UC San Diego

Physical Simulation

Fanbo Xiang

CSE291-G00 - Spring 2020

Video

Let’s start with a video.

Topics

- **Newtonian Dynamics and Integration**
  - Point mass
  - Rigidbody
  - Numerical Integration
- Rigidbody Constraints
  - Joint simulation
  - Contact, friction, and impact
- Collision Detection
  - Primitive shape collision
  - Broad and narrow phases
  - Convex shape collision

Physical Simulation for Point Mass

We have seen that Lagrangian framework is suitable for robot simulation. However, external force and collision handling are non-trivial in this framework. So, most physical simulator operates on Newtonian mechanics.

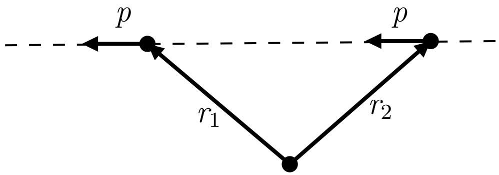
$\dot{\alpha} \triangleq d\alpha/dt$	time derivative
$\boldsymbol{x}, \boldsymbol{r}$	position
$\boldsymbol{v} = \dot{\boldsymbol{x}}$	velocity
$\boldsymbol{a} = \ddot{\boldsymbol{x}}$	acceleration
m	mass
$\boldsymbol{p} = m\boldsymbol{v}$	momentum
$\boldsymbol{f} = \dot{\boldsymbol{p}} = m\boldsymbol{a}$	force

Moment

(Note: Moment and Momentum are different things)

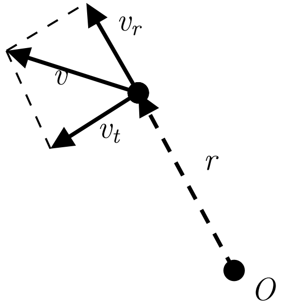
Given origin, position vector $\boldsymbol{r}$, some vector $\boldsymbol{V}$ at position $\boldsymbol{r}$, define moment of $\boldsymbol{V}$ as $\boldsymbol{r} \times \boldsymbol{V}$.

$\boldsymbol{L} = \boldsymbol{r} \times \boldsymbol{p}$	angular momentum
$\boldsymbol{\tau} = \dot{\boldsymbol{L}} = \boldsymbol{r} \times \boldsymbol{f}$	torque
$\boldsymbol{\omega} = \frac{\boldsymbol{r} \times \boldsymbol{v}}{\|\boldsymbol{r}\|_2^2}$	angular velocity



Rotational Inertia

$\boldsymbol{v}$ can be decomposed into tangential velocity $\boldsymbol{v}_t$ and radial velocity $\boldsymbol{v}_r$.



We take the object center O as the frame, then
$\boldsymbol{v}_r = 0, \boldsymbol{v}_t = v, \boldsymbol{v} = \boldsymbol{v}_t = \boldsymbol{\omega} \times \boldsymbol{r}$

Rotational Inertia

$$\boldsymbol{L} = \boldsymbol{r} \times \boldsymbol{p} = \boldsymbol{r} \times (m\boldsymbol{v}) = m\boldsymbol{r} \times (\boldsymbol{\omega} \times \boldsymbol{r})$$
$$= -m\boldsymbol{r} \times (\boldsymbol{r} \times \boldsymbol{\omega}) = -m[\hat{\boldsymbol{r}}][\hat{\boldsymbol{r}}]\boldsymbol{\omega}$$

Define rotational inertia $\boldsymbol{I}$ as

$$\boldsymbol{I} = -m[\hat{\boldsymbol{r}}][\hat{\boldsymbol{r}}]$$

Then,

$$\boldsymbol{L} = \boldsymbol{I}\boldsymbol{\omega}$$

Rotational Inertia

$$\boldsymbol{I} = \begin{bmatrix} m(r_y^2 + r_z^2) & -mr_xr_y & -mr_xr_z \\ -mr_xr_y & m(r_x^2 + r_z^2) & -mr_yr_z \\ -mr_xr_z & -mr_yr_z & m(r_y^2 + r_z^2) \end{bmatrix}$$

Rigidbody

Rigidbody is a system of particles (maybe infinitely many) whose relative positions are fixed.

$$m = \sum_i m_i (= \int \rho dV)$$ (total mass)

$$\boldsymbol{x}_{cm} = \frac{\int \rho \boldsymbol{r} dV}{\int \rho dV}$$ (center of mass)

$$\boldsymbol{I} = \begin{bmatrix} \int \rho(r_y^2 + r_z^2) dV & -\int \rho r_x r_y dV & -\int \rho r_x r_z dV \\ -\int \rho r_x r_y dV & \int \rho(r_x^2 + r_z^2) dV & -\int \rho r_y r_z dV \\ -\int \rho r_x r_z dV & -\int \rho r_y r_z dV & \int \rho(r_y^2 + r_z^2) dV \end{bmatrix}$$

(rotational inertia)

Note: Given uniform density, these integrals can be computed analytically for watertight mesh.

https://www.geometrictools.com/Documentation/PolyhedralMassProperties.pdf

Compute Volume Integration

Divergence theorem! Let $\boldsymbol{F}: \mathbb{R}^3 \to \mathbb{R}^3$

$$\int_V \nabla \cdot \boldsymbol{F} dV = \oint_S \boldsymbol{F} \cdot \boldsymbol{n} dS$$

An example: a term of $\boldsymbol{I}$,

$$-\rho \int_V r_y r_z dV$$

Let

$$\boldsymbol{F}(r_x, r_y, r_z) = \begin{bmatrix} r_x r_y r_z & 0 & 0 \end{bmatrix}^T$$

$$\nabla \cdot \boldsymbol{F} = r_y r_z$$

The integral becomes

$$\oint_S \boldsymbol{F} \cdot \boldsymbol{n} dS$$

Now we only need to do 2D integral over triangles.

Mass Properties

$\boldsymbol{I}$ is positive semi-definite (positive definite for objects with 3D volume). So it is often represented as its eigen decomposition.

$$\boldsymbol{I} = \boldsymbol{R}\boldsymbol{\Lambda}\boldsymbol{R}^T$$

where $\boldsymbol{R}$ is a rotation matrix, the columns are **principal axes**. The diagonal entries of Λ, (I_1, I_2, I_3) are **principal moments of inertia**.

$\boldsymbol{R}$ and $\boldsymbol{x}_{cm}$ form a 6D pose, which is called the pose of the center of mass.

$\{\boldsymbol{R}, \boldsymbol{x}_{cm}, m, I_1, I_2, I_3\}$ is called **mass properties** and fully determines the behavior of a rigidbody under external force.

Poll 1

Recall the mass properties

$$\{\boldsymbol{R}, \boldsymbol{x}_{cm}, m, I_1, I_2, I_3\}$$

Suppose a torque (and no net force) is applied to the rigidbody, a change in which of the following quantities will result in different object motion (choose multiple)?

A. $\boldsymbol{R}$
B. $\boldsymbol{x}_{cm}$
C. m
D. I_1, I_2, I_3

Poll 2

Suppose an object is moving in space (rotating and translating), which of the following quantities may change during the motion. (Assume all quantities are measured w.r.t. a static spatial frame)

A. $\boldsymbol{R}$

B. $\boldsymbol{x}_{cm}$

C. m

D. I_1, I_2, I_3

Newton-Euler Equations

Motion of rigidbody can be decomposed into linear motion of the center of mass and angular motion around the center of mass.

$$\boldsymbol{f} = m\boldsymbol{a} \qquad \text{(Linear motion)}$$

$$\boldsymbol{\tau} = \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{I}\dot{\boldsymbol{\omega}} \qquad \text{(angular motion)}$$

Note: even if there is no external force (thus no torque), if the object has non-zero angular velocity ω, then it can have angular acceleration $\dot{\omega}$.

From Physics to Simulation

Newton-Euler allows us to compute acceleration

$$\boldsymbol{f} = m\boldsymbol{a} \qquad \text{(Linear motion)}$$

$$\boldsymbol{\tau} = \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{I}\dot{\boldsymbol{\omega}} \qquad \text{(angular motion)}$$

How do we get position and velocity from acceleration? Through integration.

$$x = \int \dot{x} dt, \dot{x} = \int \ddot{x} dt$$

However, in reality, integration has to be computed numerically.

Rigidbody Simulation: integration

Forward Euler: $\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{v}_t + \boldsymbol{a}\Delta t, \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \boldsymbol{v}_t \Delta t$
Observe: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t \Delta t$

Semi-implicit Euler: $\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{v}_t + \boldsymbol{a}\Delta t, \boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t + \boldsymbol{v}_{t+1} \Delta t$
Observe: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t \Delta t + \boldsymbol{a}\Delta t^2$

However, we know $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t \Delta t + \frac{1}{2}\boldsymbol{a}\Delta t^2$. How do we fix it?

Rigidbody Simulation: integration

Runge-Kutta method (RK4) https://en.wikipedia.org/wiki/Runge-Kutta_methods

$$\boldsymbol{k}_1 = \boldsymbol{v}_t$$

$$\boldsymbol{k}_2 = \boldsymbol{k}_3 = \boldsymbol{v}_t + \boldsymbol{a}\frac{1}{2}\Delta t$$

$$\boldsymbol{k}_4 = \boldsymbol{k}_3 = \boldsymbol{v}_t + \boldsymbol{a}\Delta t$$

$$\boldsymbol{x}_{t+1} = \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4)\Delta t = \boldsymbol{x}_t + \boldsymbol{v}_t\Delta t + \frac{1}{2}\boldsymbol{a}\Delta t^2$$

RK4 is accurate for constant acceleration. However, most game engines uses semi-implicit Euler. Why?

- 4 times faster.
- Semi-implicit Euler preserves energy better in a lot of problems (such as orbit motion, spring-damping)

Takeaway

- One major problem that physical simulation solves is the numerical integration given by Newton-Euler.
- But that is only part of the story. Another major problem is constraints.

Poll3

We know Newton-Euler

$$\begin{aligned} \boldsymbol{f} &= m\boldsymbol{a} \\ \boldsymbol{\tau} &= \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{I}\dot{\boldsymbol{\omega}} \end{aligned}$$

But in last lecture, we have seen

$$\begin{aligned} \boldsymbol{f} &= m\boldsymbol{a}^b + \boldsymbol{\omega}^b \times m\boldsymbol{v}^b \\ \boldsymbol{\tau} &= \boldsymbol{\omega}^b \times \boldsymbol{I}\boldsymbol{\omega}^b + \boldsymbol{I}\dot{\boldsymbol{\omega}}^b \end{aligned}$$

Poll3

Why changing to the body frame changes Newton-Euler equations?

A. The body frame is moving.

B. The body frame is accelerating.

C. Last time, we derive Newton-Euler using Lagrangian, which is fundamentally different from Newtonian mechanics.

D. what the ** is this question talking about?

Topics

- Newtonian Dynamics and Integration
    - Point mass
    - Rigidbody
    - Numerical Integration
- **Rigidbody Constraints**
    - Constraint solver for joints
    - Contact, friction, and impact
- Collision Detection
    - Primitive shape collision
    - Broad and narrow phases
    - Convex shape collision

Rigidbody Constraints

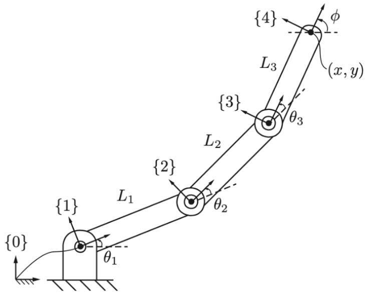
A constraint is a reduced DOF. In previous lectures, we have introduced links and joints. How do we simulate joints under the Newtonian framework?

Soft constraints

- Add a spring between links.

Hard constraints

- Compute forces between links to cancel relative motion.



This section is adapted from professor Rotenberg's slides.

https://cseweb.ucsd.edu/classes/sp19/cse291-d/Files/CSE291_12_ConstraintsAndCollisions.pdf

Constraint Solver

Main approaches (last time)

- Lagrangian method
    - Lagrange mechanics
    - Reduced coordinate (locked DOFs are gone)
    - Strict limits on DOF of the system

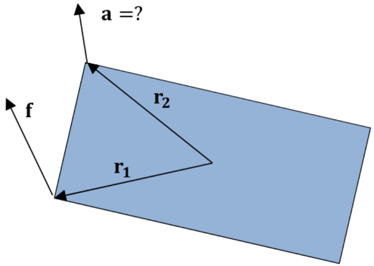
Constraint Solver

Main approaches (today)

- Featherstone's method
  - Newton-Euler equations
  - Reduced coordinate
  - Fast and accurate
- Lagrange multiplier method
  - Maximal coordinate (All DOFs are considered)
  - Easy to implement and extend

Offset Forces

If we apply force at r_1, what is the acceleration of r_2?



Offset Forces

offset acceleration formula: calculate acceleration at $\boldsymbol{r}$.

$$\boldsymbol{a_r} = \boldsymbol{a} + \dot{\boldsymbol{\omega}} \times \boldsymbol{r} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r})$$

inverse mass matrix: calculate acceleration at $\boldsymbol{r}_2$ from force at $\boldsymbol{r}_1$.

$$\boldsymbol{M}_{12}^{-1} = \begin{bmatrix} 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix} - \hat{\boldsymbol{r}}_2 \boldsymbol{I}^{-1} \hat{\boldsymbol{r}}_1$$

$$\boldsymbol{a}_2 = \boldsymbol{M}_{12} \boldsymbol{f}_1$$

(Note: it is okay to have M_{11} and M_{22} and they are very useful)

Ball Joint

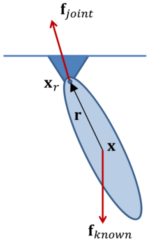
We first consider a 3 DOF ball joint. This is the simplest joint since the rotational DOFs are free and the joint does not provide torque.

We know all external forces f_{known}. By **offset acceleration** formula, we can calculate acceleration at $\boldsymbol{x_r}$.

Knowing $\boldsymbol{a_r}$, we can compute joint forces.

$$\boldsymbol{f}_{joint} = -\boldsymbol{M}_{rr}\boldsymbol{a_r}$$

(Note: $\boldsymbol{M}$ is the mass matrix from $\boldsymbol{x}_r$ to $\boldsymbol{x}_r$)



Simulate A Single Ball Joint

Algorithm

- Apply external forces to get $\boldsymbol{f}_{ext}$ and $\boldsymbol{\tau}_{ext}$
- Compute $\boldsymbol{a}$ and $\dot{\boldsymbol{\omega}}$
- Compute $\boldsymbol{a}_r$ at joint
- Compute M^{-1} and then $\boldsymbol{f}_{joint}$
- Apply $\boldsymbol{f}_{joint}$
- Simulate the motion

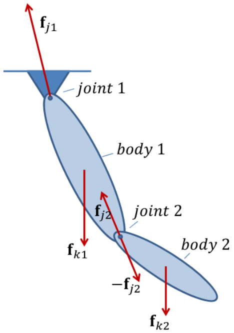
Issue

- Discrete time integration will accumulate errors.
- Solution: project back to correct position.

Simulate Kinematic Chain

Similar to previous example, $\boldsymbol{a}_1^{(1)}$, $\boldsymbol{a}_2^{(1)}$, $\boldsymbol{a}_2^{(2)}$, can be computed from external forces.

We want to make sure $\boldsymbol{a}_{j_1}^{(1)} = 0$ and $\boldsymbol{a}_{j_2}^{(2)} - \boldsymbol{a}_{j_2}^{(1)} = 0$. “No relative acceleration”



$$\boldsymbol{a}_{j_1}^{(1)} = \boldsymbol{M}_{12}^{(1)-1}(-\boldsymbol{f}_{j_2}) + \boldsymbol{M}_{11}^{(1)-1}\boldsymbol{f}_{j_1} + \boldsymbol{a}_1^{(1)}$$

$$\boldsymbol{a}_{j_2}^{(1)} = \boldsymbol{M}_{22}^{(1)-1}(-\boldsymbol{f}_{j_2}) + \boldsymbol{M}_{21}^{(1)-1}\boldsymbol{f}_{j_1} + \boldsymbol{a}_2^{(1)}$$

$$\boldsymbol{a}_{j_2}^{(2)} = \boldsymbol{M}_{22}^{(2)}\boldsymbol{f}_{j_2} + \boldsymbol{a}_2^{(2)}$$

Eventually it simplifies to the form $\boldsymbol{a} = \boldsymbol{M}^{-1}\boldsymbol{f}$ where $\boldsymbol{M}$ is 6×6.

Simulate Kinematic Chain

Algorithms in practice

- Featherstone's algorithm
  - Similar to the previous method.
  - Linear time.
  - May be modified to handle kinematic loops at a cost.

Lagrange Multiplier Method

Concatenate the position and rotation of all objects

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{\alpha}_n \\ \boldsymbol{x}_n \end{bmatrix}$$

Let $\boldsymbol{F}$ be the wrench, $\boldsymbol{M}$ be the generalized mass matrix, which combines all mass and inertia.

$$\boldsymbol{F}_C + \boldsymbol{F}_{ext} = \boldsymbol{M} \cdot \ddot{\boldsymbol{q}}$$

https://www.cs.ubc.ca/grads/resources/thesis/Nov02/Michael_Cline.pdf

Lagrange Multiplier Method

Suppose we have m equality constraints.

$$\boldsymbol{C}(\boldsymbol{q}) = \begin{bmatrix} C_1(\boldsymbol{q}) & \cdots & C_m(\boldsymbol{q}) \end{bmatrix}^T = \mathbf{0}.$$

There should not be relative velocity or relative acceleration.

$$\dot{\boldsymbol{C}} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = \mathbf{0}, \ddot{\boldsymbol{C}} = \mathbf{0}$$

Define the constraint's Jacobian matrix

$$\boldsymbol{J} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{q}}$$

Fact: $\boldsymbol{F}_C = \boldsymbol{J}^T \boldsymbol{\lambda}$ for some $\boldsymbol{\lambda}$, $\ddot{\boldsymbol{C}} = \dot{\boldsymbol{J}} \dot{\boldsymbol{q}} + \boldsymbol{J} \ddot{\boldsymbol{q}} = 0$

Lagrange Multiplier Method

Remember

$$\boldsymbol{F}_C + \boldsymbol{F}_{ext} = \boldsymbol{M}\ddot{\boldsymbol{q}}, \boldsymbol{F}_C = \boldsymbol{J}^T\boldsymbol{\lambda}, \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\ddot{\boldsymbol{q}} = 0$$

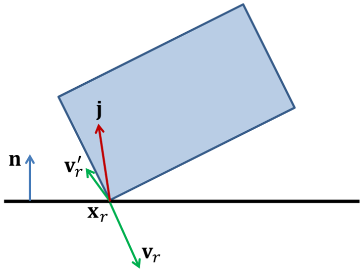
Now we only need to solve a linear system. Let $\boldsymbol{k} \triangleq \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$

$$\begin{bmatrix} \boldsymbol{M} & -\boldsymbol{J}^T \\ \boldsymbol{J} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_{ext} \\ -\boldsymbol{k} \end{bmatrix}$$

Now we only need to solve a linear system, for example, by Gauss elimination. There is also Gauss-Seidel algorithm that solves linear systems iteratively and it is very parallelizable.

Contact Simulation

(Note: **contact** and **impact** are treated differently.)



Isn't that a hinge joint?
observation: if there is not sliding, a contact is a joint.
However, contact will break easily.

Contact Simulation

We can model contact as inequality constraint.

$$\boldsymbol{C}_e(\boldsymbol{q}) = \boldsymbol{0},\, \dot{\boldsymbol{C}}_e = \boldsymbol{0},\, \ddot{\boldsymbol{C}}_e = \boldsymbol{0}$$
$$\boldsymbol{C}_c(\boldsymbol{q}) \geqslant \boldsymbol{0},\, \dot{\boldsymbol{C}}_c \geqslant \boldsymbol{0},\, \ddot{\boldsymbol{C}}_c \geqslant \boldsymbol{0}$$

Contact also needs to satisfy

- For each contact, acceleration must be non-negative, so $\boldsymbol{J}_c\ddot{\boldsymbol{q}} + \boldsymbol{k}_c \geqslant \boldsymbol{0}$
- All contact forces should be non-negative, so $\boldsymbol{\lambda}_c \geqslant \boldsymbol{0}$
- For each contact, when there is force, there is no acceleration, so $\lambda_i q_i = 0$, meaning $\boldsymbol{\lambda}^T\boldsymbol{q} = \boldsymbol{0}$.

Contact Simulation

$$\begin{bmatrix} \boldsymbol{M} & -\boldsymbol{J}_e^T & -\boldsymbol{J}_c^T \\ \boldsymbol{J}_e & \mathbf{0} & \mathbf{0} \\ \boldsymbol{J}_c & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda}_e \\ \boldsymbol{\lambda}_c \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ a \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_{ext} \\ -\boldsymbol{k}_e \\ -\boldsymbol{k}_c \end{bmatrix}$$

$$\boldsymbol{a} \geqslant \mathbf{0}, \boldsymbol{\lambda}_c \geqslant \mathbf{0}, \boldsymbol{a}^T\boldsymbol{\lambda}_c = 0, \ddot{\boldsymbol{q}} \text{ free}$$

It is a **mixed linear complementary problem** (mixed LCP). Physical simulators typically use **projected Gauss-Seidel (PGS)** to solve these problems.

Impulse Based Simulation

In fact, physical simulator does not calculate constraint forces, but constraint impulses that directly change velocities.

$$\boldsymbol{j} = \int \boldsymbol{f} dt \approx \boldsymbol{f} \Delta t$$

$$\Delta \boldsymbol{p} = \boldsymbol{j}$$

$$\Delta \boldsymbol{L} = \boldsymbol{r} \times \boldsymbol{j}$$

Impulse Based Simulation

Consider equality constraints. We take a Lagrangian view of impulse. Since $\boldsymbol{J}$ is taken, I will use $\boldsymbol{P}$ for impulse

$$\boldsymbol{P}_{ext} + \boldsymbol{P}_c = \boldsymbol{M}(\dot{\boldsymbol{q}}_2 - \dot{\boldsymbol{q}}_1)$$

$$\dot{\boldsymbol{q}}_2 = \dot{\boldsymbol{q}}_1 + \boldsymbol{M}^{-1}\boldsymbol{P}_{ext} + \boldsymbol{M}^{-1}\boldsymbol{P}_c$$

We similarly have $\boldsymbol{P}_c = \boldsymbol{J}^T\boldsymbol{\lambda}$ and $\boldsymbol{J}\dot{\boldsymbol{q}}_2 = 0$ and we can solve for the velocity $\boldsymbol{q}_2$ directly.

Which Solver is Actually Used?

The simulation backend for SAPIEN, PhysX, actually uses them all.

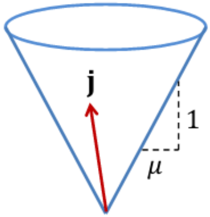
Robot simulation (Articulation): Featherstone's method.

Contact Simulation: Projected Gauss-Seidel.

Forward/inverse dynamics computation: Lagrangian view.

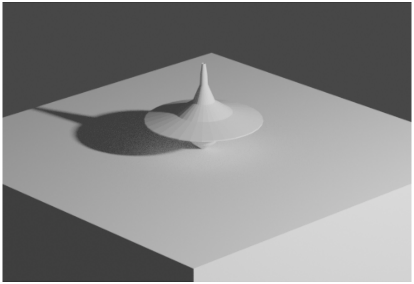
Contact Point Friction

Assuming the Coulomb friction model. $f = \mu F_N$.
If the impulse is within the friction cone, we have static friction. If not, we have kinetic friction. Including friction as a constraint is quite non-trivial.



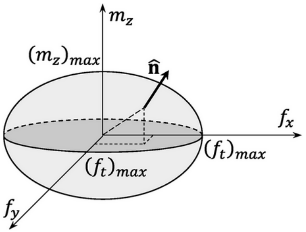
Contact Patch Friction

Point friction cannot provide **torsional friction**, but patch friction can.
Some simulators provide a torsional friction constant, but it is inaccurate it can actually be (roughly) computed from contact geometry.



Contact Patch Friction

Uniform 2D surface friction can be modeled as an ellipsoid. m_z is the friction torque, f_x, f_y are x, y direction friction force. Static friction is a point inside the ellipsoid. Kinetic friction is a point on the ellipsoid.



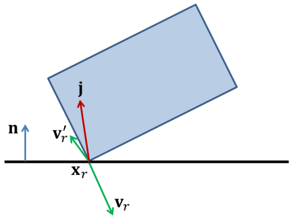
Torsional motion will cause a reduced linear friction. (Spinning objects can be pushed easier)

Determine Friction Coefficient

How do we determine pairwise friction coefficient?
Short answer: you will have to measure it for each pair.

What we end up doing: give a number to each material, and then add, multiply, or take the average. There is no correct way to do it since friction is very complicated.

Impact



When objects collide, they can bounce back, and we call this situation an impact. In this case, we compute an impact impulse j.

Perfectly inelastic collision: object stick to each other.

Perfectly elastic collision: kinetic energy is conserved.

Impact

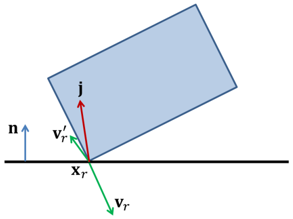
To simulate an impact, we first consider perfectly inelastic collision without sliding. Using conservation of (angular) momentum, it is very easy to compute a $\boldsymbol{j}'$. However, If $\boldsymbol{j}'$ does not lie in the friction cone, the impact has sliding, and we project $\boldsymbol{j}'$ back into the friction cone.

Next, we use the fact that the impulse of perfect elastic collision is twice the impulse of perfect inelastic collision. So the impulse will be between $\boldsymbol{j}'$ and $2\boldsymbol{j}'$. We use restitution coefficient ϵ to describe the elasticity.

$$\boldsymbol{j} = (1+\epsilon)\boldsymbol{j}'$$

Note: different people may define restitution coefficient differently.

Impact



Specifically, the impulse in a non-sliding impact with a static surface is

$$\boldsymbol{j} = -(1+\epsilon)\boldsymbol{M}_{rr} \cdot \boldsymbol{v}_r$$