Rigidbody Dynamics

Written by: Örs Mándli

1. Physics Quantites

Position

$$x(t)$$

- just a 3D vector

Linear Velocity

$$v(t) = \dot{x}(t).$$

- the derivative of the position

Angular Velocity

$$\omega(t)$$

- direction: the vector to rotate around
- length: the amount of the rotation

Rotation Matrix and Orientation Quaternion

$$R(t) = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix}$$ and $$q(t) = [s, v]$$

- the columns of the matrix are the basis vectors after the rotation
- q = s + $v_x i$ + $v_y j$ + $v_z k$; i,j,k are basically the complex basis vectors

Derivative of the above Matrix and Quaternion

$$\dot{R}(t) = \omega(t)^* R(t).$$ and $$\dot{q}(t) = \frac{1}{2}\omega(t)q(t)$$

- this is just a cross product, but we need to turn omega into a matrix to make it work (it will be omega*)
- the star version (3x3 matrix) of the vector 'a':

$$\begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}$$

- a x b = a* b

Mass of a Body

$$M = \sum_{i=1}^{N} m_i.$$

- sum of the masses of all particles that make up the body

The Intertia Tensor

$$I(t) = \sum \begin{pmatrix} m_i(r'^2_{iy} + r'^2_{iz}) & -m_i r'_{ix} r'_{iy} & -m_i r'_{ix} r'_{iz} \\ -m_i r'_{iy} r'_{ix} & m_i(r'^2_{ix} + r'^2_{iz}) & -m_i r'_{iy} r'_{iz} \\ -m_i r'_{iz} r'_{ix} & -m_i r'_{iz} r'_{iy} & m_i(r'^2_{ix} + r'^2_{iy}) \end{pmatrix}$$

- where: $r'_i = r_i(t) - x(t)$
- note that the inertia is the angular equivalent of the mass, however, while mass is constant in time, the inertia is not

$$I(t) = R(t) I_{body} R(t)^T.$$

- where: $I_{body} = \sum m_i((r_{0i}^T r_{0i})\mathbf{1} - r_{0i} r_{0i}^T)$ and R(t) is the rotaion matrix
- so the inertia tensor does not have to be recalculated all the time, it's enough to calculate it in the beginning and use the rotation matrix later

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

- I_{body}(t) and I_{body}^{-1}(t) are constant in time and can be calculated just like I(t) but with an integral instead of the sum

Velocity of a Particle

$$\dot{r}_i(t) = \omega(t) \times (r_i(t) - x(t)) + v(t).$$

- so the velocity can be split into two parts:
  - the linear component: $v(t)$
  - the angular component: $\omega \times (r_i(t) - x(t))$

Center of Mass

$$\frac{\sum m_i r_i(t)}{M}$$

Force and Torque

- external torque acting on a particle (def):
  $$\tau_i(t) = (r_i(t) - x(t)) \times F_i(t).$$
- total force on a body:
  $$F(t) = \sum F_i(t)$$
- total torque on a body:
  $$\tau(t) = \sum \tau_i(t) = \sum (r_i(t) - x(t)) \times F_i(t).$$

Linear Momentum and Force

$$P(t) = Mv(t).$$

$$\dot{P}(t) = F(t)$$

Angular Momentum and Torque

$$L(t) = I(t)\omega(t)$$

$$\dot{L}(t) = \tau(t)$$

Angular Acceleration

$$\dot{\omega}(t) = I^{-1}(t)(L(t) \times \omega(t) + \dot{L}(t))$$

- note that even if torque [L_{dot}(t)] is zero, the angular acceleration can still be non-zero
- it happens when the angular momentum and and the angular velocities point in different directions
- this is the case when the body has a rotational velocity axis that is not an axis of symmetry for the body

Kinetic Energy

$$T = \sum \tfrac{1}{2} m_i \dot{r}_i^T \dot{r}_i = \tfrac{1}{2}(v^T M v + \omega^T I \omega)$$

- so thekinetic energy can be decomposed into two terms:
  - the linear term: $\frac{1}{2} v^T M v$
  - the angular term: $\frac{1}{2} \omega^T I \omega$
- note that the kinetic energy should remain constant

2. Math

Rigid Body Equations of Motion

- The state of a body:

$$\mathbf{Y}(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix}$$

- the derivative of the state of a body:

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

  - where:

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

Quaternions

- just a better way to represent rotations than matrices
- 4D vector instead of 3x3 matrix
- a quaternion looks like this:

$$q = s + v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}$$

- q will be represented as: $[s, v]$
- a rotation of θ radians about a unit axis u as a quaterion:

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

- the composite of rotations (q2 after q1) is the product q2*q1
- the derivative of the quaterion q:

$$\dot{q}(t) = \tfrac{1}{2}\omega(t)q(t)$$

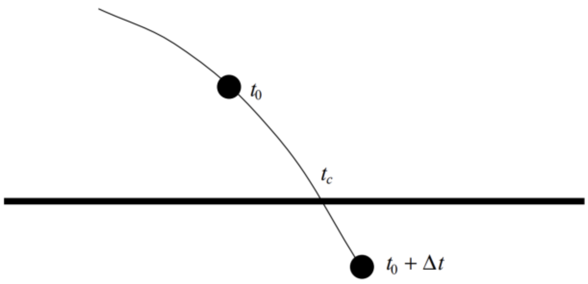
- where $\omega(t)q(t)$ is a shorthand for $[0, \omega(t)]_* \, q(t)$
- quaternion converted to a 3x3 rotation matrix:

$$\begin{pmatrix} 1 - 2v_y^2 - 2v_z^2 & 2v_xv_y - 2sv_z & 2v_xv_z + 2sv_y \\ 2v_xv_y + 2sv_z & 1 - 2v_x^2 - 2v_z^2 & 2v_yv_z - 2sv_x \\ 2v_xv_z - 2sv_y & 2v_yv_z + 2sv_x & 1 - 2v_x^2 - 2v_y^2 \end{pmatrix}$$

3. Nonpenetration Constraints

Basics

- the goal is to avoid any penetration between bodies
- so if a penetration happens, we need to go back in time to exactly (within some tolerance) where the collision happened (t_c on the picture below)



(inter-penetration detected)

- after going back, we should not have any inter-penetrations, only contacts

Contacts

- when two bodies are in contact at some point p
- either vertex-face or edge-edge contacts are considered
- there are 2 types of contacts: colliding and resting contacts
- check the relative velocity projected onto the normal:
  - if (0 <): the bodies are separating (no action needed)
  - if (0 =): resting contact
  - if (0 >): colliding contact

Colliding Contact

- when the bodies have a velocity towards each other
- instant velocity change is needed (impulse instead of force)
- impulse changes both the linear and angular velocities (impulsive torque)
- no friction -> impulse is in the normal direction: $j * n(t_0)$
- opposite, but equal size impulse is applied to A and B
- parameter: coefficient of restitution (0..1) -> bounciness

Resting Contact

- when two bodies are resting on one another
- need to apply force to prevent inter-penetration
- $f_i*n_i(t_0)$ force is applied at the ith contact
- f_is all have to be calculated at the same time, since the forces may influence bodies at other contacts

Collision Detection

- collision detection can be split into two phases: broad phase and narrow phase collision detection
- broad phase:

    - check if it is possible for two bodies to have collided (might be false positive, but very quick)
- narrow phase:
    - check if two bodies have indeed collided and extract the collision information

4. Implementation

Rigidbody Class

- constant quantities:
  - inverse mass
  - starting inertia
  - inverse of the starting inertia
- state variables:
  - position
  - rotation matrix/orientation quatertion
  - linear momentum
  - angular momentum
- derived quantities:
  - current inertia
  - linear velocity
  - angular velocity
- computed quantities:
  - force
  - torque

for fixed rigid bodies: inverse mass and inverse inertia is 0

Contact Class

- reference to rigidbody containing vertex (A)
- reference to rigidbody containing face (B)
- world-space vertex position
- normal pointing outwards from the face (B->A)
- edge direction for A
- edge direction for B
- boolean whether vertex-face or edge-edge contact

Penetration Prevention

- in case of a collision, we have to find the exact time when it happened, and go back to it
- one easy, but not too fast alg - bisection:
  - t_0: no inter-penetrtion
  - t_0 + dt: inter-penetrtion detected
  - t_c: time of collision (this has to be calculated)
  - the alg is just a binary search to find t_c between t_0 and t_0 + dt (with some epsilon tolerance)
- there faster algs that work by predicting t_c (SIGGRAPH)
- now only pregress ODE up to the collision time

Collision Detection - Broad Phase

- we can use AABBs or bounding spheres to quickly sort out bodies that can't have collided
- we can use the sort and sweep algorithm in 3D

Collision Detection - Narrow Phase

- we check collisions between convex polyhedras (no concaves!)

- we can say two bodies don't collide, if we can find a plane that separetas the two (all of their vertices lie on the other sides of the plane)
- such a plane is called a witness
- a separating plane can be 2 things:
    - contains one face (this face is the defining face)
    - contains one edge from body A, and is parallel to another edge from body B (these are the defining edges)
- we can cache witnesses between timesteps for better performance:
    - in the next frame check if the witness is still valid
    - if not, search for another (adjecent faces/edges are more likely to be witnesses)

Colliding Contact Resolution

- first, lets calculate the relative velocity:

  $$v_{rel} = \hat{n}(t_0) \cdot (\dot{p}_a(t_0) - \dot{p}_b(t_0))$$
- if it's negative, we have a colliding contact
- we need to apply an impulse:

  $$J = j\hat{n}(t_0)$$

  where n is the normal of the separating plane and j is a constant
- n can be calculated based on the type of contact:
    - vertex-face: n is the normal of the face
    - edge-edge: n is the cross product of the edges
- j can be calculate as follows:

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

  where $r_a = p - x_a(t_0)$, $r_b = p - x_b(t_0)$, e is the restitution

Resting Contact Resolution

- let $d_i(t)$ be the distance of the bodies near the contact point

  $$d_i(t) = \hat{n}_i(t) \cdot (p_a(t) - p_b(t))$$
- f_is are subjects to 3 conditions:
    - prevent inter-penetration: $\ddot{d}_i(t_0) \geq 0$
    - must be repulsive (no glueing): $f_i \geq 0$
    - must be 0 if bodies are separating: $f_i \ddot{d}_i(t_0) = 0$
- lets express $d_{i}..(t_0)$ as a function of the unknown f_is:

  $$\ddot{d}_i(t_0) = a_{i1}f_1 + a_{i2}f_2 + \cdots + a_{in}f_n + b_i$$
- the system of equations that needs to be solved:

  $$\begin{pmatrix} \ddot{d}_1(t_0) \\ \vdots \\ \ddot{d}_n(t_0) \end{pmatrix} = \mathbf{A} \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} + \begin{pmatrix} b_i \\ \vdots \\ b_n \end{pmatrix}$$
- now A and b can be calculated

lastly, solve for f_is -> Quadratic Programming problem

ODE solver (Ordinary Differential Equation)

- solves ordinary differential equations
- should have a solve function with parameters:
  - initial state
  - start time
  - end time
  - pointer to a function that differentiates the state
- should return the new state in some way
- might need to be stopped at any point (in case of collision)

Simulation steps

- initialize bodies
- run simulation loop:
  - add external forces (gravity, wind)
  - broad-phase:
    - find all body pairs that might collide
  - narrow-phase:
    - search for inter-penetration
    - step back if needed
    - find all contacts
  - solve all colliding contacts
  - solve all resting contacts
  - calculate new state of the bodies using the ODE solver

5. Sources

1. https://www.cs.cmu.edu/~baraff/sigcourse/notesd1.pdf
2. https://www.cs.cmu.edu/~baraff/sigcourse/notesd2.pdf
3. https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/previousinformation/physics4collisiondetection/2017%20Tutorial%204%20-%20Collision%20Detection.pdf