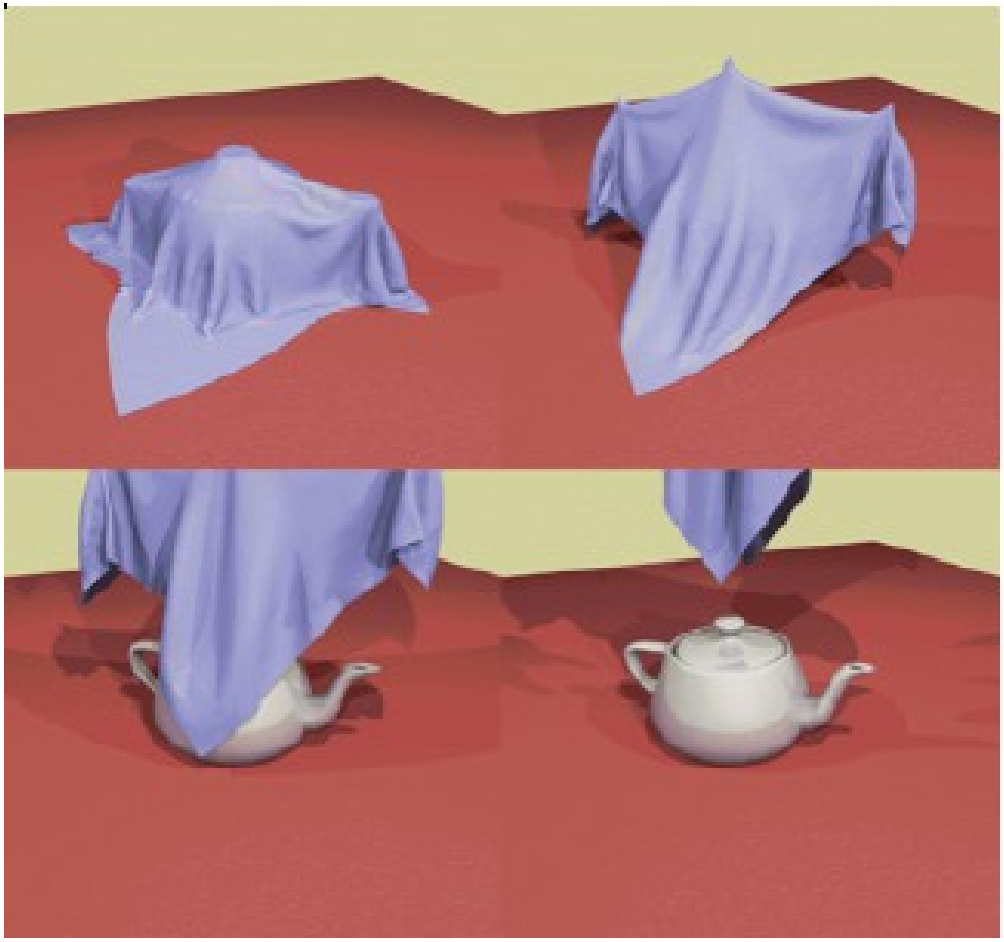
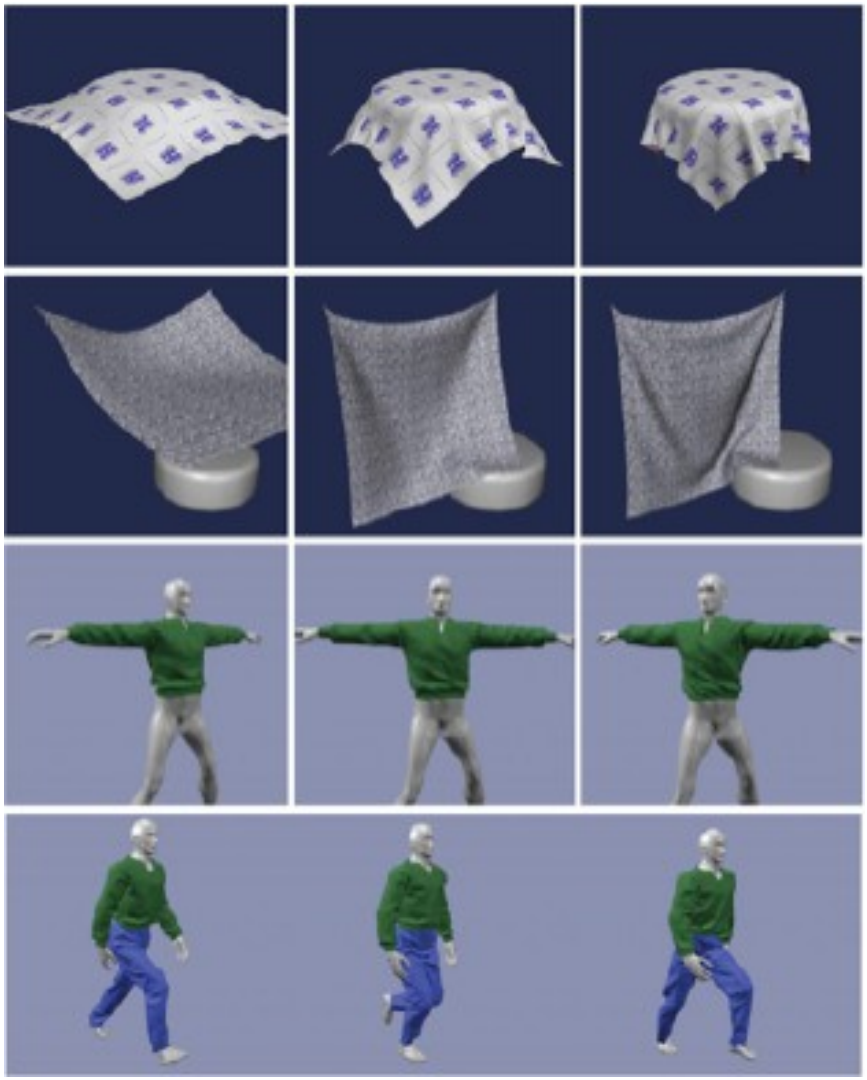
Cloth animation



Demo



Demo

What is cloth?

- 2 basic types: woven and knit
- We'll restrict to woven
  - Warp vs. weft

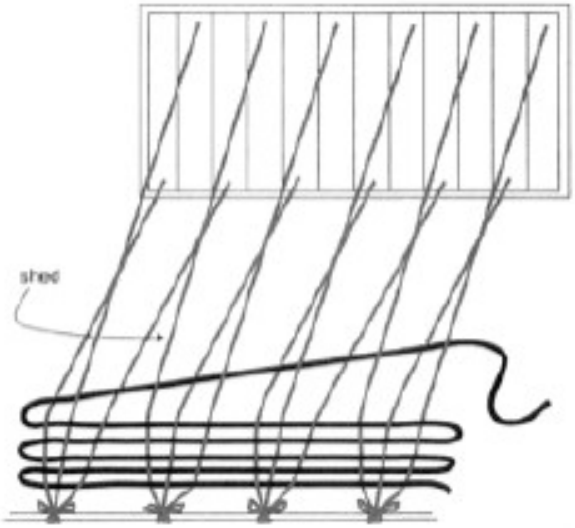
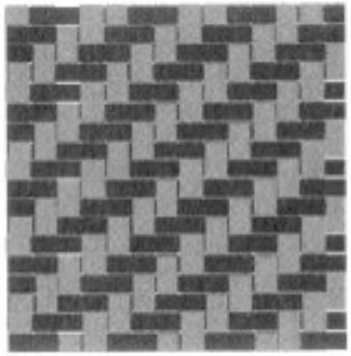
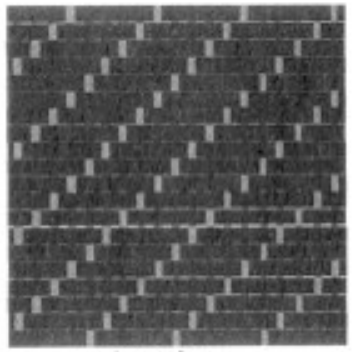


Figure 1.8. The weaving process.



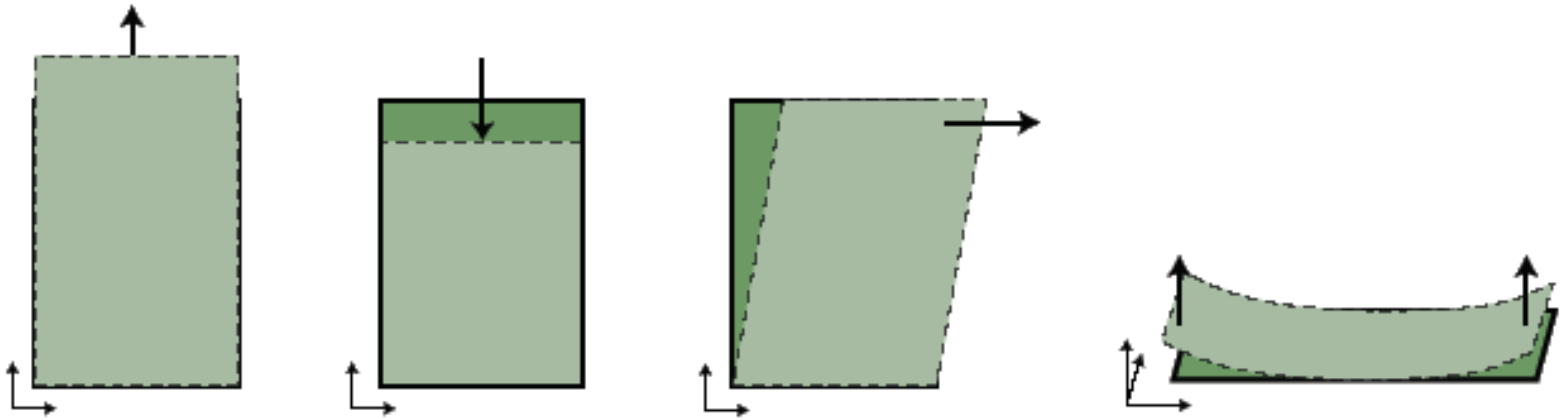
b) twill



c) satin

Cloth modeling basics

- In general, cloth resists motion in 4 directions:



In-plane stretch

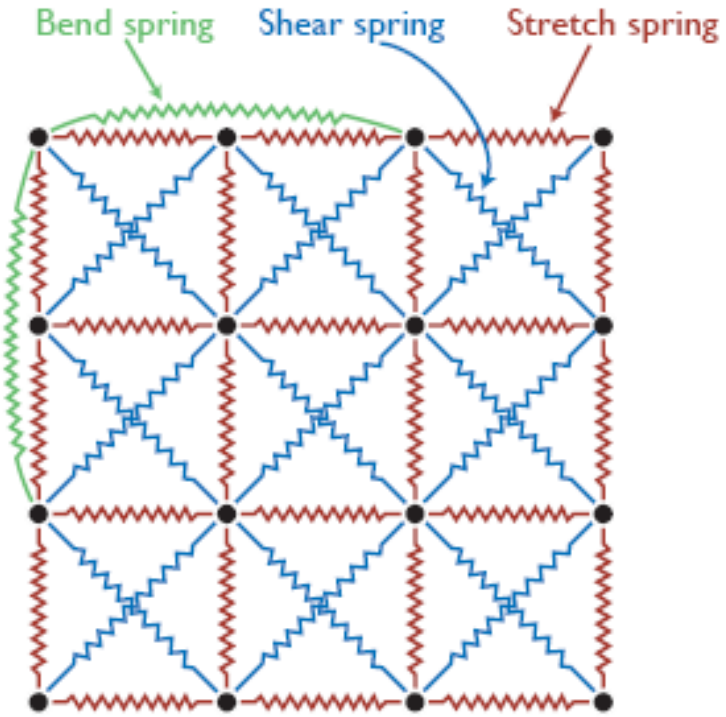
In-plane compression

In-plane shear (trellising)

Out-of-plane bending

A basic mass-spring model

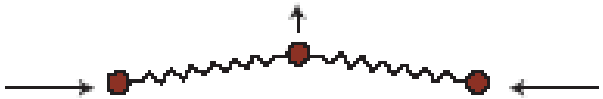
- Simple spring-mass system due to Provot [1995]
- You already know how to implement this



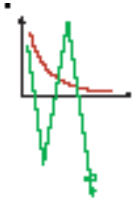
Basic problem: when we push on a piece of cloth like this,

we expect to see this:

But, in our basic particle system model, we have to make the compression forces very stiff to get significant out-of-plane motion. This is expensive.



Stiffness in ODEs -- example



Consider the following ODE:

$$\frac{dx}{dt} = -kx, \quad k \gg 1$$

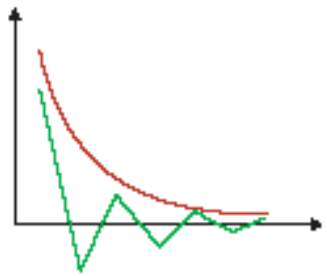
The analytical solution is

$$x(t) = Ce^{-kt}$$

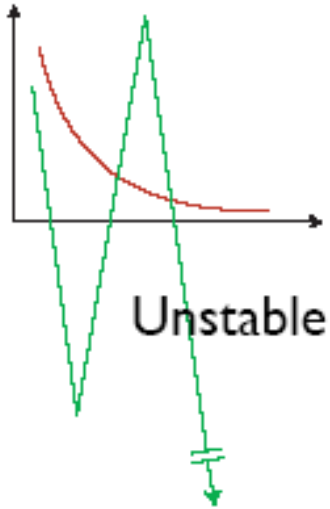
If we solve it with Euler's method,

$$x_{t+h} = x_t - hkx_t = (1 - hk)x_t$$

What happens when $hk \gg 1$?

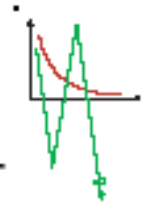


Barely stable



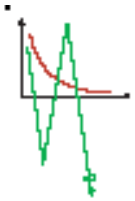
Unstable

Stiffness in cloth



- In general, cloth stretches little if at all in the plane
- To counter this, we generally have large in-plane stretch forces (otherwise the cloth looks "wiggly")
- The result: stiffness!

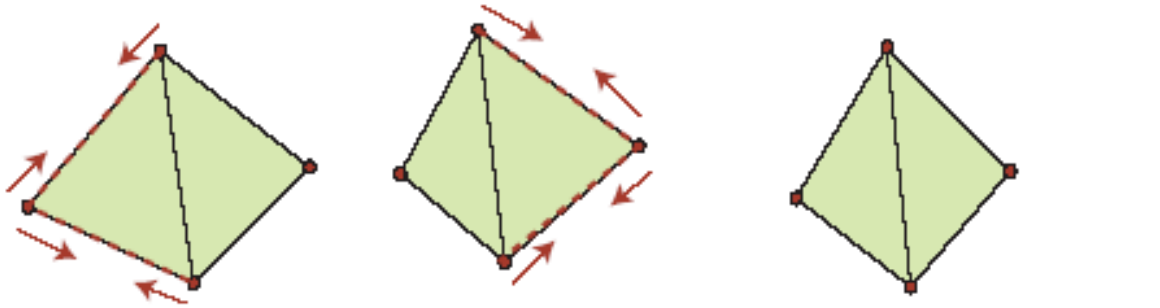
Avoiding stiffness



An alternative approach is to avoid stiffness altogether by applying only non-stiff spring forces and then "fixing" the solution at the end of the timestep.

(Provot [1995], Desbrun et al [1999], Bridson et al [2002])

We can do this with impulses and Jacobi iteration.



Iteration 1 Iteration 2 Iteration 3 (converged)

Particle-based methods

- Breen [1992]: energy-based model

$$U_i = U_{repel_i} + U_{stretch_i} + U_{bend_i} + U_{trellis_i}$$

- Find final draping position by minimizing the total energy in the cloth
  - NOT dynamic!

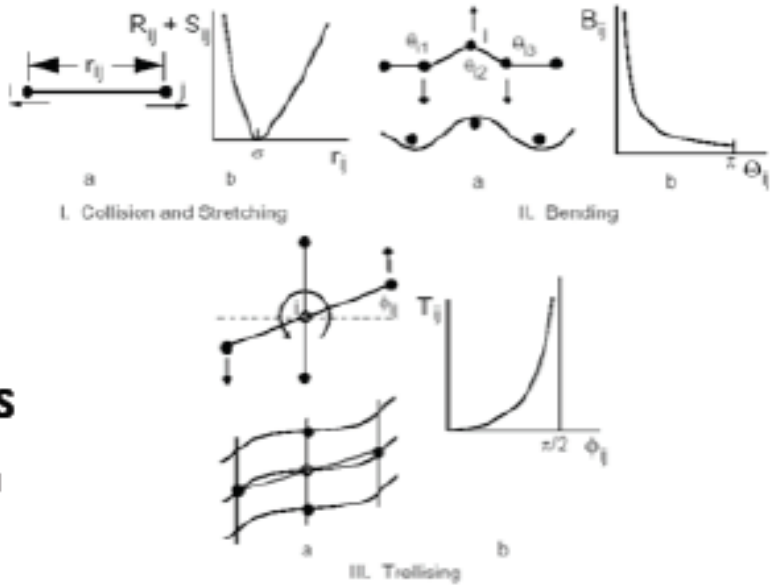


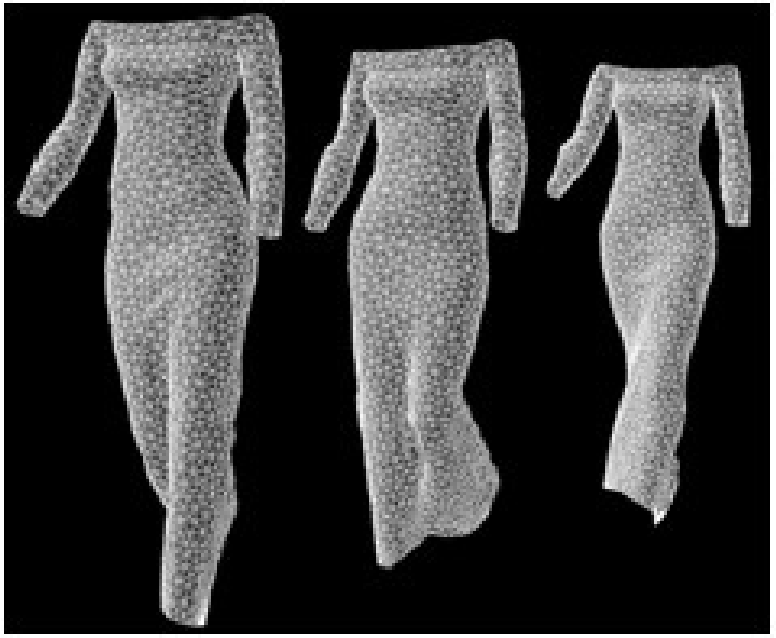
Figure 3: Cloth model energy functions

Note: You could convert this to a "normal" particle system model by differentiating energy w.r.t. position,

$$\mathbf{F} = -\nabla_{\mathbf{x}} U$$

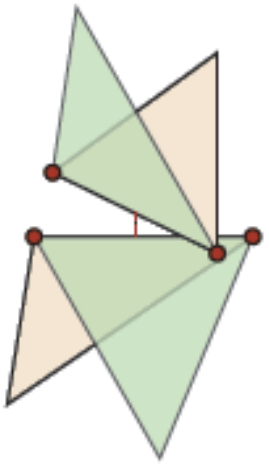
Collision Detection

- **Numerical Complexity**
  - ~ *Arises from the high number of polygons that the object meshes have (cloth and body, several thousands of polygons), and how to extract the colliding polygons quickly.*

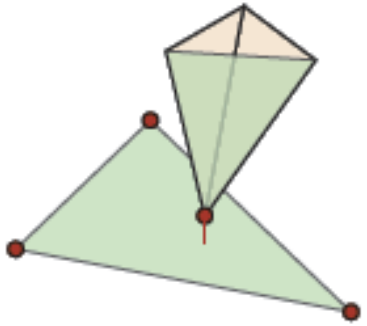


Collision detection

- We already covered this for deformable bodies
- Many of the same methods work, especially acceleration methods
- Generally need to do triangle-triangle collision checks:



Edge-edge collision



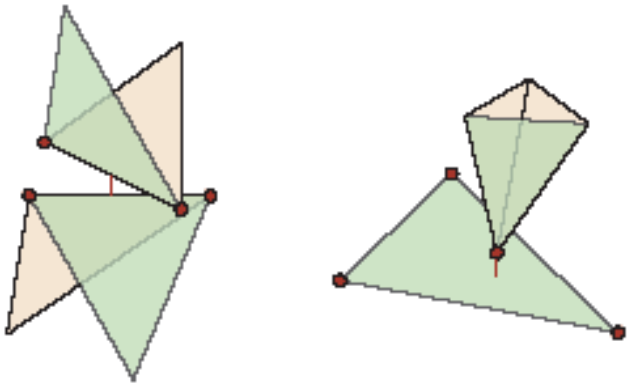
Point-face collision

Robust collision detection

If triangles are moving too fast, they may pass through each other in a single timestep.

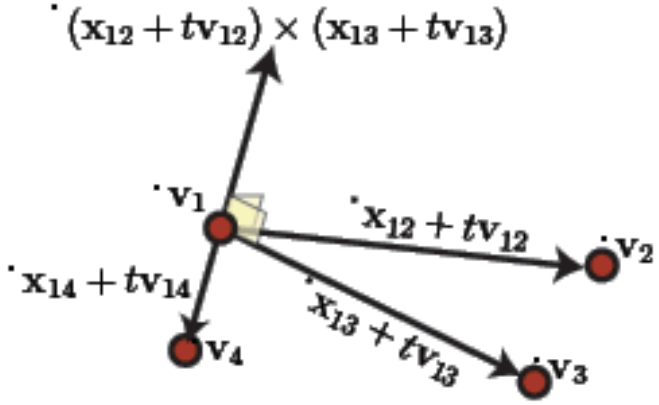
We can prevent this by checking for *any* collisions during the timestep (Provot [1997])

Note first that both point-face and edge-edge collisions occur when the appropriate 4 points are *coplanar*



Robust collision detection (2)

Detecting time of coplanarity - assume linear velocity throughout timestep:



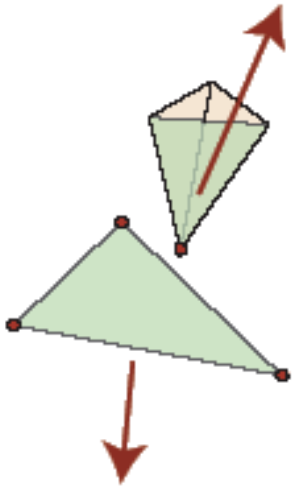
So the problem reduces to finding roots of the cubic equation

$$((\mathbf{x}_{12} + t\mathbf{v}_{12}) \times (\mathbf{x}_{13} + t\mathbf{v}_{13})) \cdot (\mathbf{x}_{14} + t\mathbf{v}_{14})$$

Once we have these roots, we can plug back in and test for triangle adjacency.

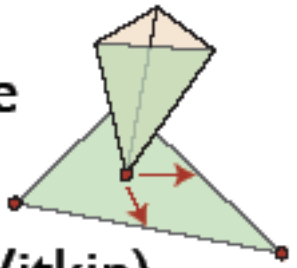
Collision response

- 4 basic options:
  - Constraint-based
  - Penalty forces
  - Impulse-based
  - Rigid body dynamics (will explain)



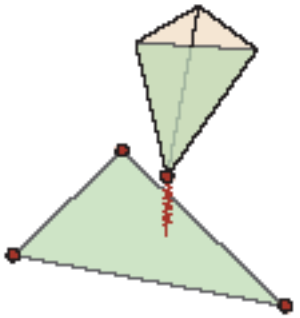
Constraint-based response

- Assume totally inelastic collision
- Constrain particle to lie on triangle surface
- Benefits:
  - Fast, may not add stiffness (e.g., Baraff/Witkin)
  - No extra damping needed
- Drawbacks
  - Only supports point-face collisions
  - Constraint attachment, release add discontinuities (constants hard to get right)
  - Doesn't handle self-collisions (generally)
- Conclusion: a good place to start, but not robust enough for heavy-duty work



Penalty forces

- Apply a spring force that keeps particles away from each other



- Benefits:
  - Easy to fit into an existing simulator
  - Works with all kinds of collisions (use barycentric coordinates to distribute responses among vertices)
- Drawbacks:
  - Hard to tune: if force is too weak, it will sometimes fail; if force is too strong, it will cause the particles to "float" and "wiggle"

Impulses

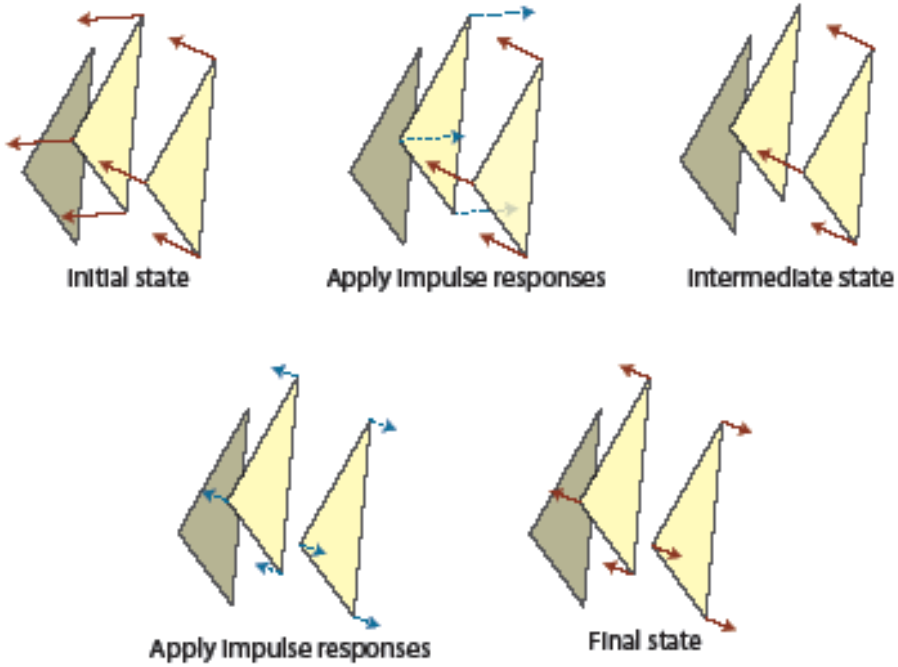
- "Instantaneous" change in momentum

$$\mathbf{J} = \int_{t_i}^{t_f} \mathbf{F}\, dt = \mathbf{p}_f - \mathbf{p}_i$$

- Generally applied outside the simulator timestep
- Benefits
  - Correctly stops all collisions (no sloppy spring forces)
- Drawbacks
  - Can have poor numerical performance
  - Handles persistent contact poorly

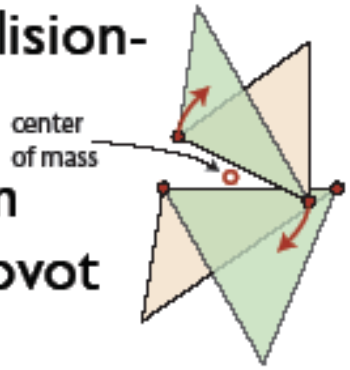
Impulses (2)

Iteration is generally necessary to remove all collisions.



Rigid collision impact zones

- Basic idea: if a group of particles *start* timestep collision-free, and move as a rigid body throughout the timestep, then they will *end* timestep collision-free.
- We can group particles involved in a collision together and move them as a rigid body (Provot [1997] -- error?, Bridson [2002])



$$x_{CM} = \frac{\sum_i m_i \mathbf{x}_i}{m_i} \qquad v_{CM} = \frac{\sum_i m_i \mathbf{v}_i}{m_i}$$ Center of mass frame

$$\mathrm{L} = \sum_i m_i (\mathbf{x}_i - \mathbf{x}_{CM}) \times (\mathbf{v}_i - \mathbf{v}_{CM})$$ Momentum

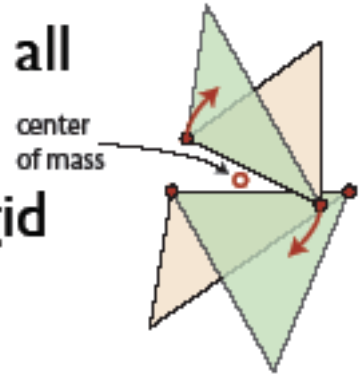
$$\mathrm{I} = \sum_i m \left(|\mathbf{x}_i - \mathbf{x}_{CM}|^2 \delta - (\mathbf{x}_i - \mathbf{x}_{CM}) \otimes (\mathbf{x}_i - \mathbf{x}_{CM}) \right)$$ Inertia tensor

$$\omega = \mathrm{I}^{-1} \mathrm{L}$$ Angular velocity

$$\mathbf{v}_i = \mathbf{v}_{CM} + \omega \times (\mathbf{x}_i - \mathbf{x}_{CM})$$ Final velocity

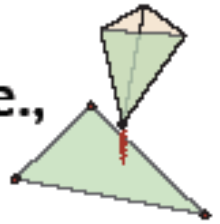
Rigid collision impact zones (2)

- Note that this is totally failsafe
- We will need to iterate, and merge impact zones as we do (e.g. until the impact zone includes all colliding particles)
- This is best used as a last resort, because rigid body cloth can be unappealing.

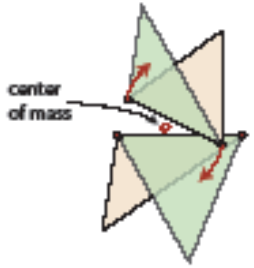


Combining methods

- So we have:
  - penalty forces - not robust, not intrusive (i.e., integrates with solver)



  - impulses - robust (esp. with iteration), intrusive - but may not converge
  - rigid impact zones - completely robust, guaranteed convergence, but very intrusive



Solution? Use all three! (Bridson et al [2002])

Combining methods (2)

Basic methodology (Bridson et al [2002]):

1. Apply penalty forces (implicitly)
2. While there are collisions left
    1. Check robustly for collisions
    2. Apply impulses
3. After several iterations of this, start grouping particles into rigid impact zones
4.

Objective: guaranteed convergence with minimal interference with cloth internal dynamics

Mastering Complexity

The Problematic

~ *High number of objects.*

~ *High number of object elements.*

- **Detecting geometrical interferences between numerous object elements efficiently needs advanced algorithms.**

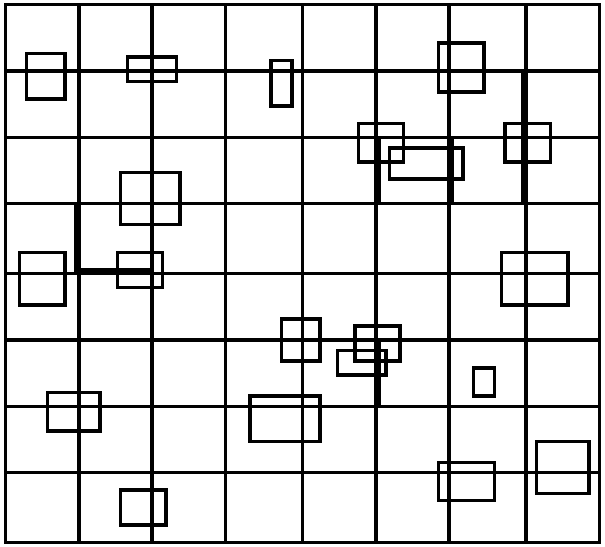
~ *Avoiding $O(n^2)$ complexity.*

Mastering Complexity
Algorithms

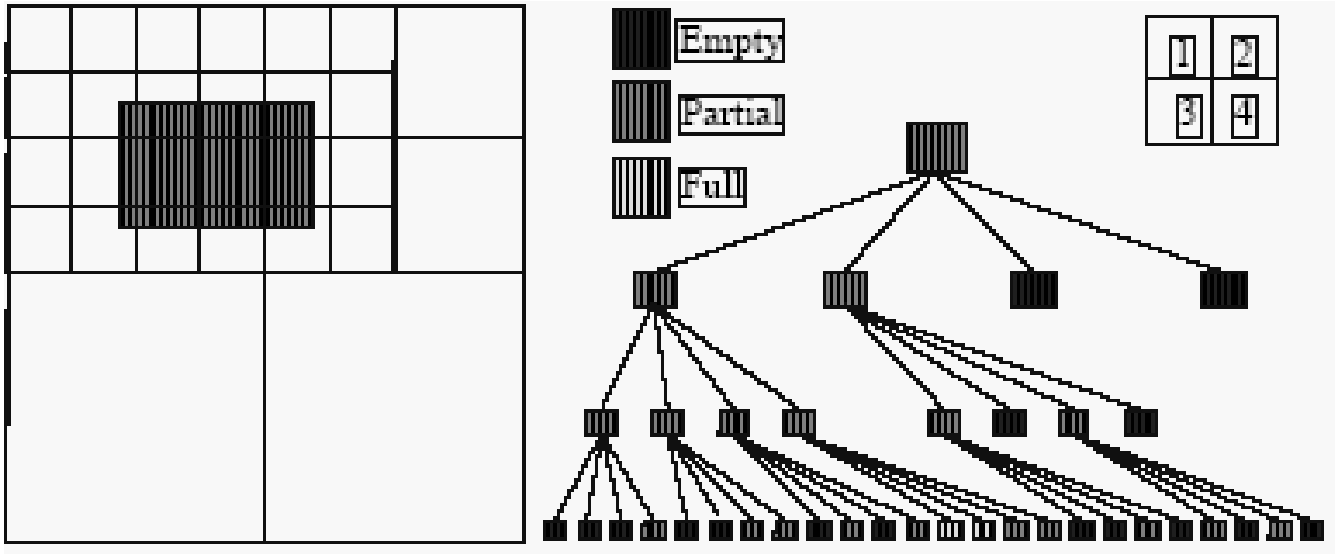
- **Space Subdivision Techniques**
  - ~ *Voxelisation*
  - ~ *Space Hierarchies (Octree)*
- **Object Subdivision Techniques**
  - ~ *Object Hierarchies*
- **Proximity Techniques**
  - ~ *Voronoi Domains*
  - ~ *Projection & Ordering*

Voxel Space Subdivision

- The space is subdivided into an array of voxels.
- Detection only between objects sharing common voxels.

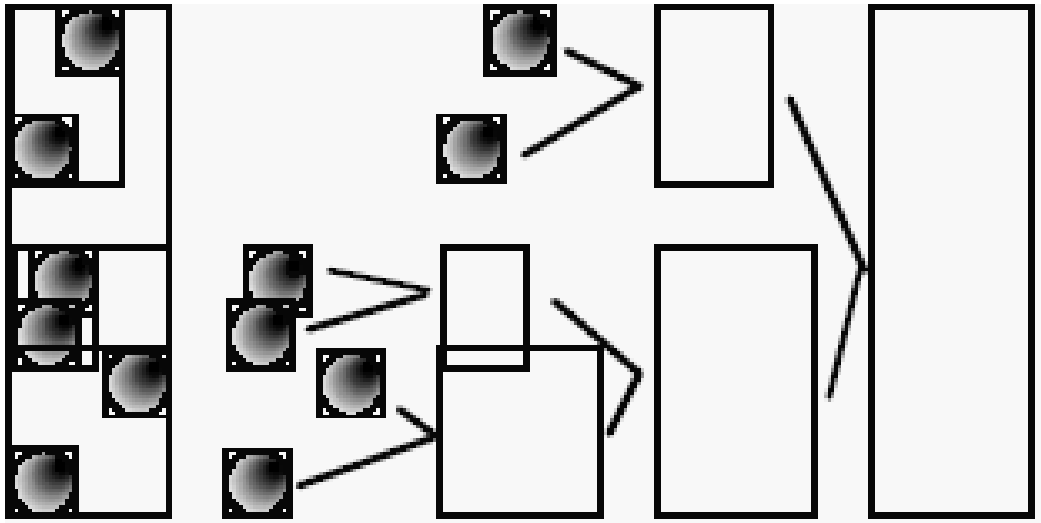


Octree Space Subdivision



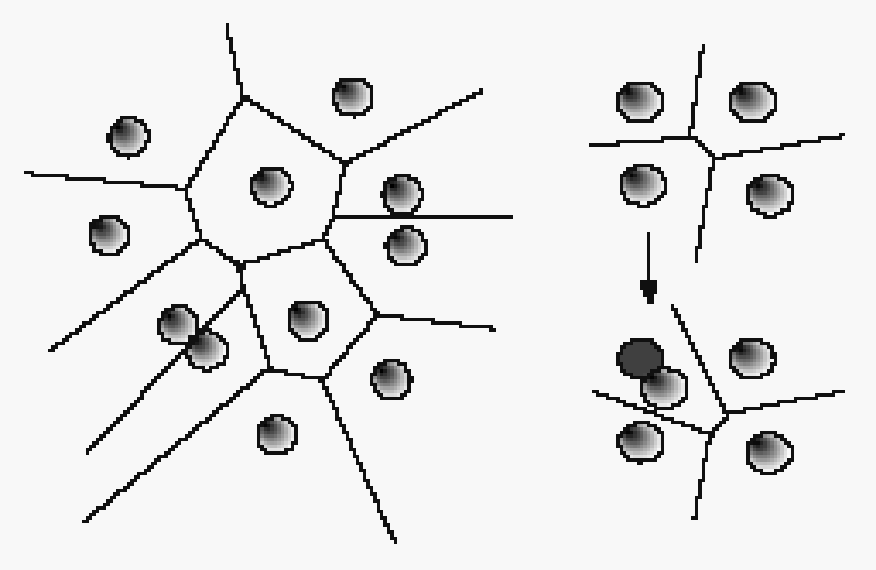
- The space is recursively subdivided into a structure representing the shape of each objects.
- Detection by exploring the stucture.

Bounding-Box Hierarchy



- The objects are grouped in a hierarchy according to proximity rules.
- Detection by exploring bounding-box intersections in the hierarchy.

Proximity Tracking



- **Keeping incremental information on the objects neighborhood relations.**
- **Ex: Voronoy domains, convex hull,...**

Incremental Techniques

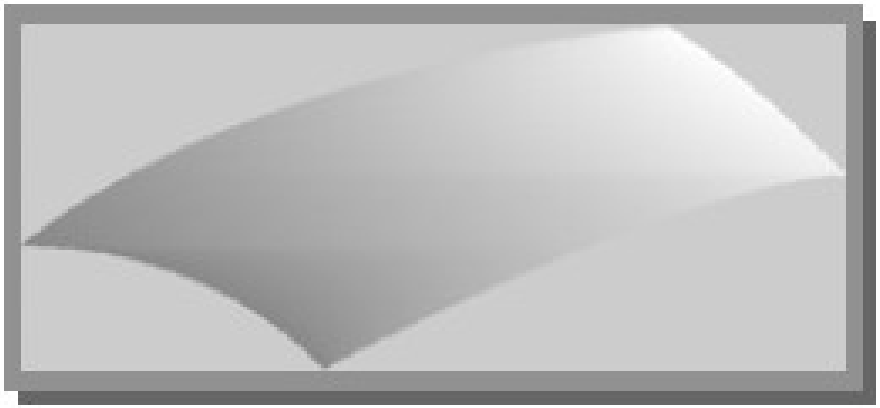
- For animations, updating the collisions as the objects move between each frame.
- A good way to speed up computation for frame-based animations.
- Difficulty to maintain the consistency of all collisions.

Collisions & Self-Collisions

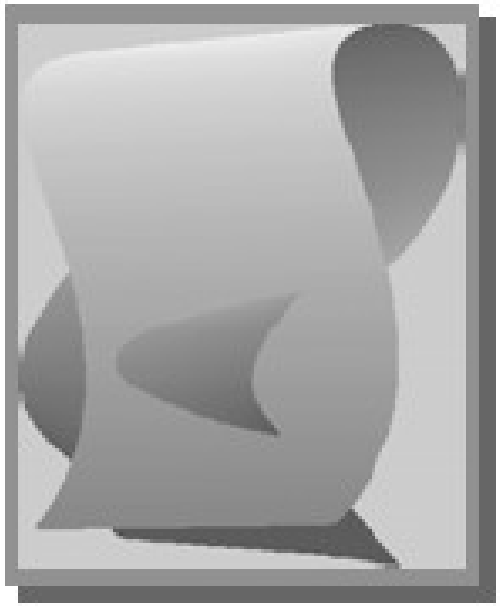
- Self-Collisions: Detecting collisions between the primitives of one object.
- The adjacency problem: Adjacent primitives are "colliding" according to usual detection algorithms.
- How to maintain the algorithm efficiency?

Efficient Self-Collision Detection

- **Curvature Optimization**
  - ~ *No self-collisions within an almost flat surface.*



  - ~ *Self-collision detection only within surface regions that are curved enough to contain some.*



Efficient Self-Collision Detection

- **Curvature Optimization**
  - *~ Combining bounding-box and curvature tests.*