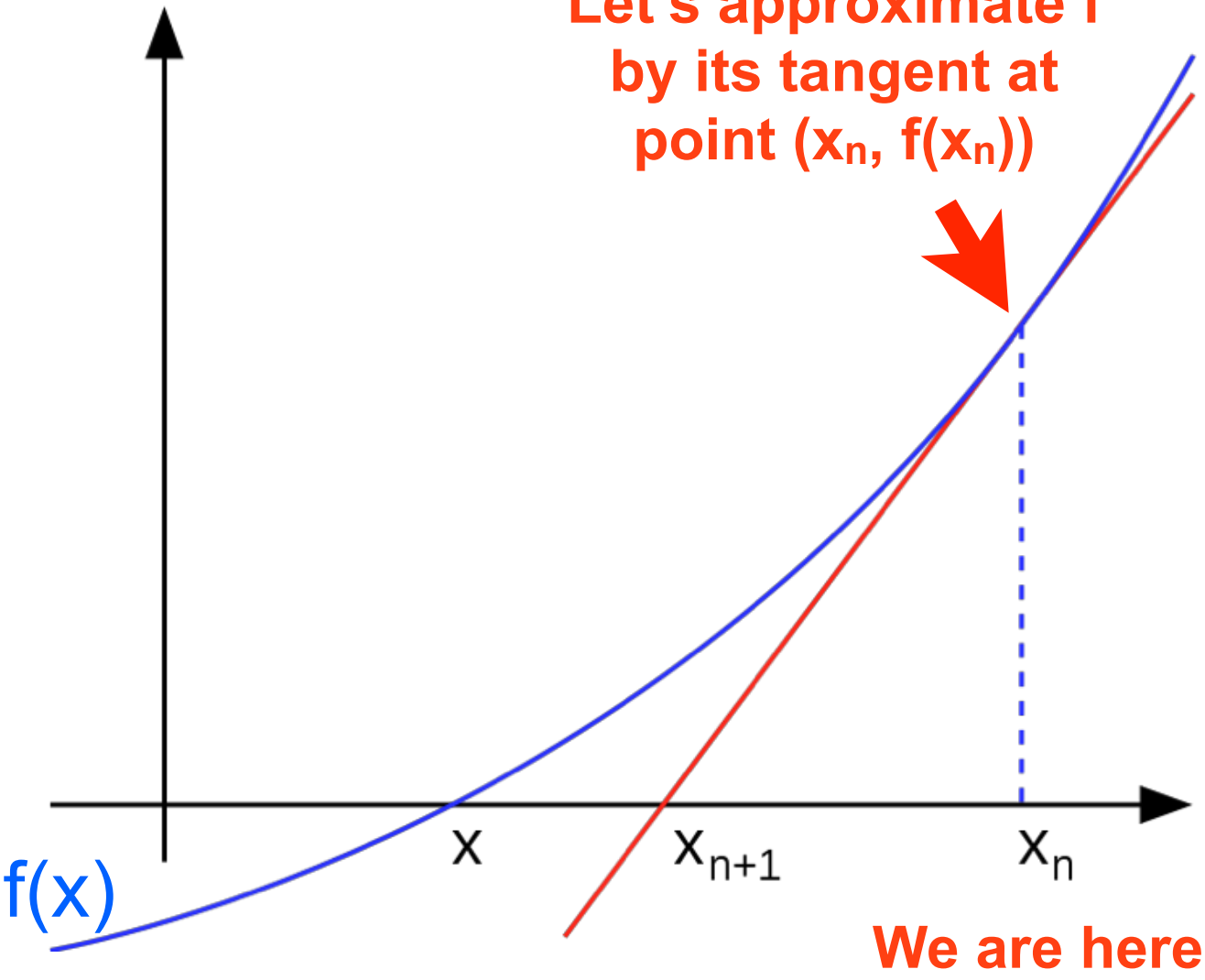
- Start from initial guess x_0, then iterate

Newton, Visually



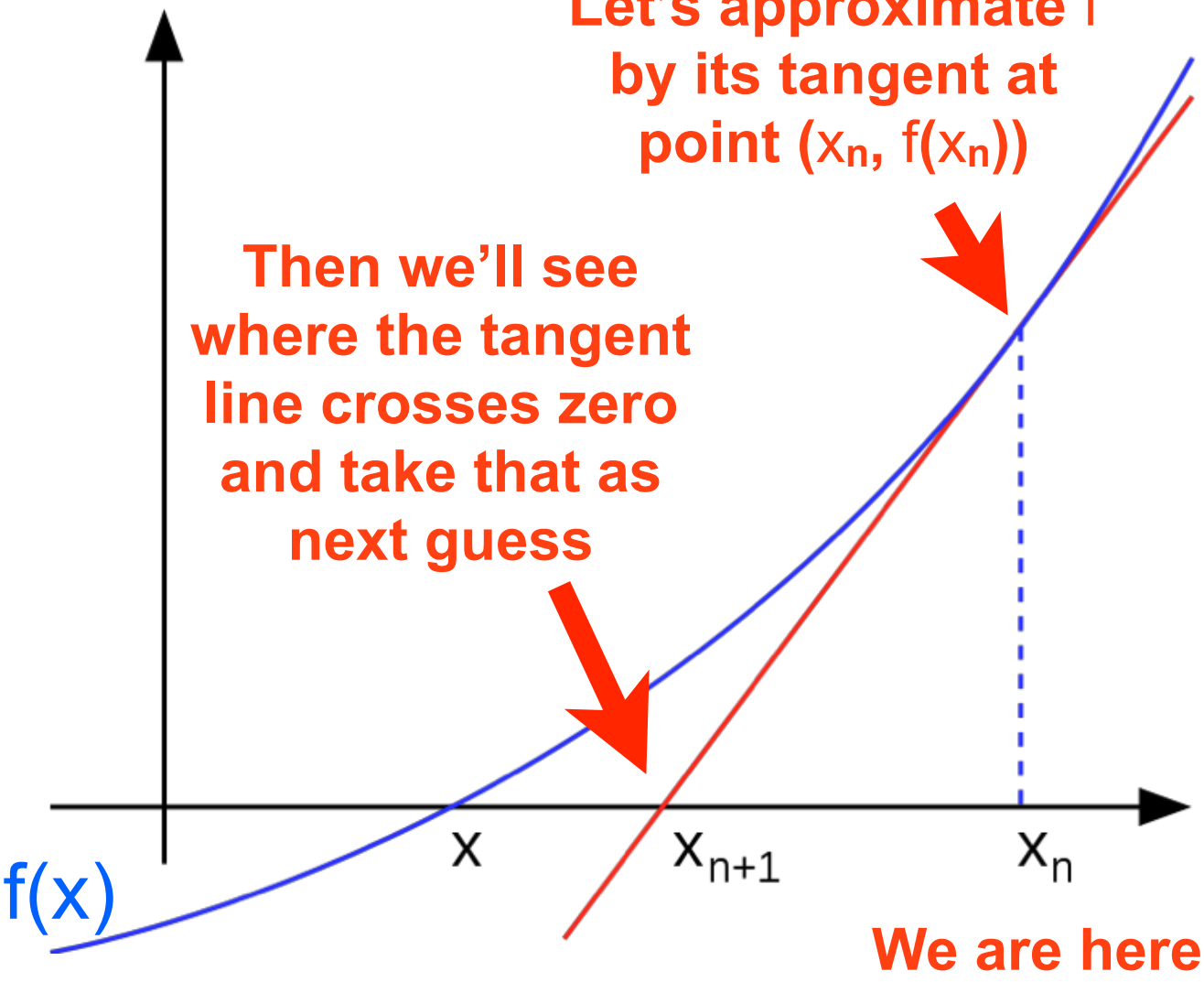
Wikipedia users Olegalexandrov, Pbroks13

Newton, Visually



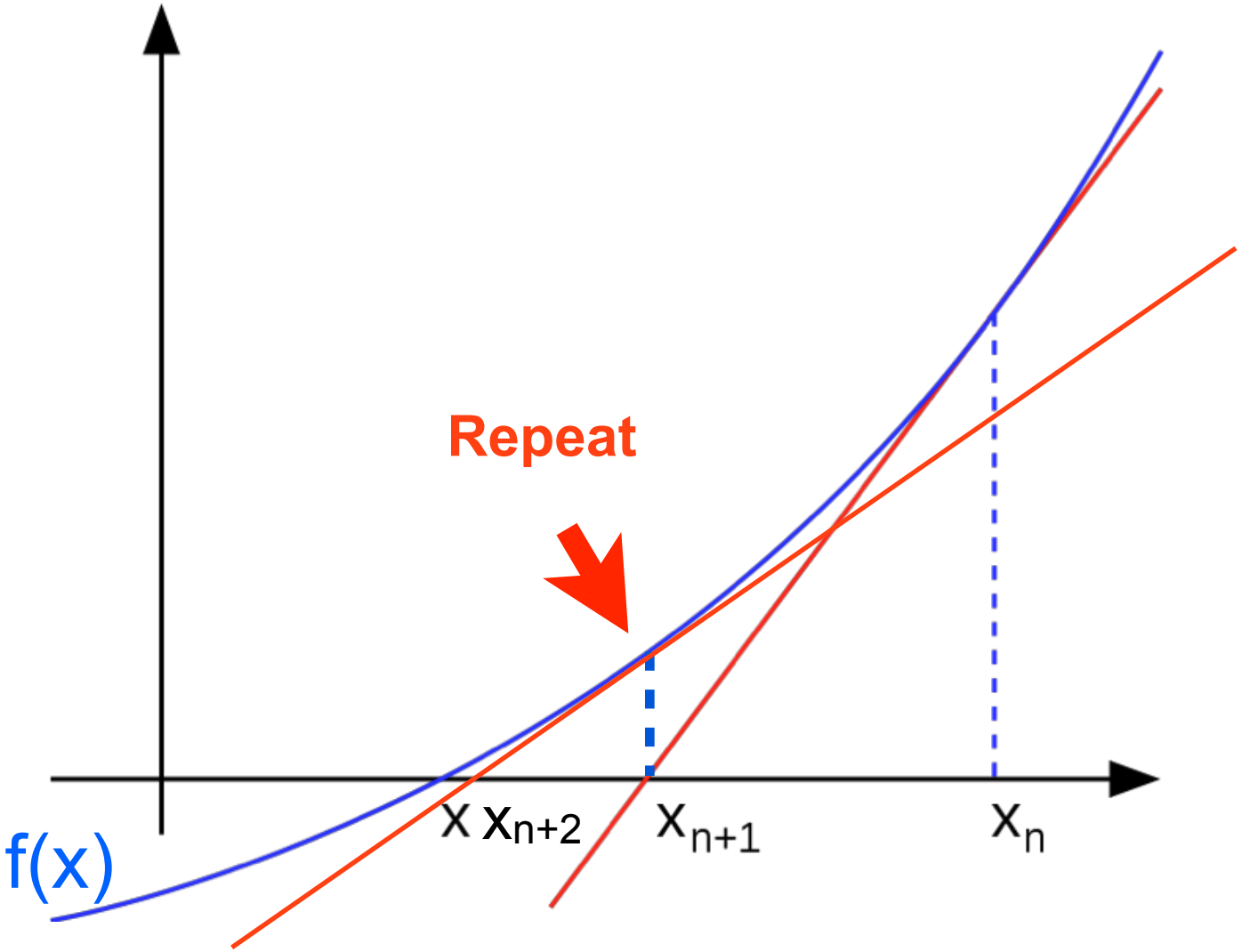
Wikipedia users Olegalexandrov, Pbroks13

Newton, Visually



Wikipedia users Olegalexandrov, Pbroks13

Newton, Visually



Wikipedia users Olegalexandrov, Pbroks13

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

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

$$\Leftrightarrow \quad f'(x_i)(x_{i+1} - x_i) = -f(x_i)$$

one step

Newton's Method – N Dimensions

- 1D: $$f'(x_i)(x_{i+1} - x_i) = -f(x_i)$$

- Now locations $\mathbf{X}_i$, $\mathbf{X}_{i+1}$ and F live in N-D phase space
- N-D Newton step is just like 1D:

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

NxN Jacobian matrix replaces f'

unknown N-D step from current to next guess

Newton's Method – N Dimensions

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

Questions?

Implicit Euler – N Dimensions

- Implicit Euler with N-D phase space:
  $\mathbf{X}_{i+1} = \mathbf{X}_i + h\, f(\mathbf{X}_{i+1})$

- Let's rewrite this as $F(\boldsymbol{Y}) = 0,$ with

$$F(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{X}_i - hf(\boldsymbol{Y})$$

Implicit Euler – N Dimensions

- Implicit Euler with N-D phase space:
  $\mathbf{X}_{i+1} = \mathbf{X}_i + h\, f(\mathbf{X}_{i+1})$

- Let's rewrite this as $F(\boldsymbol{Y}) = 0,$ with

$$F(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{X}_i - hf(\boldsymbol{Y})$$

- Then the $\mathbf{Y}$ that solves $F(\mathbf{Y})=0$ is $\mathbf{X}_{i+1}$

Implicit Euler – N Dimensions

$$F(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{X}_i - hf(\boldsymbol{Y})$$

Y is variable **X_i** is fixed

- Then iterate until F(***Y***)=0:
  - Initial guess $\boldsymbol{Y}_0 = \boldsymbol{X}_i$ (or result of explicit method)
  - For each step, solve $J_F(\boldsymbol{Y}_i)\Delta\boldsymbol{Y} = -F(\boldsymbol{Y}_i)$
  - Then set $\boldsymbol{Y}_{i+1} = \boldsymbol{Y}_i + \Delta\boldsymbol{Y}$

What is the Jacobian?

$$F(\boldsymbol{Y}) = \boldsymbol{Y} - \boldsymbol{X}_i - hf(\boldsymbol{Y})$$

- Simple partial differentiation...

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

- Where $J_f(\boldsymbol{Y}) = \left[\frac{\partial f}{\partial \boldsymbol{Y}}\right]$

The Jacobian of the derivative (force) function f

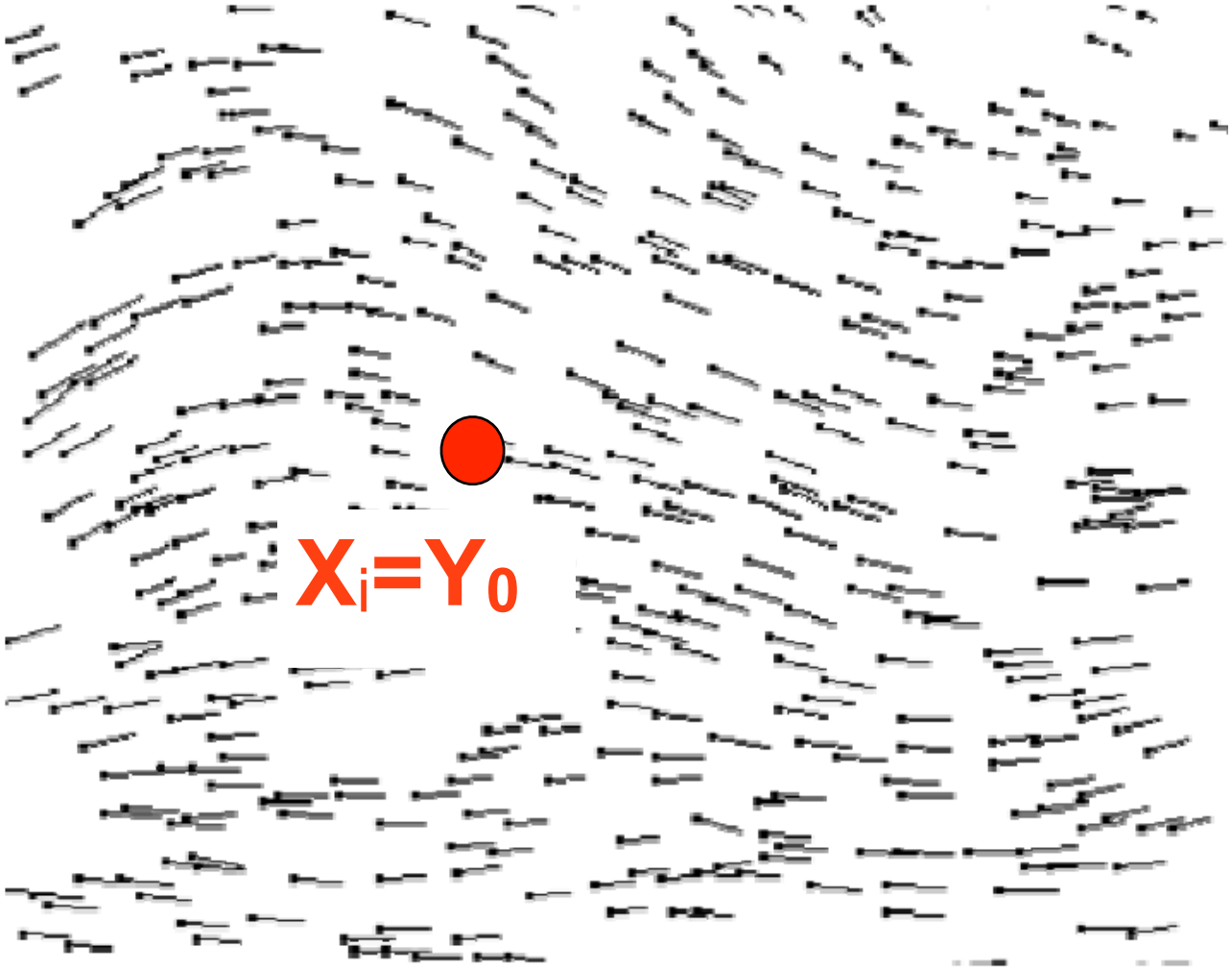
Putting It All Together

- Iterate until convergence
  - Initial guess $\boldsymbol{Y}_0 = \boldsymbol{X}_i$ (or result of explicit method)
  - For each step, solve

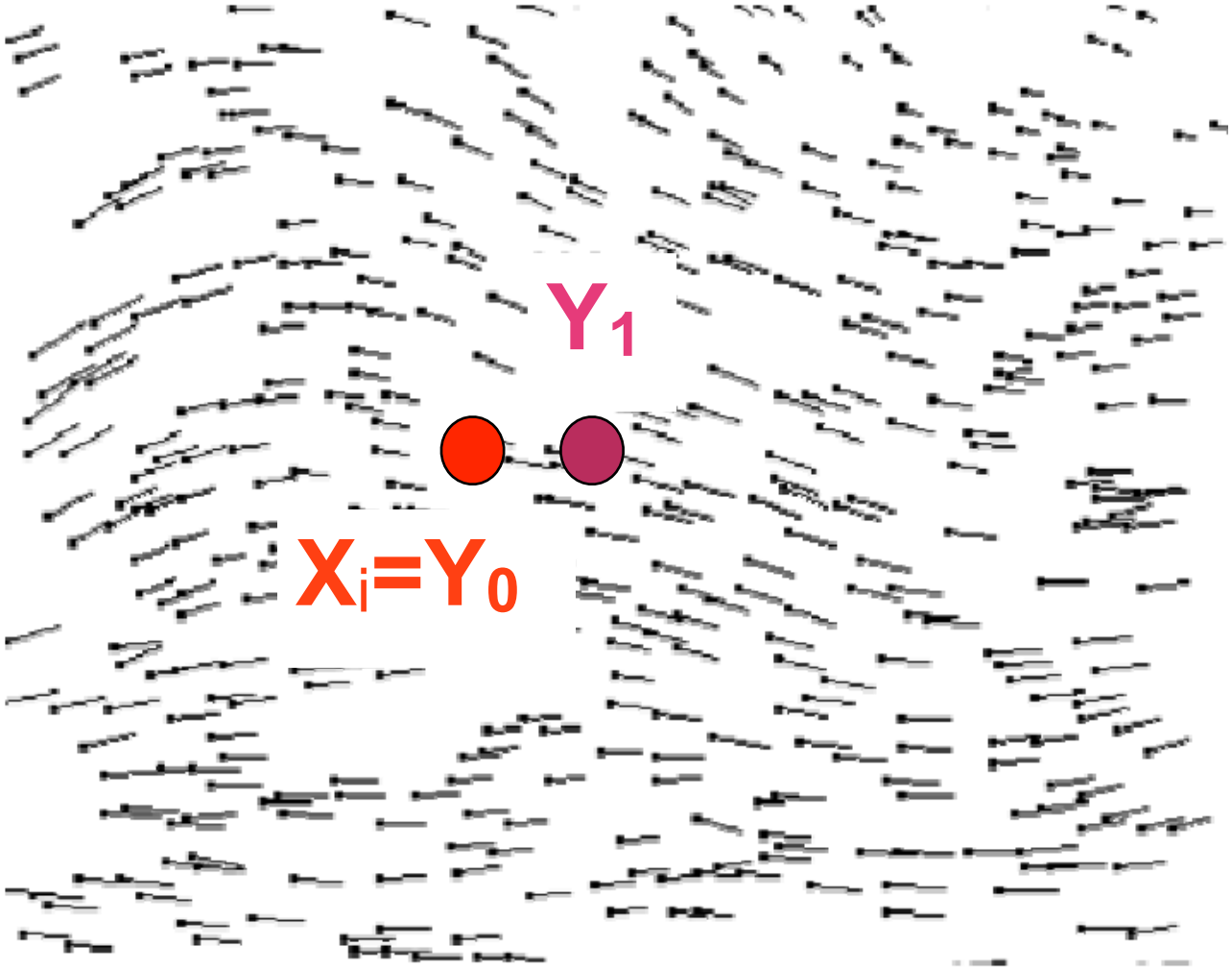
$$\Big(\boldsymbol{I} - J_f(\boldsymbol{Y}_i)\Big)\Delta\boldsymbol{Y} = -F(\boldsymbol{Y}_i)$$

  - Then set $\boldsymbol{Y}_{i+1} = \boldsymbol{Y}_i + \Delta\boldsymbol{Y}$

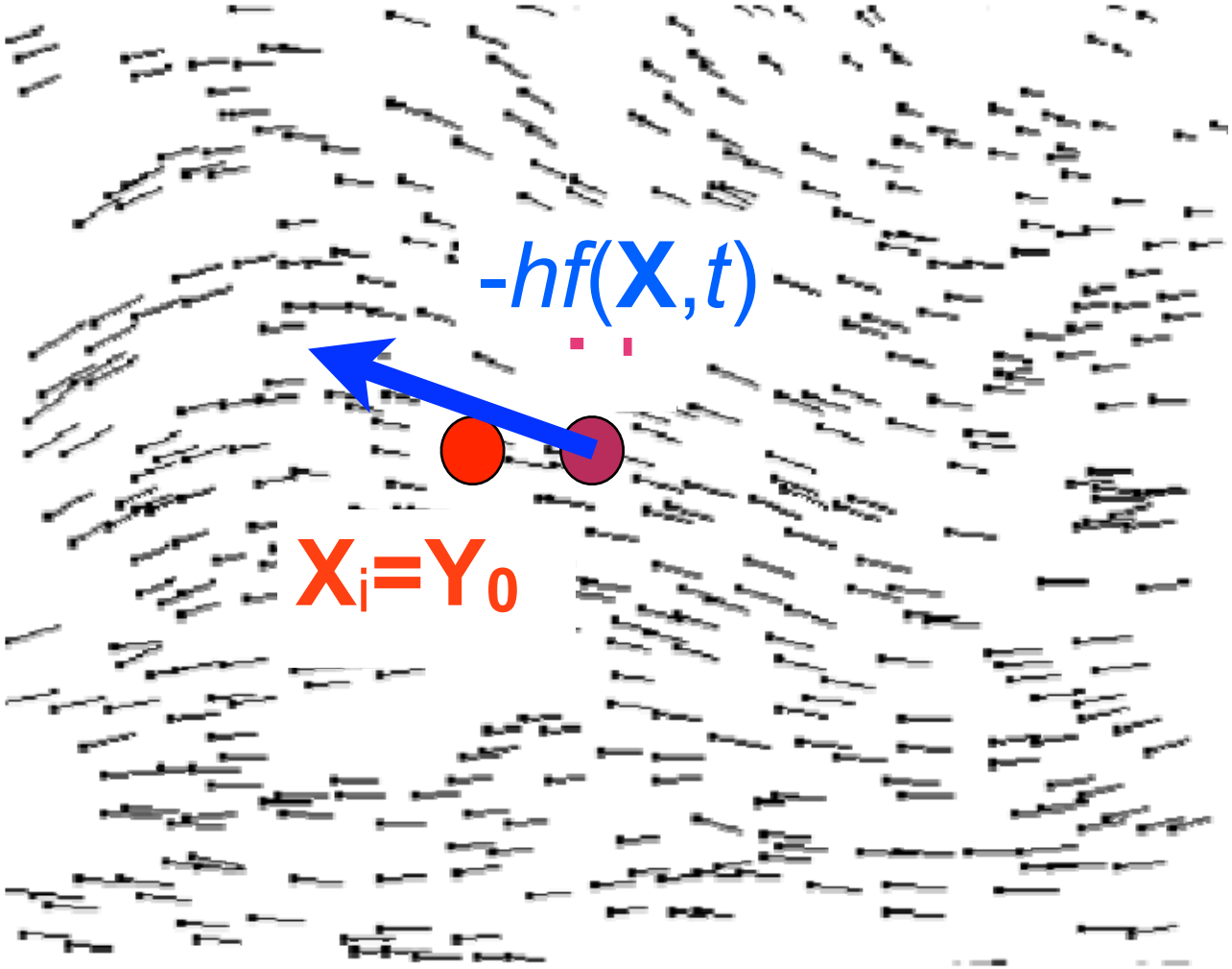
Implicit Euler with Newton, Visually



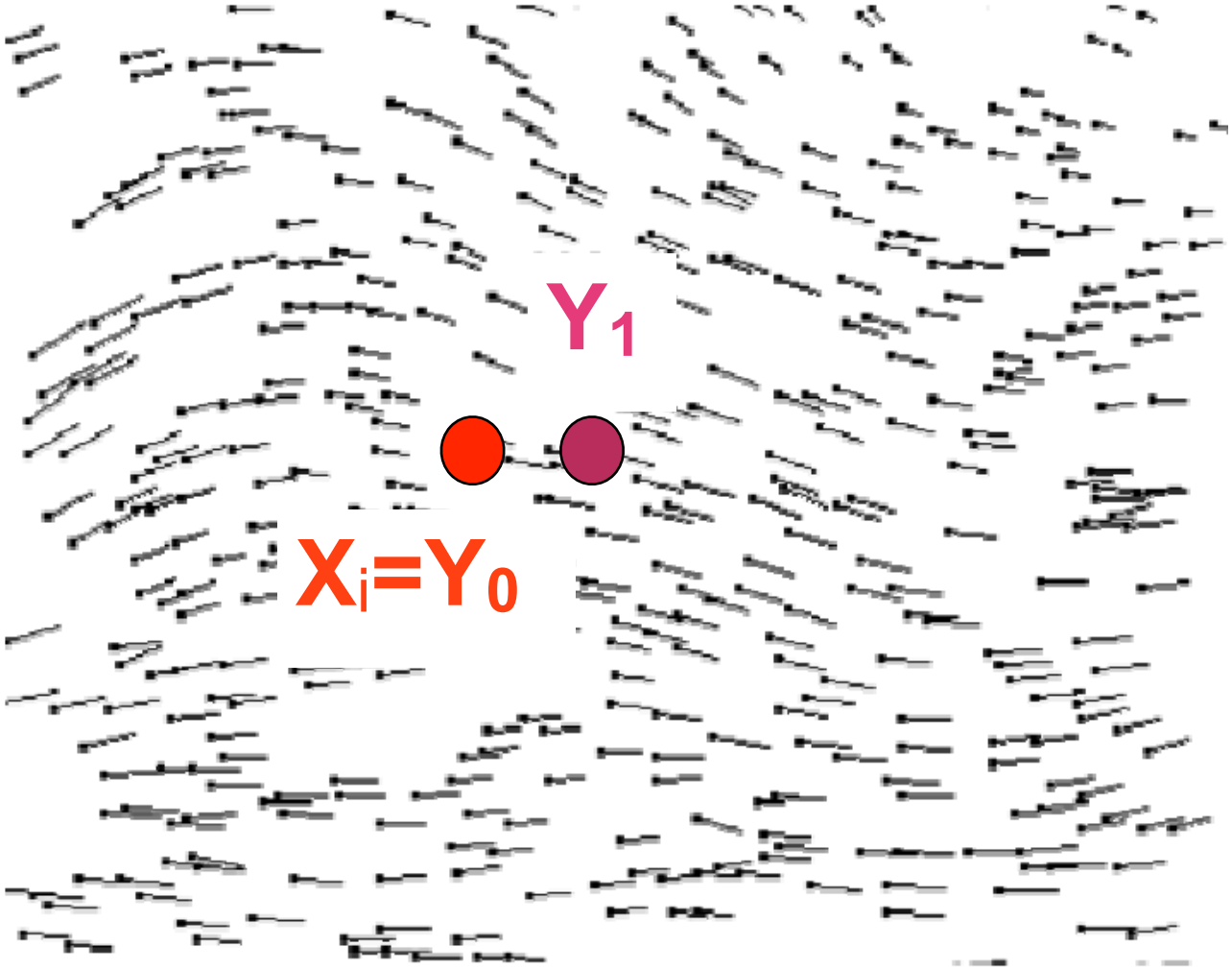
Implicit Euler with Newton, Visually



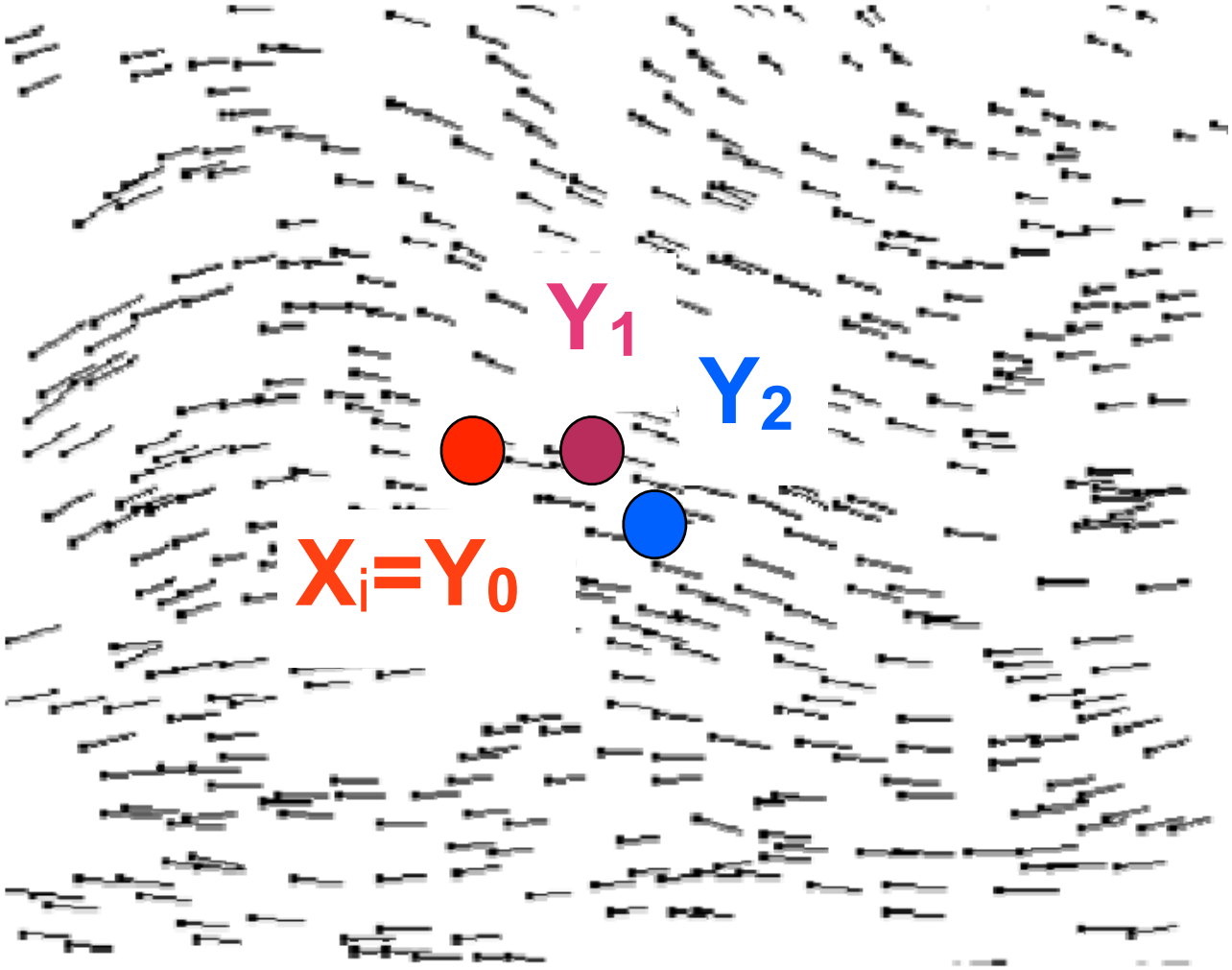
Implicit Euler with Newton, Visually



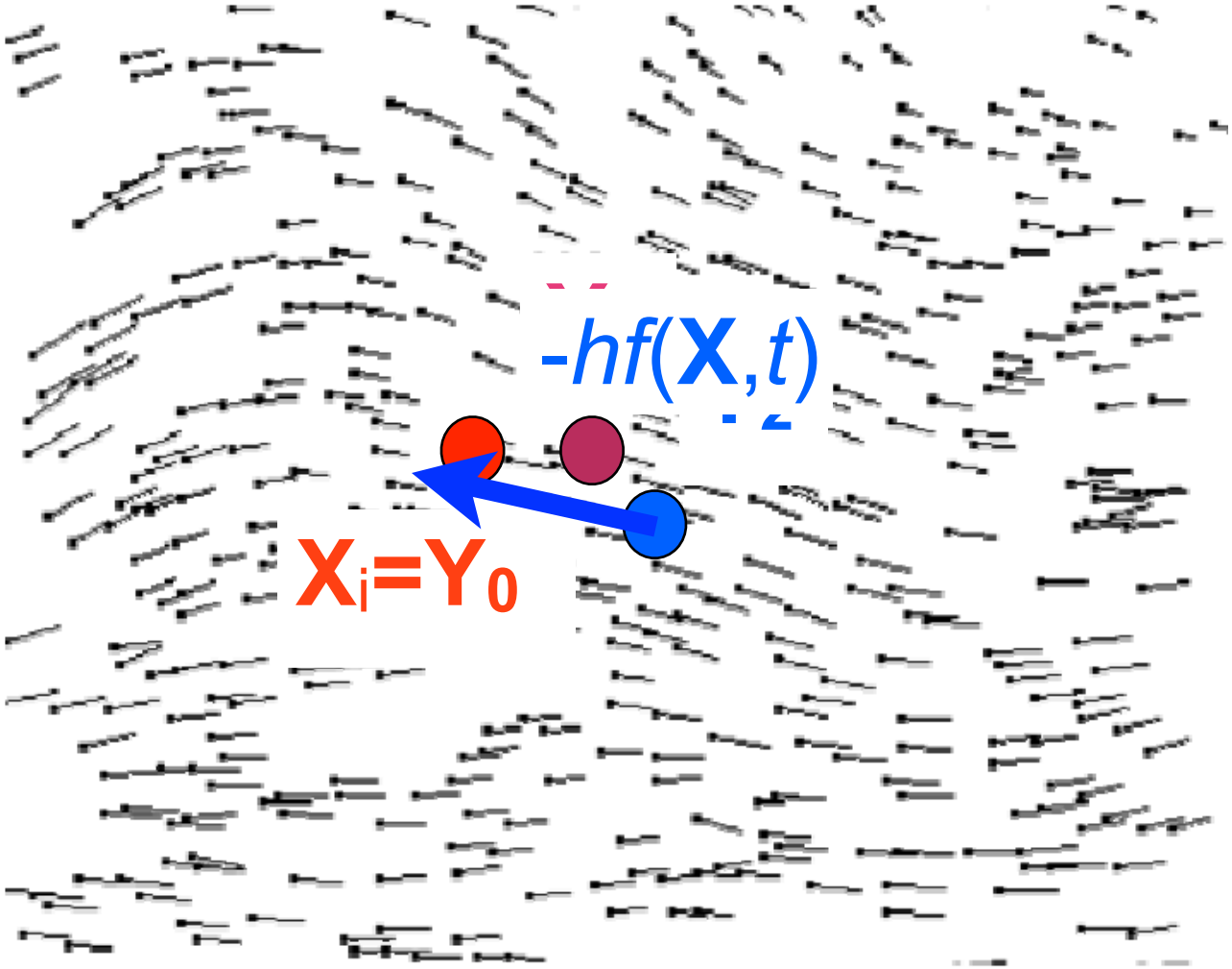
Implicit Euler with Newton, Visually



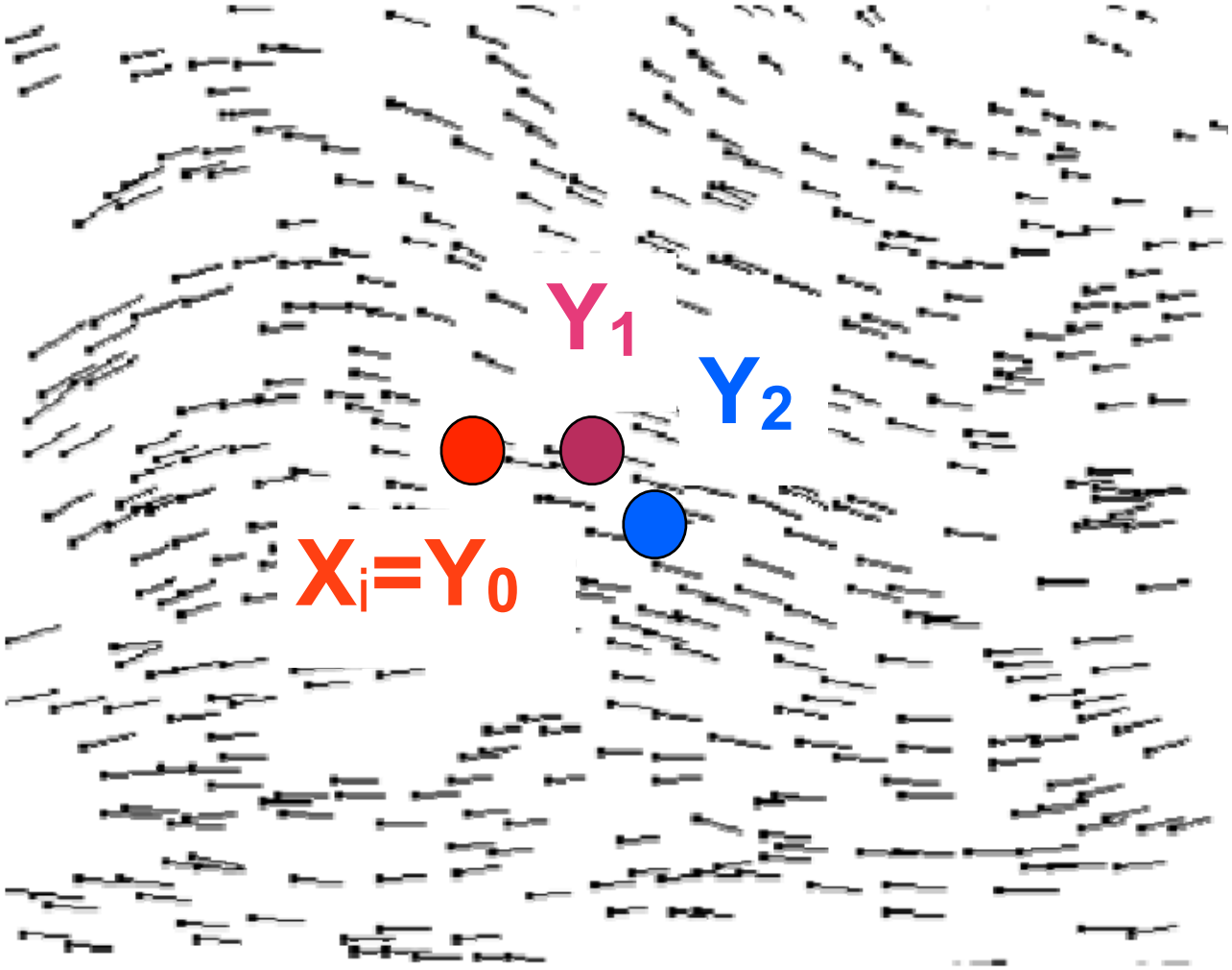
Implicit Euler with Newton, Visually



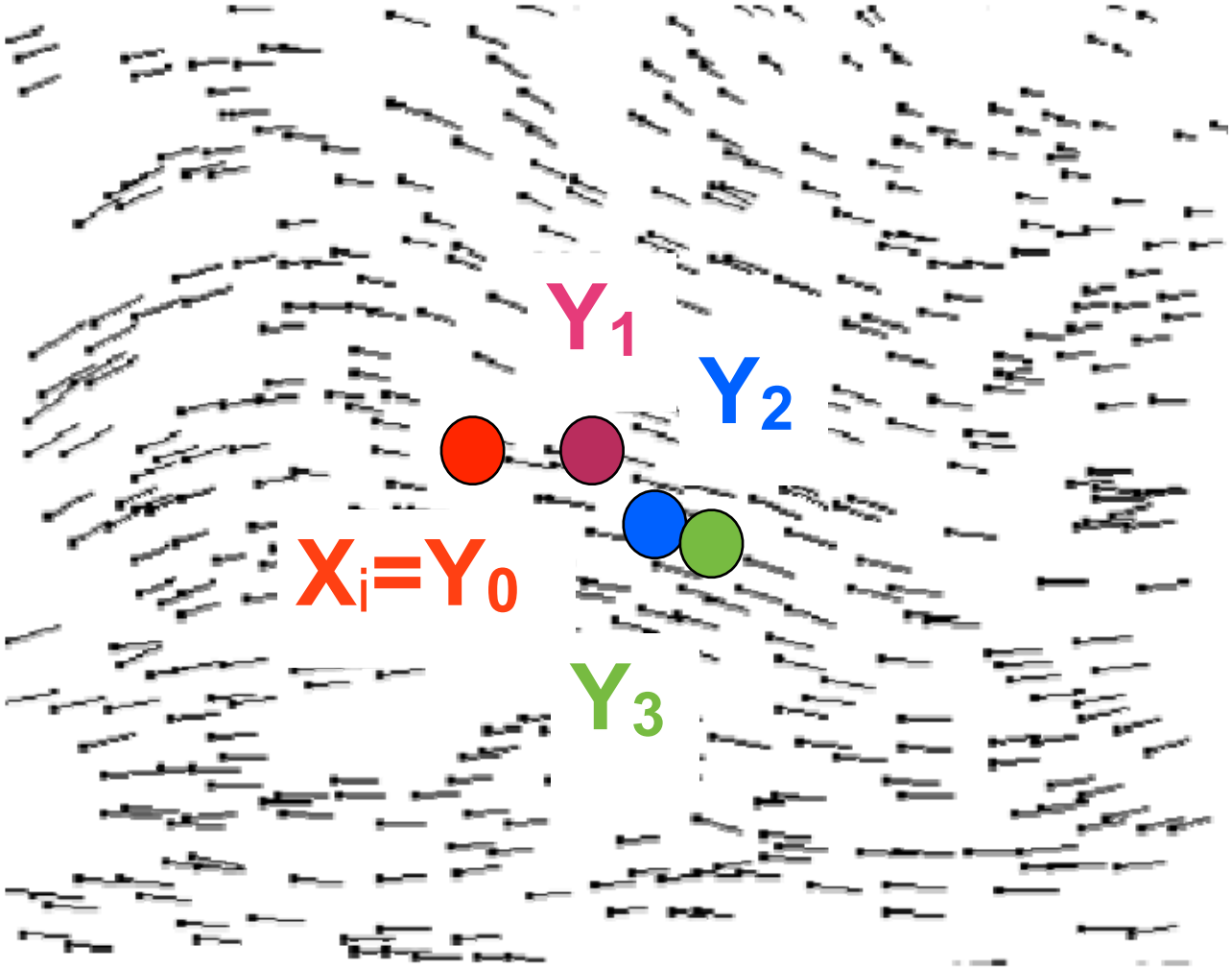
Implicit Euler with Newton, Visually



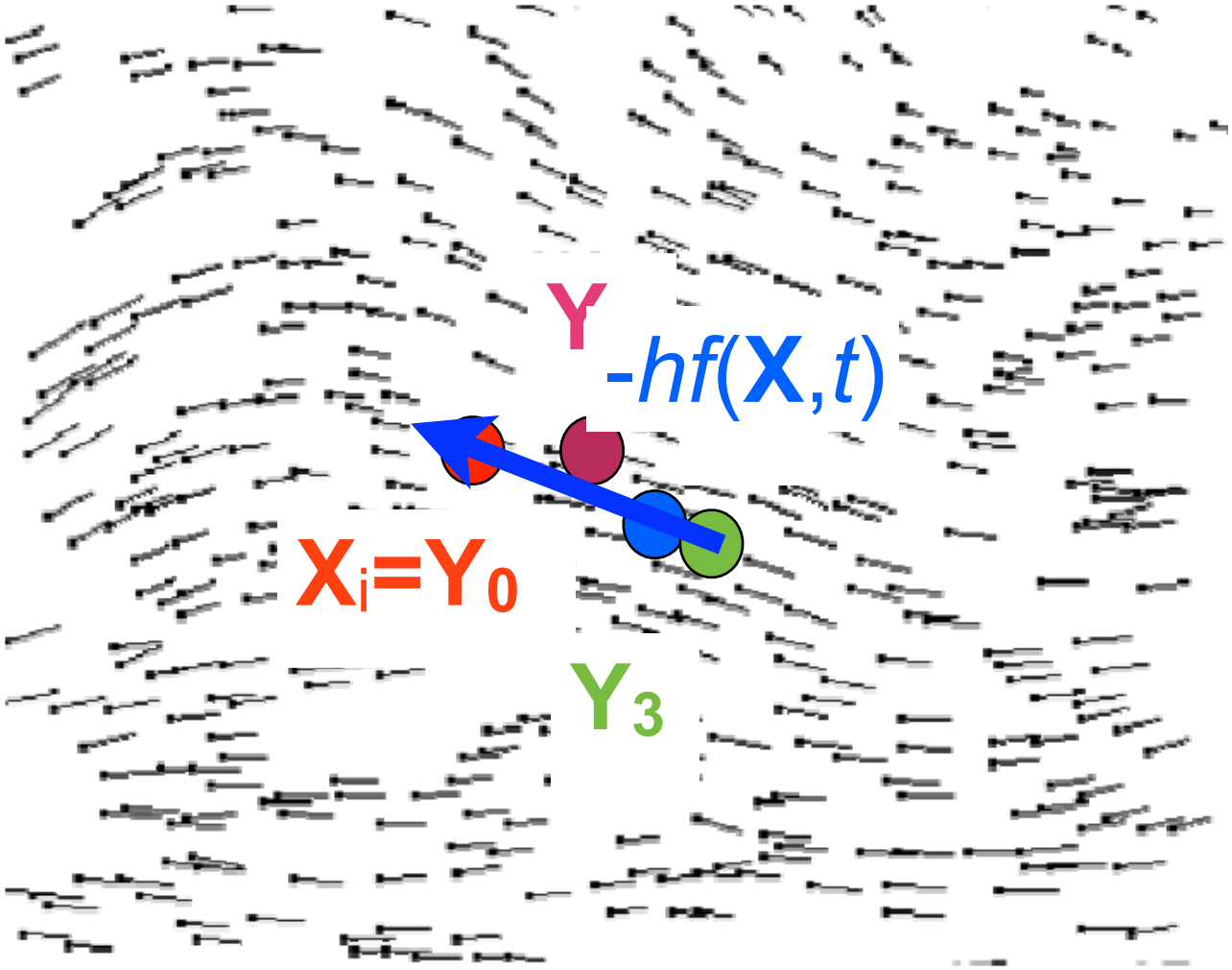
Implicit Euler with Newton, Visually



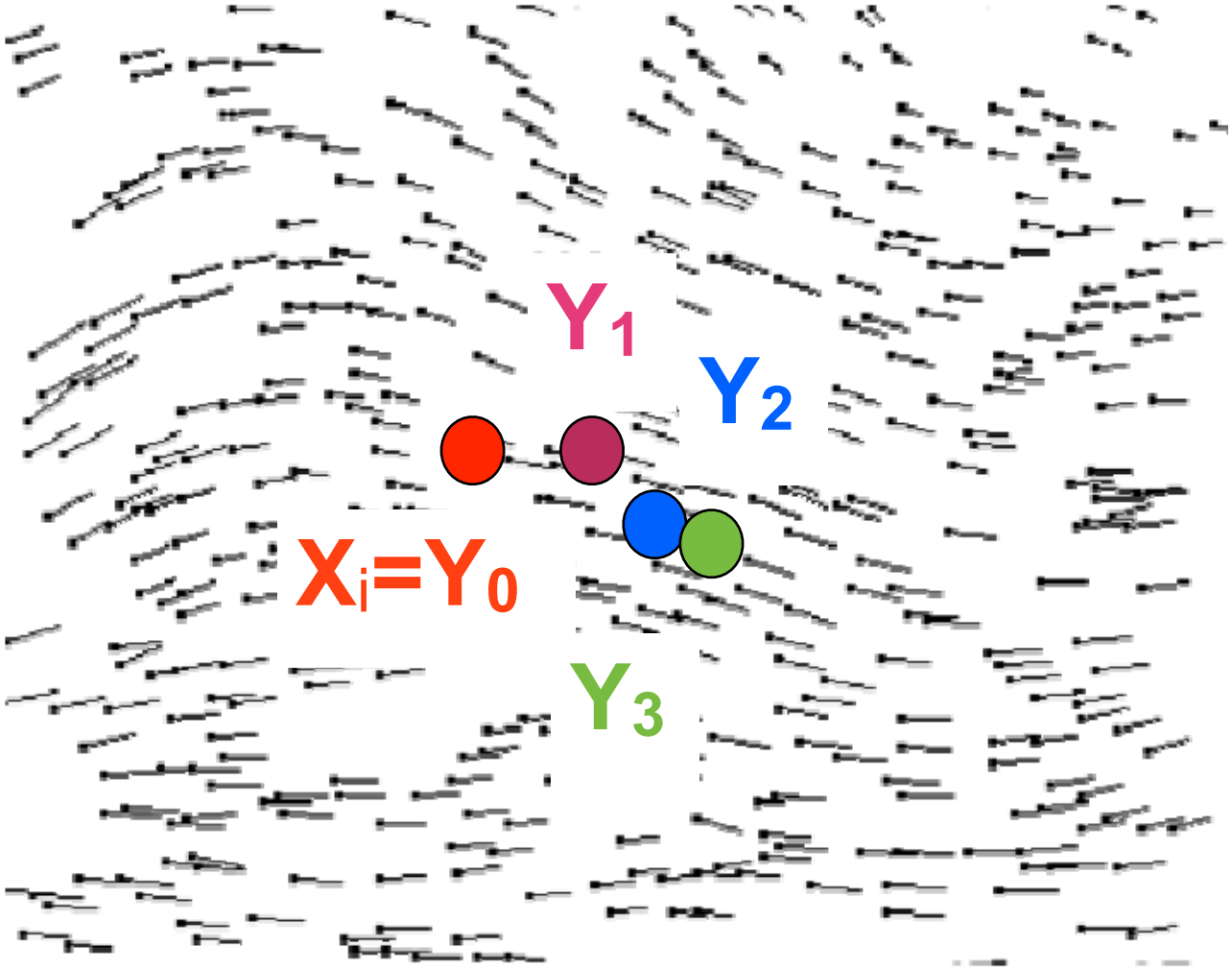
Implicit Euler with Newton, Visually



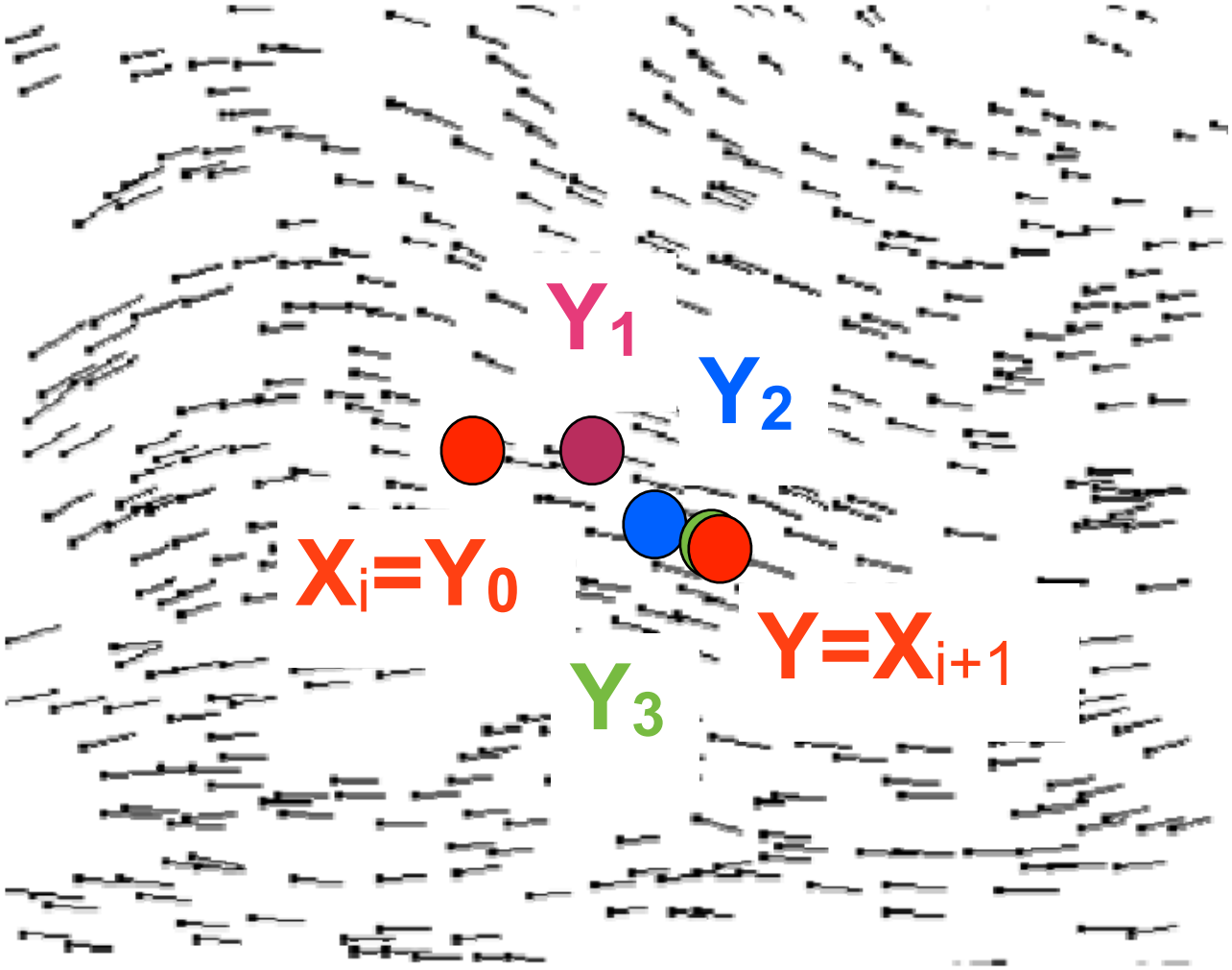
Implicit Euler with Newton, Visually



Implicit Euler with Newton, Visually

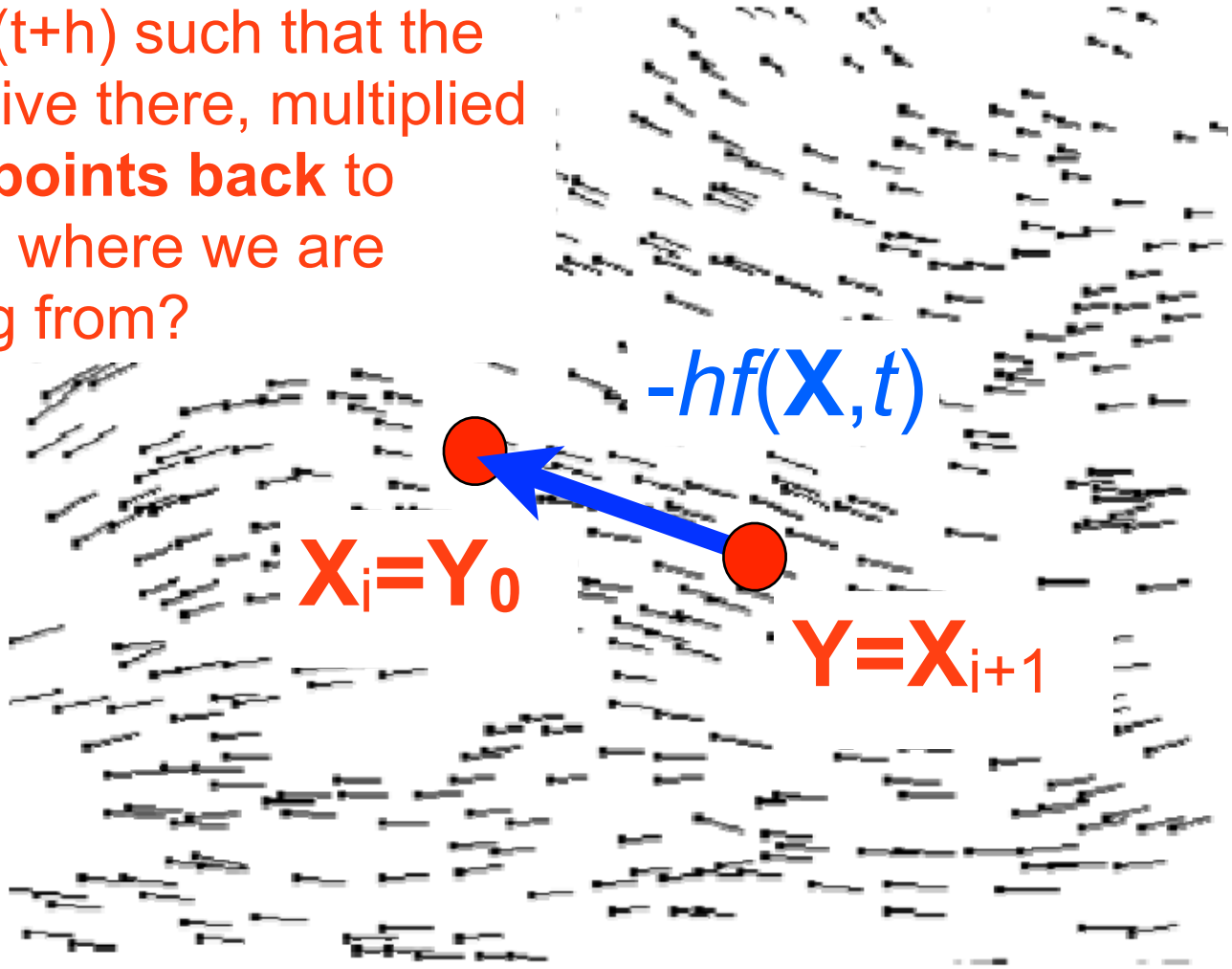


Implicit Euler with Newton, Visually



Implicit Euler with Newton, Visually

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



One-Step Cheat

- Often, the 1st Newton step may suffice
  - People often implement Implicit Euler using only one step.
  - This amounts to solving the system

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

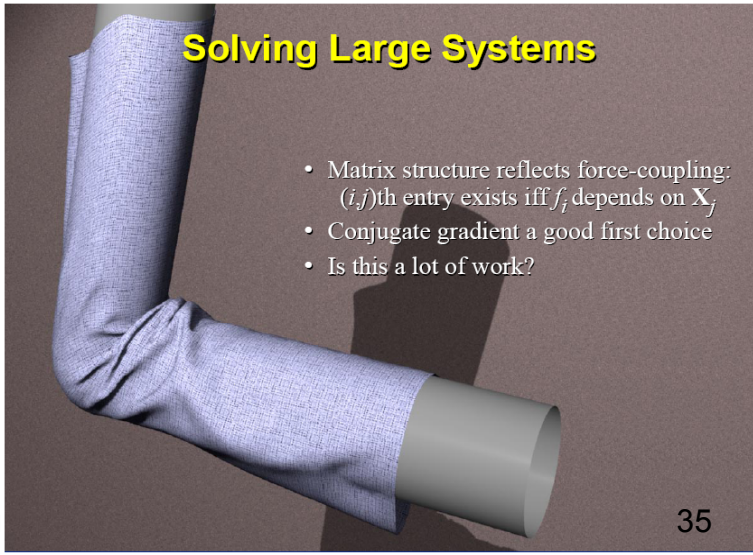
where the Jacobian and f are evaluated at $\mathbf{X}_i$, and we are using $\mathbf{X}_i$ as an initial guess.

Questions?

Good News

- The Jacobian matrix J_f is usually sparse
  - Only few non-zero entries per row

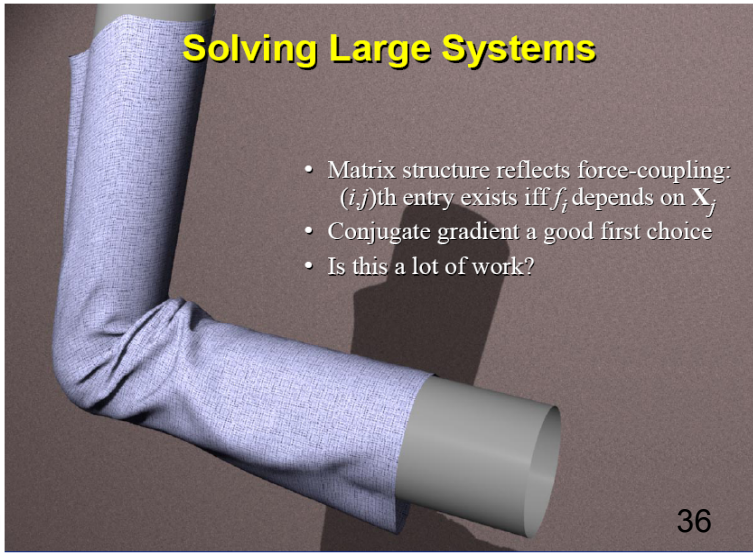
$$\left(\boldsymbol{I} - J_f(\boldsymbol{Y}_i)\right)\Delta\boldsymbol{Y} = -F(\boldsymbol{Y}_i)$$



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.

$$\left(\boldsymbol{I} - J_f(\boldsymbol{Y}_i)\right)\Delta\boldsymbol{Y} = -F(\boldsymbol{Y}_i)$$



Implicit Euler Pros & Cons

Implicit Euler Pros & Cons

- Pro: Stability!

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 don't have to worry about explosions.
    - Recall exp vs. hyperbola

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 don't 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

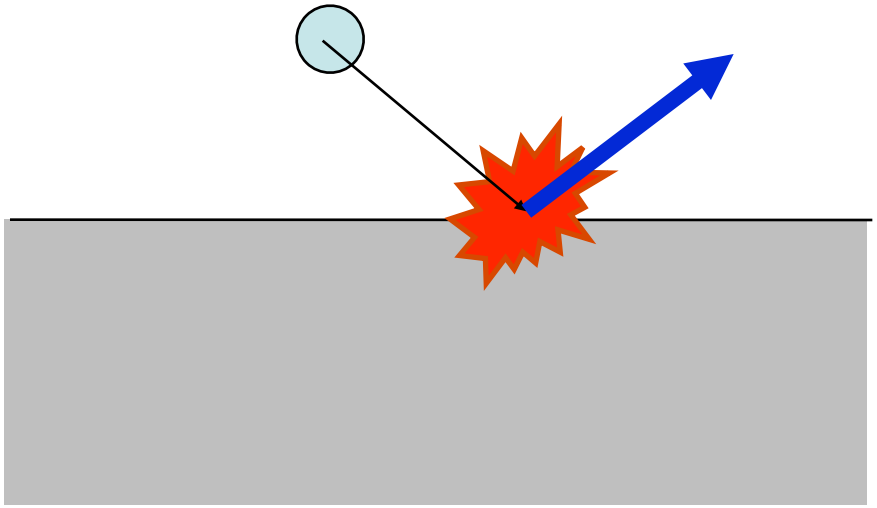
Figure 5 (top row): Dancer with short skirt; frames 110, 136 and 155. Figure 6 (middle row): Dancer with long skirt; frames 185, 215 and 236. Figure 7 (bottom row): Closeups from figures 4 and 6.

Example

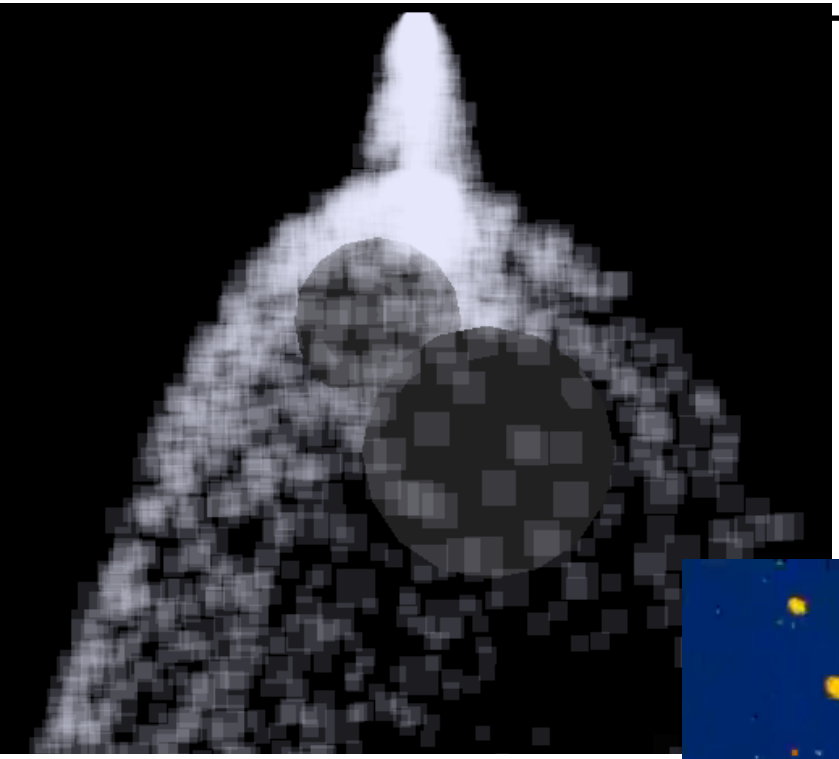
Questions?

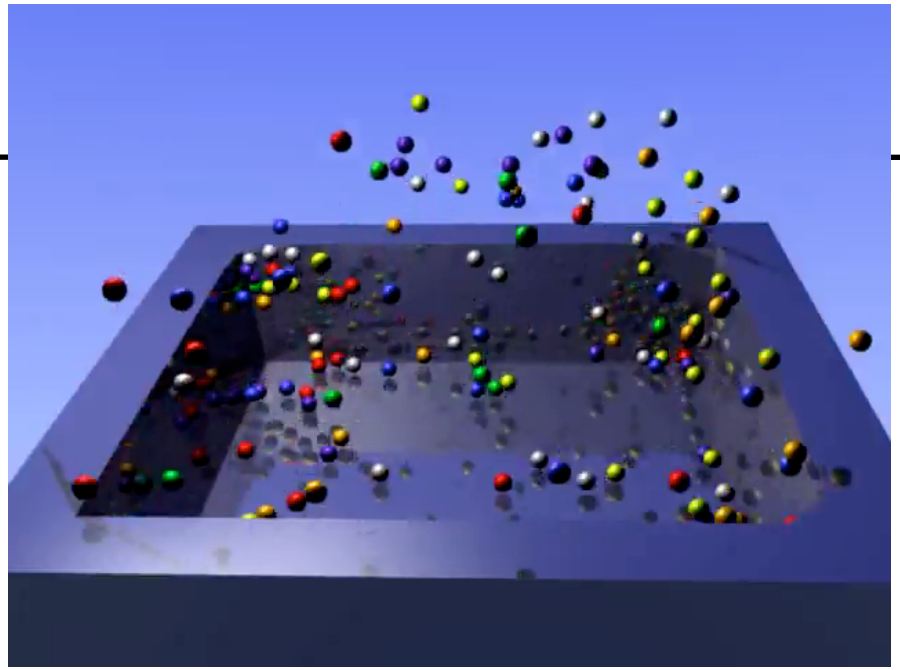
Collisions for Particles

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



Examples





Detecting Collisions

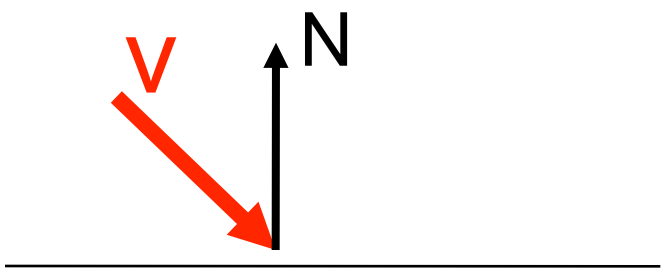
- Easy with implicit equations of surfaces:

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

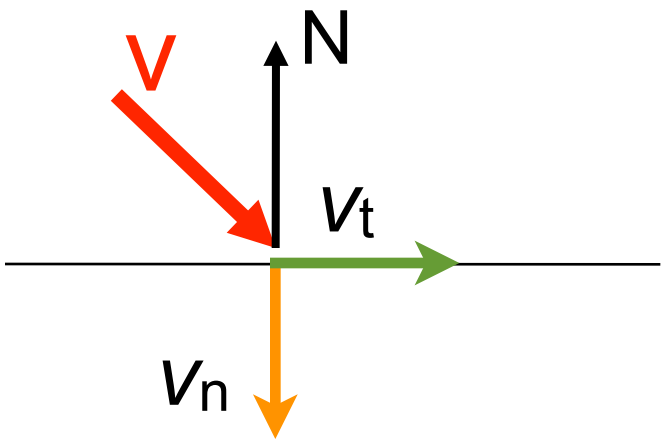
- So just compute H and you know that you're inside if it's negative

- 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

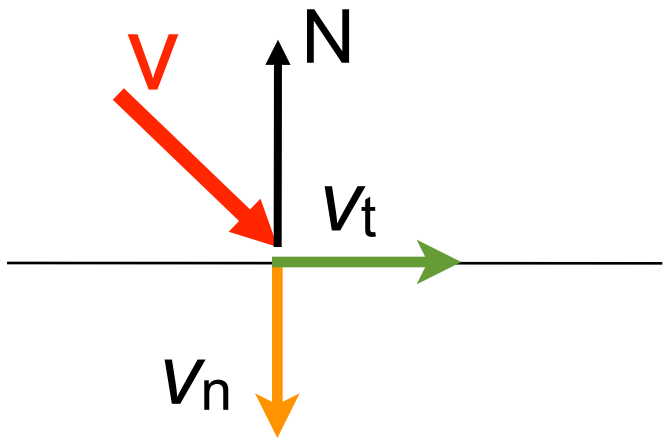
Collision Response for Particles

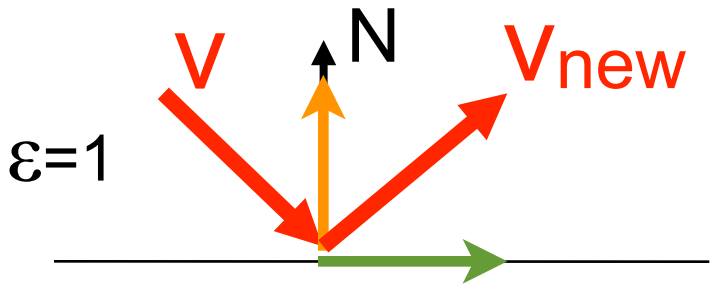
- Tangential velocity v_t *often* unchanged
- Normal velocity v_n reflects:

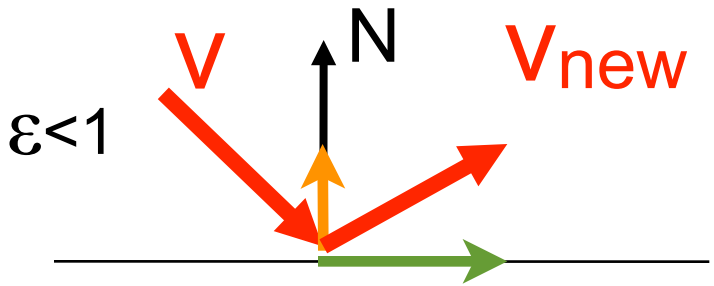
$$v = v_t + v_n$$

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

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

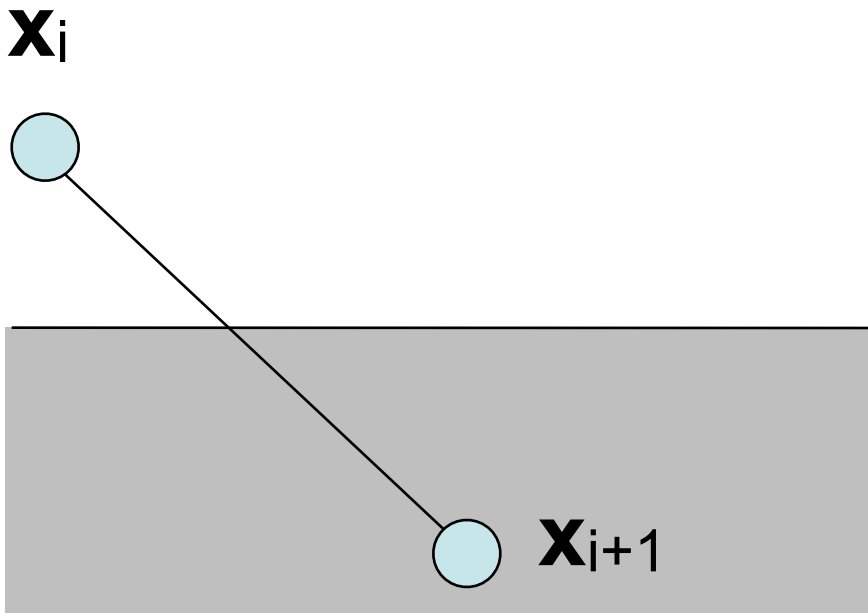






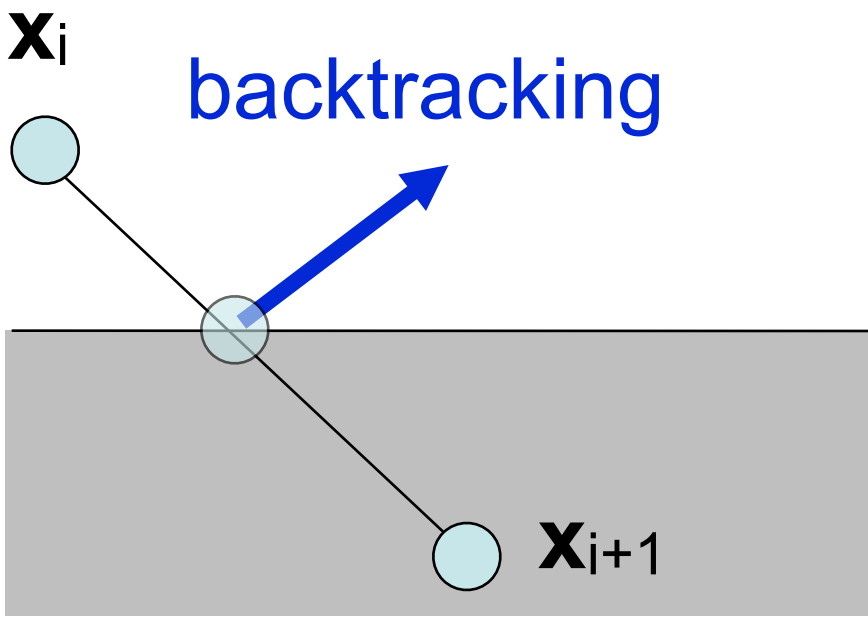
Collisions – Overshooting

- Usually, we detect collision when it's too late: we're already inside
- Your ODE integrator just evaluates forces, doesn't know about surfaces!



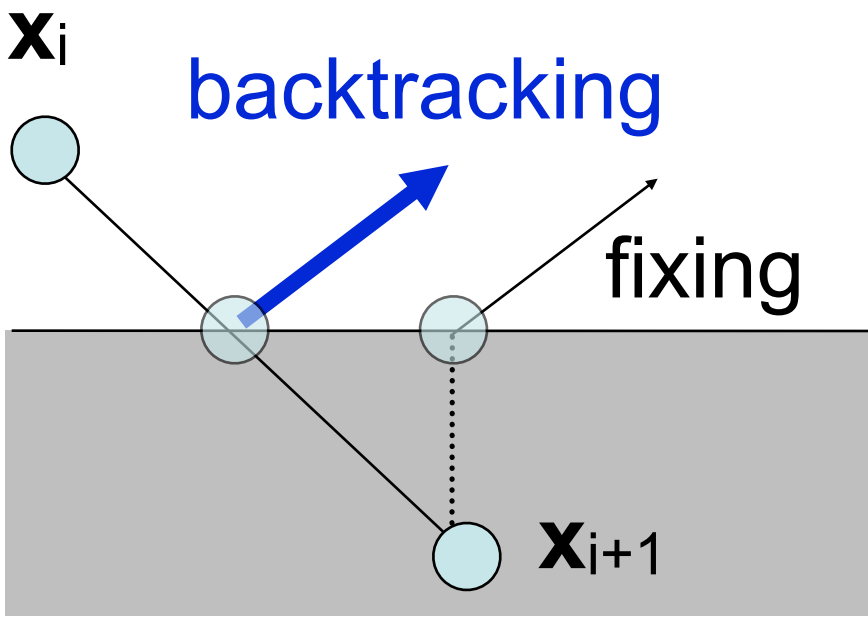
Collisions – Overshooting

- Usually, we detect collision when it's too late: we're already inside
- Solution: Back up
  - Compute intersection point
  - Ray-object intersection!
  - Compute response there
  - Advance for remaining fractional time step



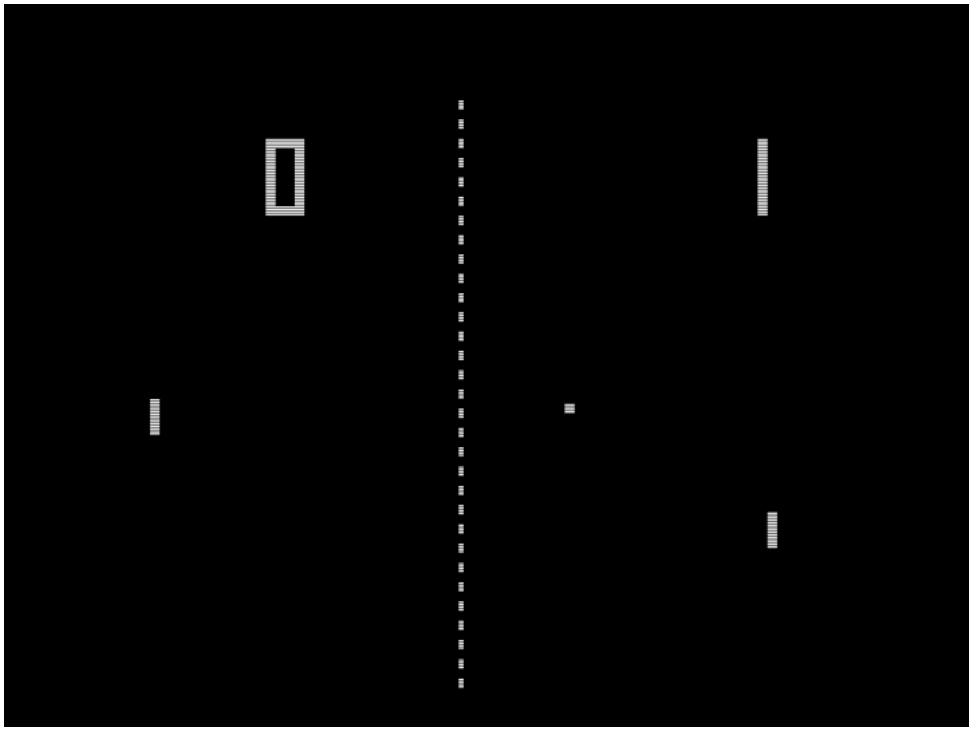
Collisions – Overshooting

- Usually, we detect collision when it's too late: we're 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 fixup
  - Just project back to object closest point



Questions?

- Pong: ε =1
- http://www.youtube.com/watch?v=sWY0Q_lMFfw
- http://www.xnet.se/javaTest/jPong/jPong.html



http://en.wikipedia.org/wiki/Pong

Collision Detection in Big Scenes

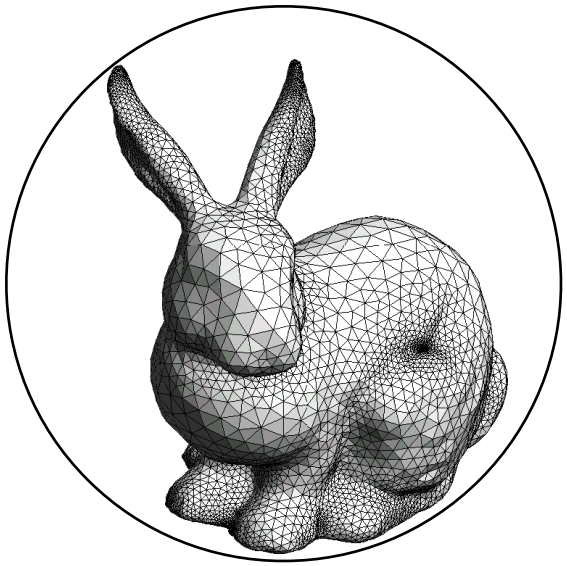
- Imagine we have *n* objects. Can we test all pairwise intersections?
  - Quadratic cost $O(n^2)$!
- Simple optimization: separate static objects
  - But still $O(\text{static} \times \text{dynamic} + \text{dynamic}^2)$
- **Can you think of ways of being smarter?**

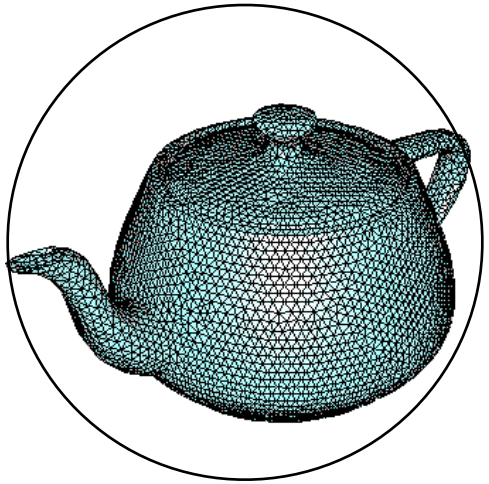
Hierarchical Collision Detection

- 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'll cover only one
- (Remember hierarchical N-body systems from two lectures ago?)

Bounding Spheres

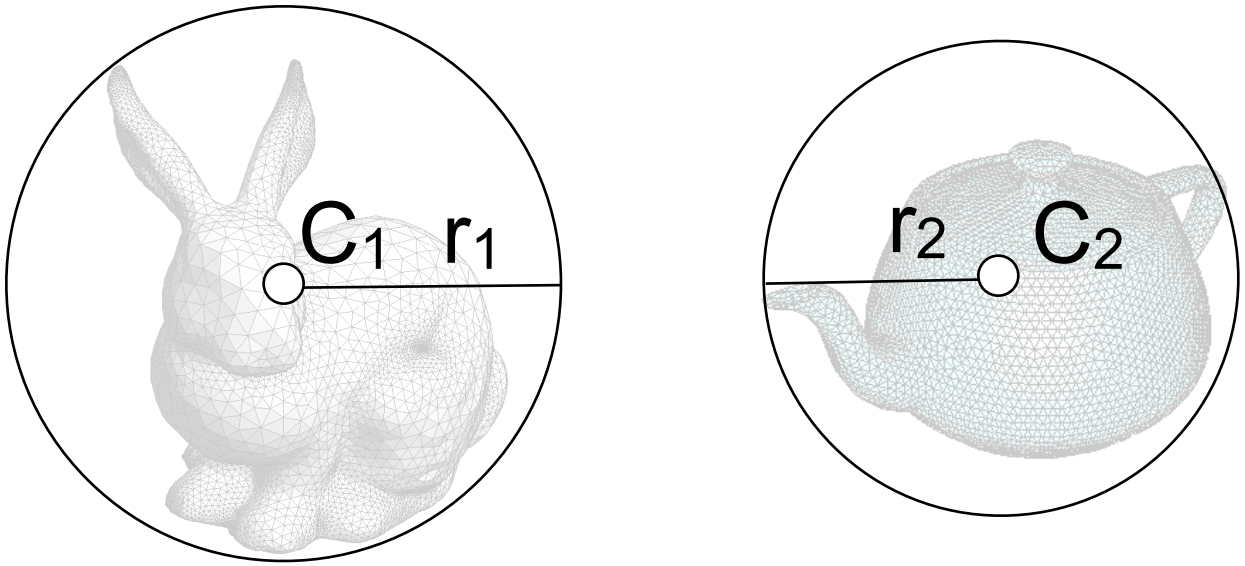
- Place spheres around objects
- If spheres don’t intersect, neither do the objects!
- Sphere-sphere collision test is easy.





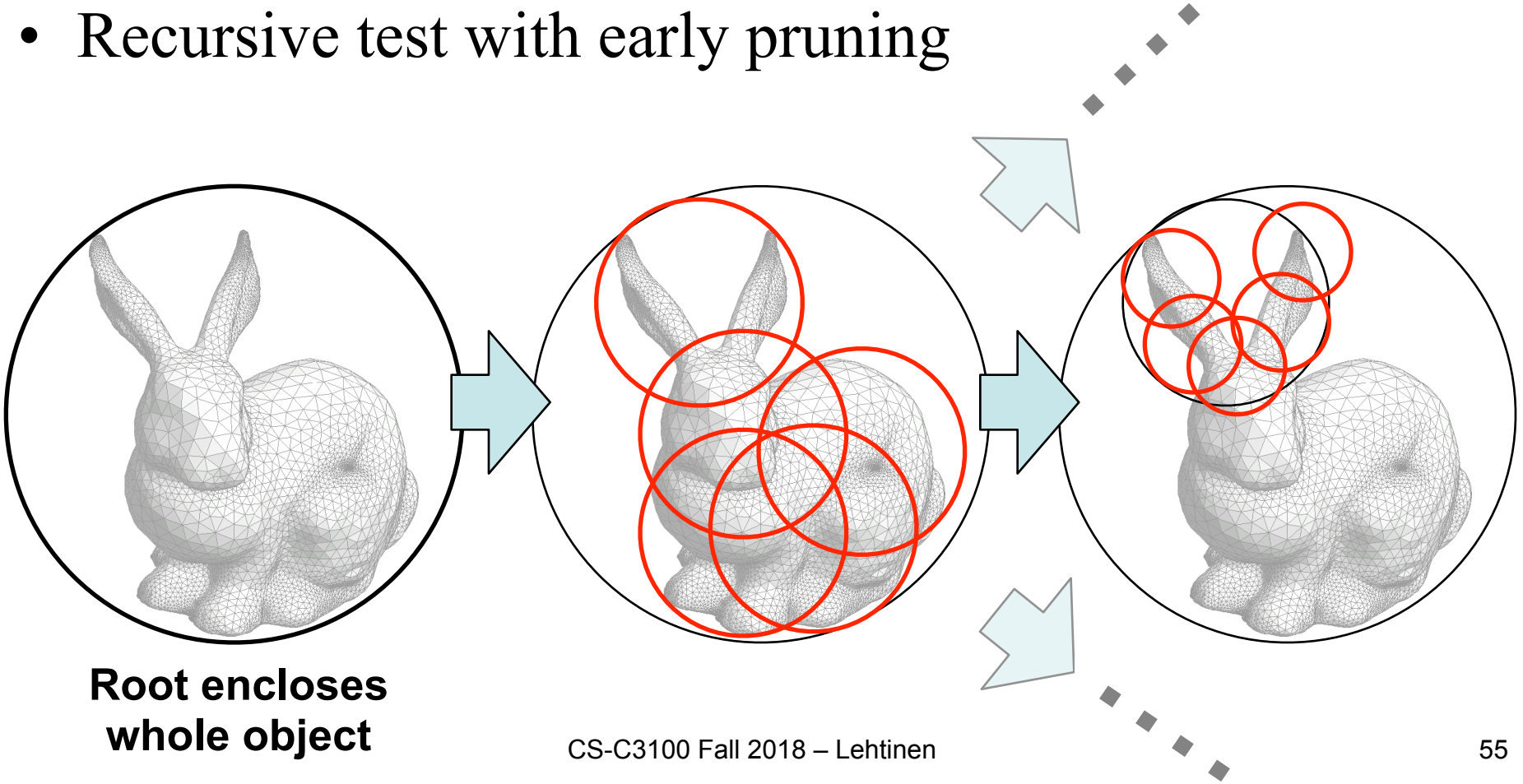
Sphere-Sphere Collision Test

- Two spheres, centers C_1 and C_2, radii r_1 and r2
- Intersect only if $||C_1-C_2||<r_1+r_2$



Hierarchical Collision Test

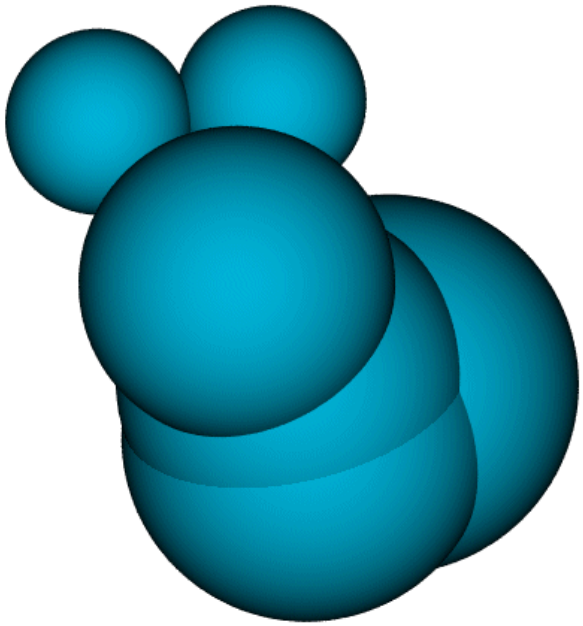
- Hierarchy of bounding spheres
  - Organized in a tree
- Recursive test with early pruning

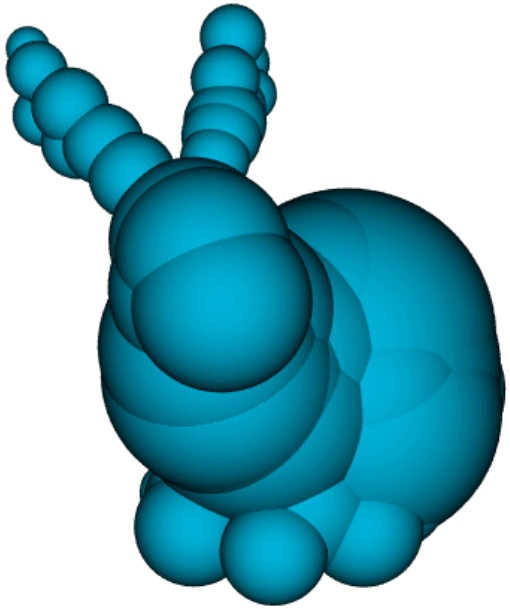


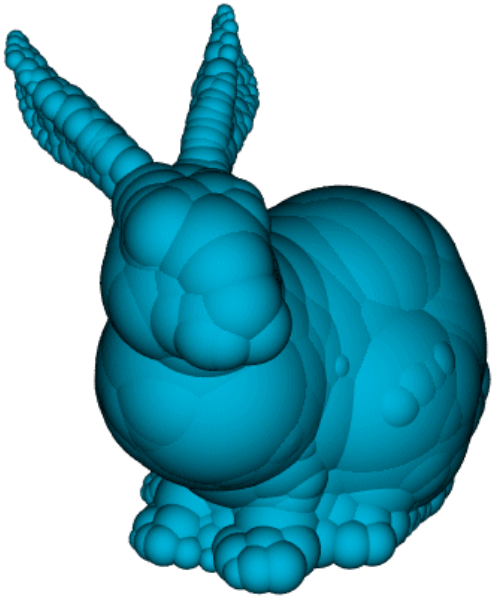
Root encloses whole object

Examples of Hierarchy

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



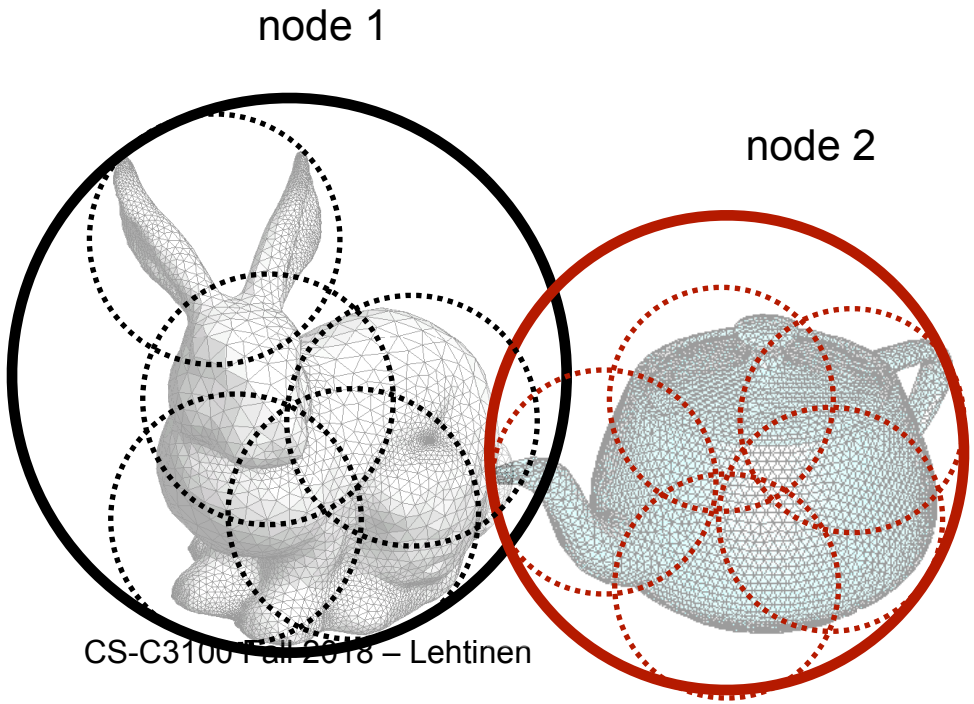




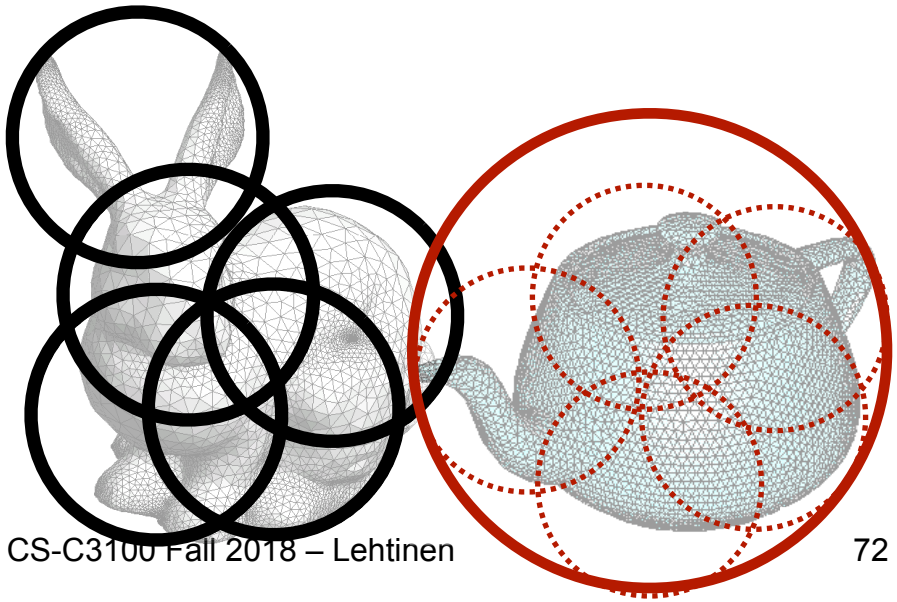
Pseudocode (simplistic version)

```
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 f node2
        if intersect(c, node1) return true

   // no intersection in the subtrees? ==> no intersection!
   return false
```

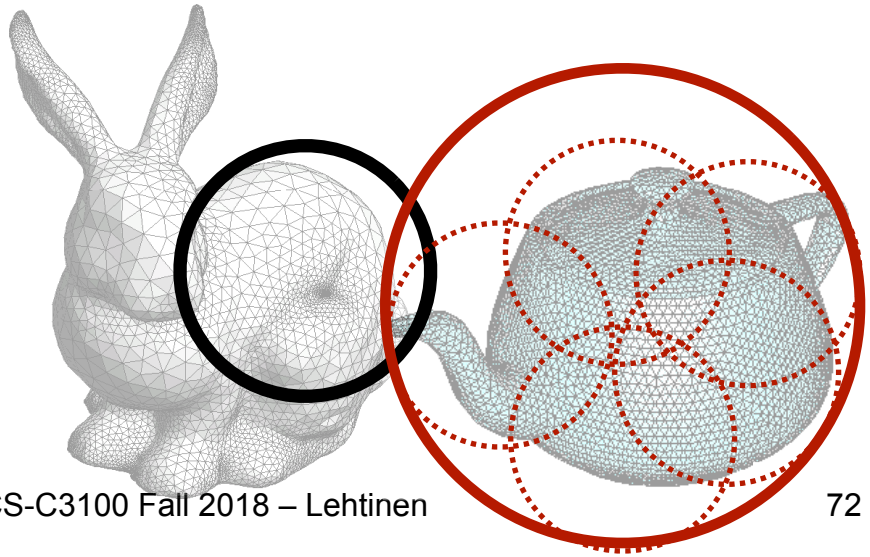
```
boolean intersect(node1, node2)
   if (!overlap(node1->sphere, node2->sphere)
      return false
   if (node1->radius()>node2->radius())
      for each child c of node1
         if intersect(c, node2) return true
   else
      for each child c f node2
        if intersect(c, node1) return true
   return false
```



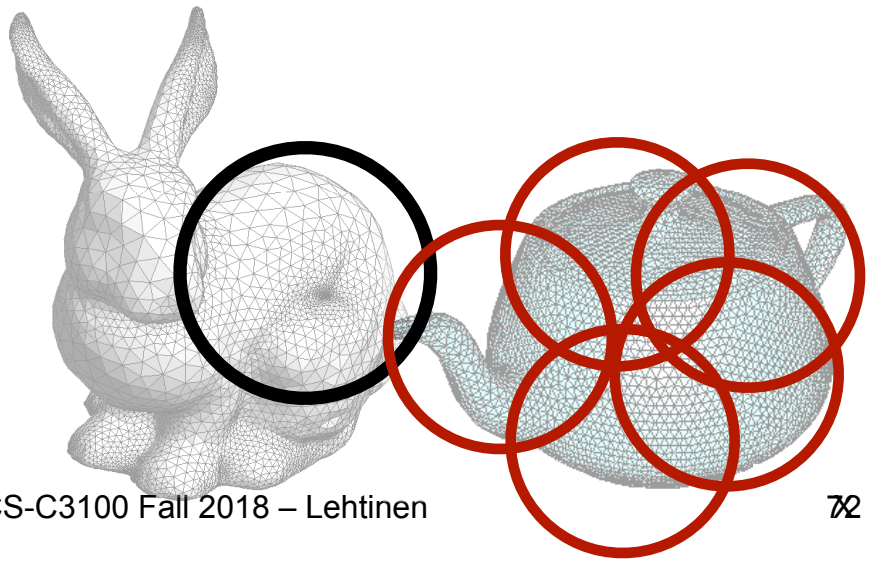
```
boolean intersect(node1, node2)
   if (!overlap(node1->sphere, node2->sphere)
      return false
   if (node1->radius()>node2->radius())
      for each child c of node1
         if intersect(c, node2) return true
   else
      for each child c f node2
        if intersect(c, node1) return true
   return false
```



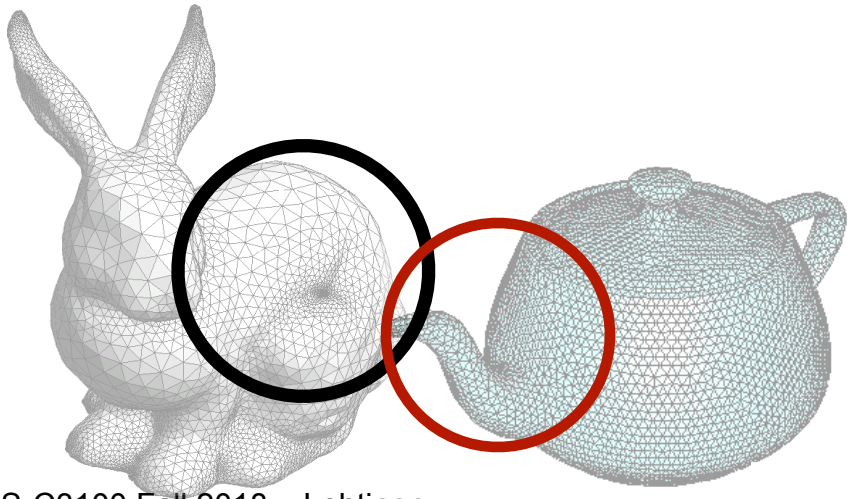
```
boolean intersect(node1, node2)
   if (!overlap(node1->sphere, node2->sphere)
      return false
   if (node1->radius()>node2->radius())
      for each child c of node1
         if intersect(c, node2) return true
   else
      for each child c f node2
        if intersect(c, node1) return true
   return false
```



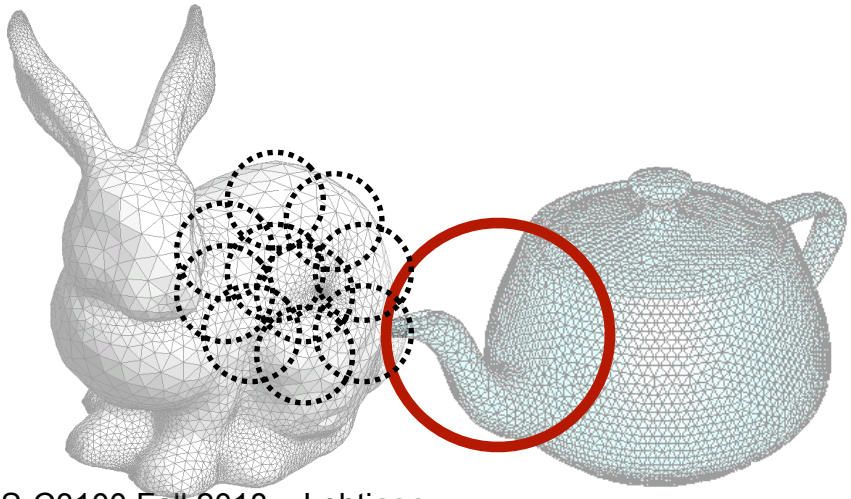
```
boolean intersect(node1, node2)
   if (!overlap(node1->sphere, node2->sphere)
      return false
   if (node1->radius()>node2->radius())
      for each child c of node1
         if intersect(c, node2) return true
   else
      for each child c f node2
        if intersect(c, node1) return true
   return false
```



```
boolean intersect(node1, node2)

    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
```



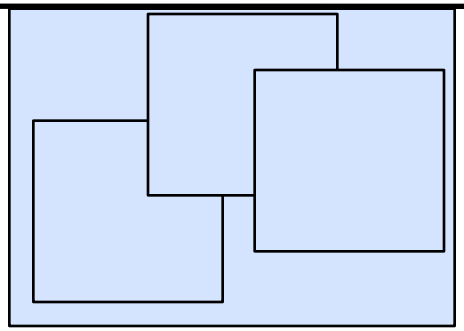
```
boolean intersect(node1, node2)
   if (!overlap(node1->sphere, node2->sphere)
      return false
   if (node1->radius()>node2->radius())
      for each child c of node1
         if intersect(c, node2) return true
   else
      for each child c f node2
        if intersect(c, node1) return true
   return false
```



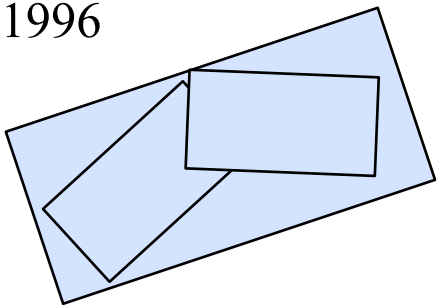
Pseudocode (with leaf case)

```
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
```

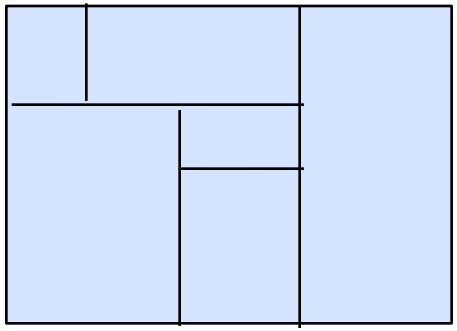
Other Options



- Axis Aligned Bounding Boxes
  - "R-Trees"
- Oriented bounding boxes
  - S. Gottschalk, M. Lin, and D. Manocha. "OBBTree: A hierarchical Structure for rapid interference detection," Proc. Siggraph 96. ACM Press, 1996



- Binary space partitioning trees; kd-trees

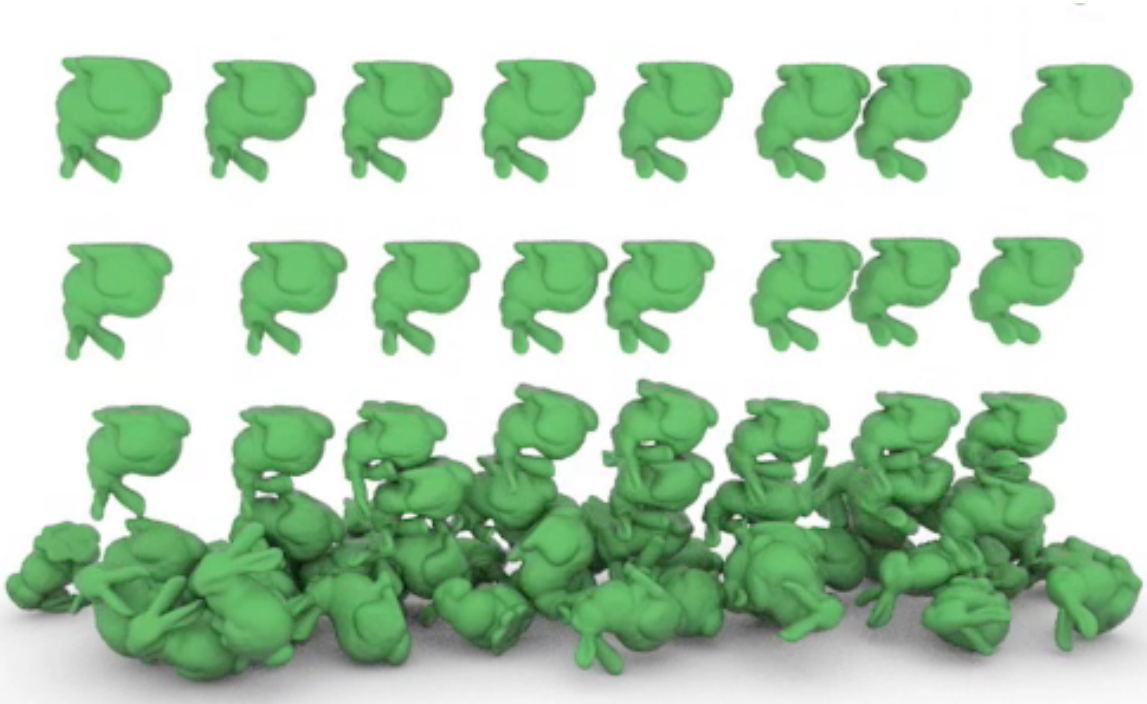


Questions?

- http://www.youtube.com/watch?v=nFd9BIcpHX4&feature=related

Deformable Objects

- When the objects deform, things get more complex, but modern methods are pretty good with those as well
  - Barbic, James Subspace Self-Collision Culling, SIGGRAPH 2010

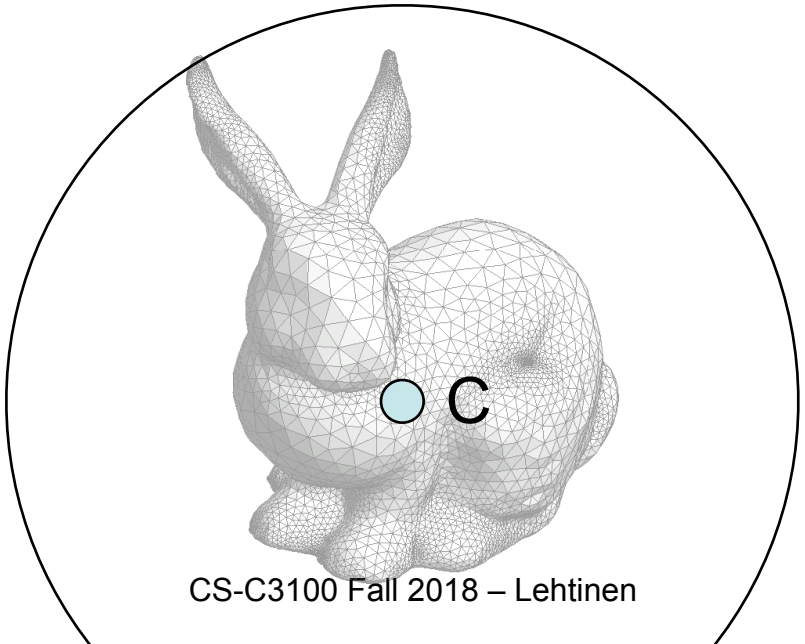


Hierarchy construction

- Top down
  - Divide and conquer
- Bottom up
  - Cluster nearby objects
- Incremental
  - Add objects one by one, binary-tree style.

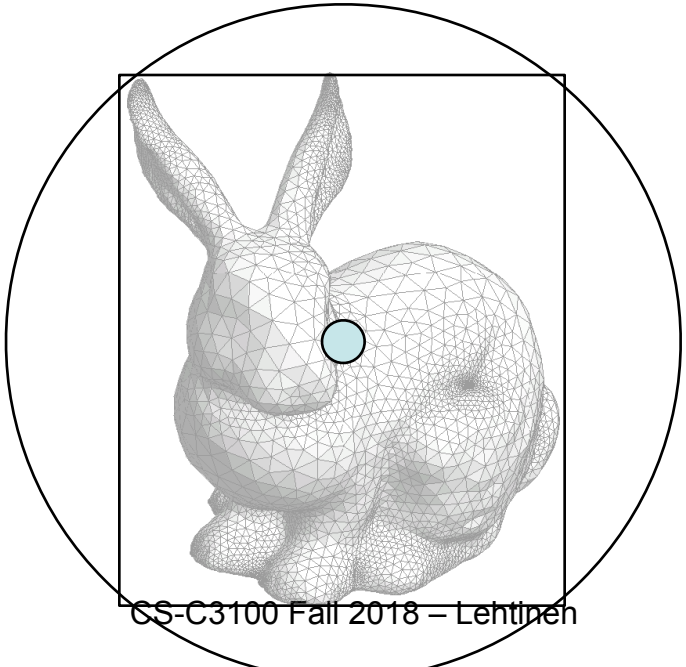
Bounding sphere of a set of points

- Trivial given center C
  - radius = $\max_i \|C-P_i\|$



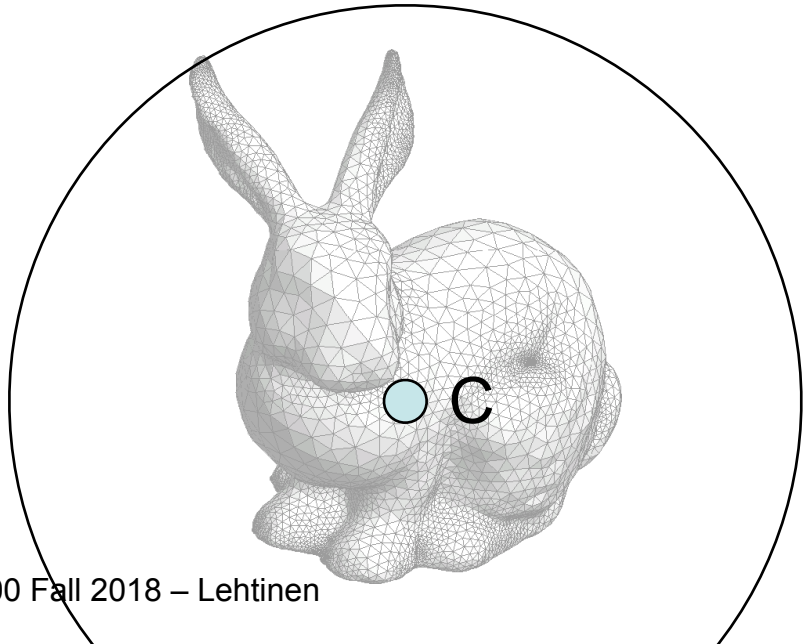
Bounding sphere of a set of points

- Using axis-aligned bounding box
  - center= $((x_{min}+x_{max})/2, (y_{min}+y_{max})/2, (z_{min}, z_{max})/2)$
  - Better than the average of the vertices because does not suffer from non-uniform tessellation



Optimal Bounding Sphere?

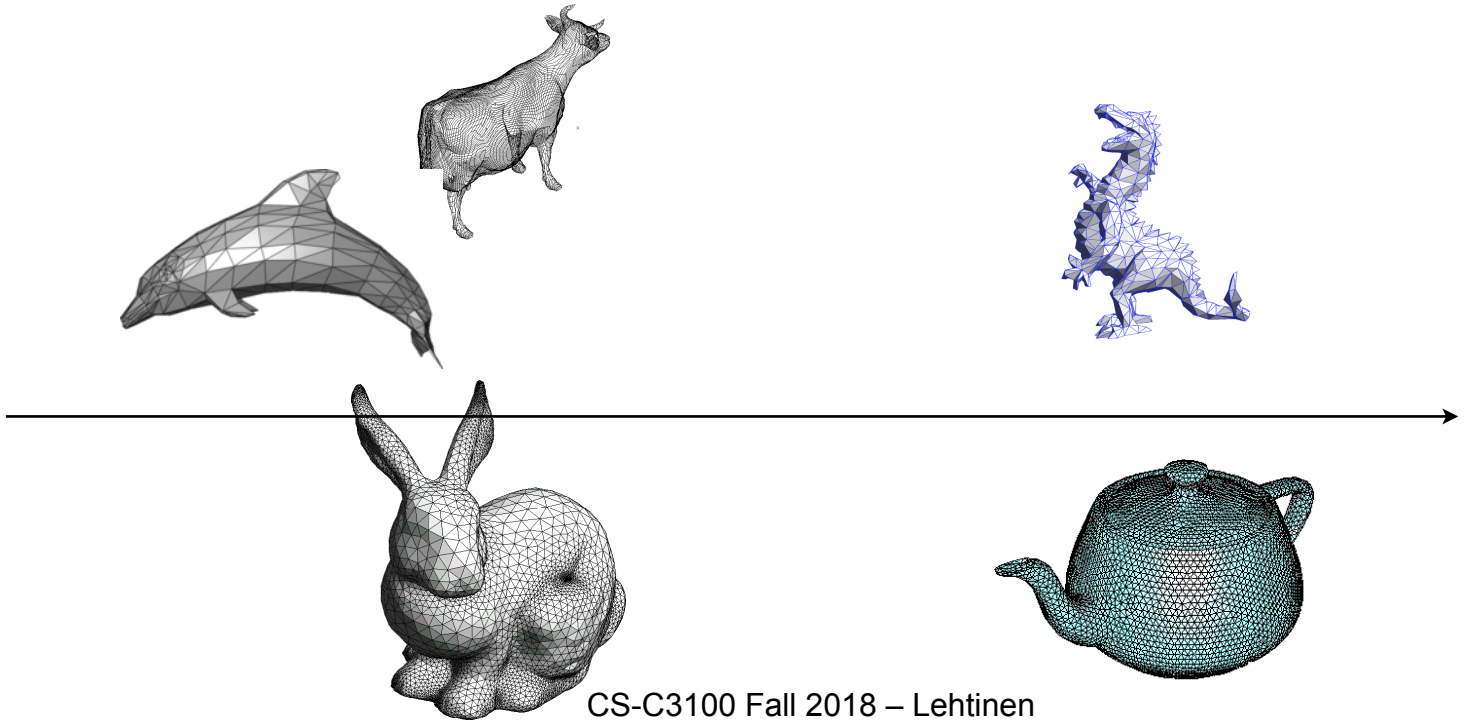
- Using the centroid of the vertices for the center C is not necessarily optimal.
- Optimal construction is difficult
- Other approximations exist.
  - We'll leave it at that.



Questions?

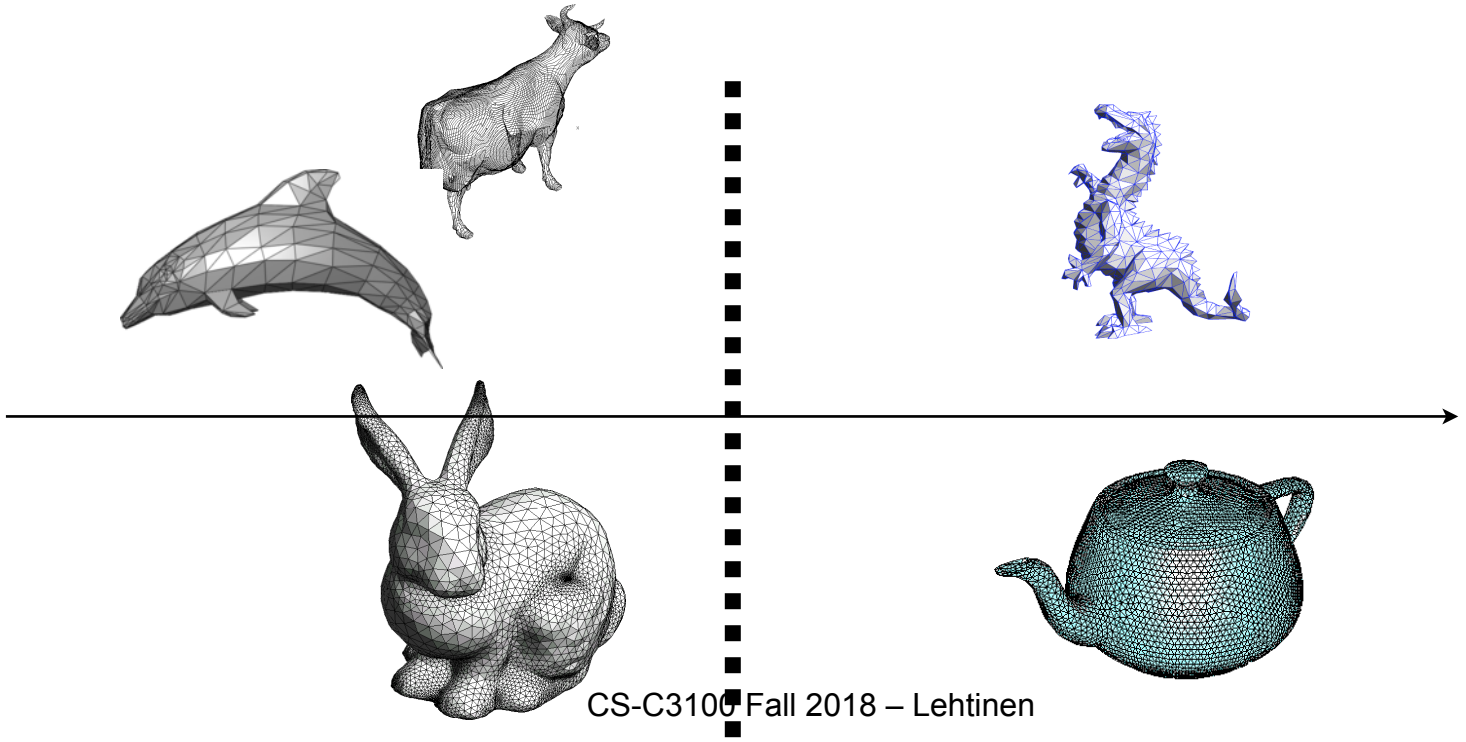
Top-Down Construction

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



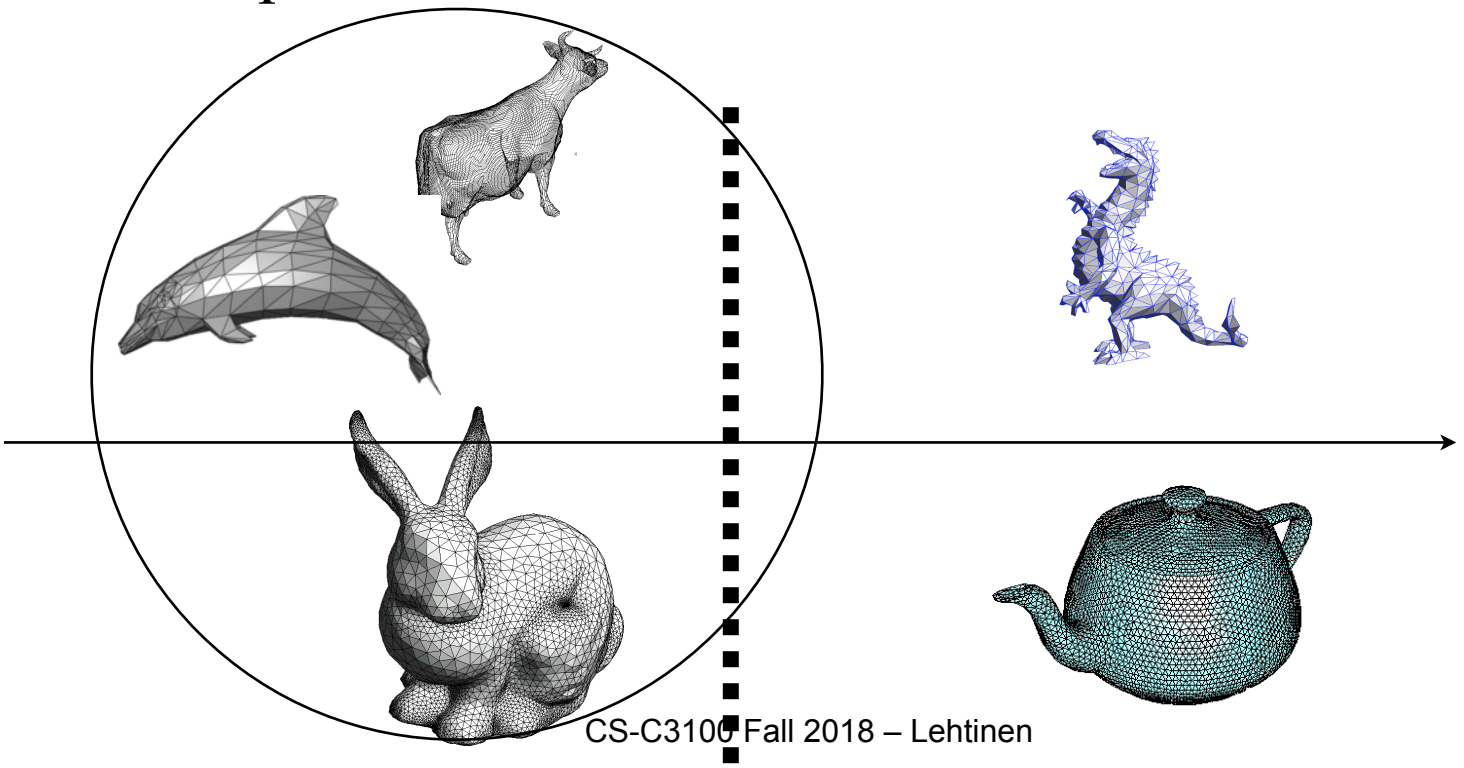
Top-Down Construction

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



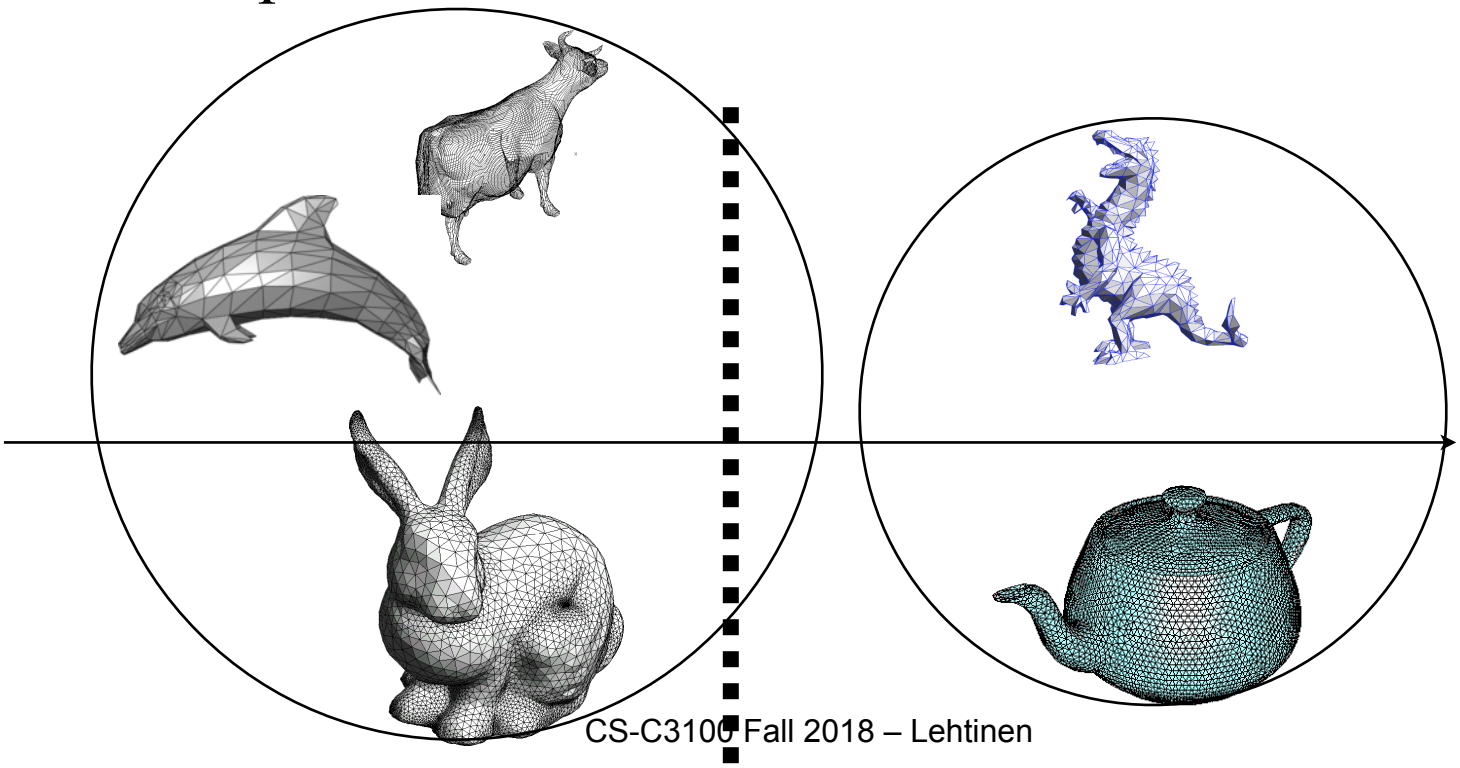
Top-Down Construction

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



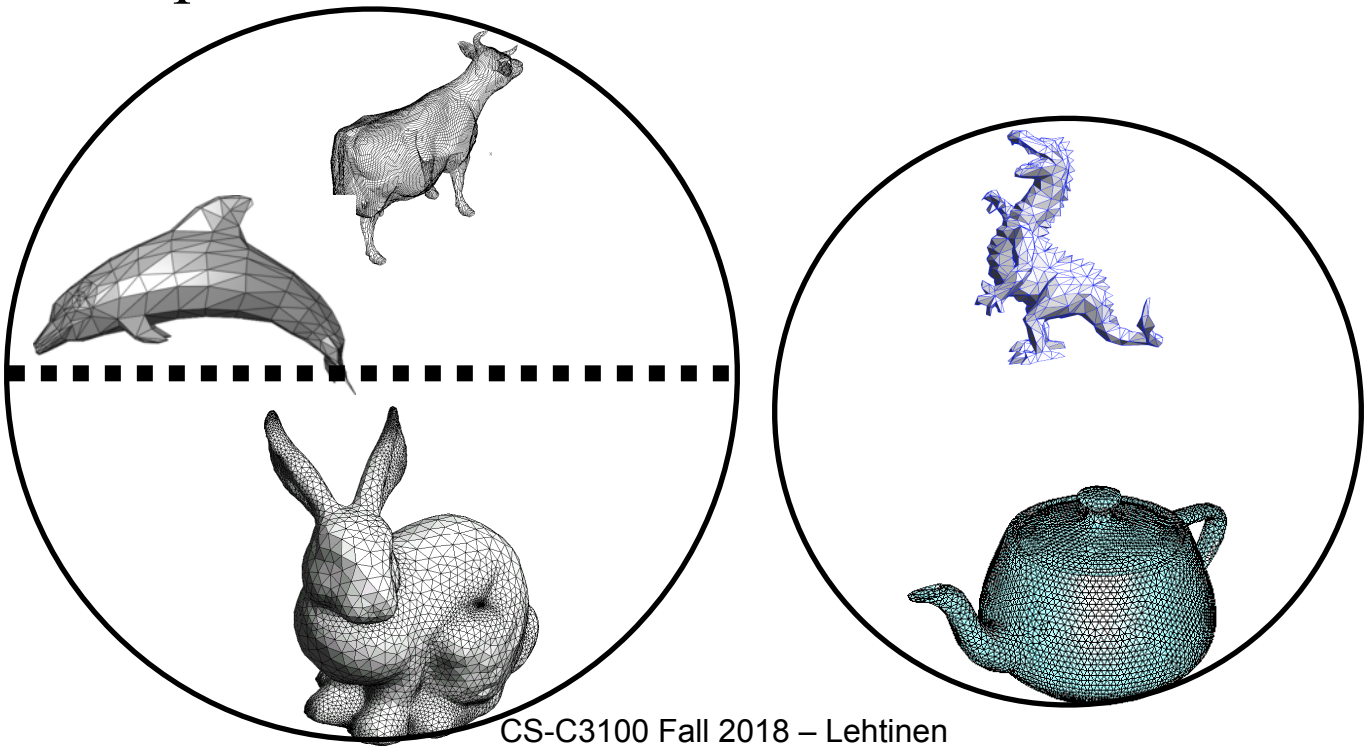
Top-Down Construction

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



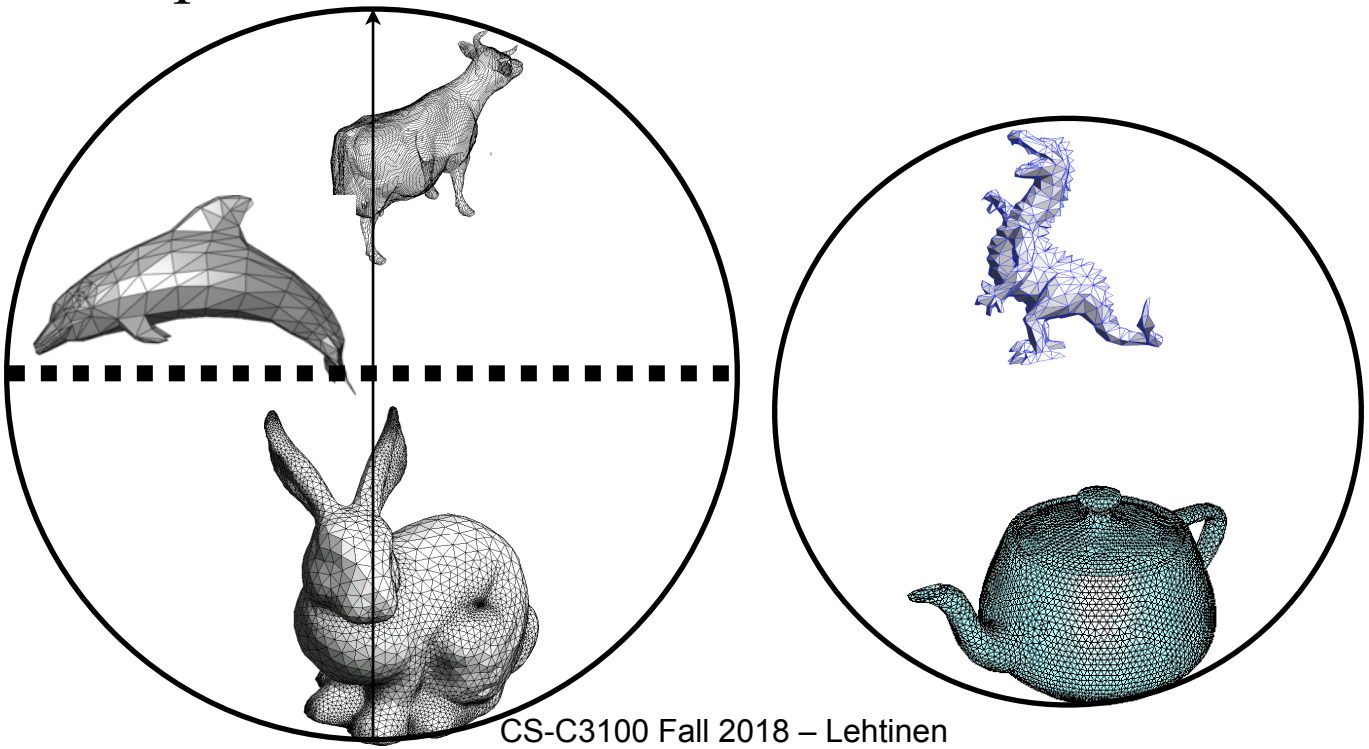
Top-Down Construction - Recurse

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



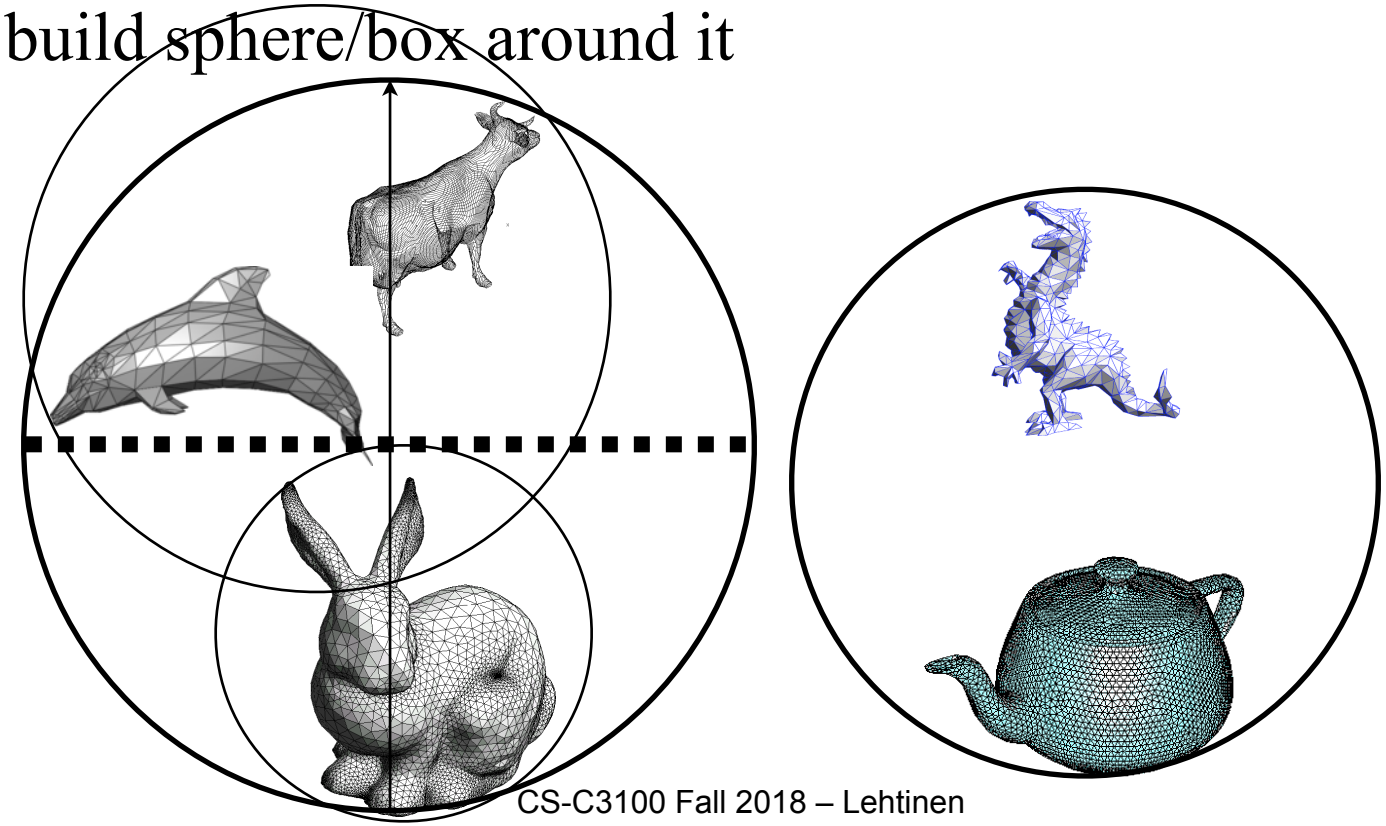
Top-Down Construction - Recurse

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



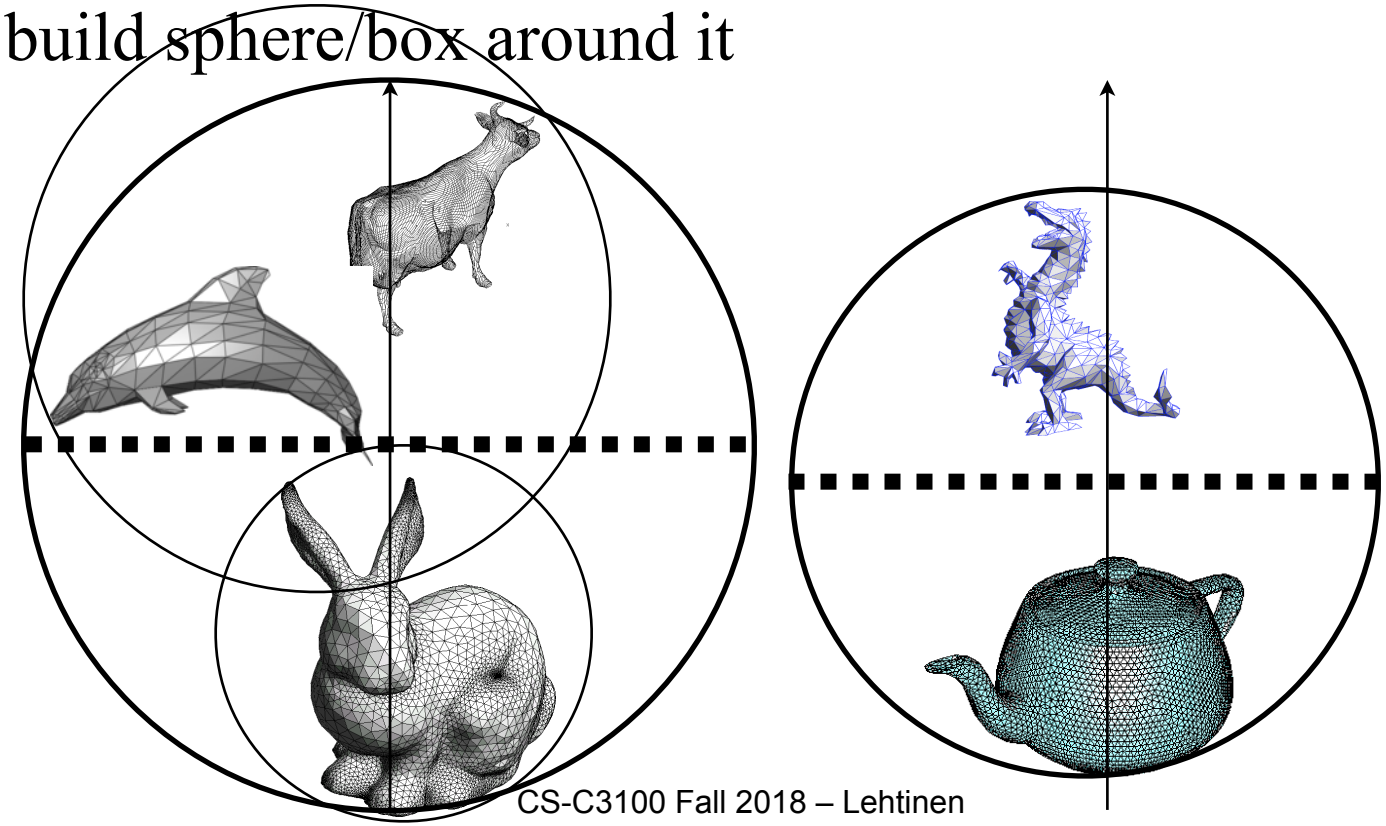
Top-Down Construction - Recurse

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



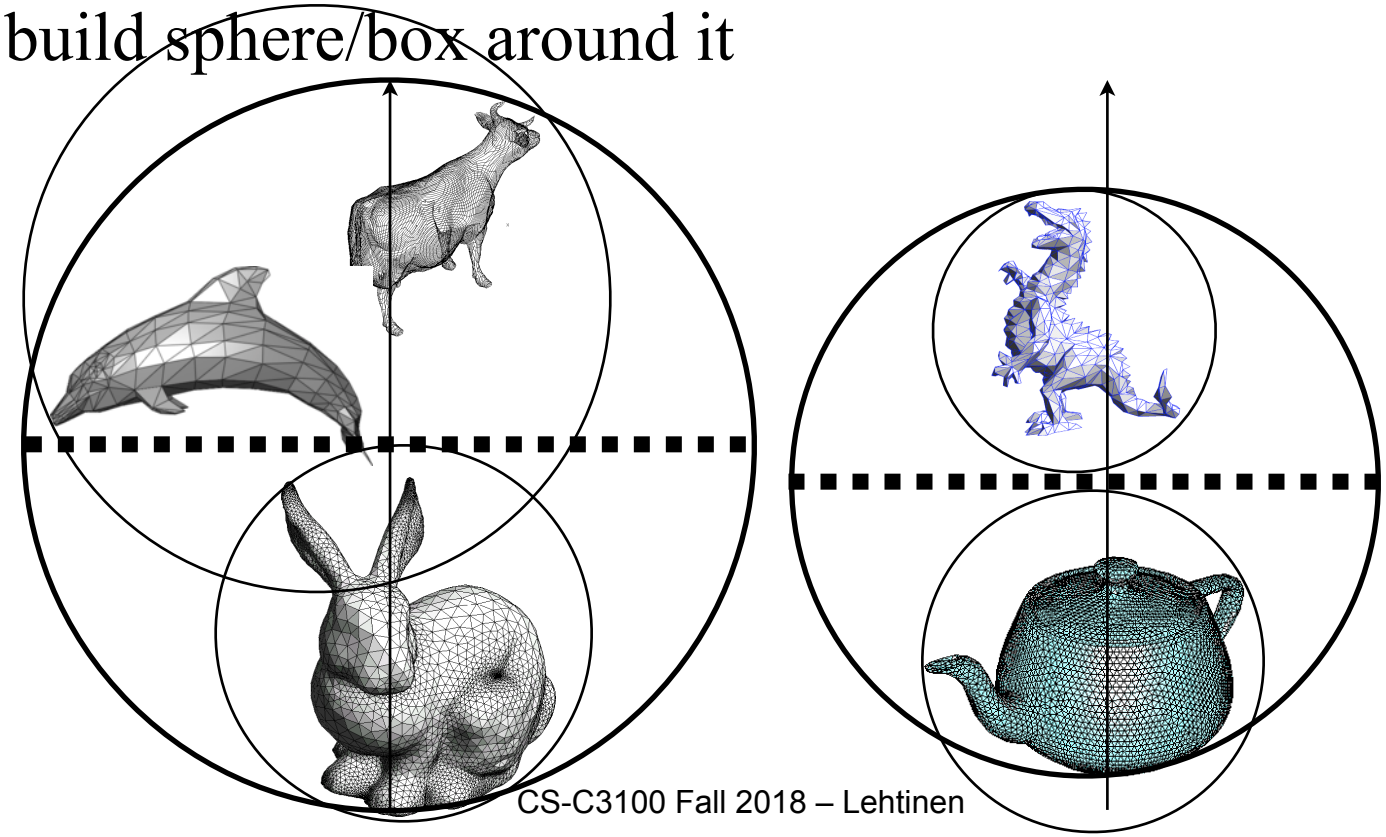
Top-Down Construction - Recurse

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



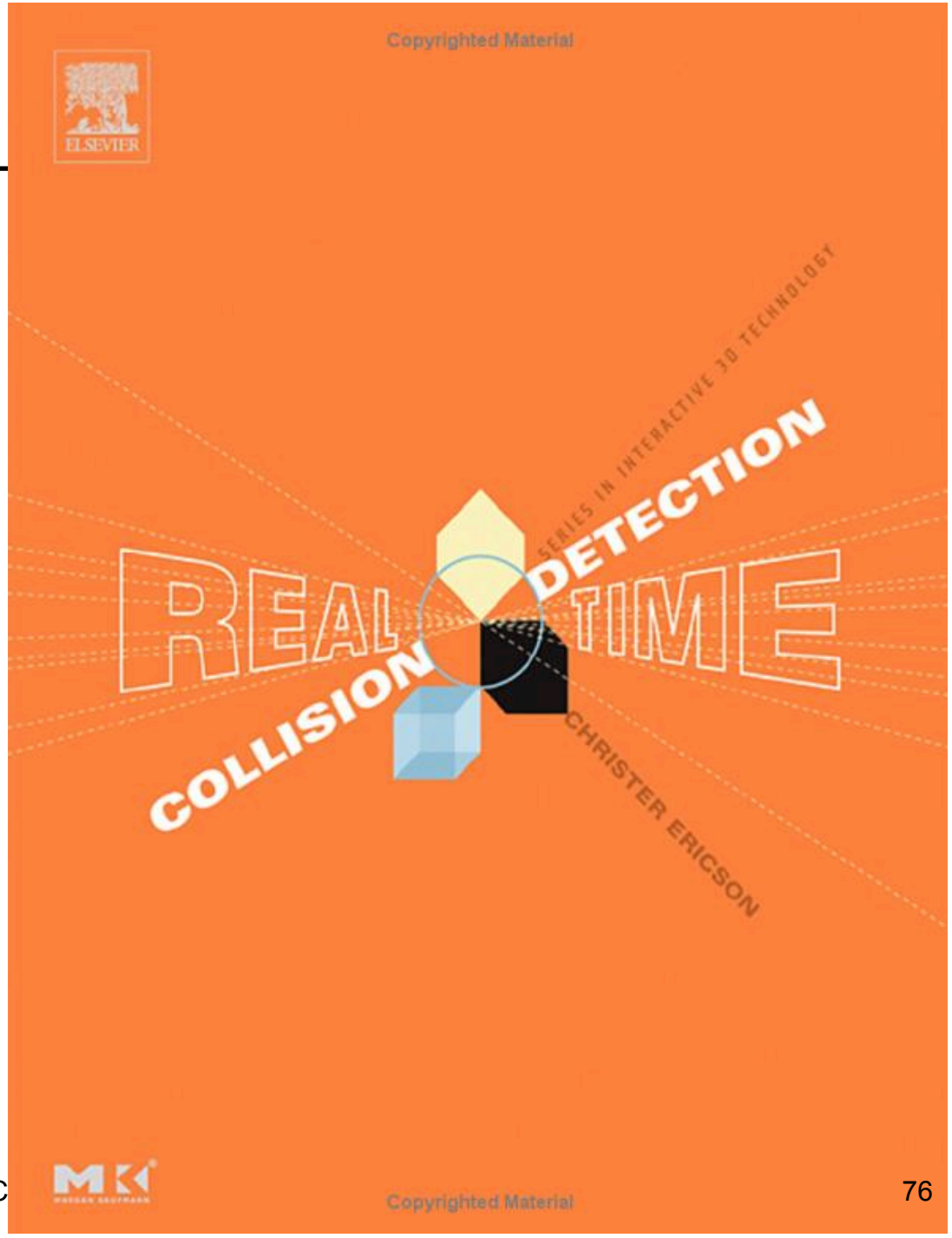
Top-Down Construction - Recurse

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



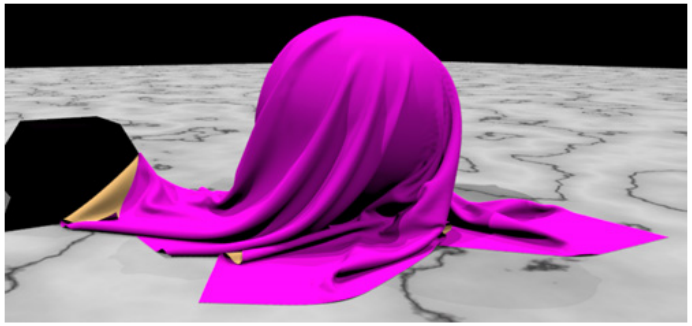
Questions?

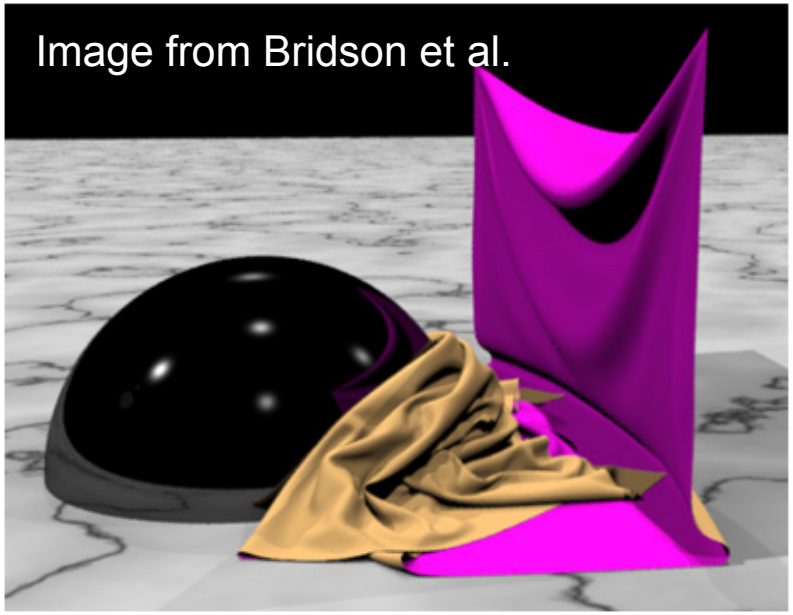
Reference

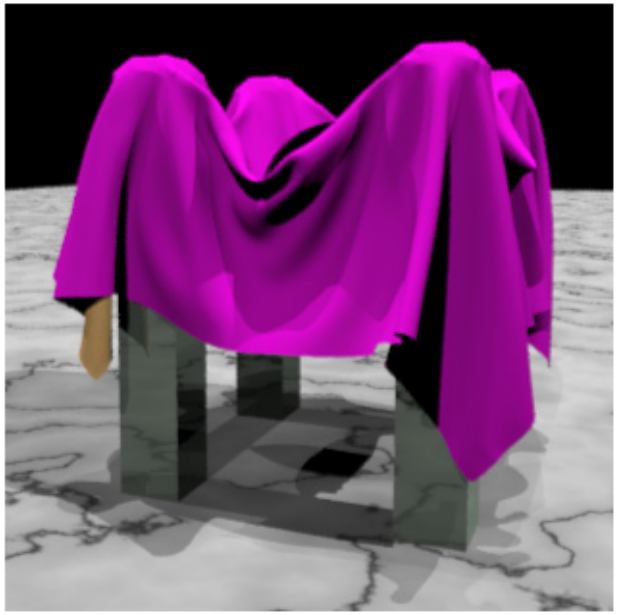

Copyrighted Material
ELSEVIER
SERIES IN INTERACTIVE 3D TECHNOLOGY
REAL-TIME
COLLISION
DETECTION
CHRISTER ERICSON
MK
Copyrighted Material

The cloth collision problem

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







Robust Treatment of Simultaneous Collisions

David Harmon, Etienne Vouga, Rasmus Tamstorf, Eitan Grinspun

Robust Treatment of Simultaneous Collisions

David Harmon, Etienne Vouga, Rasmus Tamstorf, Eitan Grinspun

That's All for Today!

- ... and for simulation and modeling, for that matter
- Next we'll start to work our way into rendering!
- But before that, good luck in your midterm!