CS277 - Experimental Haptics
Lecture 14

Six-DOF Haptic Rendering II

Outline

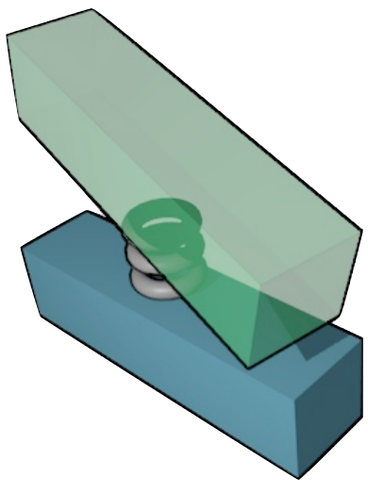
- Generic model for proxy-based rendering
- Study of three significant 6-DOF haptic rendering algorithms
  - McNeely, Puterbaugh, & Troy 1999
  - Otaduy & Lin 2005
  - Ortega, Redon, & Coquillart 2007

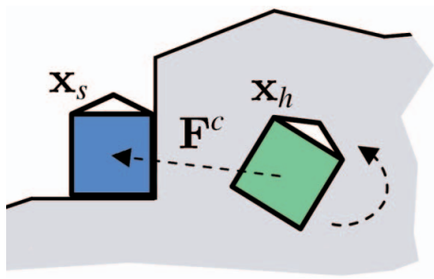
Proxy-Based Rendering

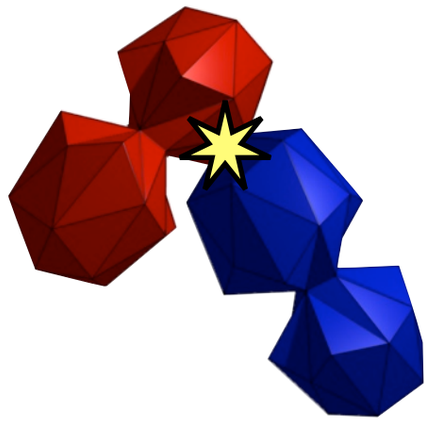
position + orientation

force + torque

position + orientation







Device Controller

Virtual Coupling

Proxy Solver

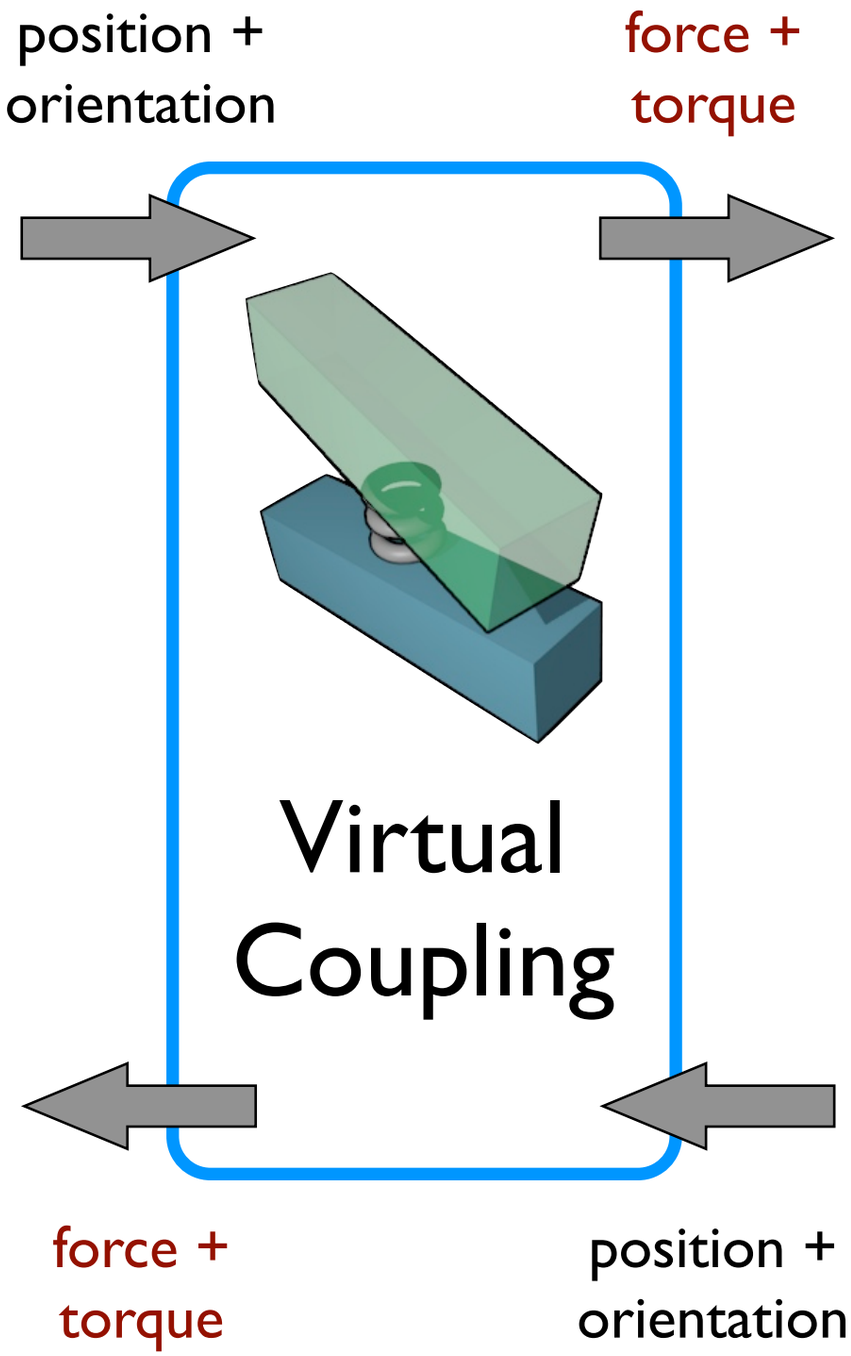
Collision Detector

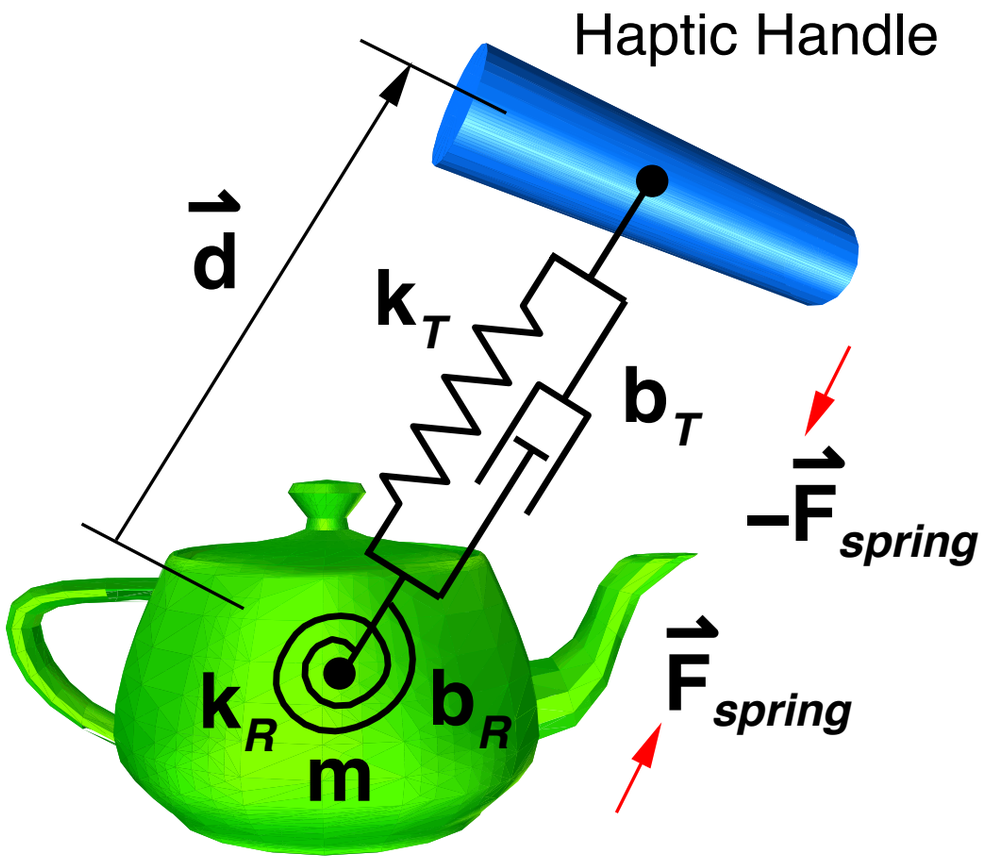
force + torque

position + orientation

contact points + normals or forces

Virtual Coupling





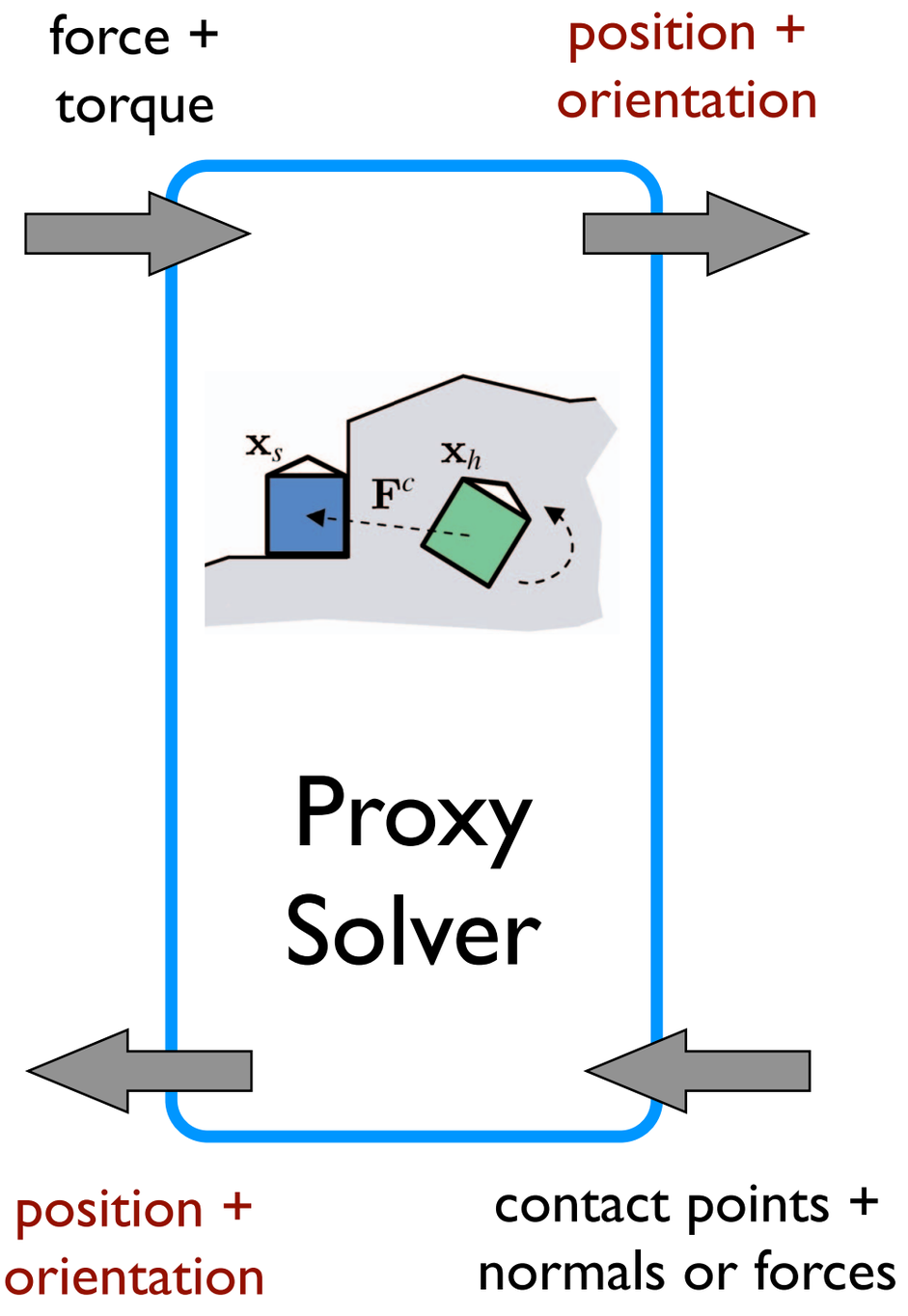
Dynamic Object

$$\mathbf{F}_c = k_T\mathbf{x} + b_T\mathbf{v}$$

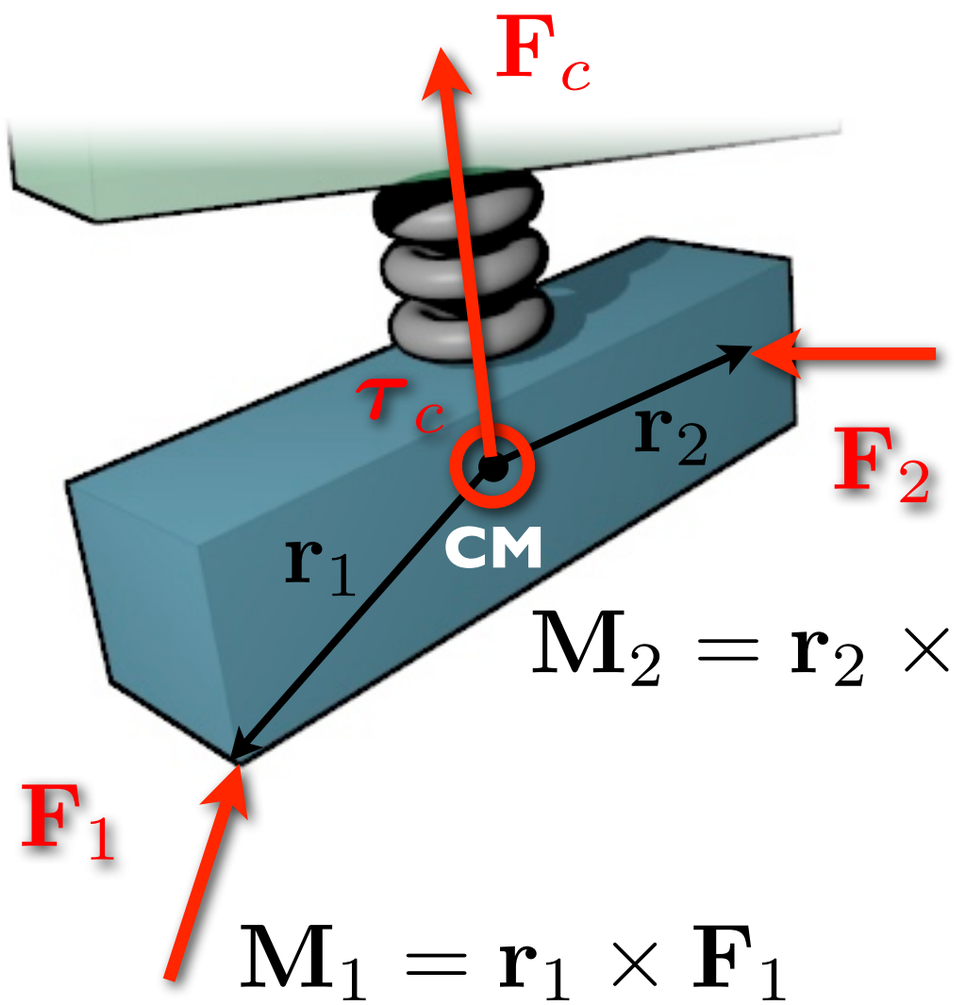
$$\boldsymbol{\tau}_c = k_R\boldsymbol{\theta} + b_R\boldsymbol{\omega}$$

Proxy Solver

- Goal is to compute position, orientation of the proxy, given
  - Applied force, torque from virtual coupling
  - Contact forces or constraints



Dynamic Proxy Simulation



$$\mathbf{M}_2 = \mathbf{r}_2 \times \mathbf{F}_2$$

$$\mathbf{M}_1 = \mathbf{r}_1 \times \mathbf{F}_1$$

$$\mathbf{F} = \mathbf{F}_c + \sum_i \mathbf{F}_i$$

$$= m\mathbf{a}$$

$$\boldsymbol{\tau} = \boldsymbol{\tau}_c + \sum_i \mathbf{M}_i$$

$$= \mathbf{I}_{\mathrm{CM}}\boldsymbol{\alpha} + \boldsymbol{\omega} \times \mathbf{I}_{\mathrm{CM}}\boldsymbol{\omega}$$

Time Integration

- Explicit Euler finite difference equation:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \, \dot{\mathbf{y}}_n$$

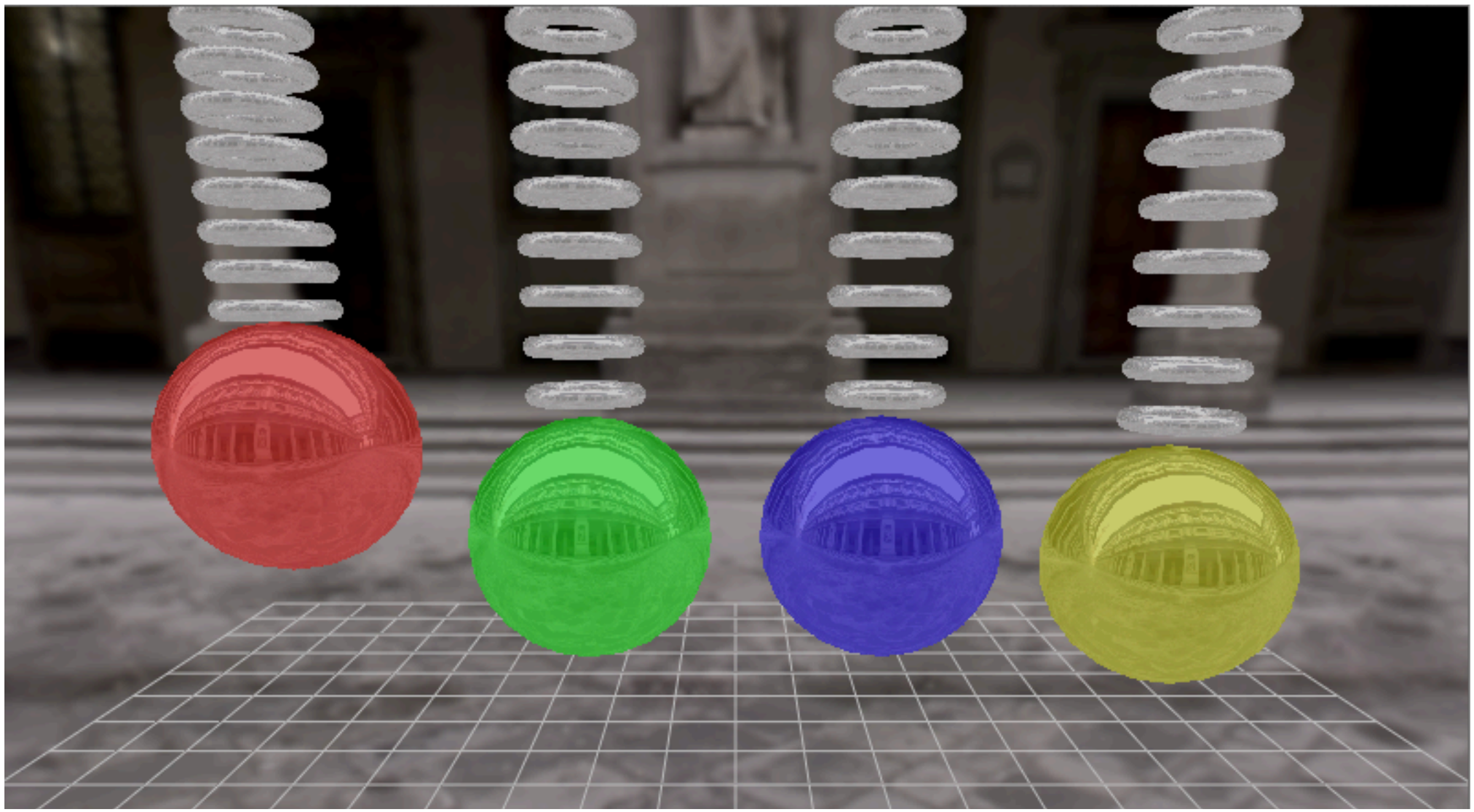
- with the state variable

$$\mathbf{y}(t) = \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\theta} \\ \mathbf{P} = m\mathbf{v} \\ \mathbf{L} = \mathbf{I}\,\boldsymbol{\omega} \end{pmatrix} \qquad \dot{\mathbf{y}}(t) = \begin{pmatrix} \dot{\mathbf{x}} \\ \dot{\boldsymbol{\theta}} \\ \dot{\mathbf{P}} \\ \dot{\mathbf{L}} \end{pmatrix} = \begin{pmatrix} \frac{1}{m}\mathbf{P} \\ \boldsymbol{\omega} \\ \mathbf{F} \\ \boldsymbol{\tau} \end{pmatrix}$$

Comments on Virtual Coupling

- The spring-damper coupling filters high frequency force variations (or discontinuities) applied to the virtual tool
  - Can be a good or a bad thing...
- A stiffer coupling spring allows the operator to feel more of the contact forces
- However, stiff coupling springs can lead to instabilities in free space (why?)

Limitations of Time Integration



What happens with a harmonic oscillator?

Implicit Time Integration

- Implicit Euler finite difference equation:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \, \dot{\mathbf{y}}_{n+1}$$

- Using first order Taylor approximation:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \left[\dot{\mathbf{y}}_n + \frac{\partial \dot{\mathbf{y}}}{\partial \mathbf{y}} \left(\mathbf{y}_{n+1} - \mathbf{y}_n \right) \right]$$

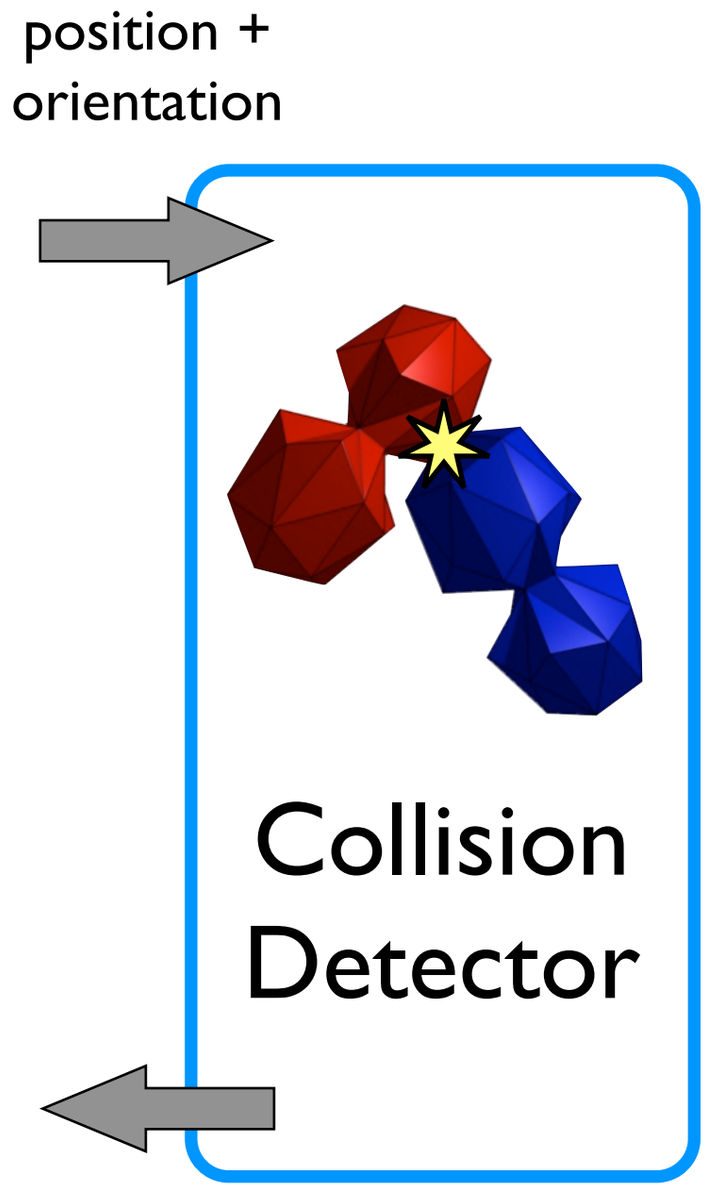
$$\left(I - \Delta t \frac{\partial \dot{\mathbf{y}}}{\partial \mathbf{y}} \right) \left(\mathbf{y}_{n+1} - \mathbf{y}_n \right) = \Delta t \, \dot{\mathbf{y}}_n$$

Summary

- Implicit Euler integration is much more stable than explicit integration
- Undershoots rather than overshoots
- Requires computing the derivatives (Jacobian) of the force vector with respect to state variables
- Allows use of stronger penalty forces, stiffer virtual coupling

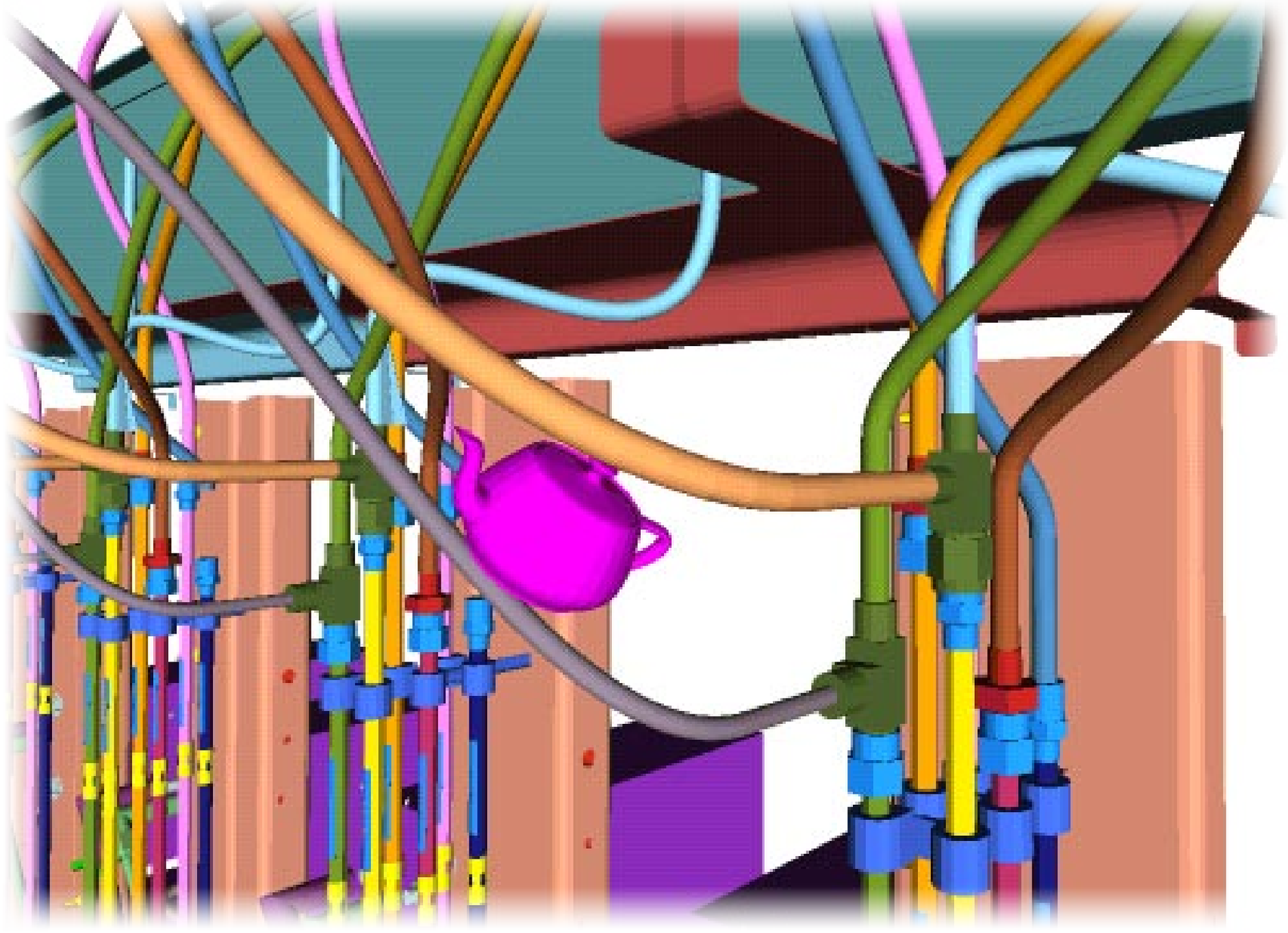
Collision Detection

- Mesh-mesh collision detection was the thoughest in the book!
  - Dynamic proxy solver also requires penetration depth
  - Poses the greatest challenge to 6-DOF haptic rendering...



Collision Detection Approaches

- Recall 1000 Hz update rate requirement for haptic rendering
  - How can we possibly get it fast enough?
- Many approaches, but we will examine two:
  - Simplify or modify geometric representation
  - Run collision detection at a lower rate if needed

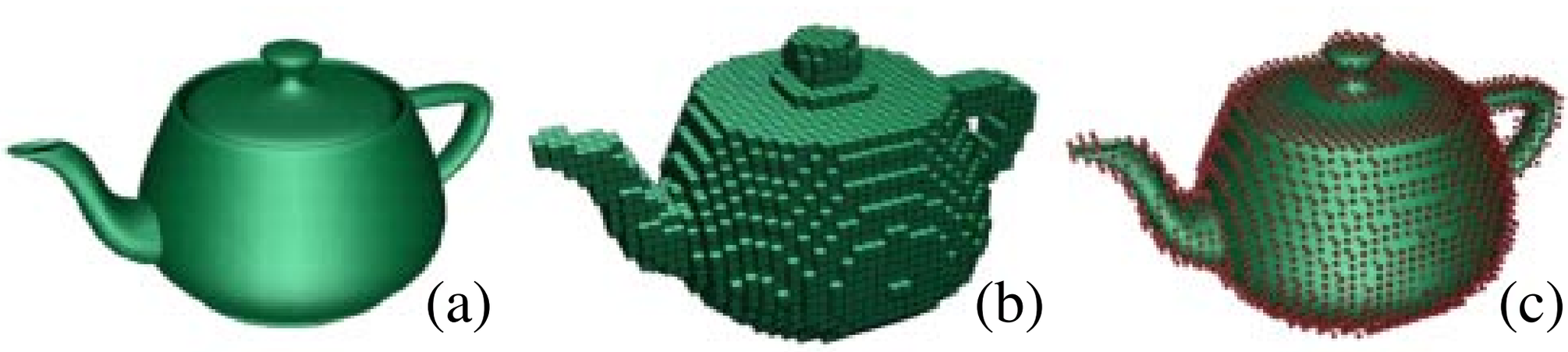


Voxmap PointShell™

[From W. A. McNeely *et al.*, *Proc. SIGGRAPH*, 1999.]

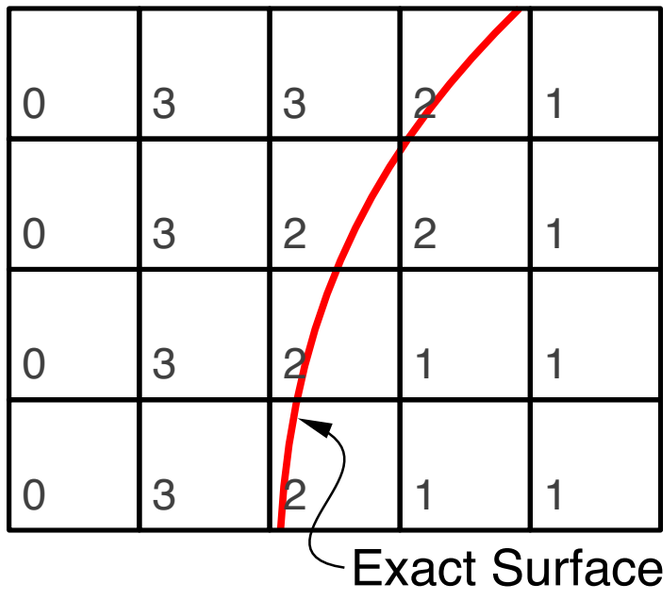
Voxelized Geometry

- Point-voxel collision tests are fast
- Idea: Voxelize all the geometry

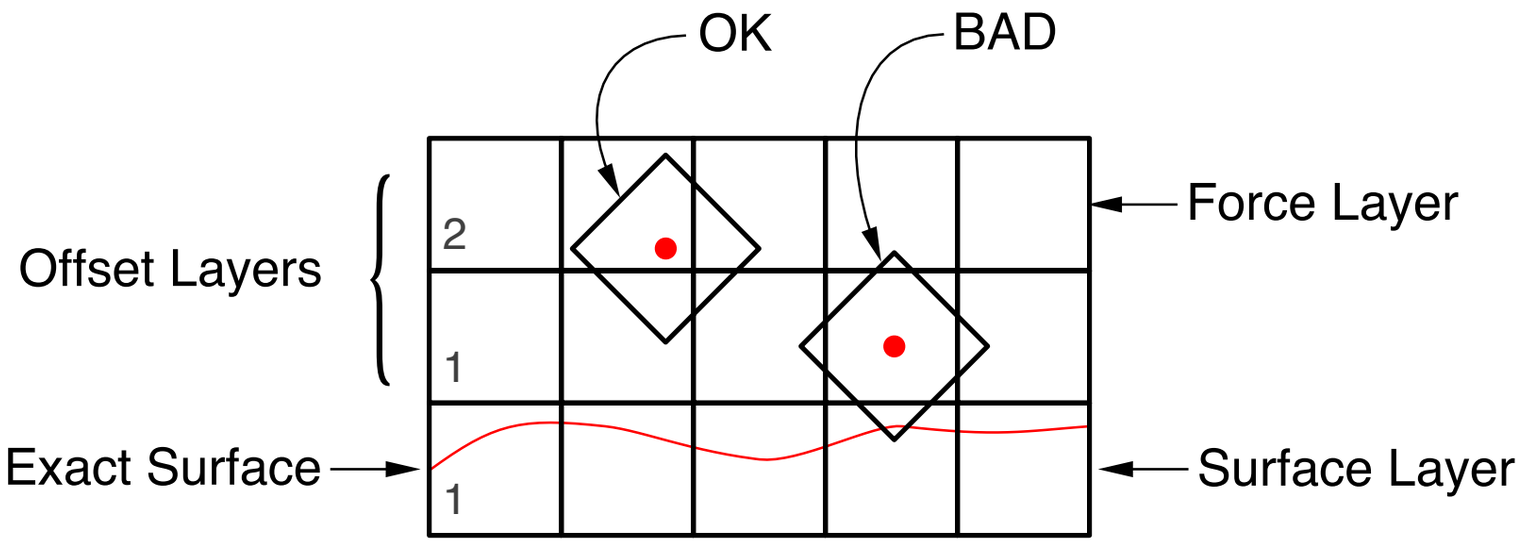


Polygonal model, "Voxmap", and "PointShell" representations of a teapot

Computing the Voxmap

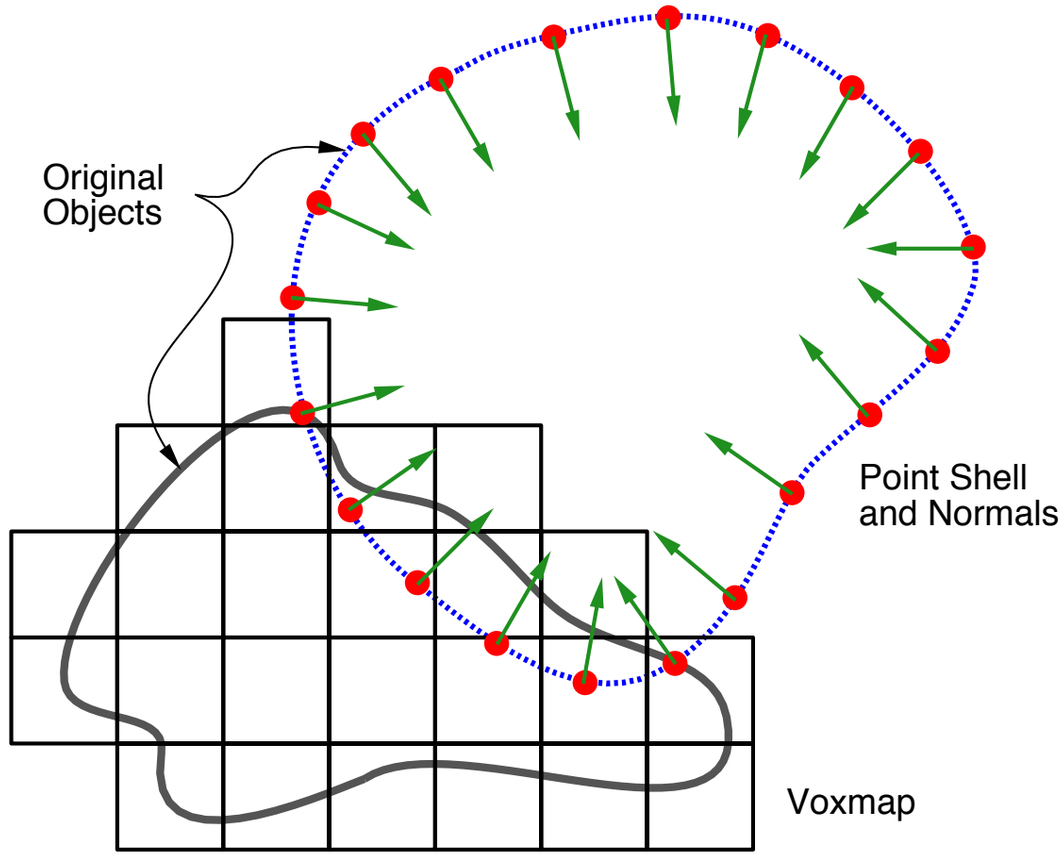


0 = free space
1 = interior
2 = surface
3 = proximity



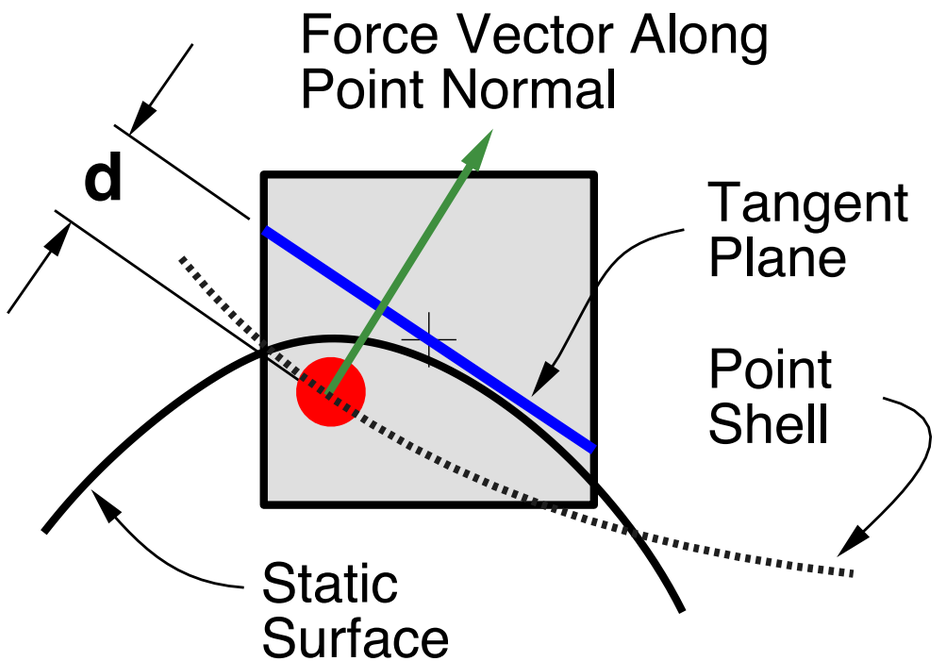
Computing the PointShell

- Approximate with centers of surface voxels
- Add inward-pointing surface normals



Collision Response

- Virtual tool is dynamically simulated, so we can apply forces to it
- Use tangent-plane force model and Hooke's Law



$$\mathbf{F} = K_{\mathrm{ff}}\,\mathbf{d}$$

Collision Response

- Net force on virtual tool is sum of penalty forces from point-voxel intersections
- Problem: What happens with multiple, simultaneous contacts?
- Solution:

$$\mathbf{F}_{net} = \begin{cases} \mathbf{F}_{total}, & N < 10 \\ \dfrac{\mathbf{F}_{total}}{\frac{1}{10}N}, & N \geq 10 \end{cases}$$

Collision Response

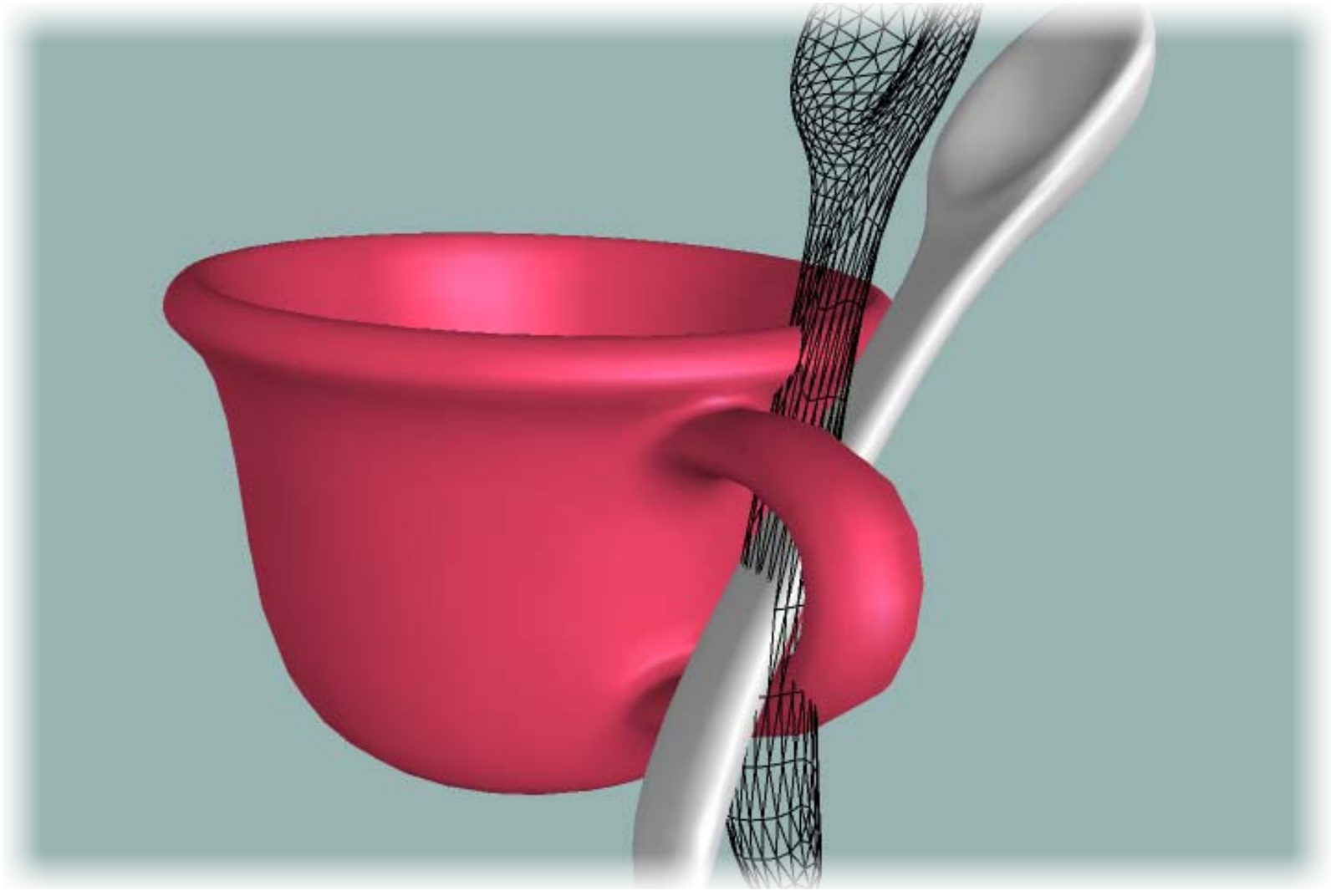
- Another problem: Can a point-voxel intersection occur on an interior voxel?
- Solution: Apply a "braking viscosity" force at the proximity voxels.

$$\mathbf{F} = \begin{cases} -b\mathbf{v}(-\mathbf{n}\cdot\mathbf{v}), & \mathbf{n}\cdot\mathbf{v} < 0 \\ 0, & \mathbf{n}\cdot\mathbf{v} \geq 0 \end{cases}$$

- Large point velocities are still a problem...

Summary

- This rendering method can provide a constant 1000 Hz update rate that includes collision detection (on a 350 MHz PC!)
- Resolution is limited by voxel size, and finer voxel grids use cubically more memory
- Many problems with *ad-hoc* solutions...
- Still one of the first highly successful 6-DoF rendering techniques

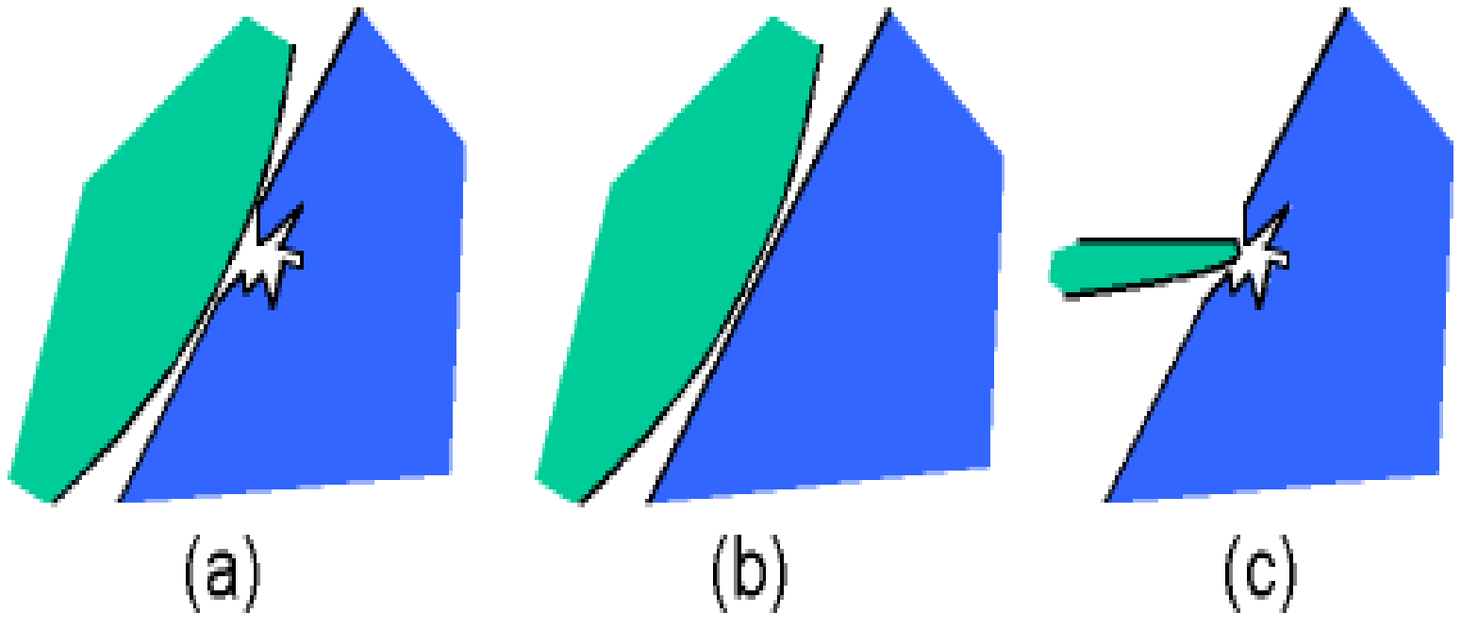


Stable & Responsive Manipulation

[From M. A. Otaduy & M. Lin, *Proc. IEEE World Haptics Conference*, 2005.]

Sensation-Preserving Simplification

- Finding all contact points between detailed polygonal models can be really expensive!
- Take advantage of perceptive limitations



[From M. A. Otaduy & M. Lin, *ACM Transactions on Graphics* 22(3), 2003.]

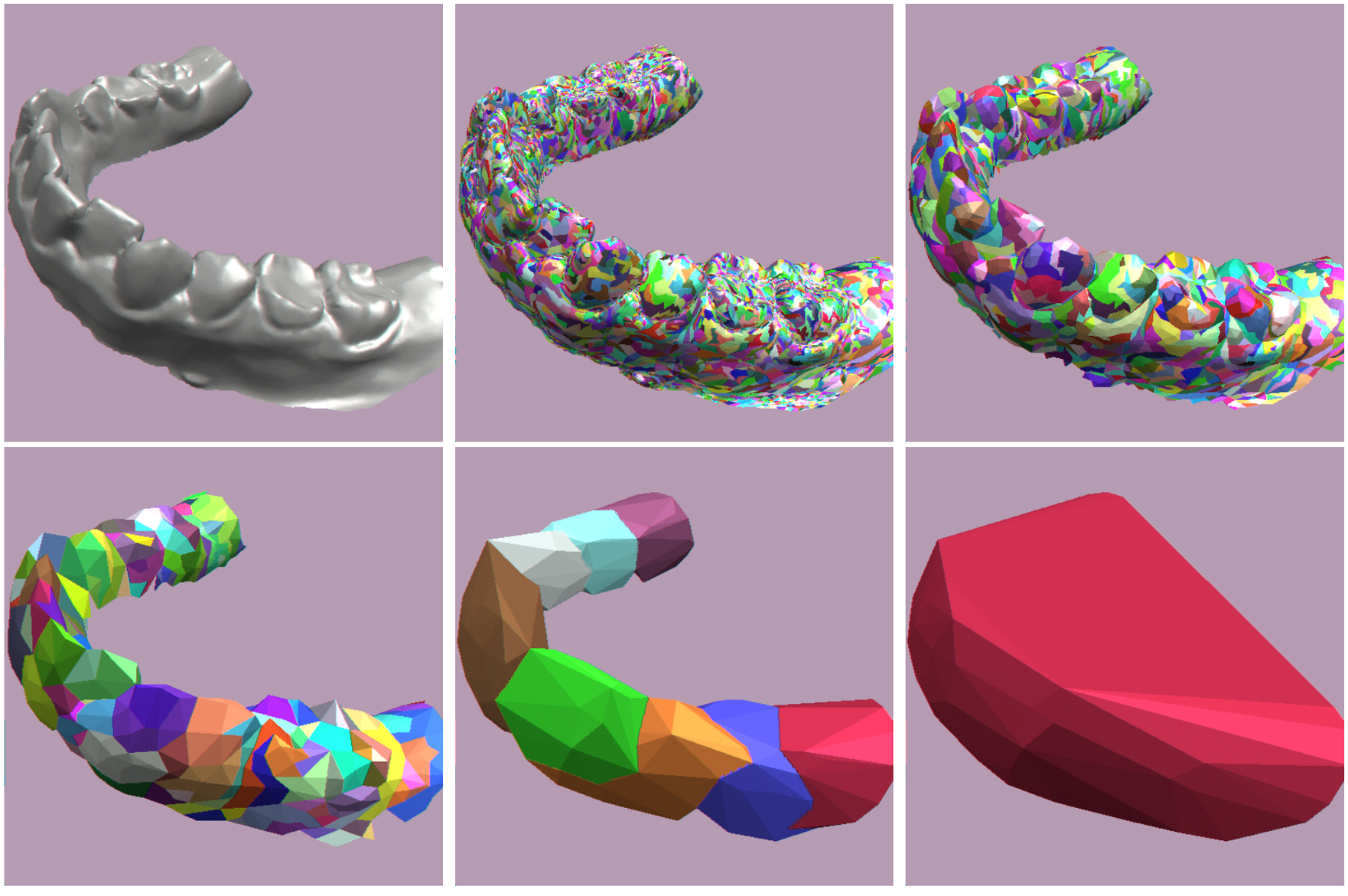
Collision Detection Strategy

- Create multi-resolution hierarchies of the meshes (levels of detail)
- Accelerate collision detection with BVH
- Only refine search where details are perceptible!

Constructing the Hierarchy

- Perform full convex decomposition on original mesh
- Then start merging pieces in priority of highest resolution (most detail)
  - Perform filtered edge collapse decimation to simplify components while preserving convexity
- Mark as level of detail whenever number of components is halved

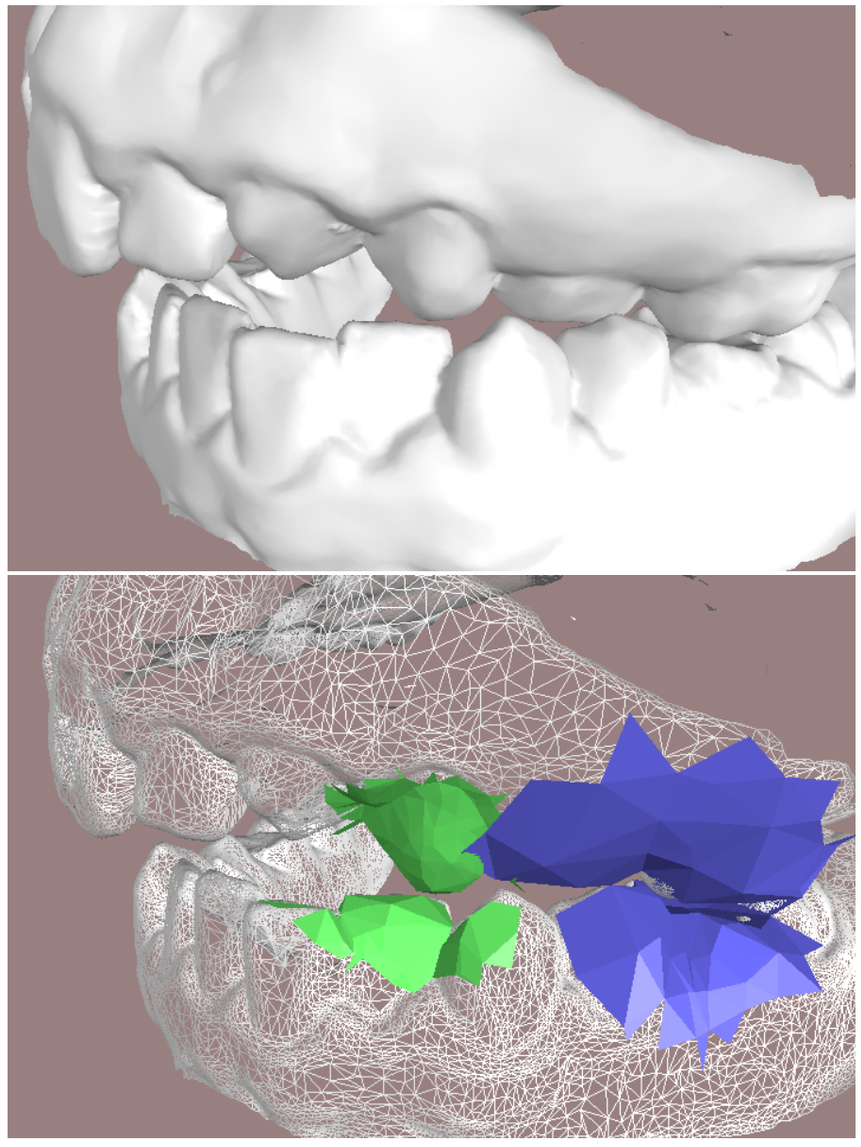
Levels of Detail



LOD hierarchy doubles as bounding volume hierarchy!

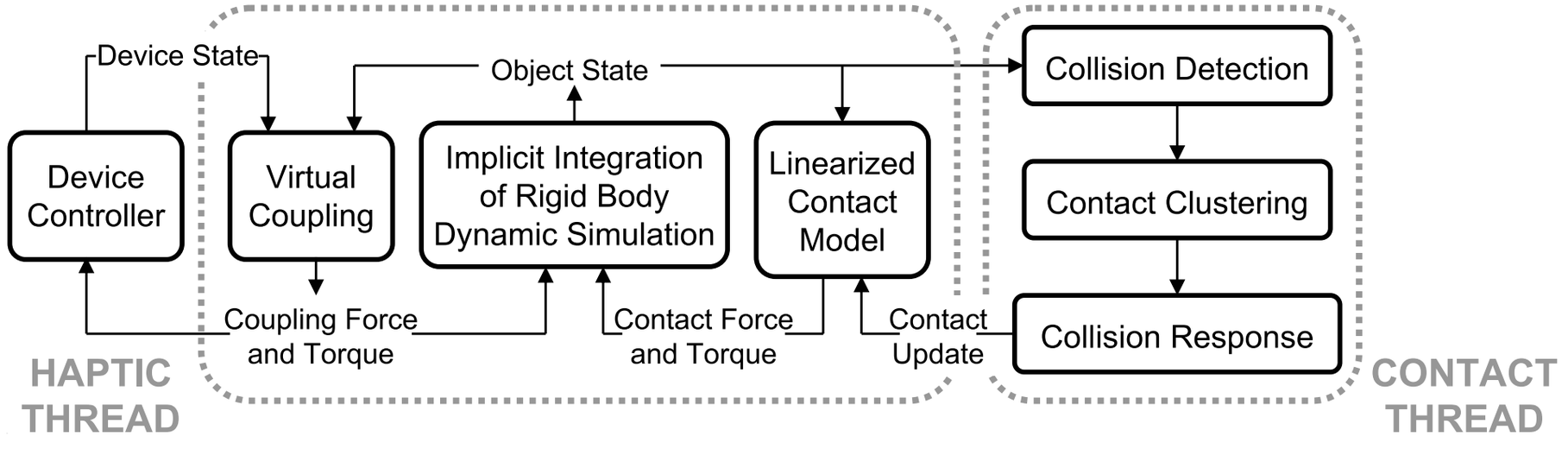
Collision Detection

- Traverse BVH as usual for collision detection, except...
  - Only recurse when the higher resolution is deemed percitible
  - Otherwise, use approximate geometry at the current LOD



Variable Rate Collision Detection

- Still cannot guarantee speed!
- As low as 100 Hz with 40k triangles



- Haptic thread can render forces at 1000 Hz while contact thread runs at a variable rate

Collision Response

- Remember problem with multiple contacts?
- K-means clustering is used to group contacts into representative points
  - Each cluster described by point and normal
  - Viscoelastic penalty-based force applied to the virtual tool for each contact:

$$\mathbf{F}_p = -kN(\mathbf{x} + R\mathbf{r} - \mathbf{p}_0) - kd\mathbf{n} - bN(\mathbf{v} + \omega \times \mathbf{r})$$

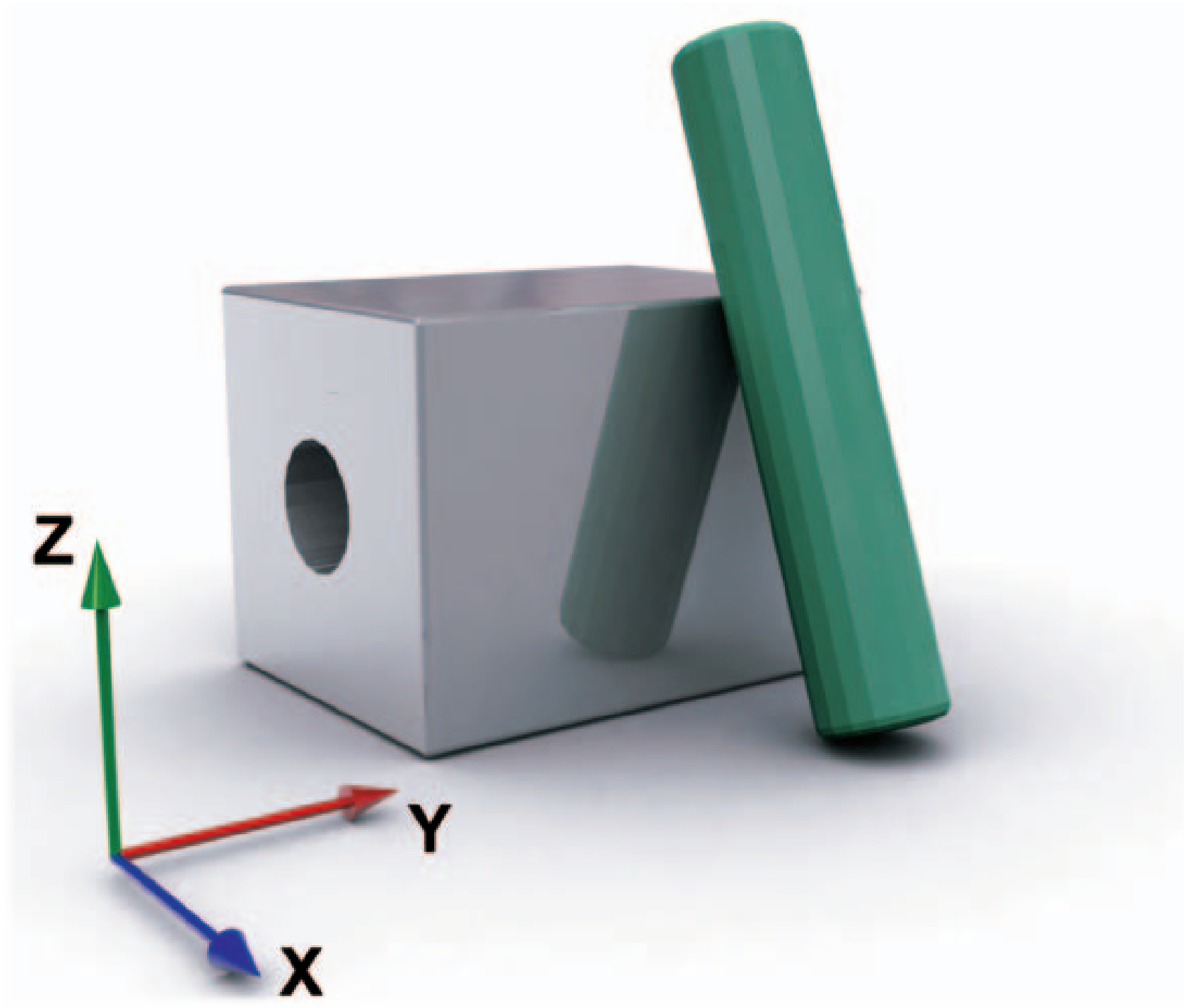
$$\mathbf{T}_p = (R\mathbf{r}) \times \mathbf{F}_p$$

Summary

- Adaptive simplification = fast collision detection between complex models
- Fidelity of haptic perception is preserved
- Variable rate collision detection allows high force and haptic update rate
- Contact clustering mitigates force discontinuities and escalating stiffness for multi-point contact

Dynamic Proxy Limitations

- Did we solve the interpenetration problem?
  - Nonpenetration enforced by high contact stiffness, can cause instability
- Are there other limitations?

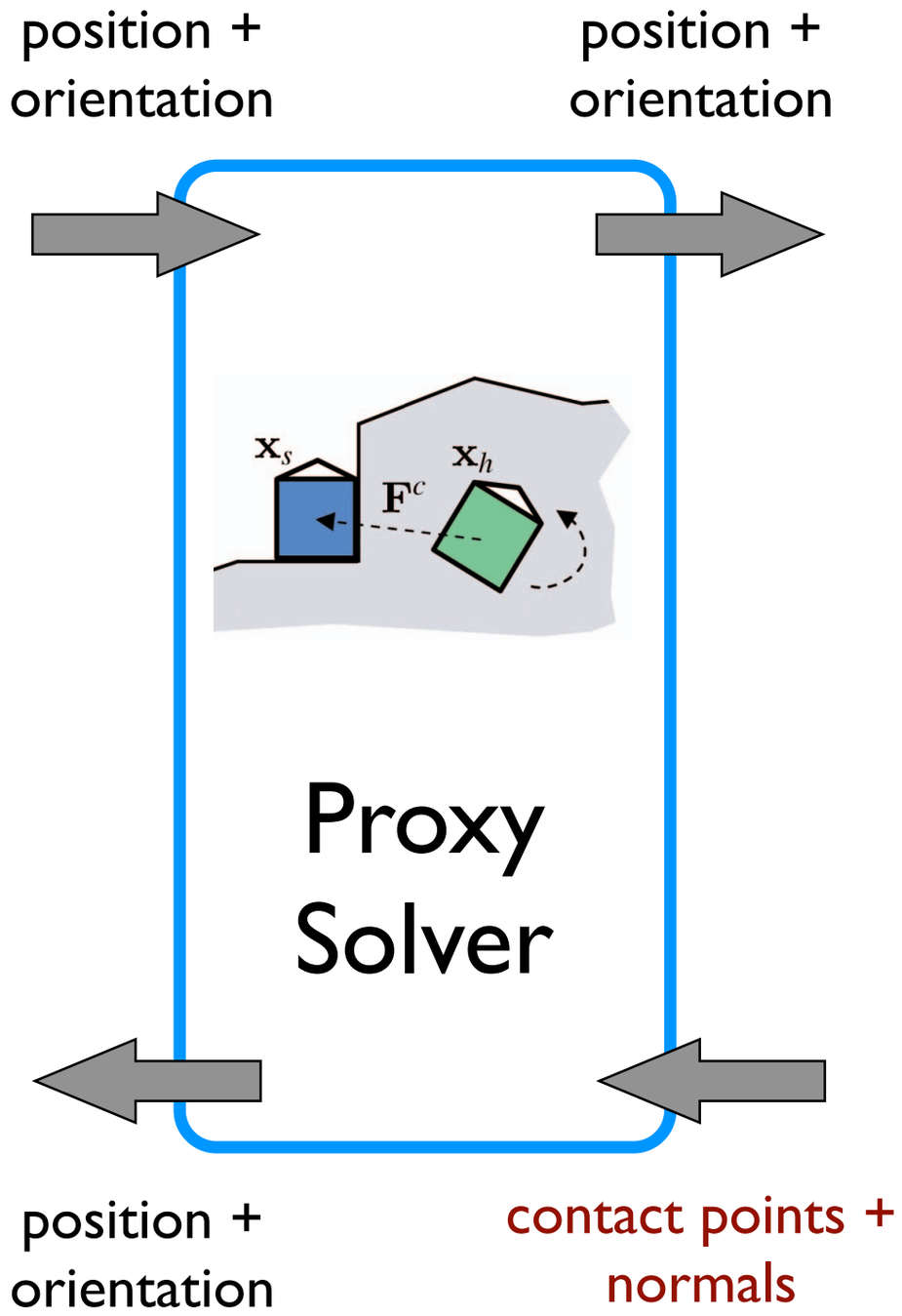


6-DOF God-Object

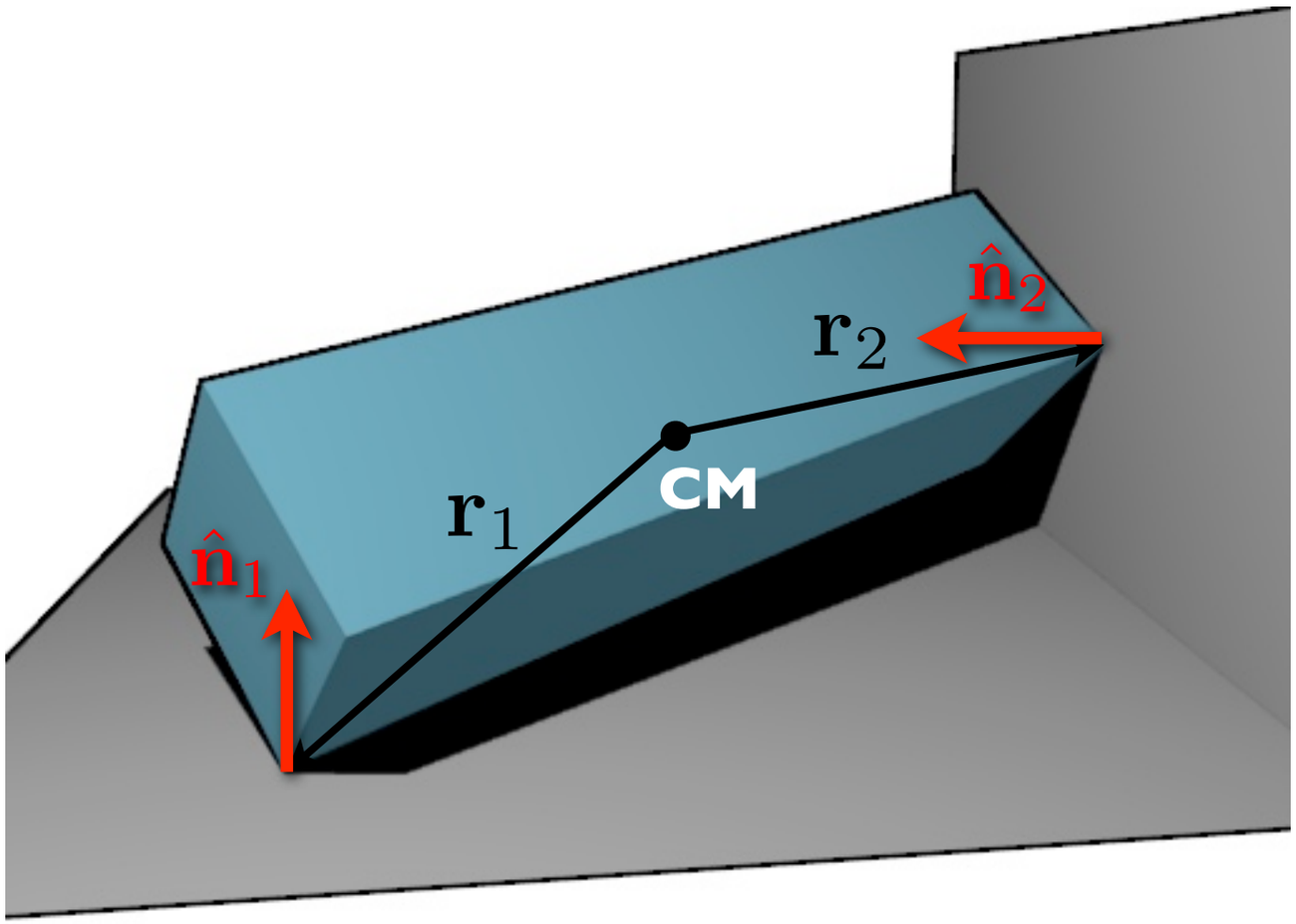
[From M. Ortega *et al.*, *IEEE Trans. Visualization and Computer Graphics* 13(3), 2005.]

Contraint-Based Proxy Solver

- Direct analogue of 3-DOF god-object
- Uses contact positions and normals only – presumes objects *do not interpenetrate*
- Computes a trajectory that does not violate contact constraints



Contact Constraints



$$\mathbf{a}_{\mathrm{CM}} \cdot \hat{\mathbf{n}}_k + \boldsymbol{\alpha} \cdot (\mathbf{r}_k \times \hat{\mathbf{n}}_k) \geq 0$$

How do we use these to determine the motion of the proxy?

Gauss' Principle of Least Constraint

- Gauss defined a kinetic distance quantity as

$$\begin{aligned}\mathcal{G}(\mathbf{a}) &= \tfrac{1}{2}\left(\mathbf{a}-\mathbf{a}^u\right)^T \mathbf{M}\left(\mathbf{a}-\mathbf{a}^u\right) \\ &= \tfrac{1}{2}||\mathbf{a}-\mathbf{a}^u||^2_{\mathbf{M}}\end{aligned}$$

- Then the motion of the constrained body is one that minimizes the kinetic distance

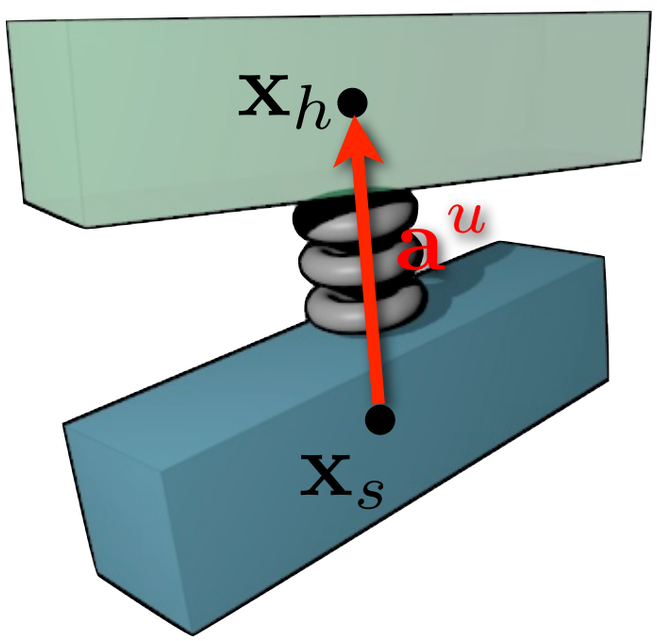
$$\mathbf{a}^c = \arg\min_{\mathbf{a}} \mathcal{G}(\mathbf{a})$$

Quasi-Static Proxy Update

- Write the generalized accelerations as

$$\mathbf{a} = (\mathbf{a}_{\mathrm{CM}}, \boldsymbol{\alpha})^T$$

- Obtain unconstrained acceleration from virtual coupling spring (proxy displacement)



$$\mathbf{a}^u = \frac{1}{2}(\mathbf{x}_h - \mathbf{x}_s)$$

Optimization Problem

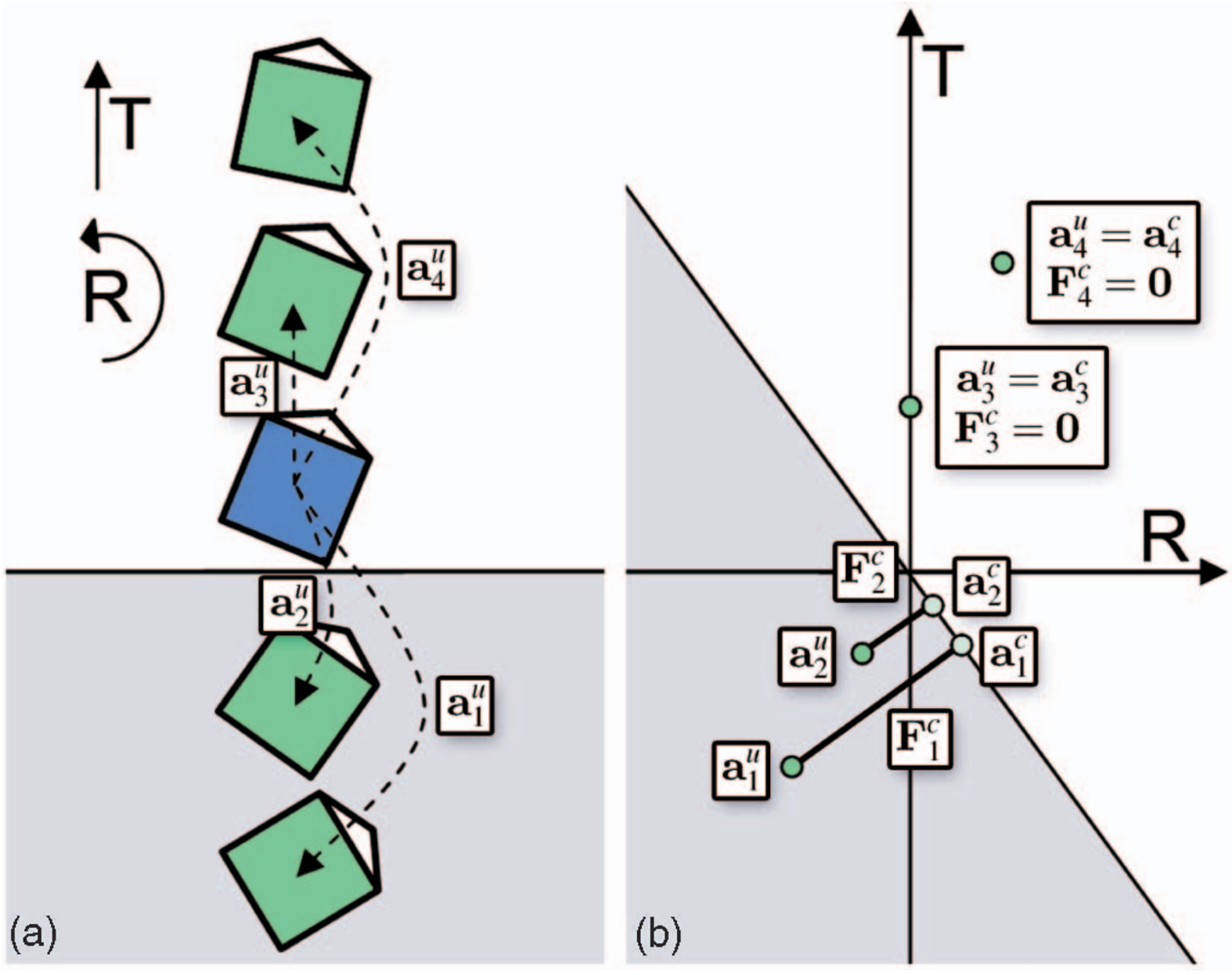
- Solve the quadratic programming problem

$$\begin{aligned} \text{minimize} \quad & \mathcal{G}(\mathbf{a}) = \tfrac{1}{2}\left(\mathbf{a}-\mathbf{a}^u\right)^T \mathbf{M}\left(\mathbf{a}-\mathbf{a}^u\right) \\ \text{subject to} \quad & \mathbf{a}_{\mathrm{CM}} \cdot \hat{\mathbf{n}}_k + \boldsymbol{\alpha} \cdot \left(\mathbf{r}_k \times \hat{\mathbf{n}}_k\right) \geq 0 \end{aligned}$$

- Then update the proxy with the constrained motion (possibly with additional collision query)

$$\mathbf{x}_s' = \mathbf{x}_s + \tfrac{1}{2}\mathbf{a}^c$$

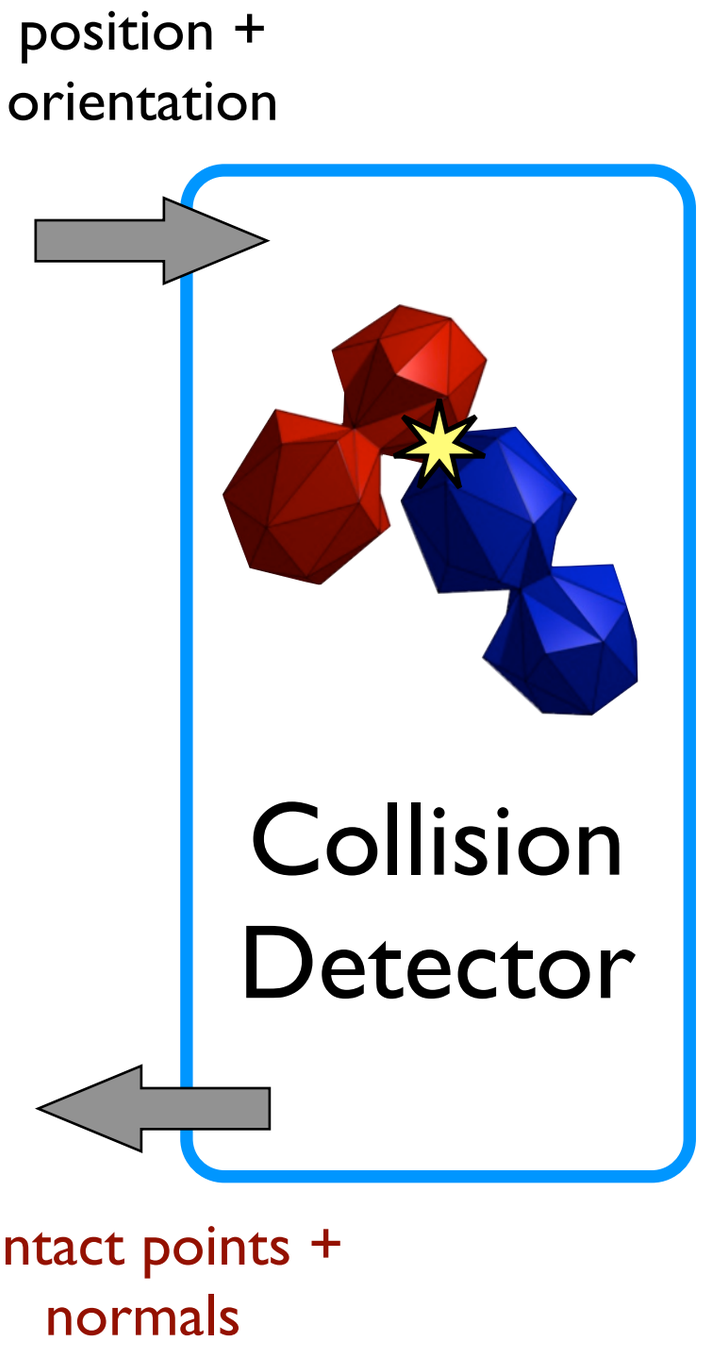
Constrained Motion



Continuous Collision Detection

- Constraint-based proxy solver requires non-interpenetrating contacts

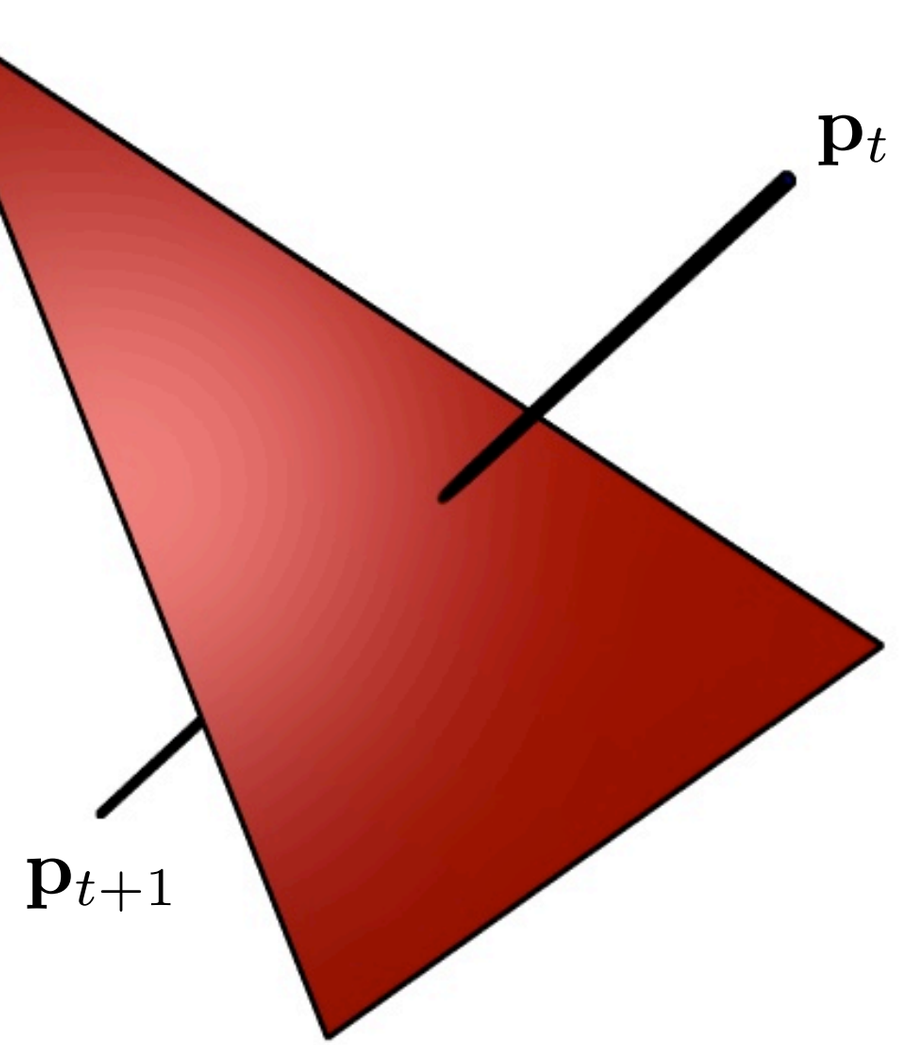
- Continuous collision detection is one method to find contacts and normals while enforcing non-interpenetration



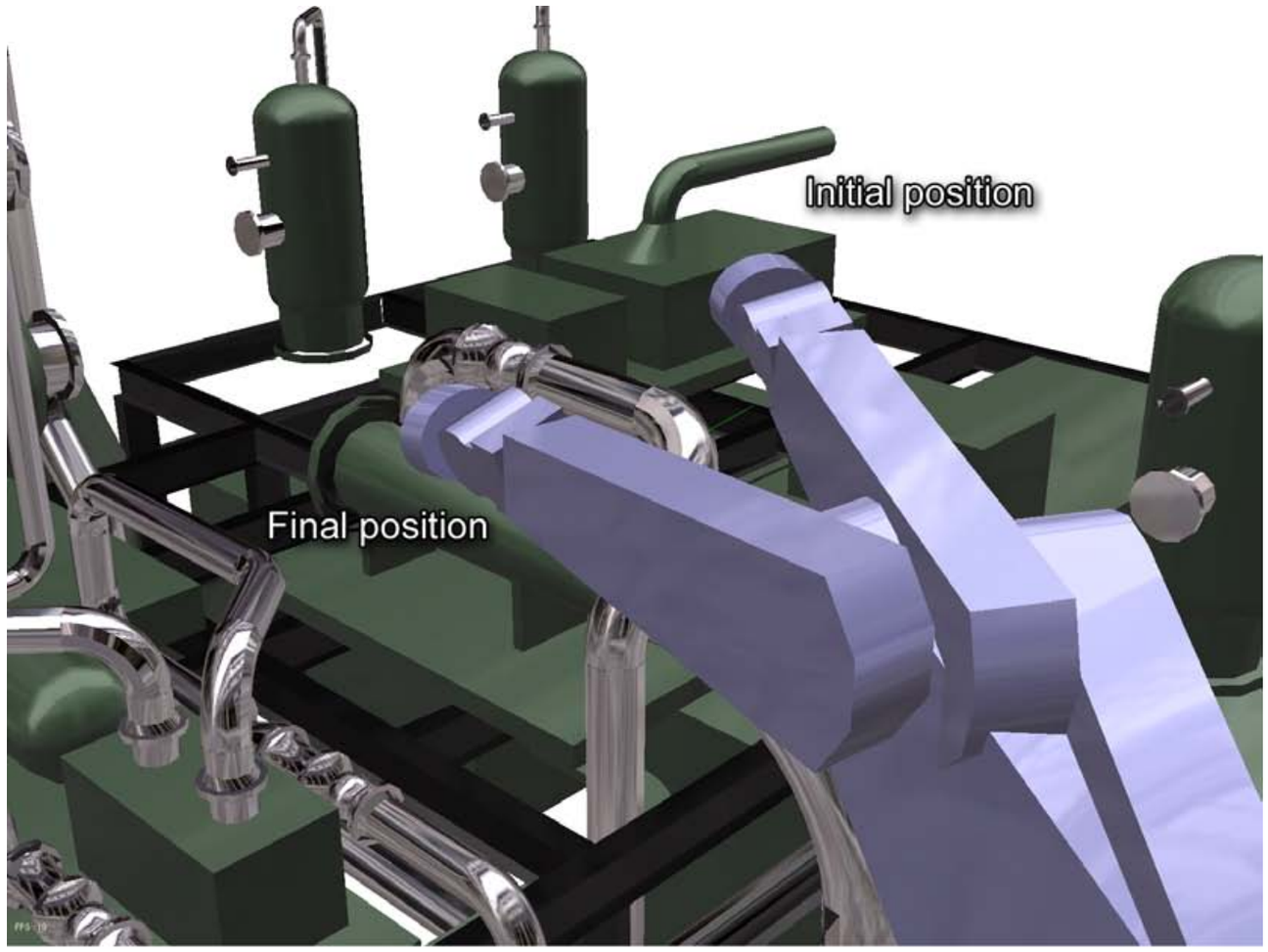
Recall 3-DOF God-Object

- The segment-triangle intersection test is a form of continuous collision detection

- The god-object is infinitely small, so it will always miss polygonal geometry unless CCD is used!



Non-Point Proxies



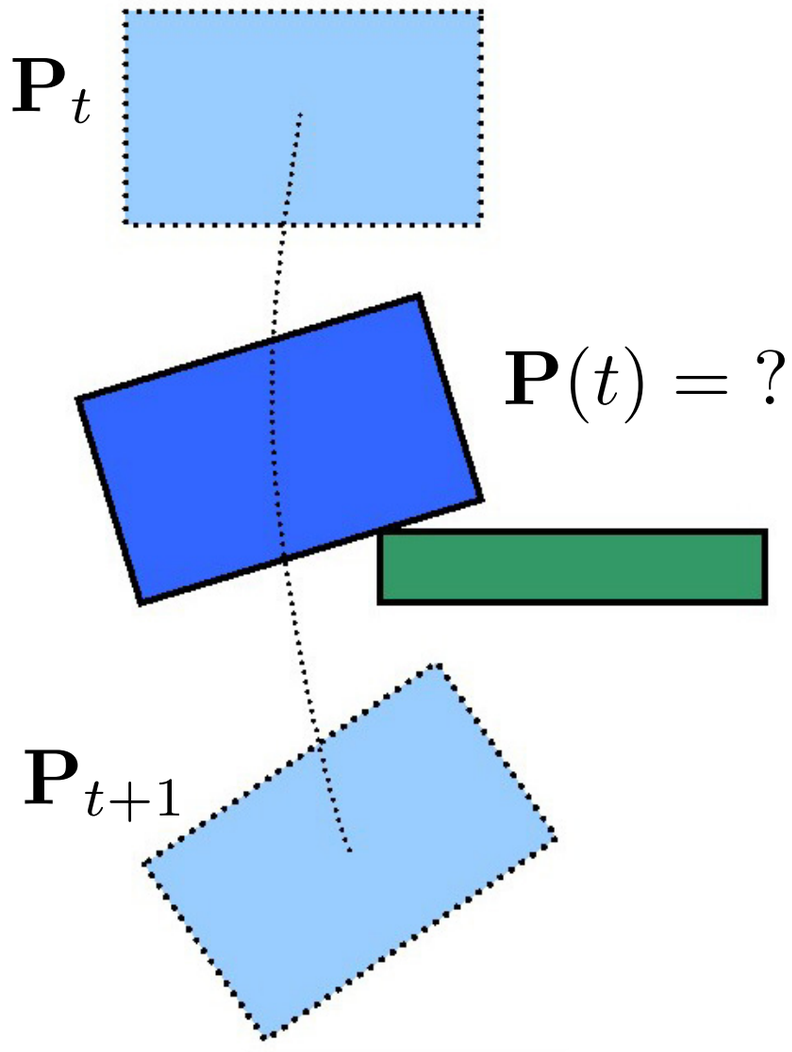
How do we generalize to a polyhedral avatar?

[From S. Redon *et al.*, *Transactions of the ASME* 5, 2005.]

Arbitrary In-Between Motions

- We only know the position of the avatar at discrete time steps

- We may assume an arbitrary object motion subject to:
  - Interpolation
  - Continuity
  - Rigidity

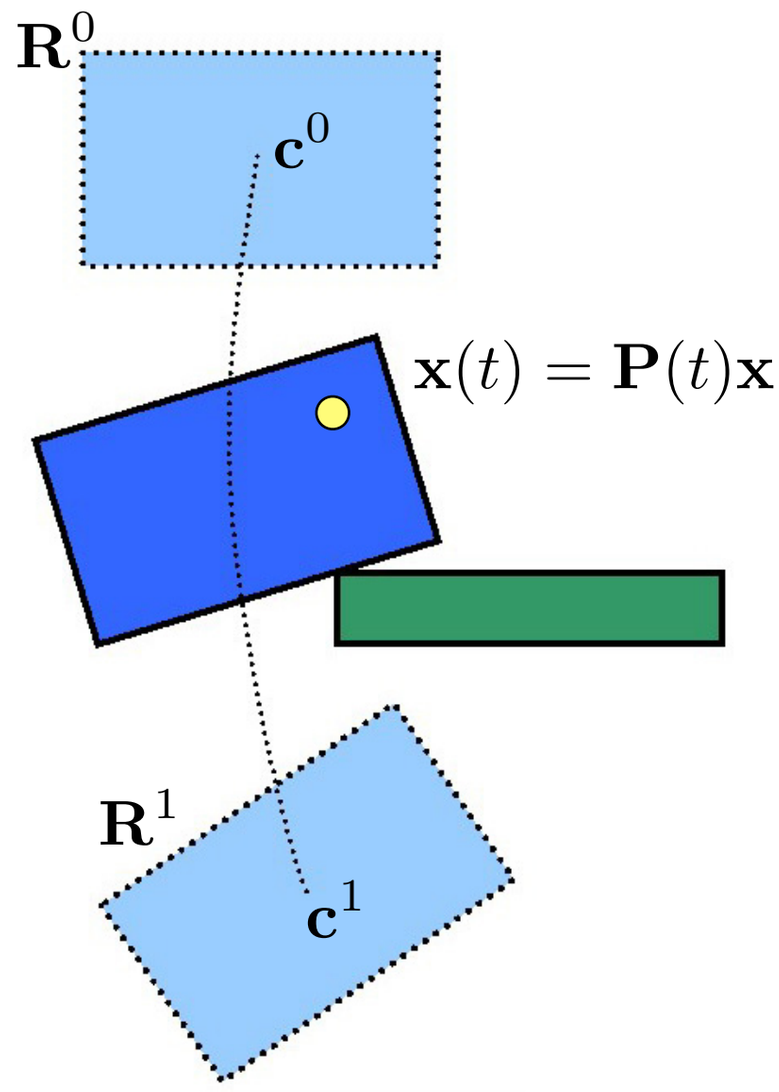


Interpolating Motion

- Describe a continuous equation for rigid-body motion between the two known positions:

$$\begin{aligned}\mathbf{T}(t) &= \mathbf{c}^0 + t(\mathbf{c}^1 - \mathbf{c}^0) \\ \mathbf{R}(t) &= \cos(\omega t)(\mathbf{I} - \mathbf{u}\mathbf{u}^T)\mathbf{R}^0 \\ &+ \sin(\omega t)\mathbf{u}^*\mathbf{R}^0 + \mathbf{u}\mathbf{u}^T\mathbf{R}^0\end{aligned}$$

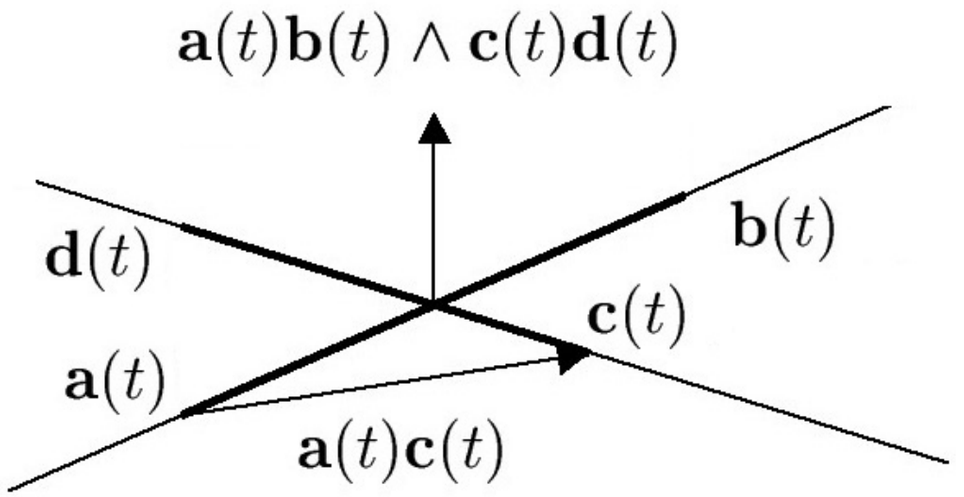
- where ω is rotation angle and **u** is the rotation axis between configurations



Testing for Intersection

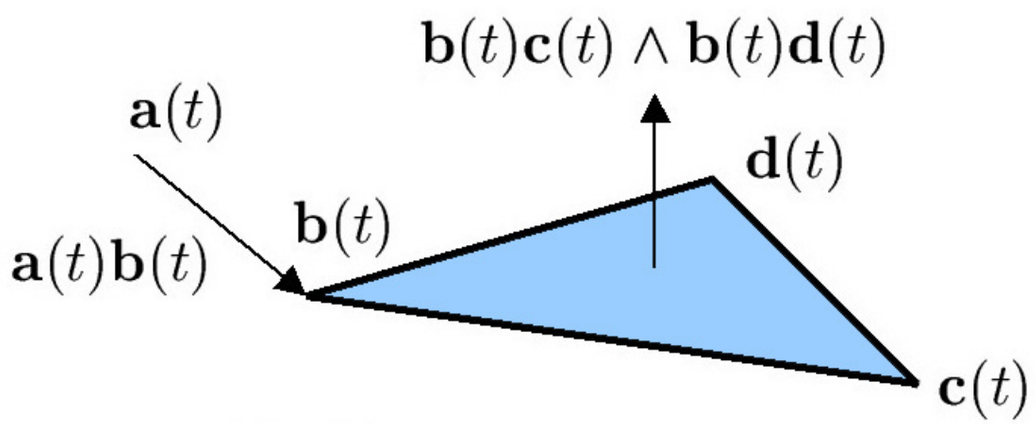
- Edges intersect at *t* if

$$\overrightarrow{\mathbf{a}(t)\mathbf{c}(t)} \cdot \left(\overrightarrow{\mathbf{a}(t)\mathbf{b}(t)} \times \overrightarrow{\mathbf{c}(t)\mathbf{d}(t)} \right) = 0$$



- Vertex/face intersect at time *t* if

$$\overrightarrow{\mathbf{a}(t)\mathbf{b}(t)} \cdot \left(\overrightarrow{\mathbf{b}(t)\mathbf{c}(t)} \times \overrightarrow{\mathbf{b}(t)\mathbf{d}(t)} \right) = 0$$



- How do we find *t*?

Interval Arithmetic

$$I = [a, b] = \{x \in \mathbb{R}, a \leq x \leq b\}$$

$$\begin{aligned}
[a, b] + [c, d] &= [a + c, b + d] \\
[a, b] - [c, d] &= [a - d, b - c] \\
[a, b] \times [c, d] &= [min(ac, ad, bc, bd), max(ac, ad, bc, bd)] \\
1/[a, b] &= [1/b, 1/a] \quad \text{if } a > 0 \text{ or } b < 0 \\
[a, b] / [c, d] &= [a, b] \times (1/[c, d]) \quad \text{if } c > 0 \text{ or } d < 0 \\
[a, b] &\leqslant [c, d] \quad \text{if } b \leqslant c
\end{aligned}$$

Solving for Intersection

- To Solve:

$$\overrightarrow{\mathbf{a}(t)\mathbf{c}(t)} \cdot \left(\overrightarrow{\mathbf{a}(t)\mathbf{b}(t)} \times \overrightarrow{\mathbf{c}(t)\mathbf{d}(t)}\right) = 0, \;\; \overrightarrow{\mathbf{a}(t)\mathbf{b}(t)} \cdot \left(\overrightarrow{\mathbf{b}(t)\mathbf{c}(t)} \times \overrightarrow{\mathbf{b}(t)\mathbf{d}(t)}\right) = 0$$

- Use interval arithmetic, evaluate using the interval t = [0,1]
- If zero is in the result interval, halve and repeat:

$$t = [0, 1]$$
$$t = [0, \tfrac{1}{2}] \qquad t = [\tfrac{1}{2}, 1]$$
$$t = [0, \tfrac{1}{4}] \quad t = [\tfrac{1}{4}, \tfrac{1}{2}] \quad t = [\tfrac{1}{2}, \tfrac{3}{4}] \quad t = [\tfrac{3}{4}, 1]$$

Bounding Volumes

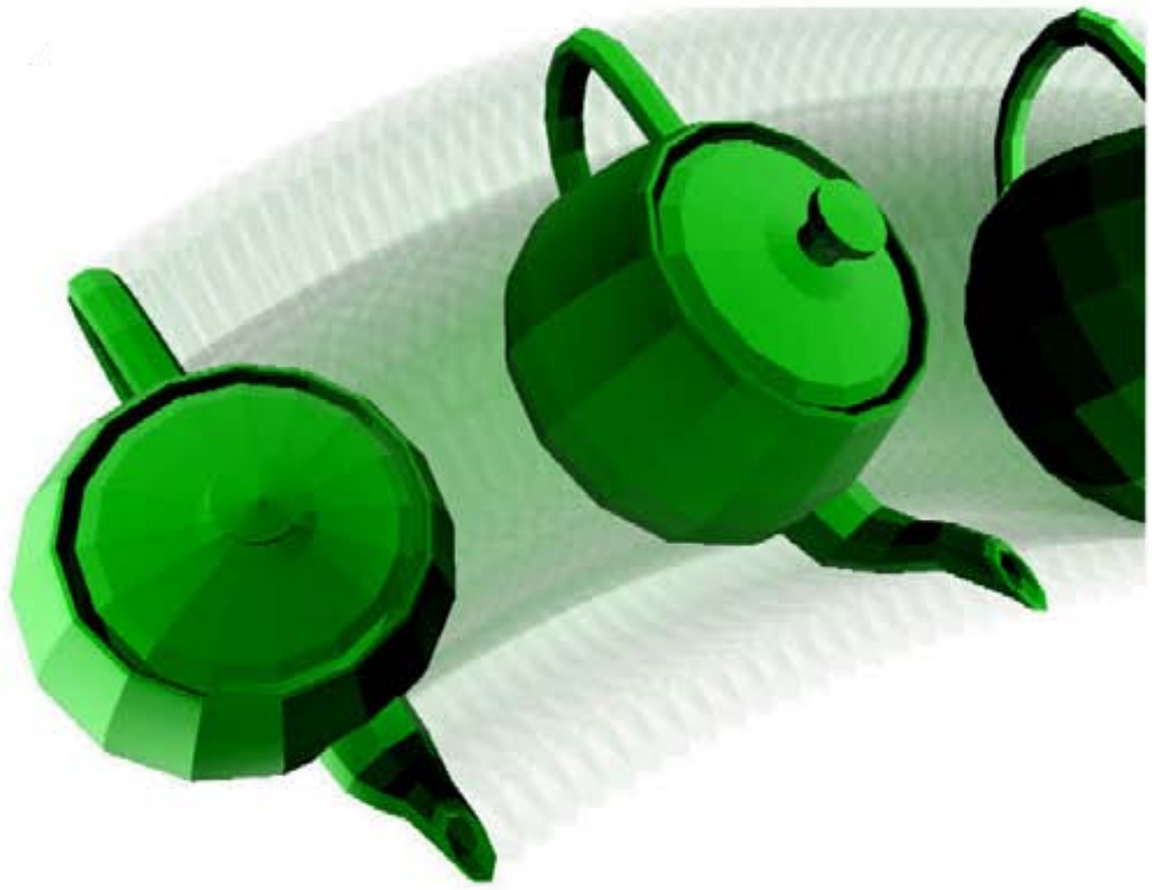
- Continuous collision detection also works with bounding volume intersection tests
- For example, the sphere test becomes

$$\begin{aligned} \|\overrightarrow{\mathbf{c_1}(t)\mathbf{c_2}(t)}\| &\leq r_1 + r_2 \\ (\mathbf{c_2}(t) - \mathbf{c_1}(t))^2 &\leq (r_1 + r_2)^2 \end{aligned}$$

- Conservative test:
  - There may be an intersection if the lower bound on the left is less than the right side

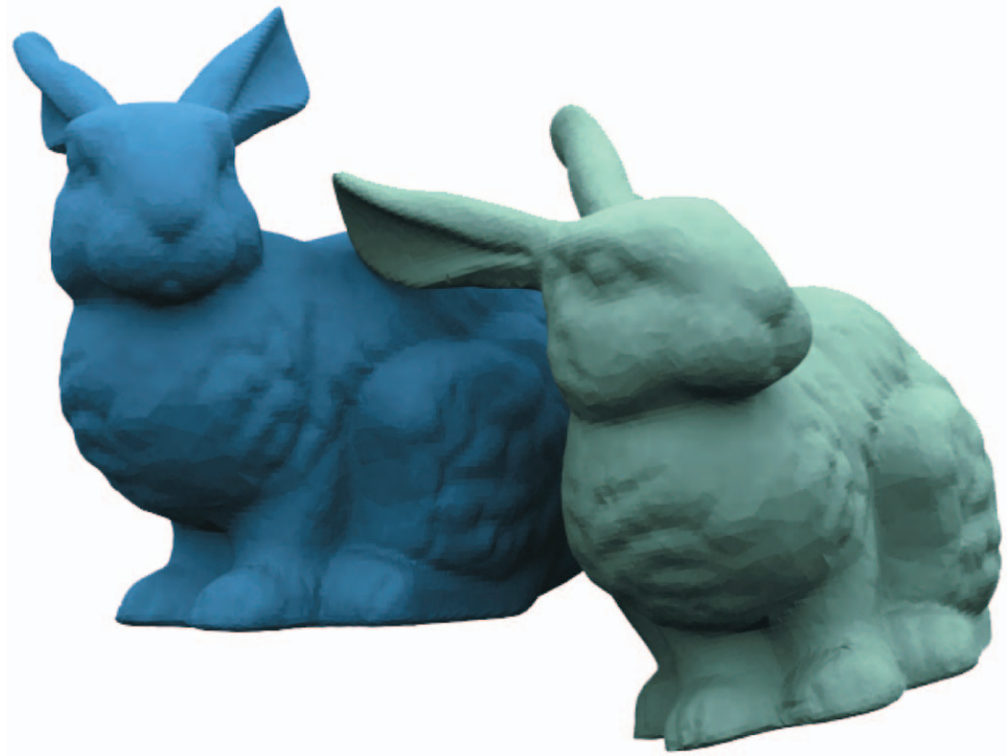
CCD Summary

- Finds the first contact between the avatar and the scene along a motion path
- Not quite "*continuous*", but computes time of contact to a precision
- Can combine with structures like BVHs



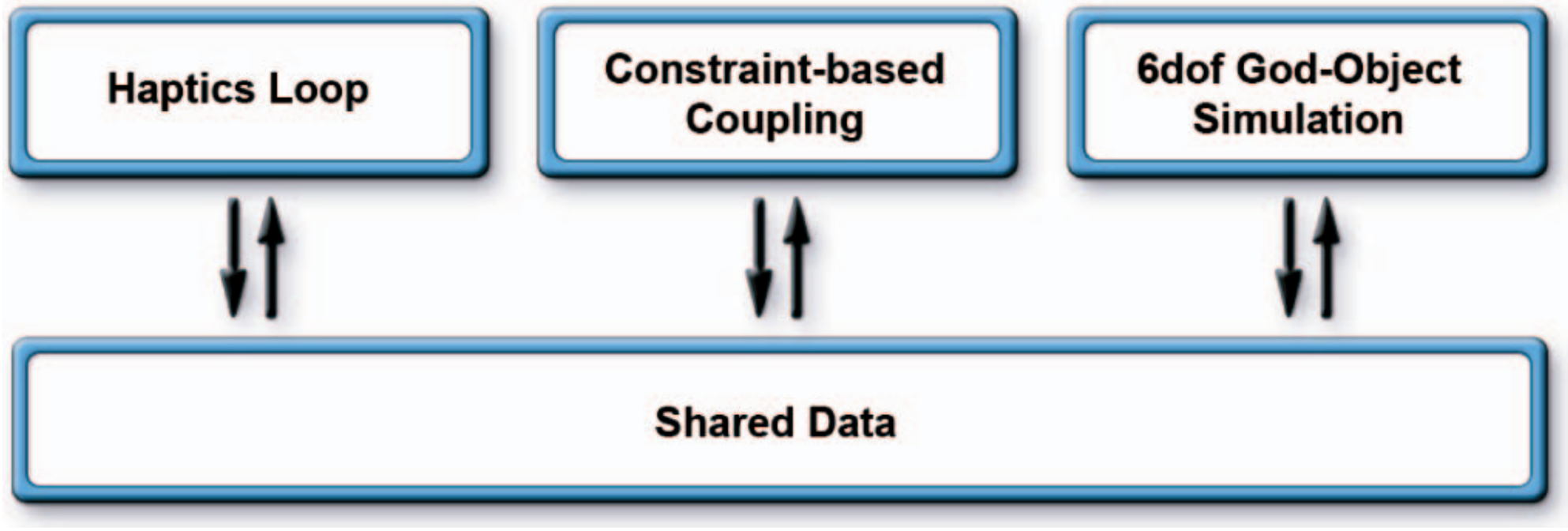
Collision Detection Performance

- Fast, but not haptic rates for large meshes



- Execution time varies:
  - 70 Hz for 27k triangles
- Again, use multiple threads at different rates...

Implementation Diagram



(1 ms) (~ µs) (~ ms)

Summary

- Advantages:
  - Continuous collision detection ensures no object penetration
  - No forces are felt in free space

- Disadvantages?

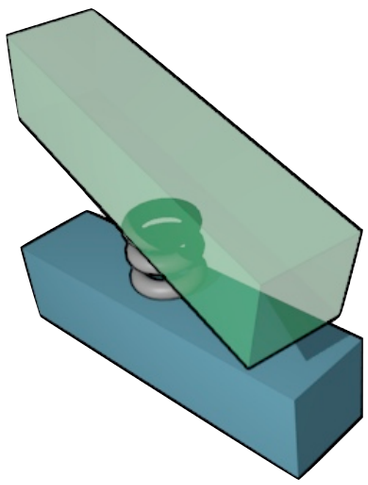
Proxy-Based Rendering

position + orientation

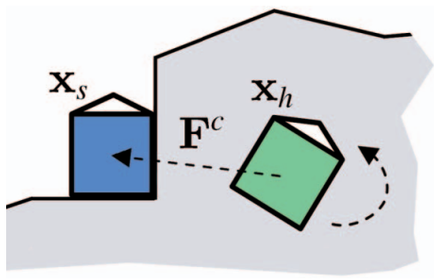
force + torque

position + orientation

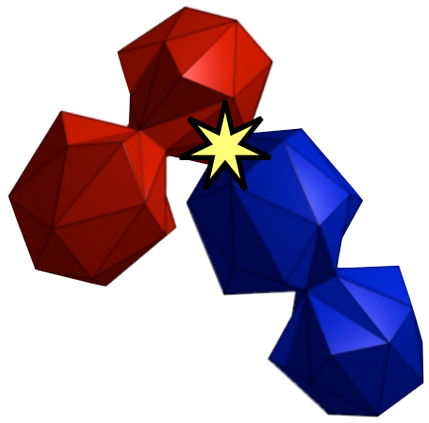
Device Controller



Virtual Coupling



Proxy Solver



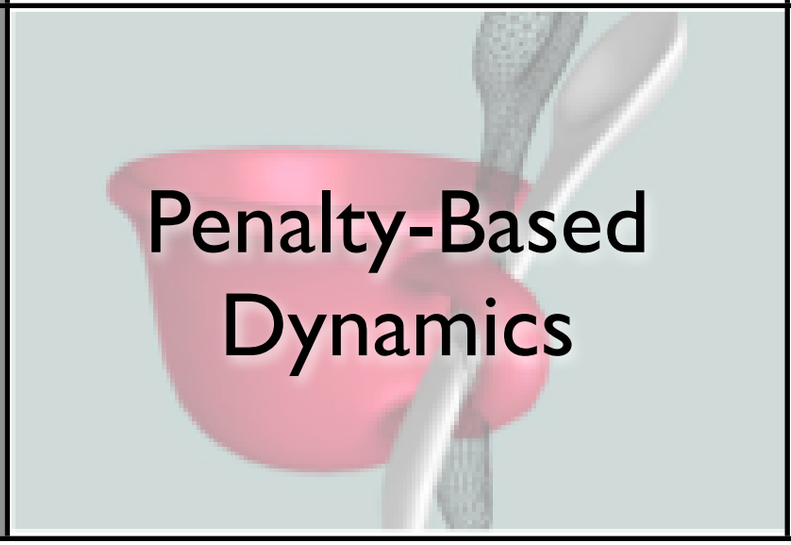
Collision Detector

force + torque

position + orientation

contact points + normals or forces

Proxy Rendering Taxonomy

	Soft Constraints	Hard Constraints
Massless Proxy	Quasi-Static Equilibrium	Distance Minimization 
Proxy with Mass	Penalty-Based Dynamics 	Constrained Dynamics