Rigid Body Dynamics (I)

COMP259 March 28, 2006

Nico Galoppo von Borries

From Particles to Rigid Bodies

- Particles
  - No rotations
  - Linear velocity v only
- Rigid bodies
  - Body rotations
  - Linear velocity v
  - Angular velocity ω

Outline

- Rigid Body Preliminaries
  - Coordinate system, velocity, acceleration, and inertia
- State and Evolution
- Quaternions
- Collision Detection and Contact Determination
- Colliding Contact Response

Coordinate Systems

- Body Space (Local Coordinate System)
  - bodies are specified relative to this system
  - center of mass is the origin (for convenience)
    - We will specify body-related physical properties (inertia, …) in this frame

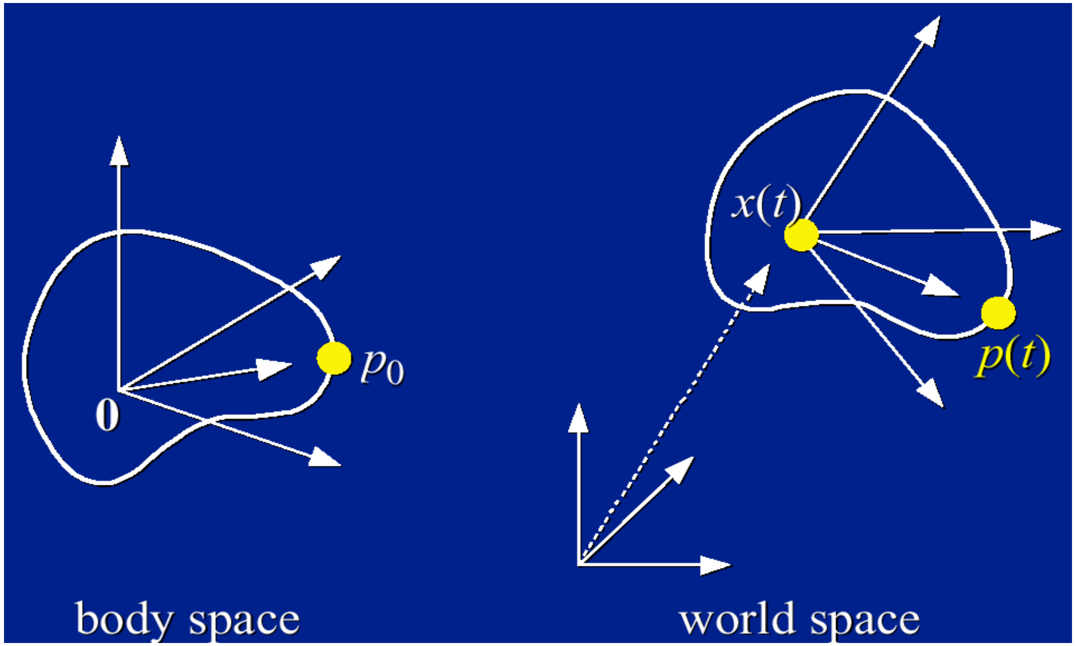
Coordinate Systems

- World Space
  - bodies are transformed to this common system

$$p(t) = R(t)\ p_0 + x(t)$$

  - x(t) represents the position of the body center
  - R(t) represents the orientation
    - Alternatively, use *quaternion* representation

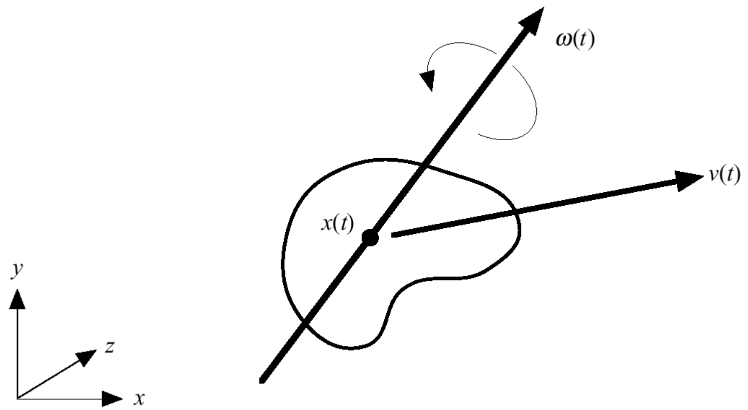
Coordinate Systems



Meaning of R(t): columns represent the coordinates of the body space base vectors (1,0,0), (0,1,0), (0,0,1) in world space.

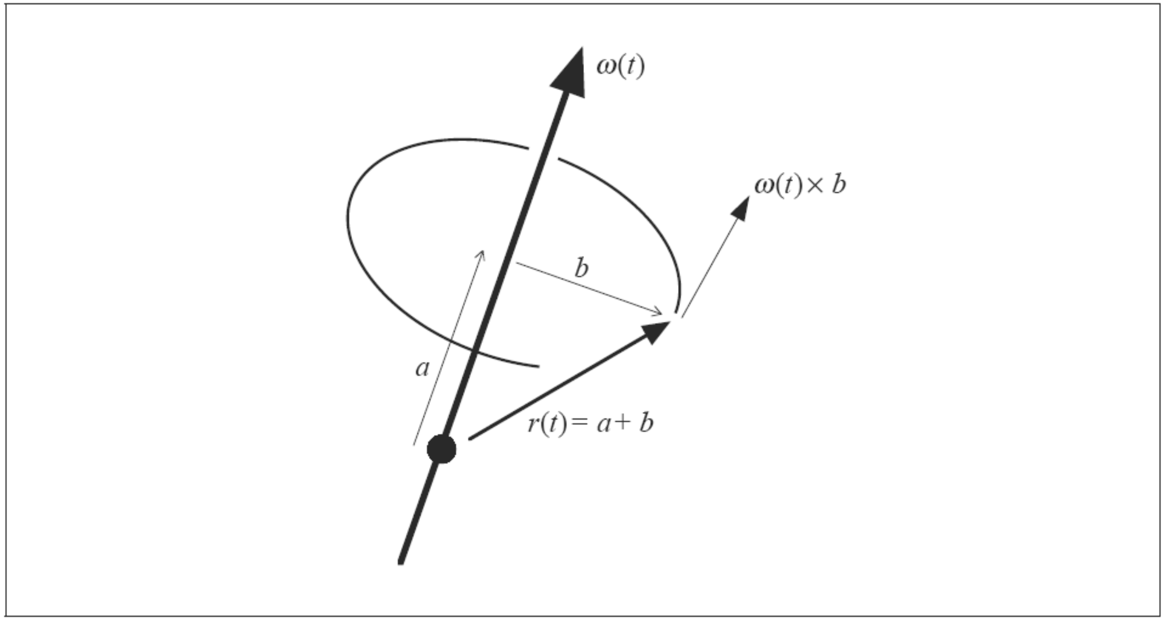
Kinematics: Velocities

- How do x(t) and R(t) change over time?
- Linear velocity v(t) = dx(t)/dt is the same:
  - Describes the velocity of the center of mass (m/s)
- Angular velocity $\omega(t)$ is new!
  - Direction is the axis of rotation
  - Magnitude is the angular velocity about the axis (degrees/time)
  - There is a simple relationship between R(t) and $\omega(t)$



Kinematics: Velocities

Then



$$\dot{R} = \left(\omega(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \omega(t) \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \omega(t) \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right)$$

Angular Velocities

- $\mathbf{R}(t)$ and $\omega(t)$ are related by:

$$\frac{d}{dt}\mathbf{R}(t) = \begin{pmatrix} 0 & -\omega_z(t) & \omega_y(t) \\ \omega_z(t) & 0 & -\omega_x(t) \\ -\omega_y(t) & \omega_x(t) & 0 \end{pmatrix} \mathbf{R}(t)$$

$$= \omega(t)^* \mathbf{R}(t)$$

Dynamics: Accelerations

- How do v(t) and dR(t)/dt change over time?
- First we need some more machinery
  - Forces and Torques
  - Momentums
  - Inertia Tensor
- Simplify equations by formulating accelerations terms of *momentum derivatives* instead of velocity derivatives

Forces and Torques

- External forces $F_i(t)$ act on particles
  - Total external force $F=\sum F_i(t)$
- Torques depend on distance from the center of mass:

$$\tau_i (t) = (r_i(t) - x(t)) \pounds F_i(t)$$

  - Total external torque

$$\tau = \sum ((r_i(t)-x(t)) \pounds F_i(t)$$

- F(t) doesn't convey any information about where the various forces act
- $\tau(t)$ does tell us about the *distribution* of forces

Linear Momentum

- Linear momentum $P(t)$ lets us express the effect of total force $F(t)$ on body (simple, because of conservation of energy):

  $$F(t) = dP(t)/dt$$

- Linear momentum is the product of mass and linear velocity
  - $P(t) = \sum m_i dr_i(t)/dt$
    $= \sum m_i v(t) + \omega(t)$ **£** $\sum m_i (r_i(t) - x(t))$
- But, we work in body space:
  - $P(t) = \sum m_i v(t) = Mv(t)$ (linear relationship)
  - Just as if body were a particle with mass M and velocity v(t)
  - Derive v(t) to express acceleration:

    $$dv(t)/dt = M^{-1}\, dP(t)/dt$$

- Use P(t) instead of v(t) in state vectors

Angular momentum

- Same thing, angular momentum L(t) allows us to express the effect of total torque τ(t) on the body:

  L♉☎✆🖫 ¿t

- Similarily, there is a linear relationship between momentum and velocity:

  L ☎✆🖫 | ¡ 🗀! t

  – I(t) is inertia tensor, plays the role of mass

- Use L(t) instead of ω(t) in state vectors

Inertia Tensor

- 3x3 matrix describing how the shape and mass distribution of the body affects the relationship between the angular velocity and the angular momentum $I(t)$
- Analogous to mass – rotational mass
- We actually want the inverse $I^{-1}(t)$ for computing $\omega(t)=I^{-1}(t)L(t)$

Inertia Tensor

$$\begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

Bunch of volume integrals:

Ixx = $\int_{B_i} (y^2 + z^2)\ dV$ Iyy = $\int_{B_i} (z^2 + x^2)\ dV$ Izz = $\int_{B_i} (x^2 + y^2)\ dV$

Ixy = Iyx = $\int_{B_i} xy\ dV$ Ixz = Izx = $\int_{B_i} zx\ dV$ Iyz = Izy = $\int_{B_i} yz\ dV$

Inertia Tensor

- Compute I in body space I_{body} and then transform to world space as required
  - $I(t)$ varies in world space, but I_{body} is constant in body space for the entire simulation
- $I(t) = R(t)\, I_{body}\, R^{-1}(t) = R(t)\, I_{body}\, R^{T}(t)$
- $I^{-1}(t) = R(t)\, I_{body}^{-1}\, R^{-1}(t) = R(t)\, I_{body}^{-1}\, R^{T}(t)$
- Intuitively: transform $\omega(t)$ to body space, apply inertia tensor in body space, and transform back to world space

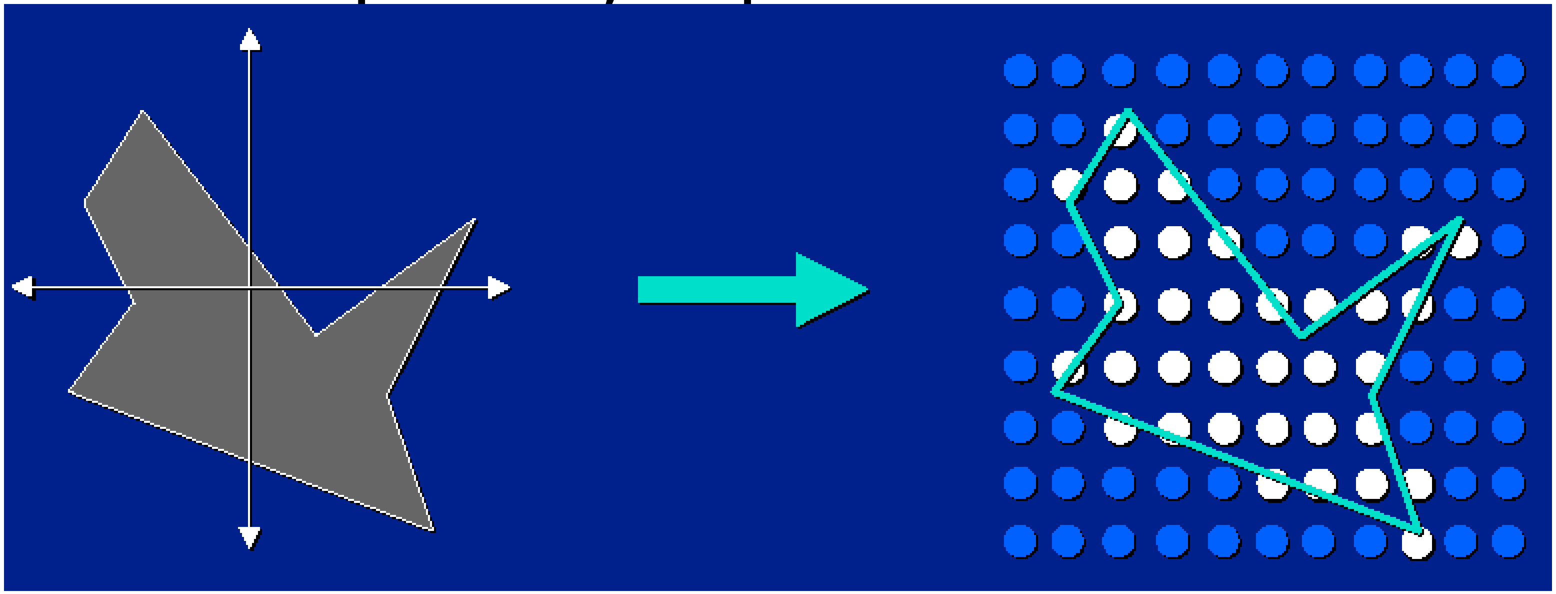
Computing I_{body}^{-1}

- There exists an orientation in body space which causes I_{xy}, I_{xz}, I_{yz} to all vanish
  - Diagonalize tensor matrix, define the eigenvectors to be the local body axes
  - Increases efficiency and trivial inverse
- Point sampling within the bounding box
- Projection and evaluation of Greene's thm.
  - Code implementing this method exists
  - Refer to Mirtich's paper at

    http://www.acm.org/jgt/papers/Mirtich96

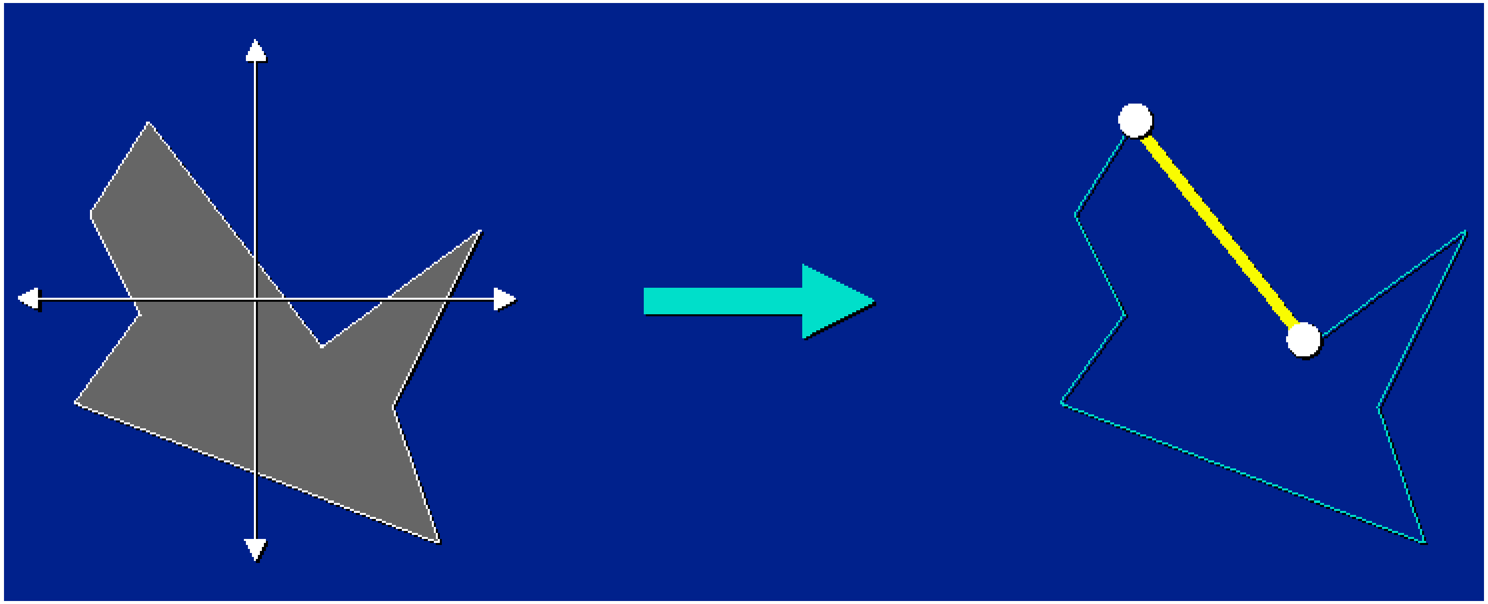
Approximation w/ Point Sampling

- Pros: Simple, fairly accurate, no B-rep needed.
- Cons: Expensive, requires volume test.



Use of Green's Theorem

- Pros: Simple, exact, no volumes needed.
- Cons: Requires boundary representation.



Outline

- Rigid Body Preliminaries
- State and Evolution
  - Variables and derivatives
- Quaternions
- Collision Detection and Contact Determination
- Colliding Contact Response

New State Space

$$\mathbf{X}(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} \rightarrow \text{Spatial information} \\ \left.\begin{array}{l} \\ \\ \end{array}\right\} \rightarrow \text{Velocity information} \end{array}$$

v(t) replaced by linear momentum P(t)

ω(t) replaced by angular momentum L(t)

Size of the vector: (3+9+3+3)N = 18N

Taking the Derivative

$$\frac{d}{dt}\mathbf{X}(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \omega(t)^{*}R(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

Conservation of momentum (P(t), L(t)) lets us express the accelerations in terms of forces and torques.

Simulate: next state computation

- From *X(t)* certain quantities are computed

$$\begin{cases} I^{-1}(t) = R(t)\, I_{body}^{-1}\, R^T(t) \\ v(t) = P(t) / M \\ \omega(t) = I^{-1}(t)\, L(t) \end{cases}$$

- We cannot compute the state of a body at all times but must be content with a finite number of discrete time points, ***assuming that the acceleration is continuous***
- Use your favorite ODE solver to solve for the new state X(t), given previous state X(t-Δt) and applied forces F(t) and τ(t)

$$X(t) \leftarrow \text{Solver::Step}(X(t-\Delta t), F(t), \tau(t))$$

Simple simulation algorithm

X **Ã** InitializeState()

For $t=t_0$ to t_{final} with step Δ t

  ClearForces(F(t), τ(t))

  AddExternalForces(F(t), τ(t))

  X_{new} **Ã** Solver::Step(X, F(t), τ(t))

  X **Ã** X_{new}

  t **Ã** t + Δt

End for

Outline

- Rigid Body Preliminaries
- State and Evolution
- Quaternions
  - Merits, drift, and re-normalization
- Collision Detection and Contact Determination
- Colliding Contact Response

Unit Quaternion Merits

- Problem with rotation matrices: numerical drift
- $R(t) = dR(t)/dt * \Delta t \ R(t_{-1})R(t_{-2})R(t_{-3})\cdots$
- A rotation in 3-space involves 3 DOF
- Unit quaternions can do it with 4
- Rotation matrices R(t) describe a rotation using 9 parameters
- Drift is easier to fix with quaternions
  - renormalize

Unit Quaternion Definition

- $q = [s,v]$ -- s is a scalar, v is vector
- A rotation of θ about a unit axis u can be represented by the unit quaternion:

$$[\cos(\theta/2), \sin(\theta/2) * u]$$

- $\| [s,v] \| = 1$ -- the length is taken to be the Euclidean distance treating $[s,v]$ as a 4-tuple or a vector in 4-space

Unit Quaternion Operations

- Multiplication is given by:

$$[s_1, v_1][s_2, v_2] = [s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2]$$

- dq(t)/dt = ½ ω(t) q(t) = [0, ½ ω(t)] q(t)

- R =

$$\begin{pmatrix} 1 - 2v_y^2 - 2v_z^2 & 2v_x v_y - 2s v_z & 2v_x v_z + 2s v_y \\ 2v_x v_y + 2s v_z & 1 - 2v_x^2 - 2v_z^2 & 2v_y v_z - 2s v_x \\ 2v_x v_z - 2s v_y & 2v_y v_z + 2s v_x & 1 - 2v_x^2 - 2v_y^2 \end{pmatrix}$$

Unit Quaternion Usage

- To use quaternions instead of rotation matrices, just substitute them into the state as the orientation (save 5 variables)

$$\mathbf{X}(t) = \frac{d}{dt}\begin{pmatrix} x(t)\\ R(t)\\ P(t)\\ L(t)\end{pmatrix} = \begin{pmatrix} v(t)\\ \frac{1}{2}\omega(t)\, q(t)\\ F(t)\\ \tau(t)\end{pmatrix} = \begin{pmatrix} \frac{P(t)}{M}\\ \frac{1}{2}\left(I^{-1}(t)\, L(t)\right) q(t)\\ F(t)\\ \tau(t)\end{pmatrix}$$

where

$R =$ `QuatToMatrix(`$q(t)$`)`

$I^{-1}(t) = R\, I_{body}^{-1}\, R^T$

Outline

- Rigid Body Preliminaries
- State and Evolution
- Quaternions
- Collision Detection and Contact Determination
  - Contact classification
  - Intersection testing, bisection, and nearest features
- Colliding Contact Response

What happens when bodies collide?

- Colliding
  - Bodies *bounce* off each other
  - Elasticity governs 'bounciness'
  - Motion of bodies changes ***discontinuously*** within a discrete time step
  - 'Before' and 'After' states need to be computed
- In contact
  - Resting
  - Sliding
  - Friction

Detecting collisions and response

- Several choices
  - Collision detection: which algorithm?
  - Response: Backtrack or allow penetration?
- Two primitives to find out if response is necessary:
  - Distance(A,B): cheap, no contact information ! fast intersection query
  - Contact(A,B): expensive, with contact information

Distance(A,B)

- Returns a value which is the minimum distance between two bodies
- Approximate may be ok
- Negative if the bodies intersect
- Convex polyhedra
  - Lin-Canny and GJK -- 2 classes of algorithms
- Non-convex polyhedra
  - much more useful but hard to get distance fast
  - PQP/RAPID/SWIFT++
- Remark: most of these algorithms give inaccurate information if bodies intersect, except for DEEP

Contacts(A,B)

- Returns the set of features that are nearest for disjoint bodies or intersecting for penetrating bodies
- Convex polyhedra
  - LC & GJK give the nearest features as a bi-product of their computation – only a single pair. Others that are equally distant may not be returned.
- Non-convex polyhedra
  - much more useful but much harder problem especially contact determination for disjoint bodies
  - Convex decomposition: SWIFT++

Prereq: Fast intersection test

- First, we want to make sure that bodies will intersect at next discrete time instant
- If not:
  - X_{new} is a valid, non-penetrating state, proceed to next time step
- If intersection:
  - Classify contact
  - Compute response
  - Recompute new state

Bodies intersect ! classify contacts

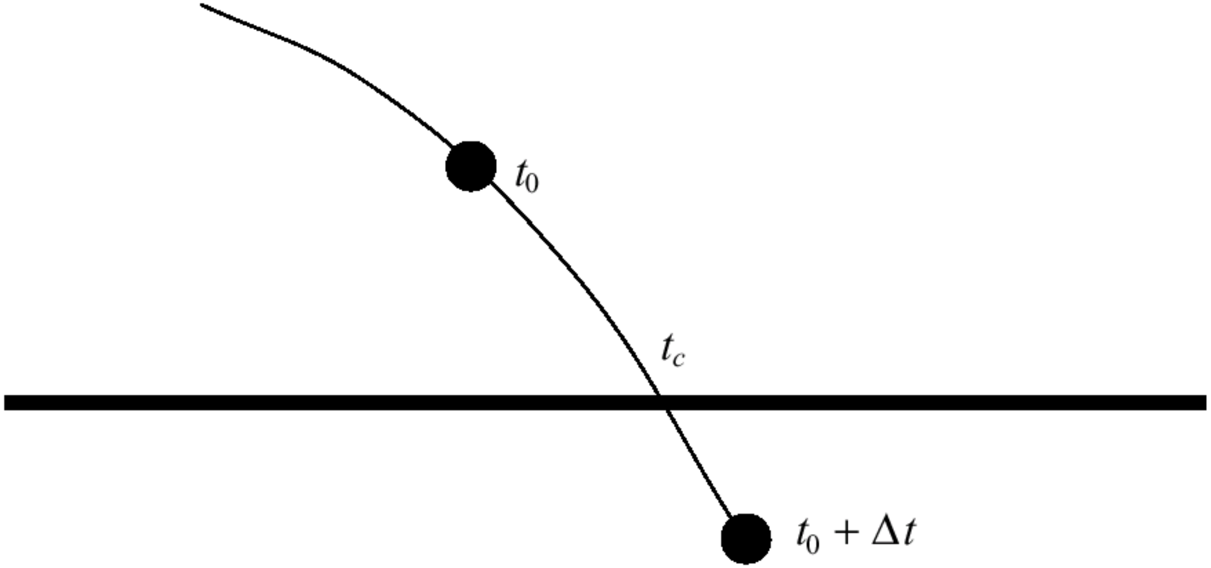
- Colliding contact (Today)
  - $v_{rel} < -\varepsilon$
  - Instantaneous change in velocity
  - Discontinuity: requires restart of the equation solver
- Resting contact (Thursday)
  - $-\varepsilon < v_{rel} < \varepsilon$
  - Gradual contact forces avoid interpenetration
  - No discontinuities
- Bodies separating
  - $v_{rel} > \varepsilon$
  - No response required

Collisiding contacts

- At time t_i, body A and B intersect and $v_{rel} < -\varepsilon$
- Discontinuity in velocity: need to stop numerical solver
- Find time of collision t_c
- Compute new velocities $v^+(t_c) \rightarrow X^+(t)$
- Restart ODE solver at time t_c with new state $X^+(t)$

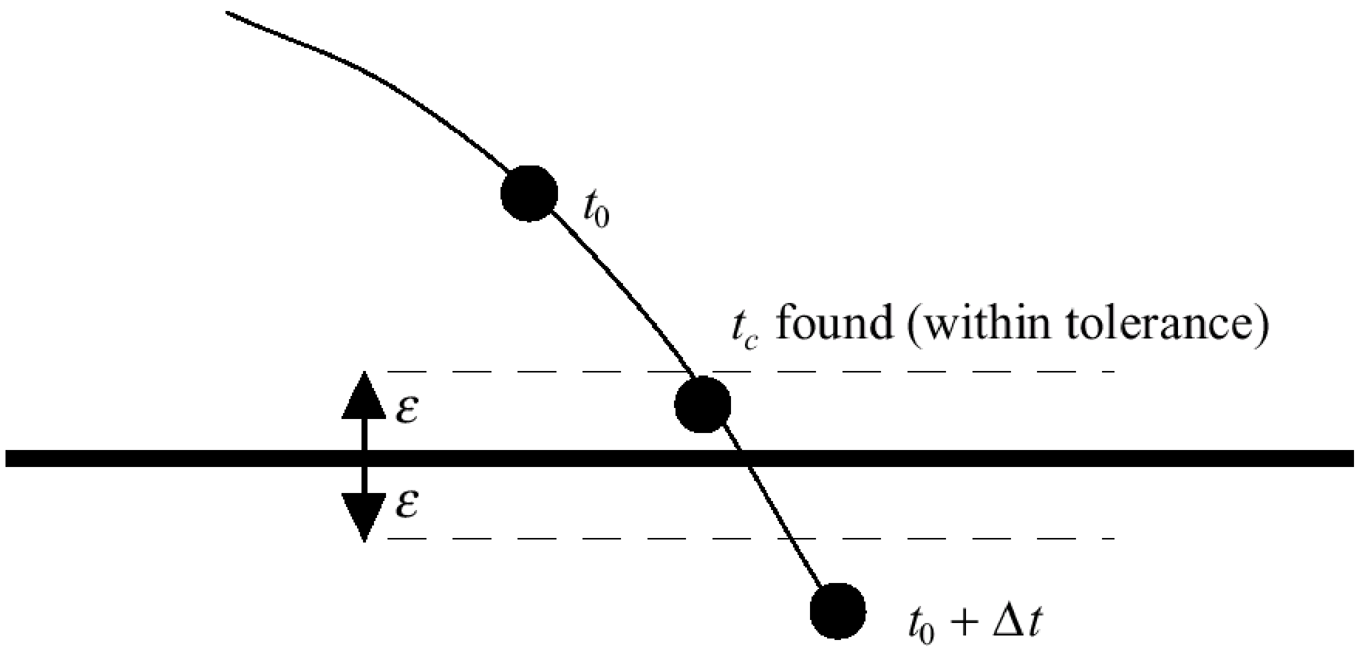
Time of collision

- We wish to compute when two bodies are “close enough” and then apply contact forces
- Let’s recall a particle colliding with a plane



Time of collision

- We wish to compute t_c to some tolerance



Time of collision

1. A common method is to use **bisection search** until the distance is positive but less than the tolerance
2. Use **continuous collision detection** (cf.Dave Knott's lecture)
3. $\mathbf{t_c}$ not always needed
   ! **penalty-based methods**

findCollisionTime(X,t,Δt)

```
for each pair of bodies (A,B) do
 Compute_New_Body_States(Scopy, t, H);
 hs(A,B) = H;    // H is the target timestep
 if Distance(A,B) < 0 then
     try_h = H/2;   try_t = t + try_h;
     while TRUE do
         Compute_New_Body_States(Scopy, t, try_t - t);
         if Distance(A,B) < 0 then
                 try_h /= 2;   try_t -= try_h;
         else if Distance(A,B) < ε then
                 break;
         else
                 try_h /= 2;   try_t += try_h;
     hs(A,B) = try_t – t;
h = min( hs );
```

Outline

- Rigid Body Preliminaries
- State and Evolution
- Quaternions
- Collision Detection and Contact Determination
- Colliding Contact Response
  - Normal vector, restitution, and force application

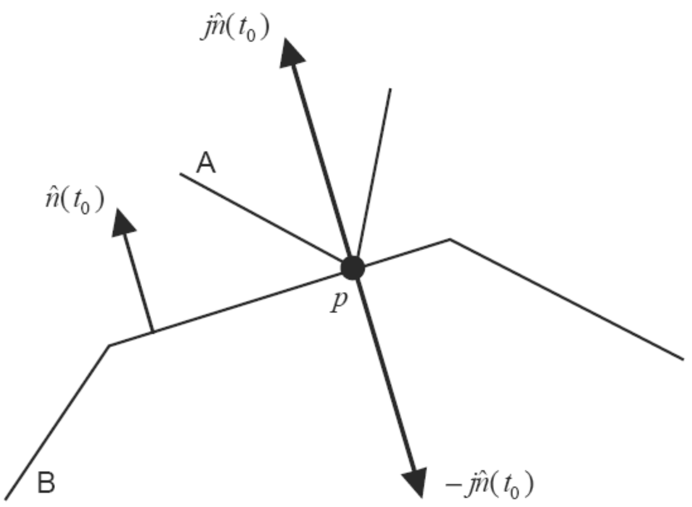
What happens upon collision

- Impulses provide instantaneous changes to velocity, unlike forces

$$\Delta(P) = J$$

- We apply impulses to the colliding objects, at the point of collision
- For frictionless bodies, the direction will be the same as the normal direction:

$$J = j^T n$$



Colliding Contact Response

- Assumptions:
  - Convex bodies
  - Non-penetrating
  - Non-degenerate configuration
    - edge-edge or vertex-face
    - appropriate set of rules can handle the others
- Need a contact unit normal vector
  - Face-vertex case: use the normal of the face
  - Edge-edge case: use the cross-product of the direction vectors of the two edges

Colliding Contact Response

- Point velocities at the nearest points:

$$\dot{p}_a(t_0) = v_a(t_0) + \omega_a(t_0) \times (p_a(t_0) - x_a(t_0))$$

$$\dot{p}_b(t_0) = v_b(t_0) + \omega_b(t_0) \times (p_b(t_0) - x_b(t_0))$$

- Relative contact normal velocity:

$$v_{rel} = \hat{n}(t_0) \cdot (\dot{p}_a(t_0) - \dot{p}_b(t_0))$$

Colliding Contact Response

- We will use the empirical law of frictionless collisions: $v_{rel}^{+} = -\epsilon v_{rel}^{-}$
  - Coefficient of restitution ϵ [0,1]
    - $\epsilon = 0$ -- bodies stick together
    - $\epsilon = 1$ – loss-less rebound
- After some manipulation of equations...

$$j = \frac{-(1+\epsilon)v_{rel}^{-}}{\frac{1}{M_a} + \frac{1}{M_b} + \hat{n}(t_0) \cdot \left(I_a^{-1}(t_0)\left(r_a \times \hat{n}(t_0)\right)\right) \times r_a + \hat{n}(t_0) \cdot \left(I_b^{-1}(t_0)\left(r_b \times \hat{n}(t_0)\right)\right) \times r_b}$$

Apply_BB_Forces()

- For colliding contact, the computation can be local

```
for each pair of bodies (A,B) do
        if Distance(A,B) < ε then
  Cs = Contacts(A,B);
  Apply_Impulses(A,B,Cs);
```

Apply_Impulses(A,B,Cs)

- The impulse is an instantaneous force – it changes the velocities of the bodies instantaneously: $\Delta v = J / M$

```
for each contact in Cs do
    Compute n
    Compute j
    J = j^T n
    P(A) +=  J
    L(A) += (p - x(t)) x J
    P(B) -=  J
    L(B) -= (p - x(t)) x J
```

Simulation algorithm with Collisions

X Ã InitializeState()

For $t=t_0$ to t_{final} with step Δt

ClearForces(F(t), τ(t))
AddExternalForces(F(t), τ(t))
X_{new} Ã Solver::Step(X, F(t), τ(t), t, Δt)

t Ã findCollisionTime()
X_{new} Ã Solver::Step(X, F(t), τ(t), t, Δt)

C Ã Contacts(X_{new})
while (!C.isColliding())
    applyImpulses(X_{new})
end if

X Ã X_{new}
t Ã t + Δt
End for

Penalty Methods

- If we don't look for time of collision t_c then we have a simulation based on penalty methods: the objects are allowed to intersect.
- **Global** or **local** response
  - **Global**: The penetration depth is used to compute a spring constant which forces them apart (dynamic springs)
  - **Local:** Impulse-based techniques

Global penalty based response

Global contact force computation

for each pair of bodies (A,B) do

if Distance(A,B) < ε then

Flag_Pair(A,B);

Solve For_Forces(flagged pairs);

Apply_Forces(flagged pairs);

Local penalty based response

Local contact force computation

```
for each pair of bodies (A,B) do
if Distance(A,B) < ε then
        Cs = Contacts(A,B);
        Apply_Impulses(A,B,Cs);
```

References

- D. Baraff and A. Witkin, “Physically Based Modeling: Principles and Practice,” Course Notes, SIGGRAPH 2001.
- B. Mirtich, “Fast and Accurate Computation of Polyhedral Mass Properties,” Journal of Graphics Tools, volume 1, number 2, 1996.
- D. Baraff, “Dynamic Simulation of Non-Penetrating Rigid Bodies”, Ph.D. thesis, Cornell University, 1992.
- B. Mirtich and J. Canny, “Impulse-based Simulation of Rigid Bodies,” in Proceedings of 1995 Symposium on Interactive 3D Graphics, April 1995.
- B. Mirtich, “Impulse-based Dynamic Simulation of Rigid Body Systems,” Ph.D. thesis, University of California, Berkeley, December, 1996.
- B. Mirtich, “Hybrid Simulation: Combining Constraints and Impulses,” in Proceedings of First Workshop on Simulation and Interaction in Virtual Environments, July 1995.
- COMP259 Rigid Body Simulation Slides, Chris Vanderknyff 2004