

# Collision and Contact

*David Baraff*

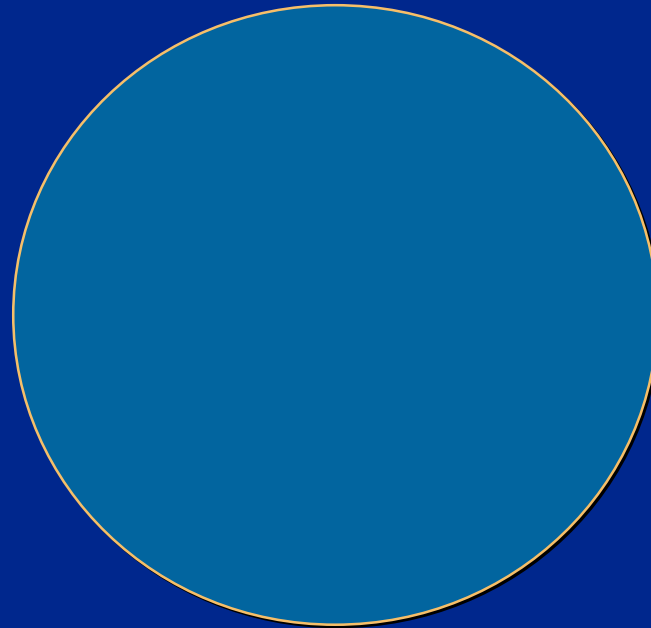


# Collision and Contact

We want objects to behave as if they were solid and not interpenetrate. When collisions or contacts occur we need to:

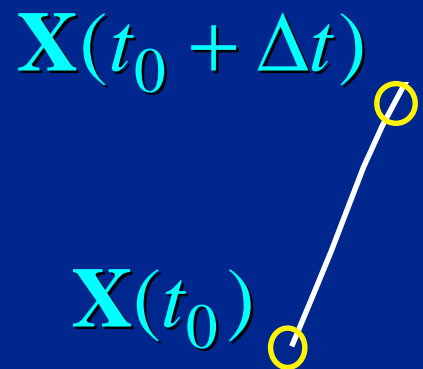
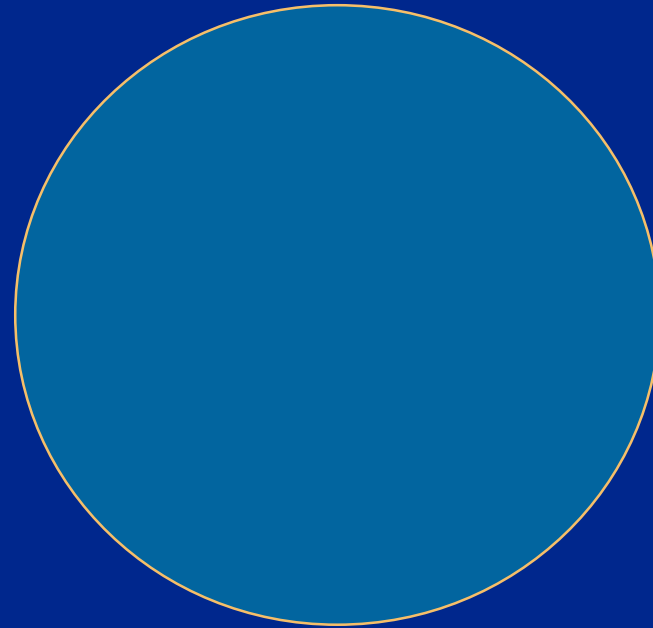
- Detect them.
- Fix them (if they're wrong).
- Maintain them.

# Simulations with Collisions

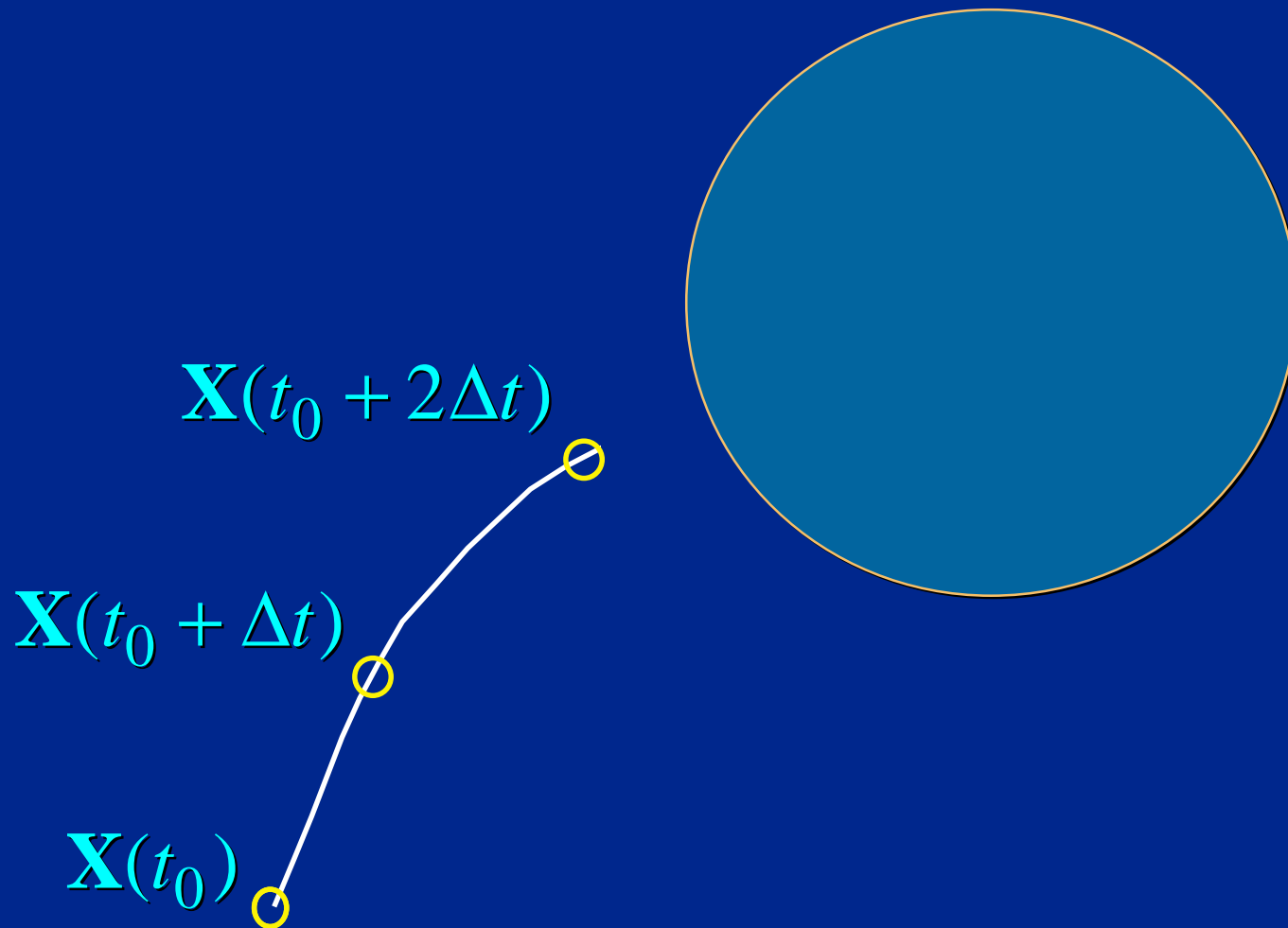


$X(t_0)$  ○

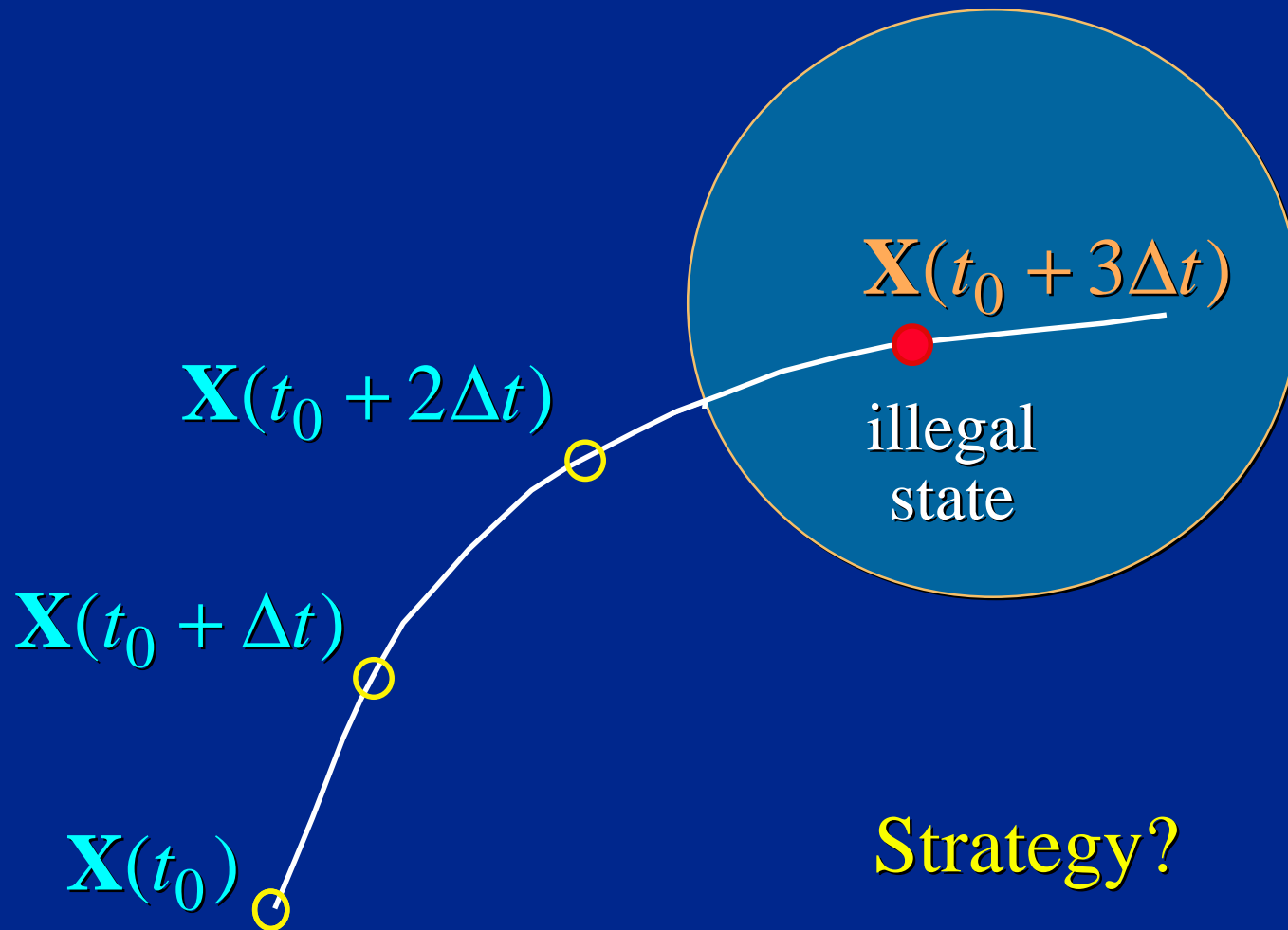
# Simulations with Collisions



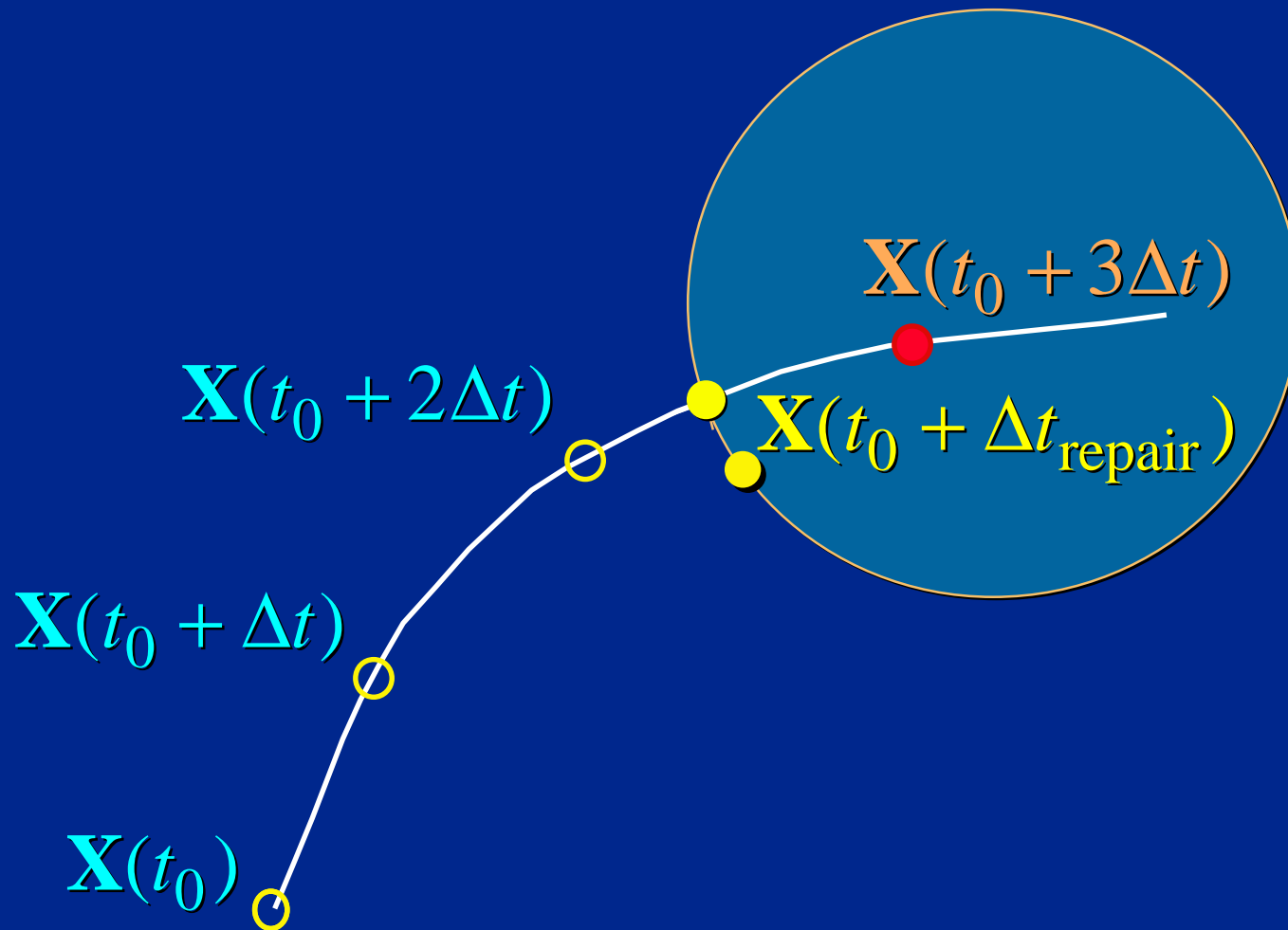
# Simulations with Collisions



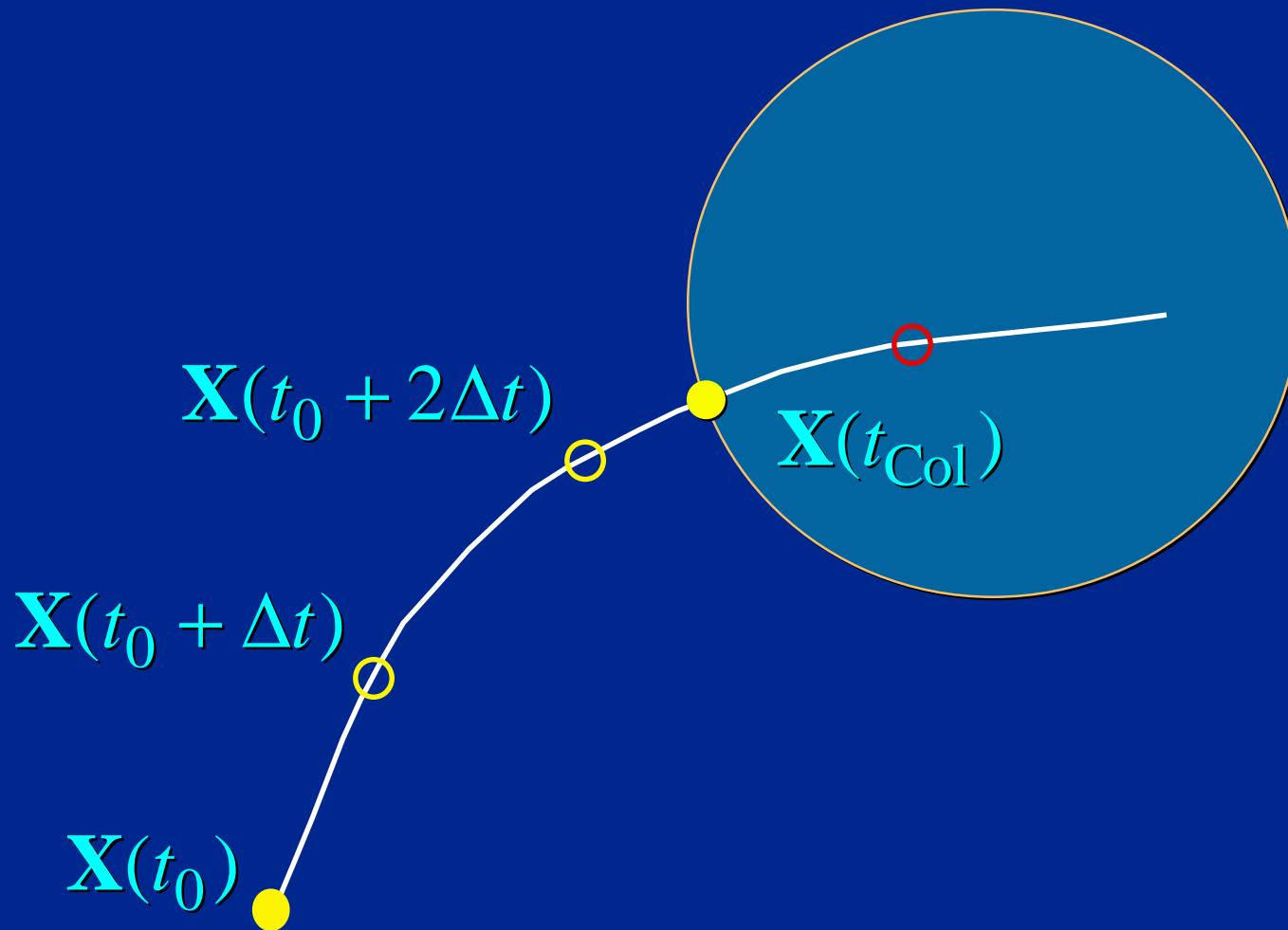
# An Illegal State X



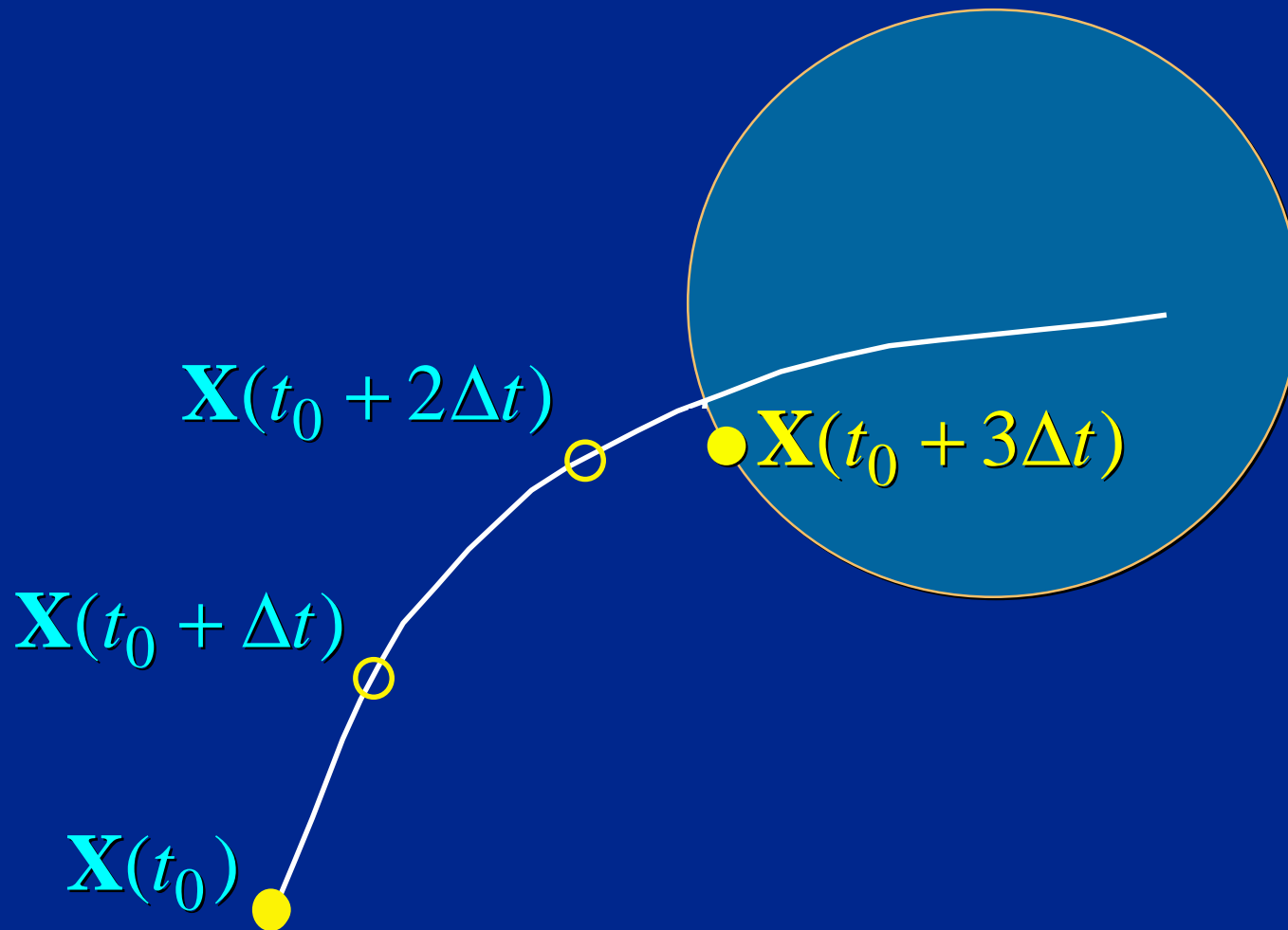
# Plan 1: Gradual Repair



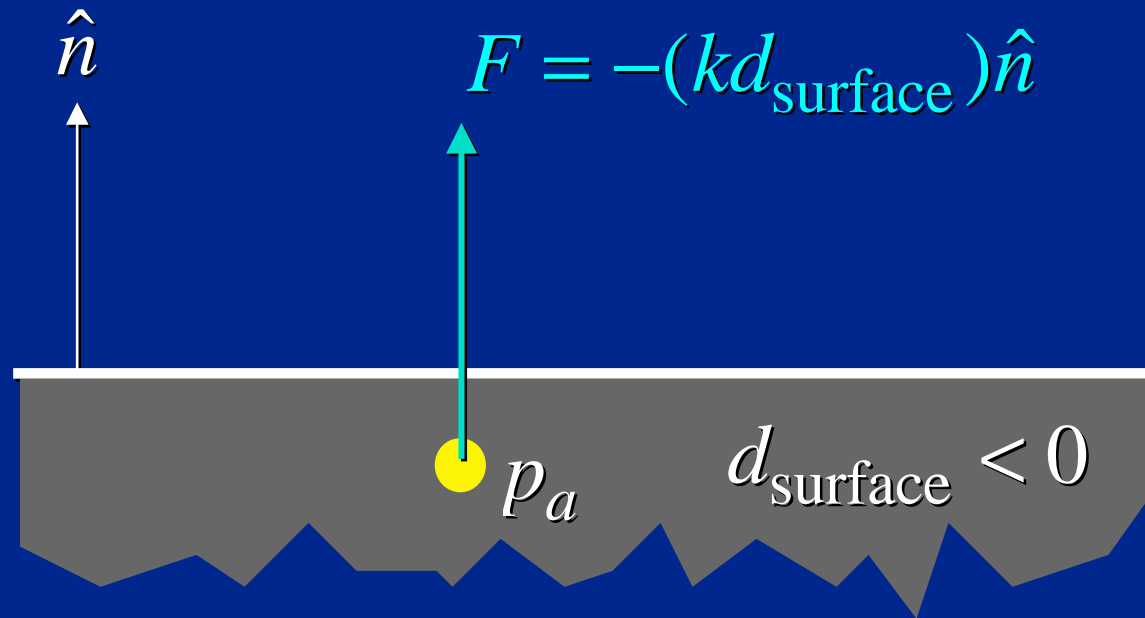
# Plan 2: Backstep



# Plan 3: Just Lie About It!



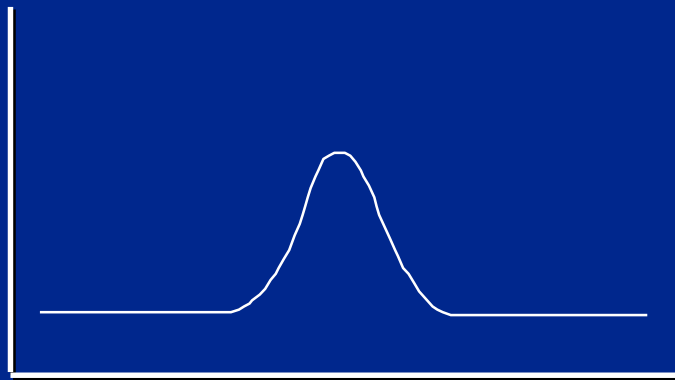
# Penalty-Method Approach





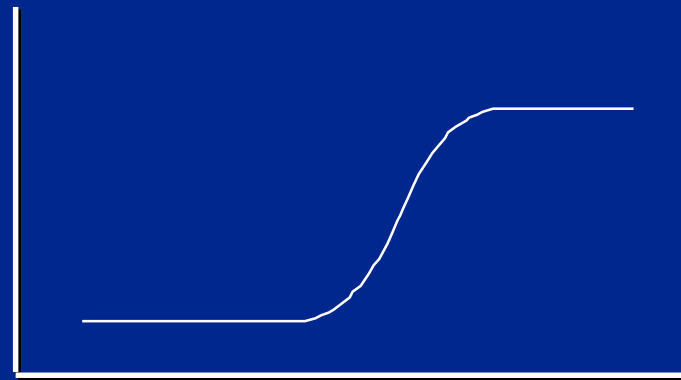
# A Soft Collision

force



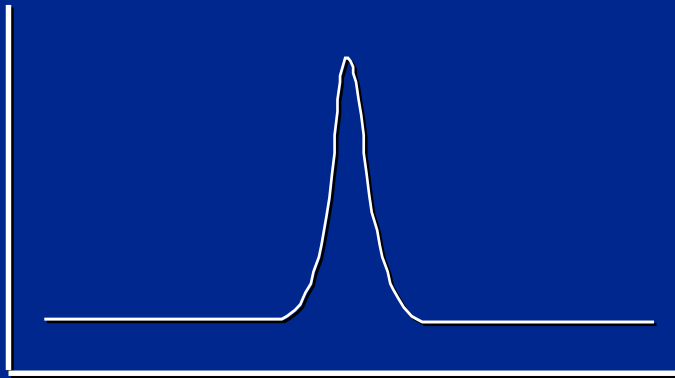
$\Delta t$

velocity



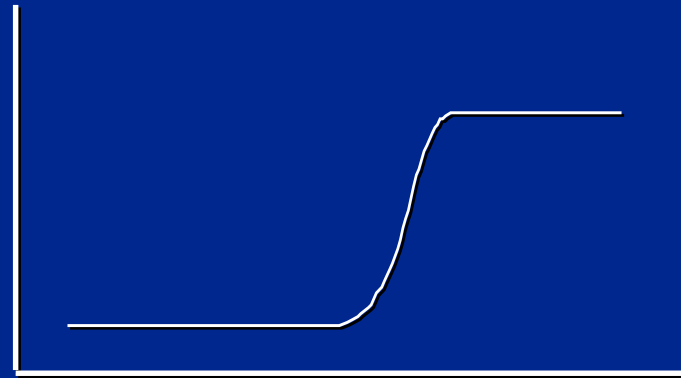
# A Harder Collision

force

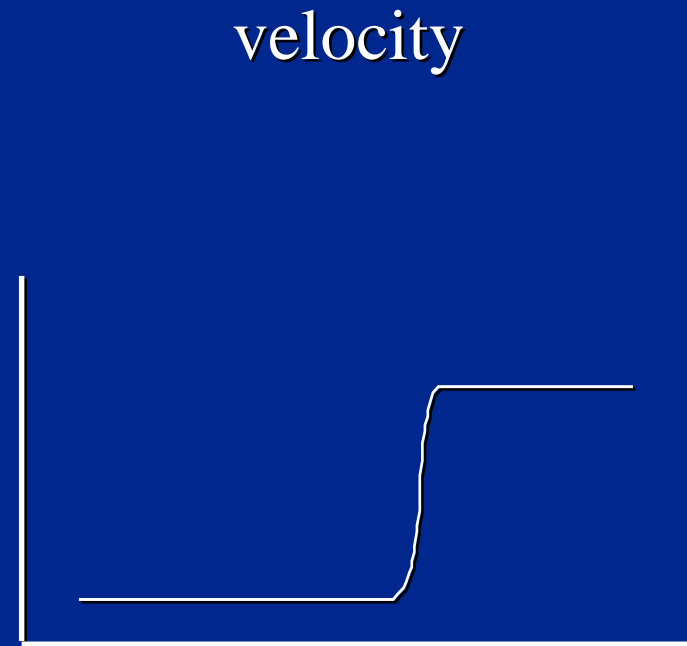
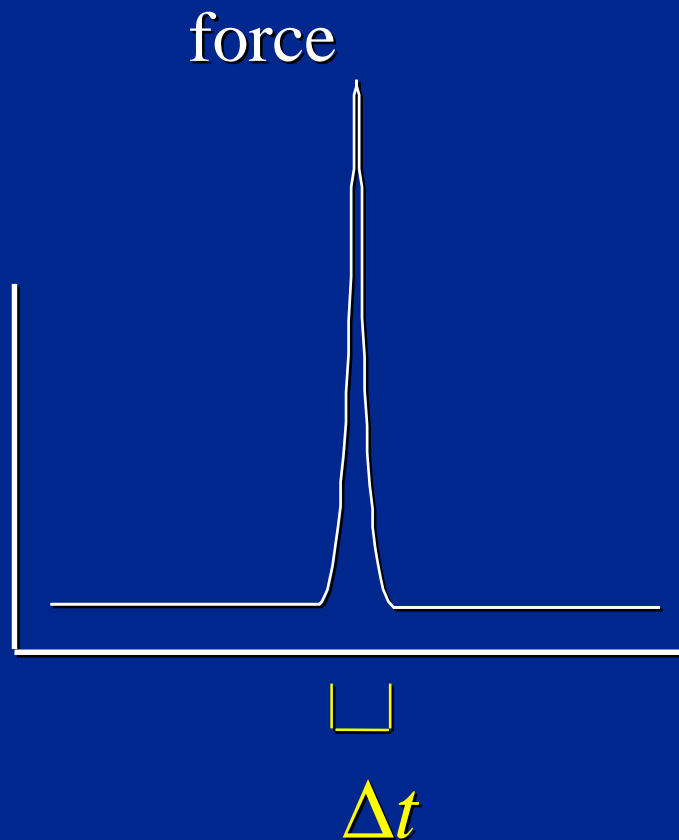


$\Delta t$

velocity

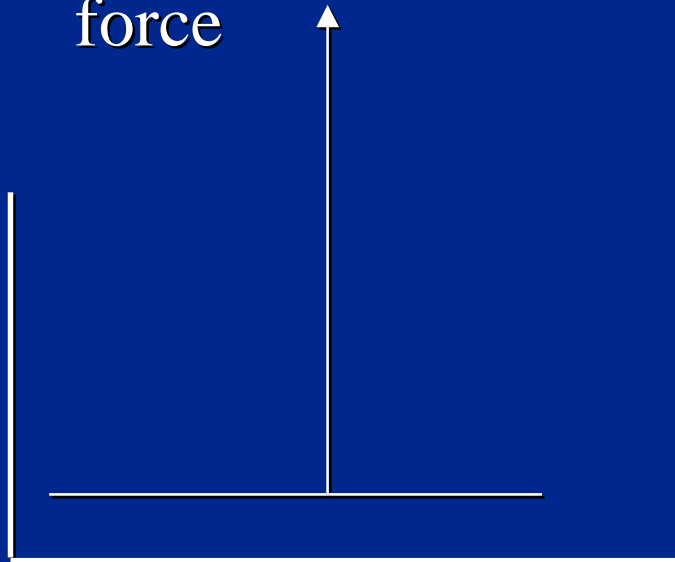


# A Very Hard Collision

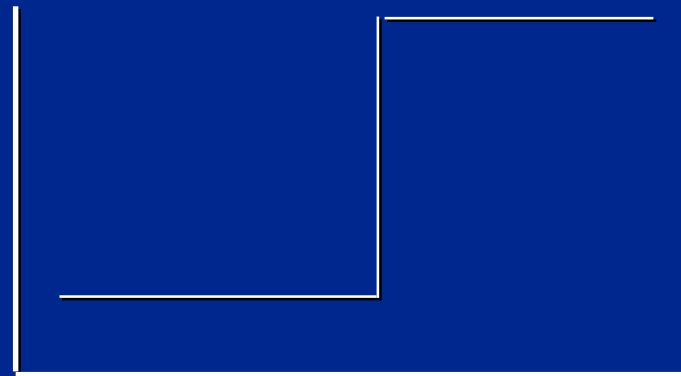


# An Infinitely Hard Collision

impulsive  
force



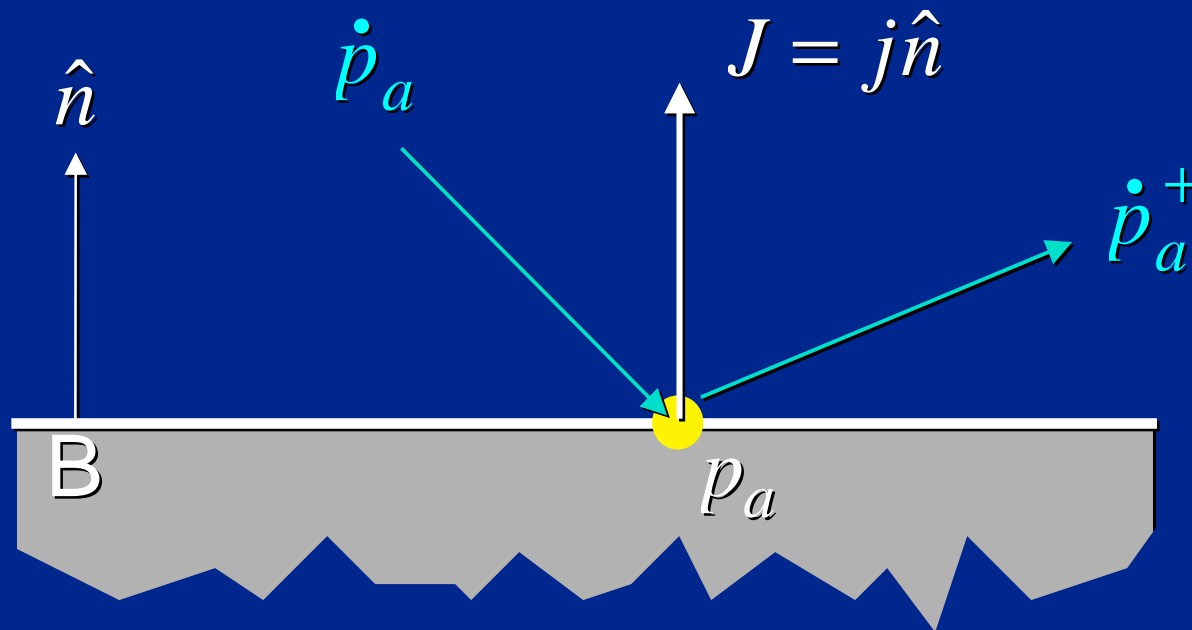
velocity



$$f_{imp} = \infty$$

$$\Delta t = 0$$

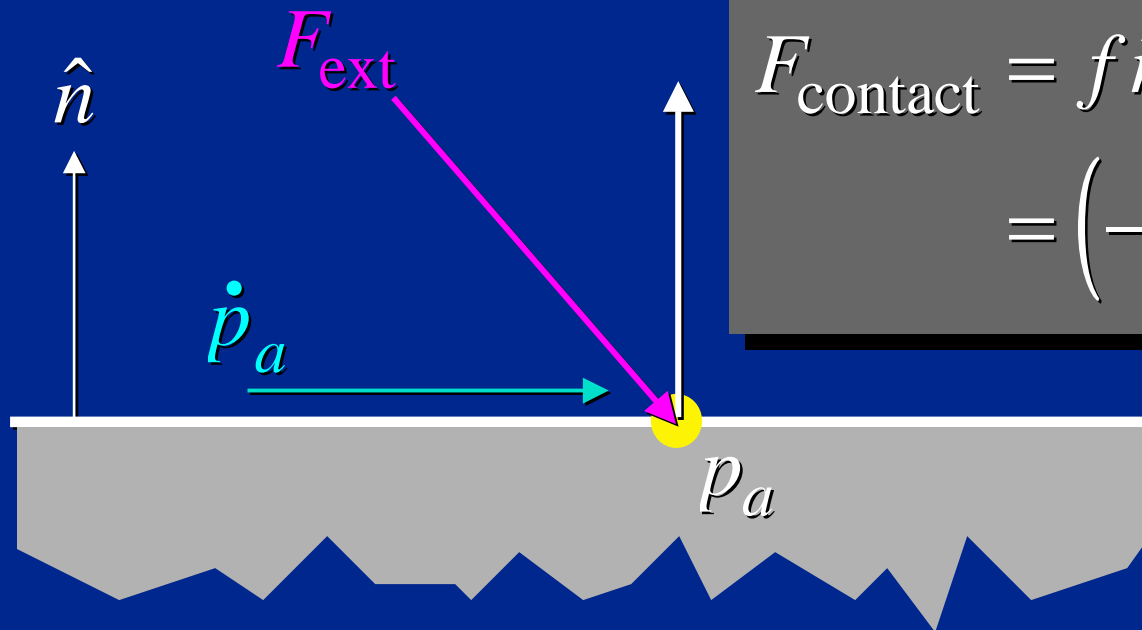
# Colliding Contact



$$\hat{n} \cdot \dot{p}_a < 0$$

$$\dot{p}_a^+ = \frac{J}{m_a} + \dot{p}_a$$

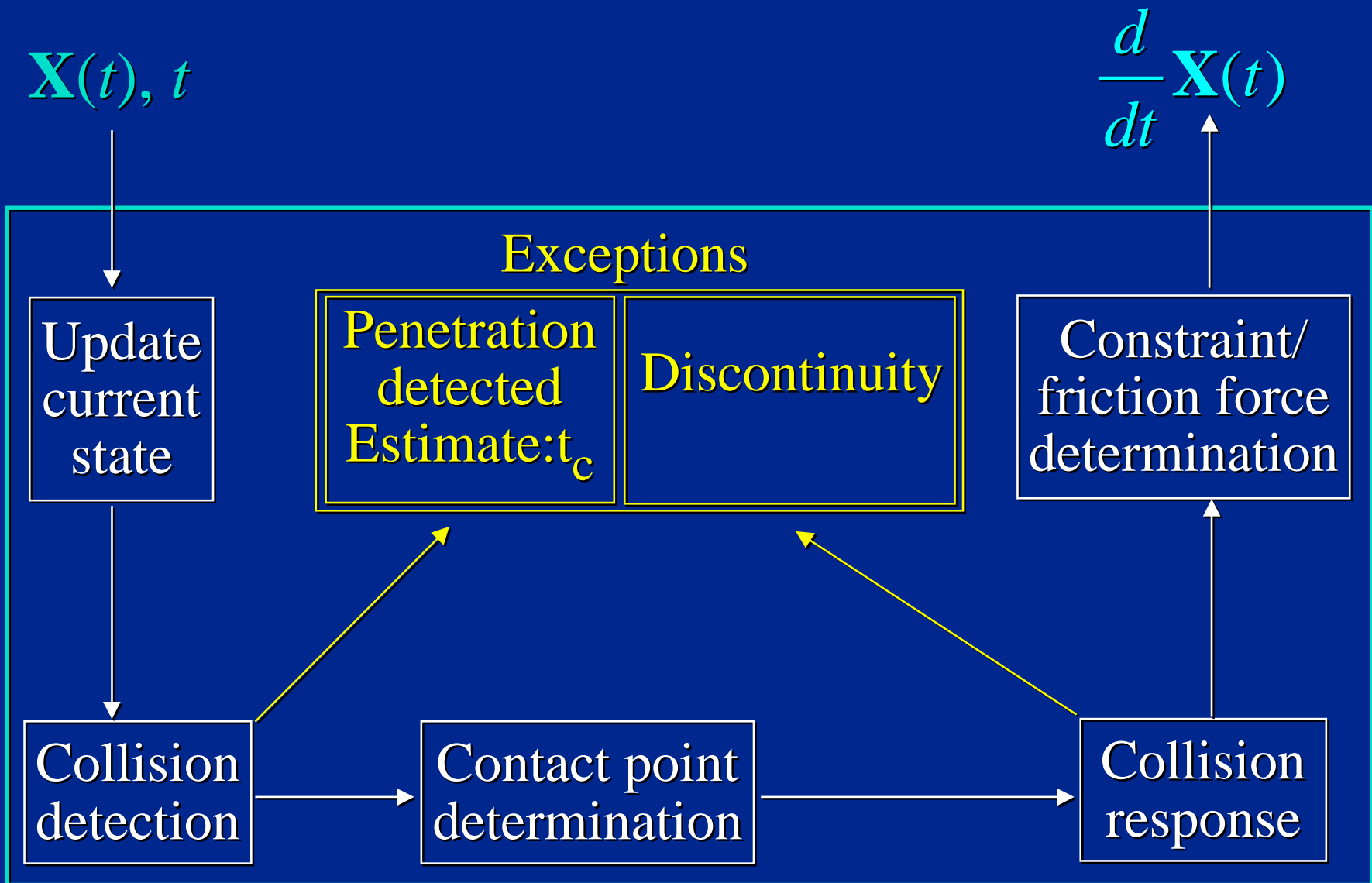
# Resting Contact



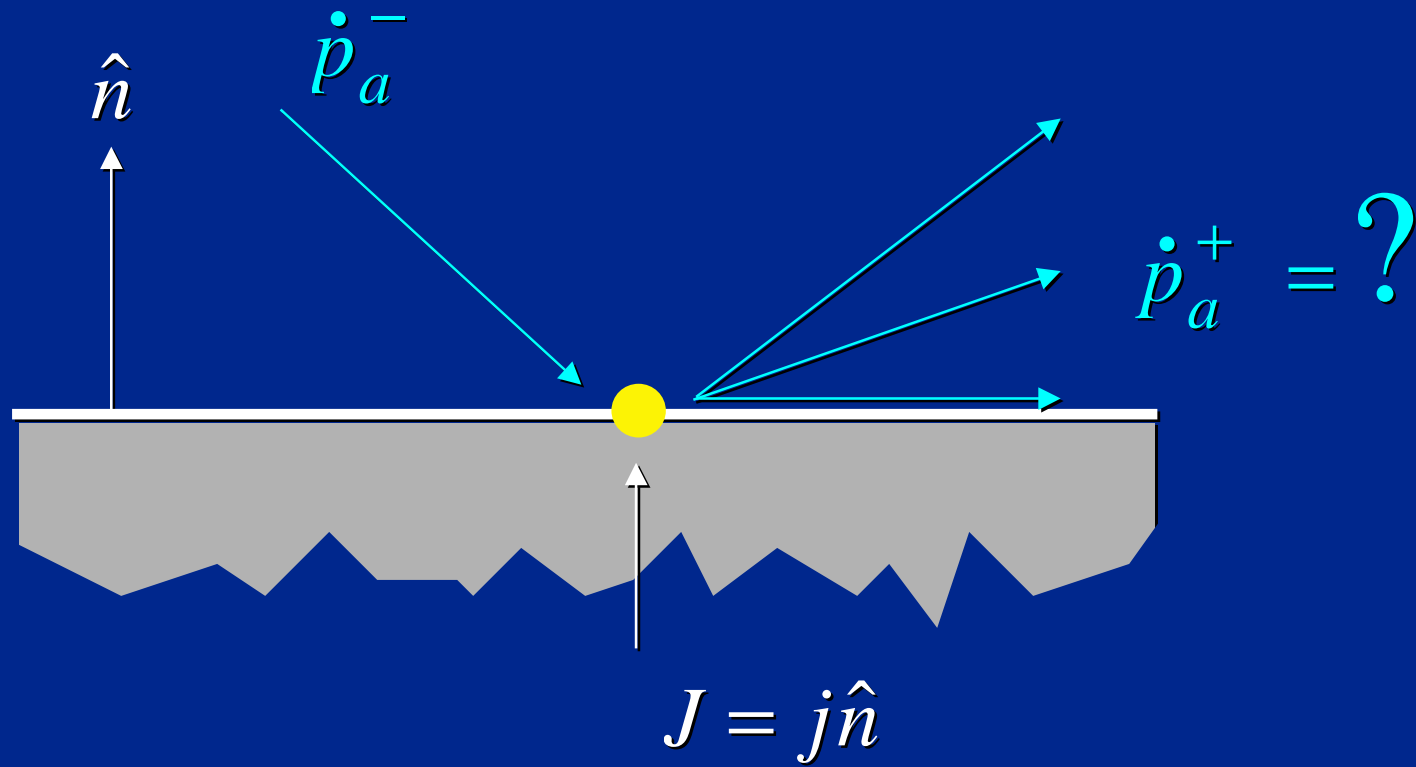
$$\begin{aligned} F_{\text{contact}} &= f \hat{n} \\ &= \left( -F_{\text{ext}} \cdot \hat{n} \right) \hat{n} \end{aligned}$$

$$\hat{n} \cdot \dot{p}_a = 0$$

# Dxdt() for Contacts

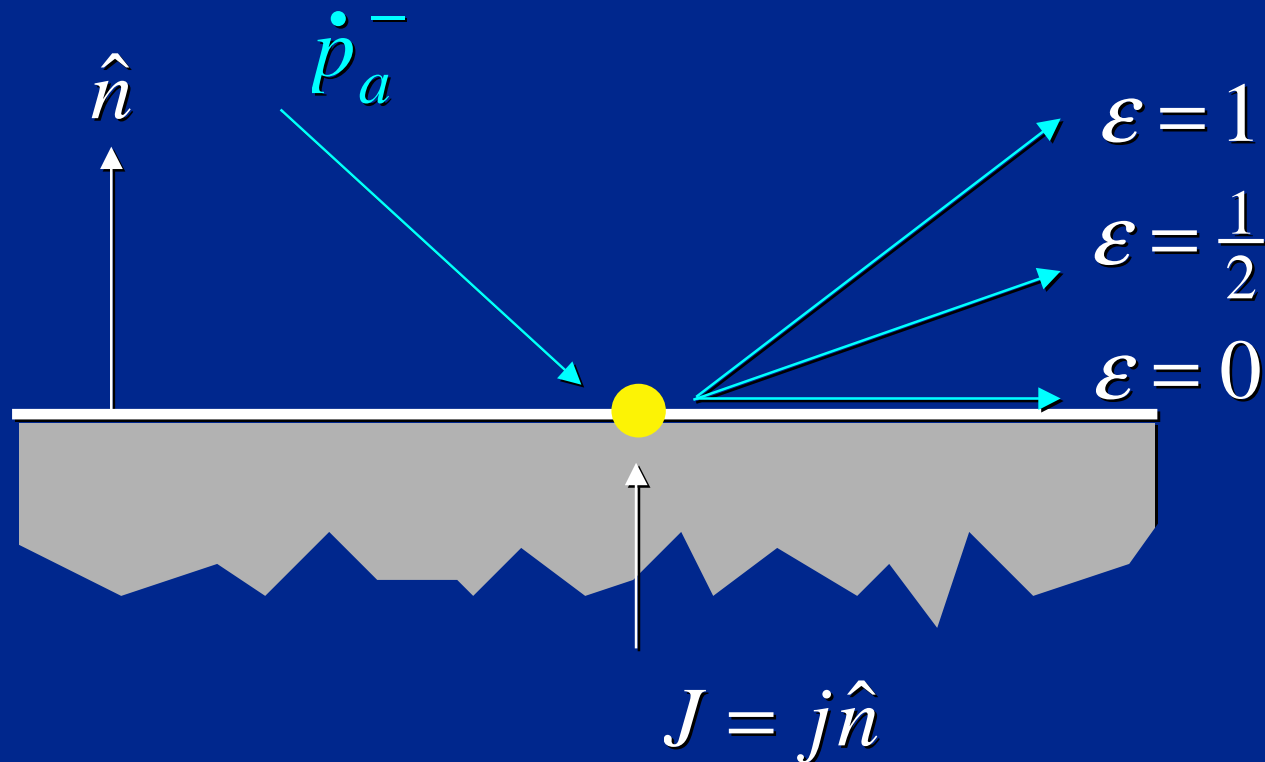


# Computing Impulses

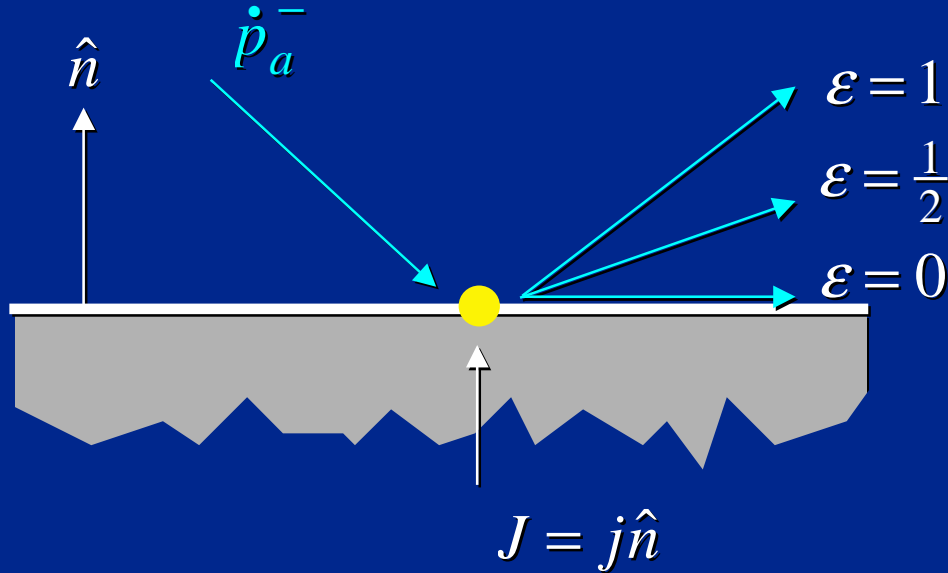


# Coefficient of Restitution

$$\hat{n} \cdot \dot{p}_a^+ = -\varepsilon(\hat{n} \cdot \dot{p}_a^-)$$



# Computing $j$

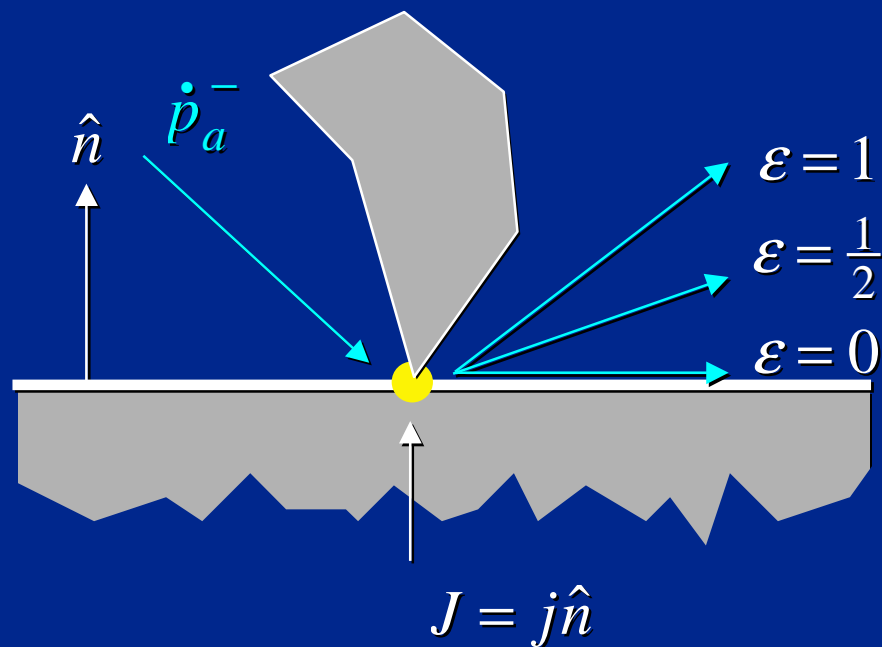


$$\hat{n} \cdot \dot{p}_a^+ = -\epsilon(\hat{n} \cdot \dot{p}_a^-)$$

$$\dot{p}_a^+ = \frac{j\hat{n}}{m_a} + \dot{p}_a^-$$

$$aj = b$$

# Computing $j$



$$\hat{n} \bullet \dot{p}_a^+ = -\epsilon(\hat{n} \bullet \dot{p}_a^-)$$

$$\dot{p}_a^+ = (m_a, \mathbf{I}_a, v^-, \omega^-, j)$$

$$aj = b$$

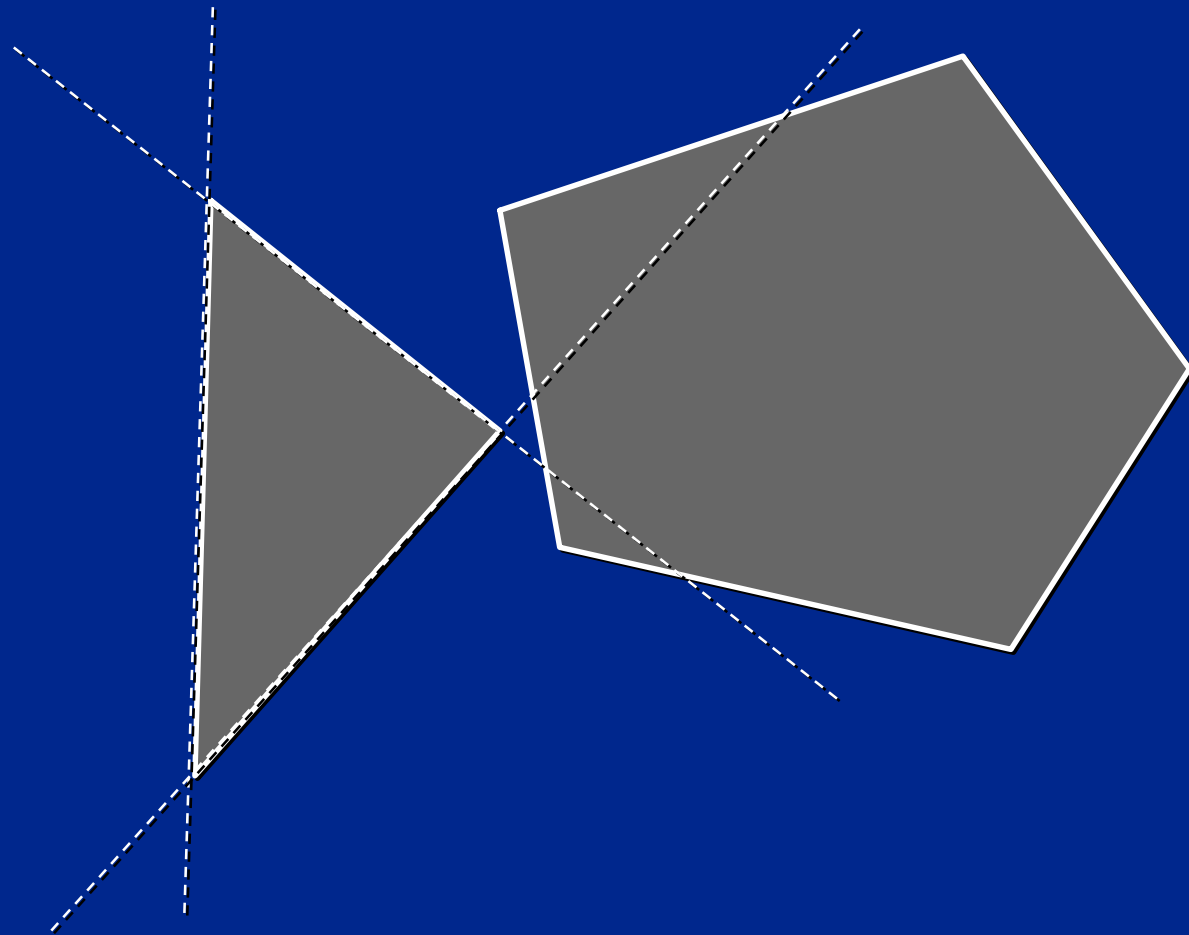
# In the Course Notes — Collision Response

Data structures to represent contacts (found by the collision detection phase).

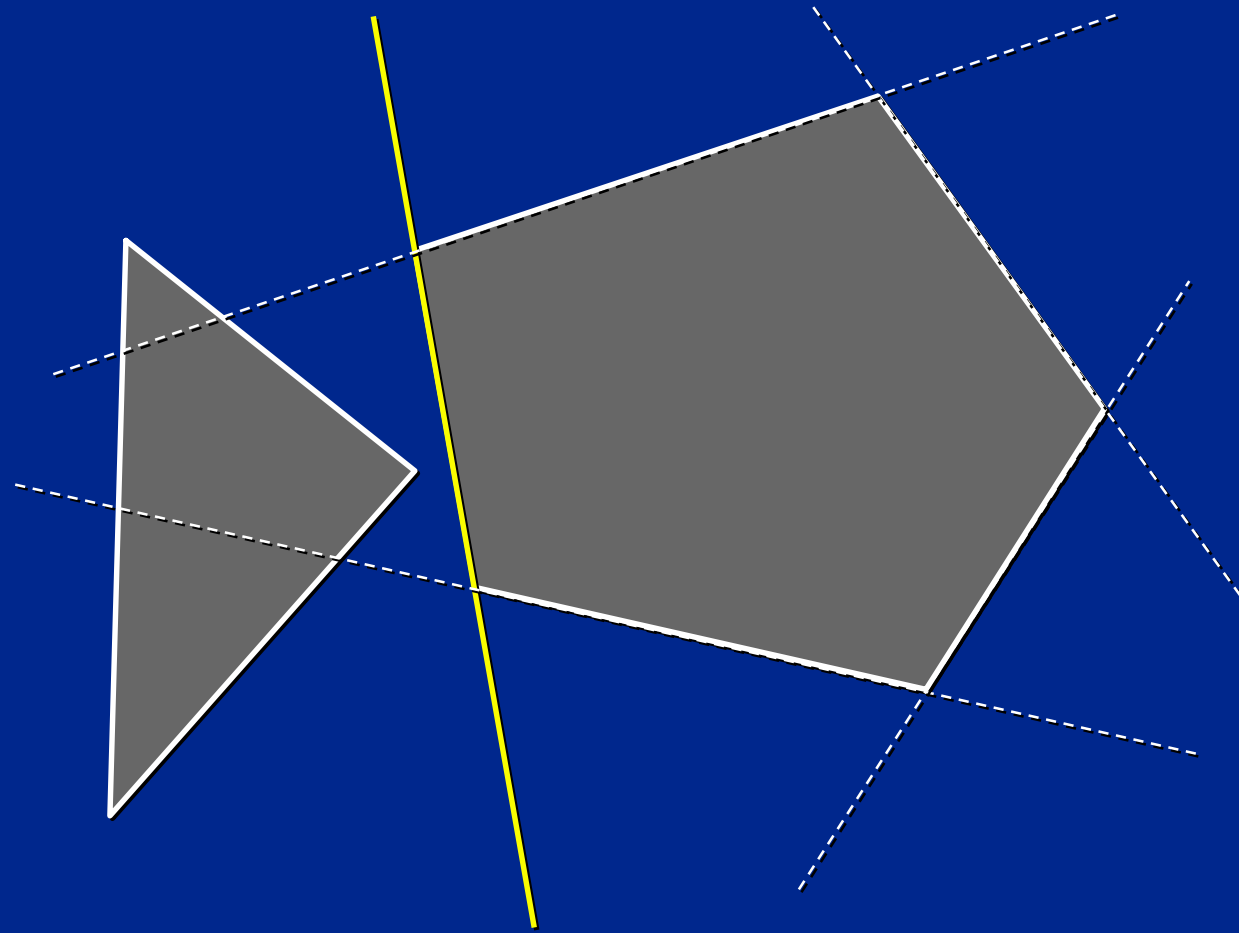
Derivations and code for computing the impulse between two colliding frictionless bodies for a particular coefficient of  $\epsilon$ .

Code to detect collisions and apply impulses.

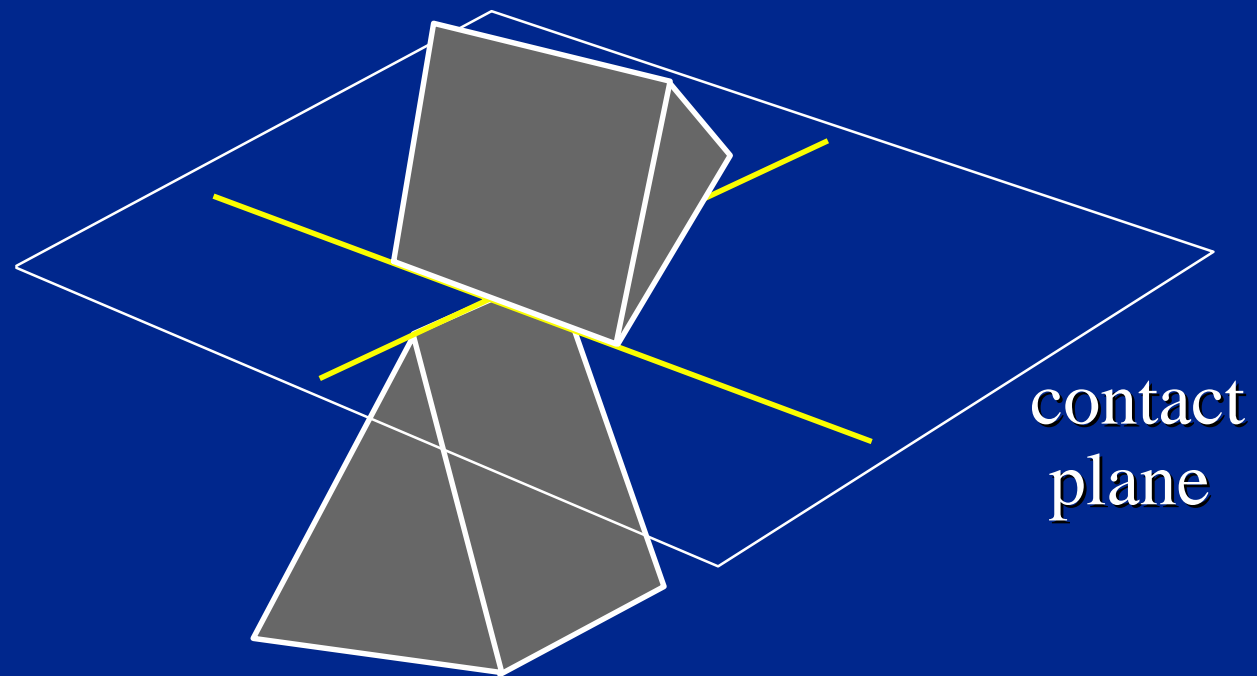
# Separating Planes



# Separating Planes



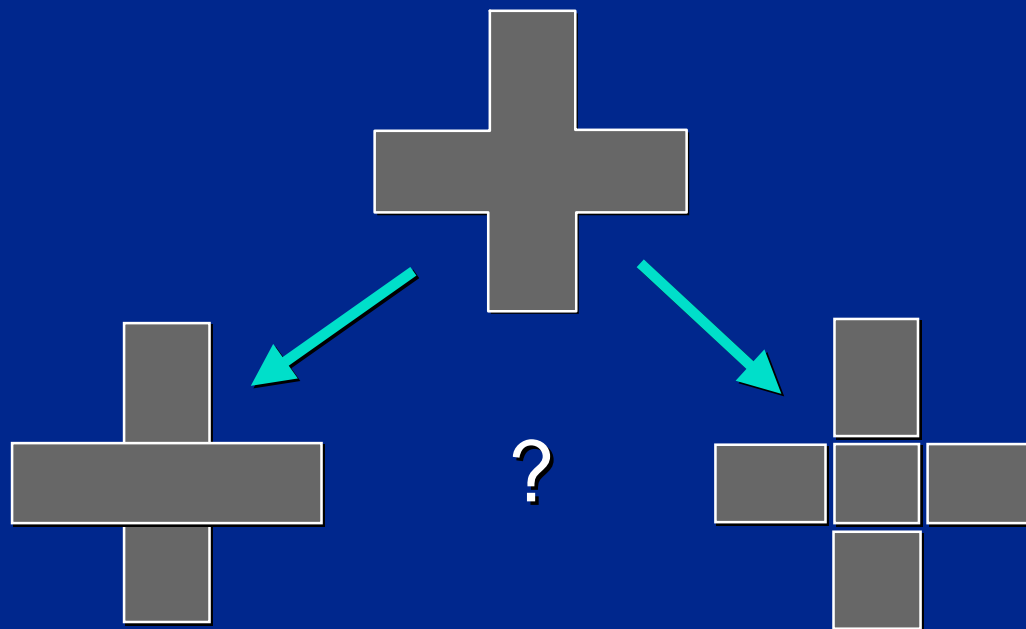
# Separating Planes (in 3D)



# Does It Really Work?

Yes, but...

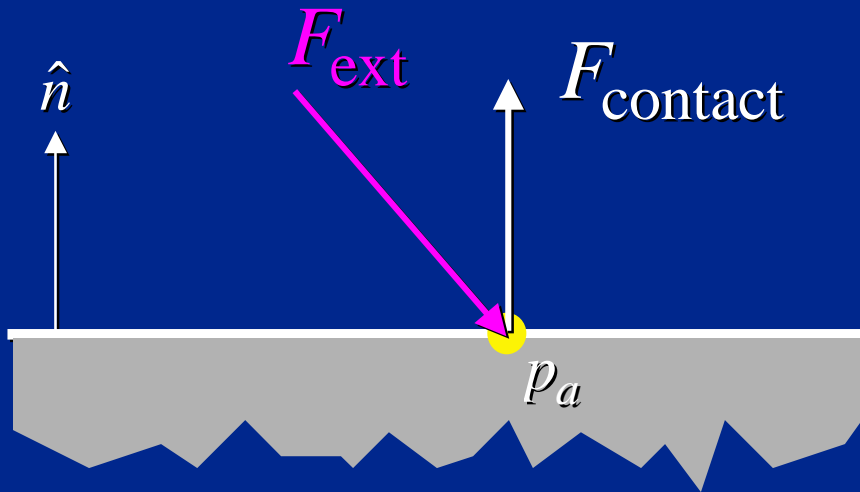
- Requires convex decomposition
- Needs a good decomposition:



# An Actual Implementation

- **Coriolis<sup>TM</sup>**—rigid body dynamics engine in Alias|Wavefront's *Maya<sup>TM</sup>*
- Fast and reliable
- Compares pairs of polygons from non-convex topologically specified polyhedra (using a coherence-based separating plane approach)
- Hierarchical bounding-box tree to eliminate most false hits

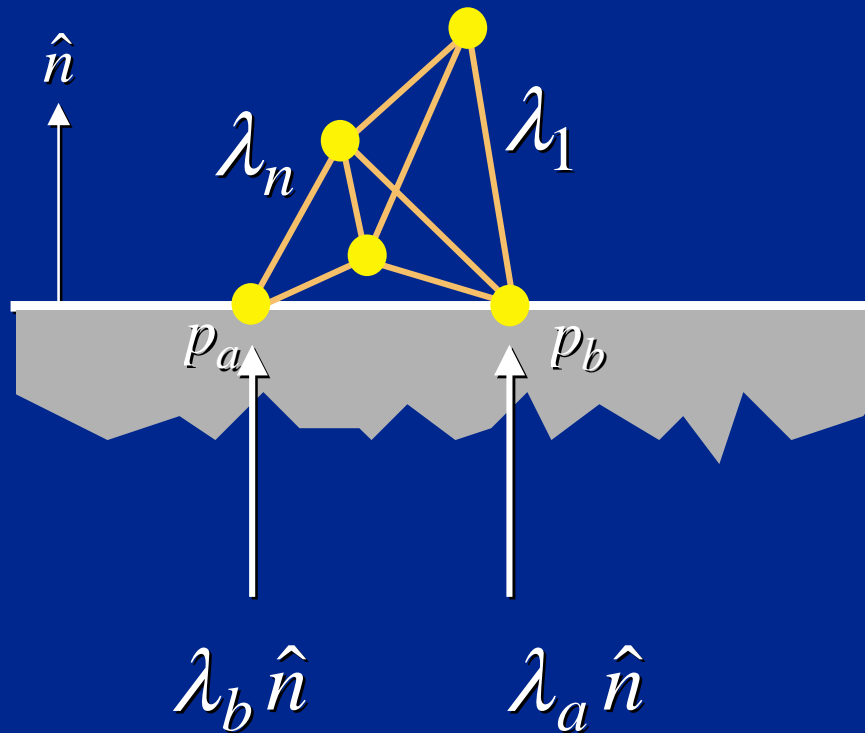
# Resting Contact



$$\begin{aligned} F_{contact} &= f \hat{n} \\ &= \left( -F_{ext} \cdot \hat{n} \right) \hat{n} \end{aligned}$$

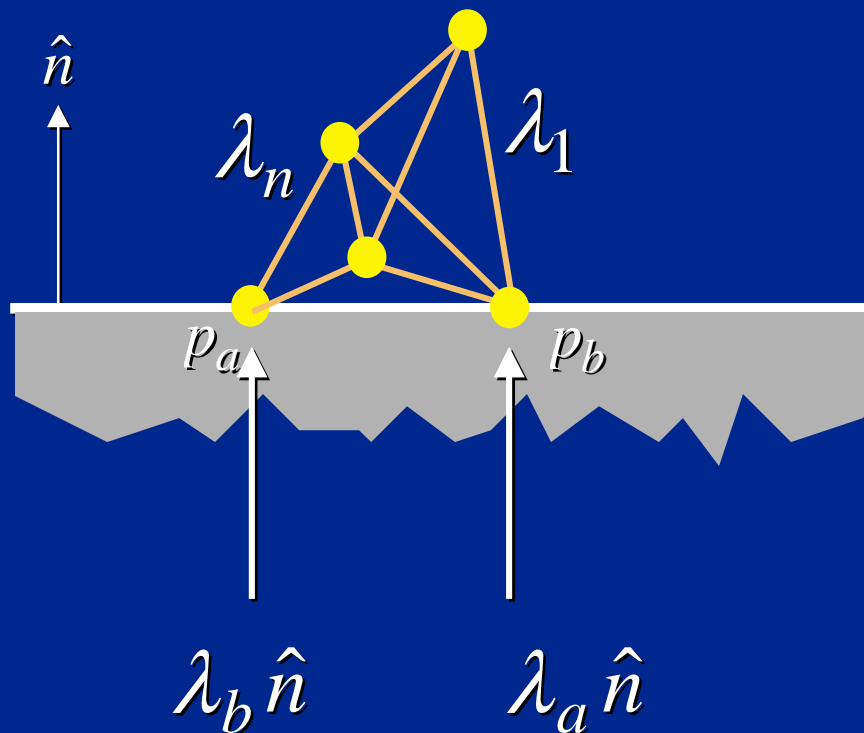
$$\begin{aligned} f &= -F_{ext} \cdot \hat{n} \\ &\text{(if } f \geq 0\text{)} \end{aligned}$$

# Resting Contact



$$\mathbf{r} = \mathbf{J}\mathbf{W}\mathbf{J}^T \begin{pmatrix} \lambda \\ \lambda_a \\ \lambda_b \end{pmatrix} - \mathbf{c} = \mathbf{0}$$

# Resting Contact: Quadratic Program



$$\mathbf{r} = \mathbf{J}\mathbf{W}\mathbf{J}^T \begin{pmatrix} \lambda \\ \lambda_a \\ \lambda_b \end{pmatrix} - \mathbf{c} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

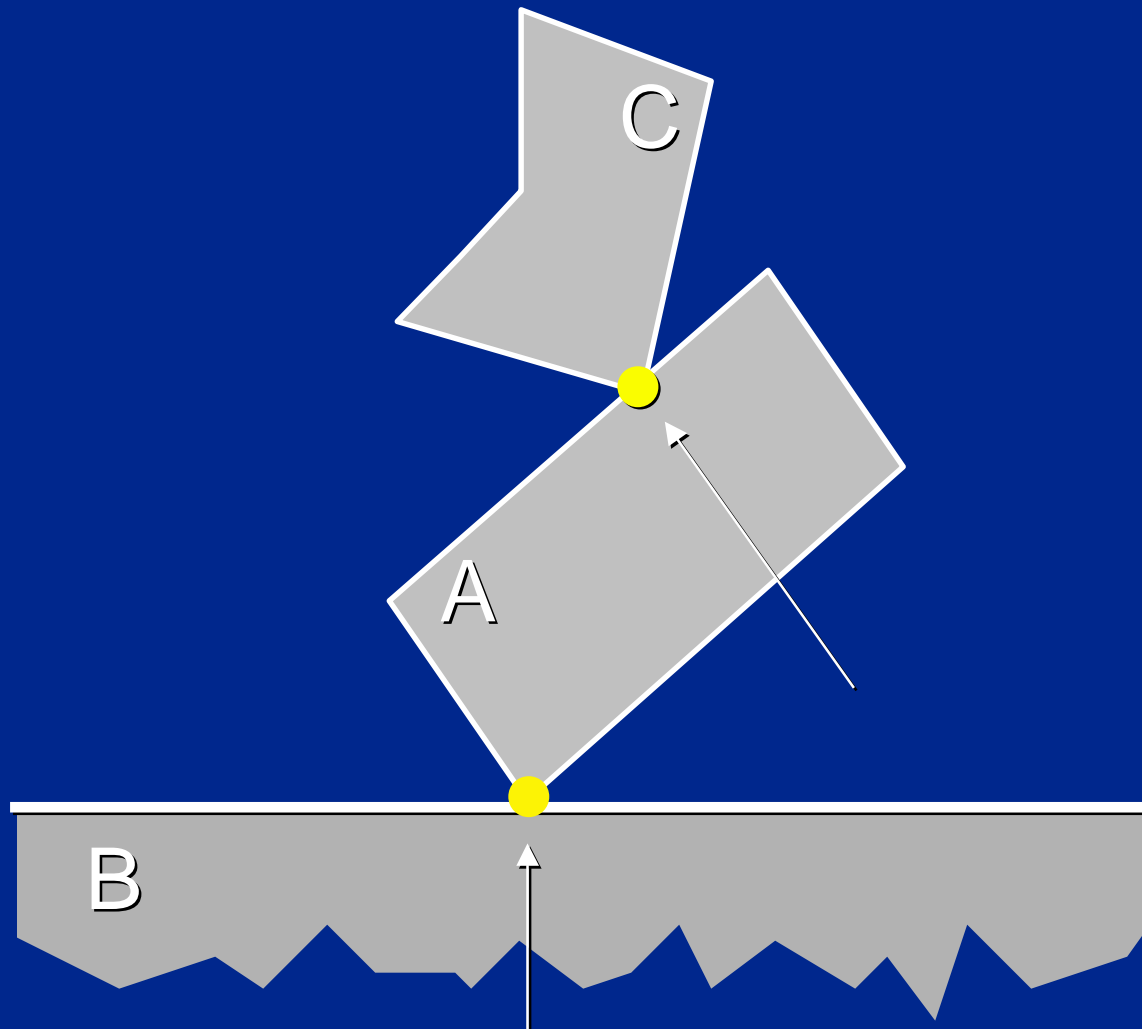
$$\lambda_a \geq 0$$

$$\lambda_b \geq 0$$

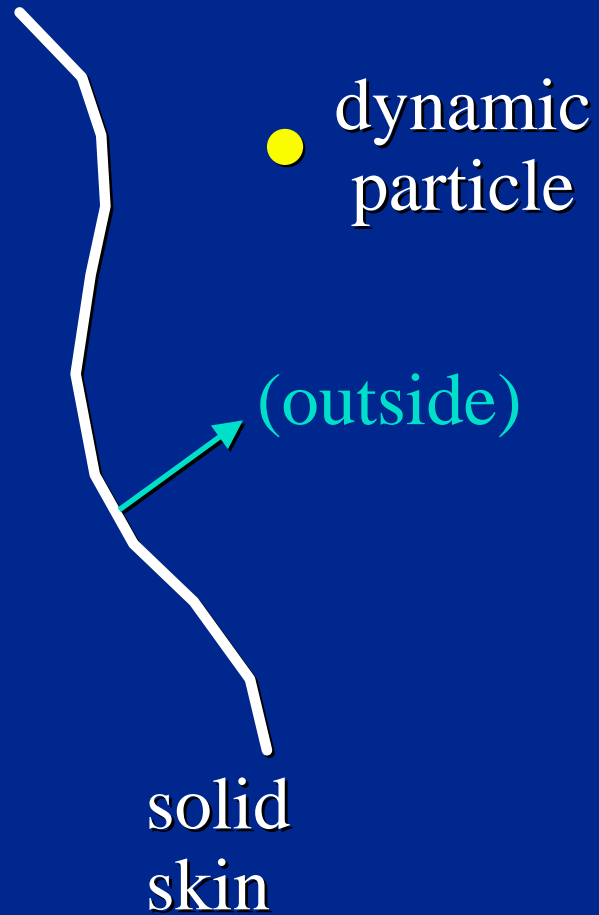
$$\lambda_a = 0 \text{ if } a_{n+1} > 0$$

$$\lambda_b = 0 \text{ if } a_{n+2} > 0$$

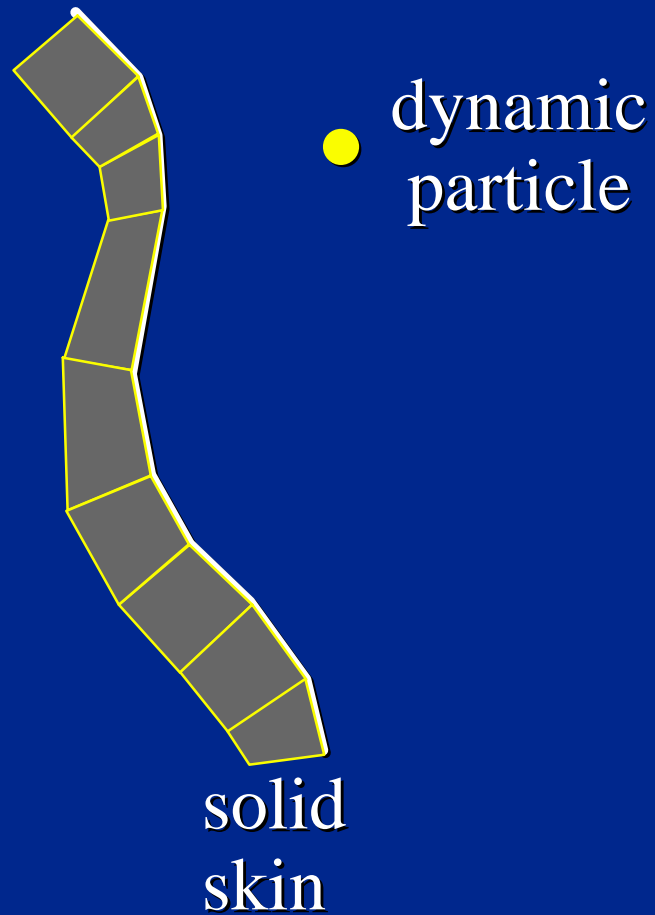
# Rigid Bodies: Same Thing!



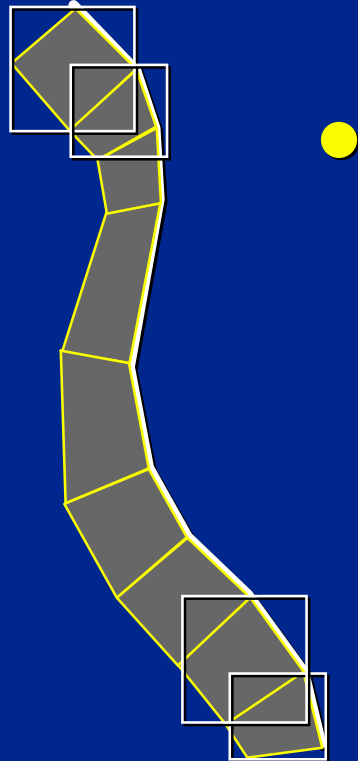
# Cloth/Fur Collision Detection



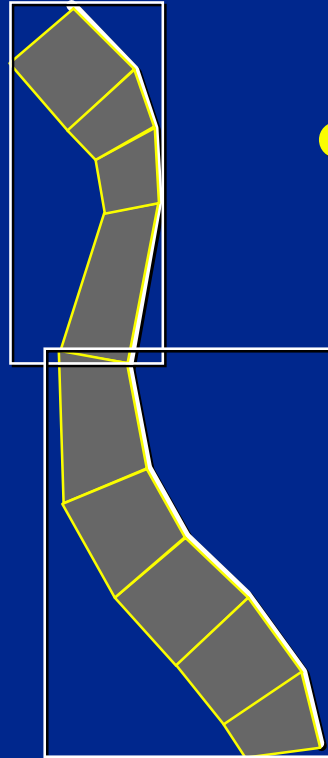
# Cloth/Fur Collision Detection



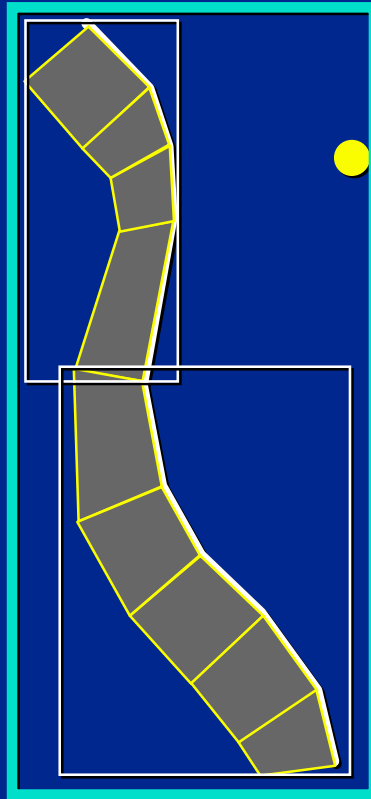
# Leaf-level Bounding Boxes



# Mid-level Bounding Boxes



# Root-level Bounding Box



# Cloth/Fur: Establishing Contacts



- New contacts: generally inside
- Too many for partial time steps.
- Gradual correction—bad.
- Arbitrary displacements—worse.
- Solution: combine information about displacements with implicit step method.

# Cloth/Fur: Maintaining Constraints

- Penalty force—cloth/cloth contact.
- Exact constraints for cloth/solids:
  - Must work with conjugate-gradient algorithm
  - Lagrange multipliers?
  - Change of variables?