Visual Interactive Simulation

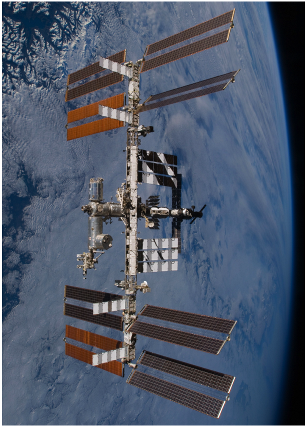
Rigid bodies & Introduction to constraints

martin.servin@physics.umu.se

November 14, 2011

Todays lecture

- Simple 3D rigid body simulations
- Geometry and algebra for 3D
- Rigid body mechanics
- Rigid body simulation
- Introduction to constraints

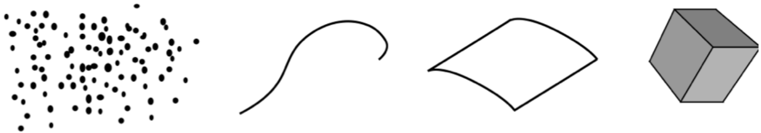


Recall: Particle system

definitions
initialization
while running **do**
  collision detection and collision response
  compute forces and constraints
  stepforward:
    solve to update state variables $(v_{n+1},\ x_{n+1})$
    udate derived quantities
  simulation I/O
end while
post-processing

$$
\begin{aligned}
m\dot{v} = f \quad &\rightarrow \quad v_{n+1} = v_n + hm^{-1}f_n \\
\dot{x} = v \quad &\rightarrow \quad x_{n+1} = x_n + hv_{n+1}
\end{aligned}
$$

Rigid body

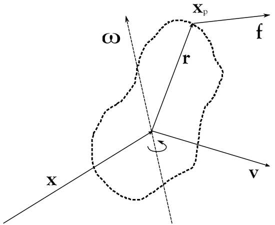


- One particle has 3 degrees of freedom (DOF) - translation
- A system of N_p particles has $3N_p$ DOF
- Rigid body = the distance between all parts are constant
- Ex: rigid particle system, 1D, 2D, 3D rigid shapes
- A rigid body has 6 DOF 3 translational and 3 rotational
- Translation: motion of bodys center of mass
- Rotation: motion around bodys center of mass

Rigid body

Newton-Euler equations of motion

$$
\begin{aligned}
m\dot{\mathbf{v}} &= \mathbf{f} \\
\dot{\mathbf{x}} &= \mathbf{v} \\
I\dot{\boldsymbol{\omega}} &= \boldsymbol{\tau} + \boldsymbol{\tau}_g \\
\dot{\mathbf{q}} &= T(\mathbf{q})\boldsymbol{\omega}
\end{aligned}
$$



Inertia tensor I. Torque

$$\boldsymbol{\tau} = \sum_i \mathbf{r}_i \times \mathbf{f}_i$$

Gyroscopic 'force'

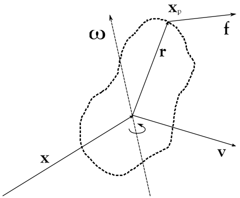
$$\boldsymbol{\tau}_g = -\boldsymbol{\omega} \times (I\boldsymbol{\omega})$$

Follows from the Newton equations of motion for particles - with the rigid body assumption.

Rotations

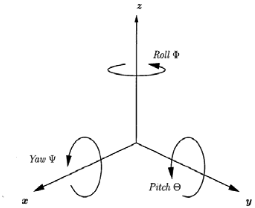
- Current state is a linear transformation local to global
- translation + rotation
- Position of a point

$$\mathbf{x}_\mathrm{p} = \mathbf{x} + \mathbf{r} = \mathbf{x} + R\mathbf{r}'$$



- Rotation matrix represented by Euler angles (Ψ, Θ, Φ)

$$R = R_z(\Psi)R_y(\Theta)R_x(\Phi)$$



- Drawbacks: singularities, hard to interpolate, numerical drift leading to non-orthonormality.
- We will use quaternions instead!

Quaternions

- A quaternion $q = [w, \mathbf{a}]$ is a 4D complex
- real scalar w, real 3D vector $\mathbf{a}$
- Product between $q_1 = [w, \mathbf{a}]$ and $\mathbf{q}_2 = [v, \mathbf{b}]$

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = [wv - \mathbf{a}^T\mathbf{b}, w\mathbf{b} + v\mathbf{a}, \mathbf{a} \times \mathbf{b}]$$

- Conjugate

$$\mathbf{q}^* = [w, -\mathbf{a}]$$

- Norm

$$|\mathbf{q}| = \sqrt{w^2 + \mathbf{a}^T\mathbf{a}}$$

Geometrical interpretation of quaternions

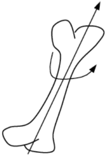
The unit quaternions $|\mathbf{q}| = 1$ form a *group* of 3D rotations

$$\mathbf{q} = [w, \mathbf{a}] = [\cos(\theta/2), \sin(\theta/2)\mathbf{n}]$$

is counter clockwise rotation of

$$\theta = 2\cos^{-1}(w) \in [-\pi, \pi]$$

about unit vector $\mathbf{n} = \mathbf{a}/|\mathbf{a}|$



- Rotation of a vector $\mathbf{r}'$

$$\mathbf{r} = \mathbf{q}\cdot\mathbf{r}'\cdot\mathbf{q}^* \quad , \qquad \text{where } \mathbf{q}\cdot\mathbf{r}' = \mathbf{q}\cdot[0, \mathbf{r}']$$

- Compound rotation $\mathbf{q} = \mathbf{q}_1\mathbf{q}_2$

$$\mathbf{r} = \mathbf{q}\cdot\mathbf{r}'\cdot\mathbf{q}^* = \mathbf{q}_2\cdot(\mathbf{q}_1\cdot\mathbf{r}'\cdot\mathbf{q}_1^*)\cdot$$

Quaternion, Euler angles and rotation matrix

Euler angles to quaternion $(\Psi, \Theta, \Phi) \rightarrow \mathbf{q} = \mathbf{q}_\Phi \cdot (\mathbf{q}_\Theta \cdot \mathbf{q}_\Psi)$

$$\begin{aligned} q_\Psi &= [\cos(\Psi/2), \sin(\Psi/2), 0, 0] \\ q_\Theta &= [\cos(\Theta/2), 0, \sin(\Theta/2), 0] \\ q_\Phi &= [\cos(\Phi/2), 0, 0 \sin(\Phi/2)] \end{aligned}$$

Quaternion $q = [w, \mathbf{a}]$ to rotation matrix

$$R = \begin{pmatrix} 1 - 2(a_y^2 + a_z^2) & 2a_x a_y - 2wa_z & 2a_x a_z + 2wa_y \\ 2a_x a_y + 2wa_z & 1 - 2(a_x^2 + a_z^2) & 2a_y a_z - 2wa_x \\ 2a_x a_z - 2wa_y & 2a_y a_z + 2wa_x & 1 - 2(a_x^2 + a_y^2) \end{pmatrix}$$

Quaternion time derivative

It can be shown that the time derivative of a quaternion is

$$\frac{d\mathbf{q}}{dt} = \mathrm{T}(\mathbf{q})\boldsymbol{\omega}$$

with the transformation matrix

$$\mathrm{T}(q) = \frac{1}{2}\begin{pmatrix} -a_1 & -a_2 & -a_3 \\ w & a_3 & -a_2 \\ -a_3 & w & a_1 \\ a_2 & -a_1 & w \end{pmatrix}$$

implying

$$\mathbf{x}_p = \mathbf{x} + \mathbf{r} = \mathbf{x} + \mathbf{q}\cdot\mathbf{r}'\cdot\mathbf{q}^*$$
$$\dot{\mathbf{x}}_p = \mathbf{v} + \boldsymbol{\omega}\times\mathbf{r}$$

Angular momentum and inertia tensor

Angular momentum of a particle

$$\mathbf{L} = m\mathbf{x} \times \mathbf{v}$$

separate in translational and rotational motion

$$\mathbf{L}_{tot} = \mathbf{L}_{CM} + \mathbf{L} = m\mathbf{x} \times \mathbf{v} + \sum_{(i)} m_{(i)} \mathbf{r}_{(i)} \times (\omega \times \mathbf{r}_{(i)})$$

Introducing the inertia tensor by $\mathbf{L} = I\omega$ as the 3×3 matrix

$$I = [I_{jk}] = \sum_{(i)} m_{(i)} (|r^{(i)}|^2 \delta_{jk} - r_j^{(i)} r_k^{(i)})$$

$$\rightarrow \int \rho(r)(|r|^2 \delta_{jk} - r_j r_k) dV$$

with mass density distribution function $\rho(r)$.

Angular momentum and inertia tensor

Example: solid homogenous block with sides a, b and c in the inertia tensor is

$$I' = \frac{m}{12}\begin{pmatrix} b^2+c^2 & 0 & 0 \\ 0 & a^2+c^2 & 0 \\ 0 & 0 & b^2+c^2 \end{pmatrix}$$

In a general rotation

$$I = R(q)I'R(q)^T$$

Angular momentum is changed only by an external force producing a torque $\boldsymbol{\tau} = \mathbf{r} \times \mathbf{f}$

$$\dot{\mathbf{L}} = I\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (I\boldsymbol{\omega}) = \boldsymbol{\tau}$$

Observe that if $\boldsymbol{\tau} = 0$, angular momentum is conserved $\dot{\mathbf{L}} = 0$

Summary of the Newton-Euler equations of motion

$$\begin{aligned} m\dot{\mathbf{v}} &= \mathbf{f} \\ \dot{\mathbf{x}} &= \mathbf{v} \\ I\dot{\boldsymbol{\omega}} &= \boldsymbol{\tau} + \boldsymbol{\tau}_g \\ \dot{\mathbf{q}} &= T(\mathbf{q})\boldsymbol{\omega} \end{aligned}$$

$$\begin{aligned} \boldsymbol{\tau} &= \sum_i \mathbf{r} \times \mathbf{f}_i \\ \boldsymbol{\tau}_g &= -\boldsymbol{\omega} \times (I\boldsymbol{\omega}) \\ I &= R(q)I'R(q)^T \\ |\mathbf{q}| &= 1 \end{aligned}$$

$$T(q) = \frac{1}{2}\begin{pmatrix} -a_1 & -a_2 & -a_3 \\ w & a_3 & -a_2 \\ -a_3 & w & a_1 \\ a_2 & -a_1 & w \end{pmatrix}$$

Total energy for a rigid body in gravity field

$$E = K + U = \tfrac{1}{2}m\mathbf{v}^T\mathbf{v} + \tfrac{1}{2}\boldsymbol{\omega}^T I\boldsymbol{\omega} + m\mathbf{g}^T\mathbf{x}$$

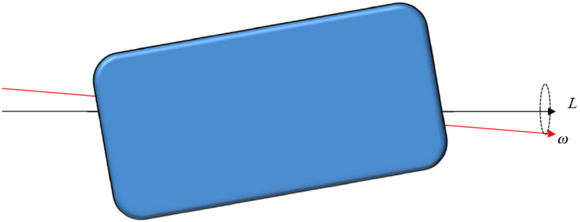
Discretization with semi-implicit Euler

$$
\begin{aligned}
v_{n+1} &= v_n + h m^{-1} f_n \\
\omega_{n+1} &= \omega_n + h I^{-1}(\tau_n + \tau_{g_n}) \\
x_{n+1} &= x_n + h v_{n+1} \\
q_{n+1} &= q_n + h T_n \omega_{n+1}
\end{aligned}
$$

where

$$
\begin{aligned}
f_n &= \sum_i f_n^i \quad , \quad \tau_n = \sum_i \tau_n^i = \sum_i r_n^i \times f_n^i \\
\tau_n^g &= \omega_n \times (I_n \omega_n) \\
I_n^{-1} &= I(q_n)^{-1} = R(q_n) I'^{-1} R(q_n)^T \\
T_n &= T(q_n) \\
q_n &= q_n / |q_n|
\end{aligned}
$$

Example: gyroscopic force



- Mass is distributed asymmetric about the rotation axis
- Angular momentum is constant
- Angular velocity wobbles - due to the gyroscopic "force"
- Gyroscopic force can lead to numerical instabilities
- Gyroscopic force is often discarded in game engines

Rigid body system data

- State variables: `x`, `q`, `v`, `omega`
- Derived quantities: $\mathtt{I}_{\mathtt{inv}}$, `I`
- Computed quantities: `f`, `tau`
- Constants: `m`, $\mathtt{I}_{\mathtt{body}}$, $\mathtt{I}_{\mathtt{inv\_body}}$

- Where is the geometry of the body?
- The inertia tensor contains enough geometry information to compute the dynamics
- For collision detection we will need more geometrical information

Stepping a rigid body system

definitions
initialization
while running **do**
  collision detection and collision response
  compute forces and constraints
  stepforward:
    solve to update state variables
    udate derived quantities
  simulation I/O
end while
post-processing

$$
\begin{aligned}
v_{n+1} &= v_n + hm^{-1}f_n \\
\omega_{n+1} &= \omega_n + hI^{-1}(\tau_n + \tau_{g_n}) \\
x_{n+1} &= x_n + hv_{n+1} \\
q_{n+1} &= q_n + hT_n\omega_{n+1}
\end{aligned}
$$

Reading instructions

Physics Based Animation, Erleben et al

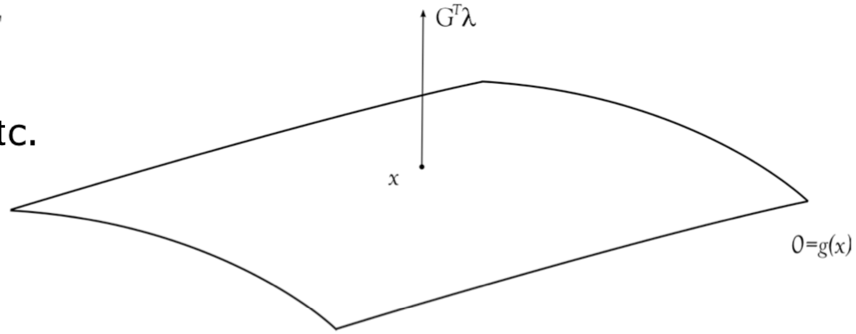
- Chp 22.1-22.3, 22.6 Basic Classical Mechanics
- Chp 18.1,18.2,18.5 Vectors, matrices and quaternions
- Chp 7.1 Equations of motion

SIGGRAPH 97 lecture notes *An Introduction to Physically Based Modeling*

- Rigid Body Dynamics I, D Baraff
- Constrained dynamics, Witkin

Notes on Discrete Mechanics, M. Servin, collected lecture notes

Constrained rigid bodies



- General system position $x = [x_1, x_2, ..., x_i, ...x_N]^T$

- A constraint is a restriction of free motion: joints etc.

- A holonomic constraint can limits the motion to a hypersurface – valid positions

$$0 = g(x) \qquad g = [g_1, g_2, ..., g_i, ...g_M]^T$$

- Valid velocities – tangential to the constraint surface

$$0 = \dot{g}(x) = \frac{\partial g}{\partial x^T}\frac{dx}{dt} = Gv$$

$$M\dot{v} = f + G^T\lambda$$
$$0 = g(x)$$

- We assume there is a constraint force that maintains the system on the surface

$$f_c = G^T\lambda = G_{ji}\lambda_j$$

$G = \frac{\partial g}{\partial x^T}$ Jacobian

λ Lagrange multiplier

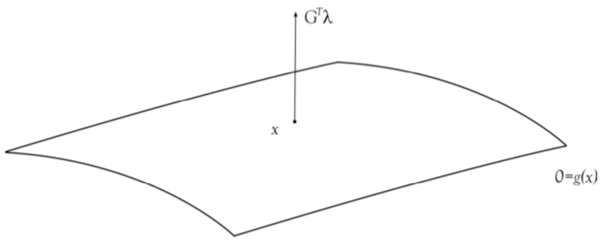
- The constraint force is orthogonal to the surface and does no work (add/removes no energy)

$$f_c^T v = (G^T\lambda)^T v = G_{ji}\lambda_j v_i = \lambda_j^T(Gv) = 0$$

Non-holonomic constraints

- Unilateral constraints

$$0 \geq g(x)$$



- Non-holonomic constraints cannot be integrated into a holonomic, e.g.,

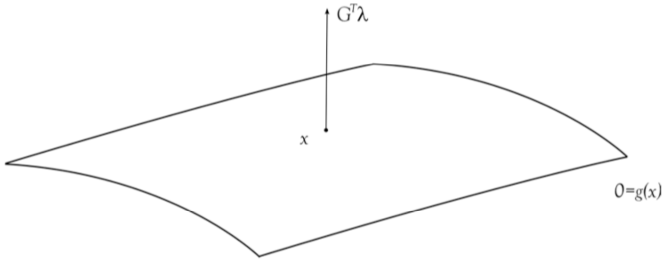
$$0 \;=\; g(v) \;\equiv\; Gv$$

- These are building blocks to stable modelling of frictional contacts

Time-integration of constrained multibodis

- Adding constraints modifies the ODE to a system of differential algebraig equations (DAE)

$$
\begin{aligned}
M\dot{v} &= f + G^T\lambda \\
0 &= g(x)
\end{aligned}
$$



- First approach. The acceleration method. Differentiate the constraint twice.

$$0 = \dot{G}v + G\dot{v}$$

- Multiply and substitute to find

$$[GM^{-1}G^T]\lambda = -GM^{-1}f - \dot{G}v$$

- Solve to find the Lagrange multiplier. Then compute the consraint force and integrate the velocity equation.

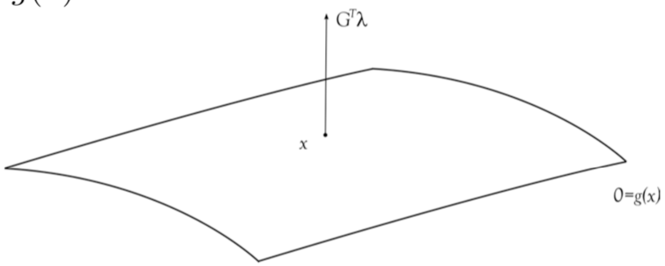
- Unstable!!! Singular matrix -> regularize
- Constraint drift!!! Blind to constraint violation -> include stabilization terms

$$\lambda \rightarrow \lambda + \alpha g(x) + \beta G v$$

Time-integration of constrained multibodis

- Second approach: perturb (regularize) the constraint $-\varepsilon\lambda = g(x)$

$$M\dot{v} = f + G^T\lambda$$
$$-\lambda = \varepsilon^{-1} g(x)$$



- Discretize the DAE directly

$$M v_{n+1} - G_n^T \lambda_{n+1} = M v_n + h f_n$$
$$G_n v_{n+1} - h^{-1}\varepsilon\lambda_{n+1} = h^{-1} g_n$$

$$-\varepsilon\lambda_{n+1} = g(x_{n+1}) \approx g(x_n) + hG_n v_{n+1}$$

- In matrix form

$$\begin{pmatrix} M & -G_n^T \\ G_n & \varepsilon/h \end{pmatrix} \begin{bmatrix} v_{n+1} \\ \lambda_{n+1} \end{bmatrix} = \begin{bmatrix} f_n \\ -h^{-1} g_n \end{bmatrix}$$

$$Au = b$$

- Can be solved directly. Sparse matrix solver for large systems. Or first Schur complement

$$\boxed{\left[GM^{-1}G^T + \varepsilon/h\right]\lambda_{n+1} = -hG_n M^{-1} f_n - G_n v_n - h^{-1} g_n}$$

$$f_n^{c} = G_n^T \lambda_{n+1} \qquad v_{n+1} = v_n + hM^{-1}(f_n + f_n^{c})$$

Time-integration of constrained multibodis

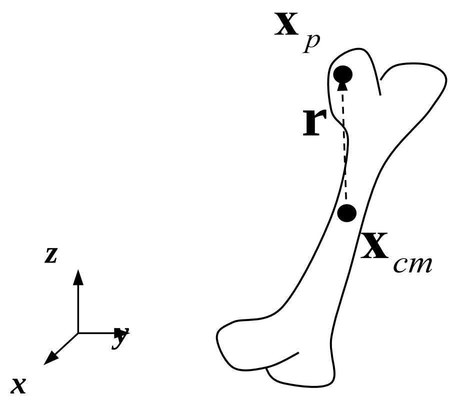
- For a rigorous treatment see

C. Lacoursière, PhD thesis 2007

Ghost and Machines: Regularized Variational Methods for Interactive Simulation of Multibodies with Dry Frictional Contacts

Example: point-to-point constraint

$$\mathbf{x}_p = \mathbf{x}_{cm} + \mathbf{r} = \mathbf{x}_{cm} + R\mathbf{r}'$$



$$0 = \mathbf{g}(\mathbf{x}_p) \equiv \mathbf{x}_p - \mathbf{p}$$

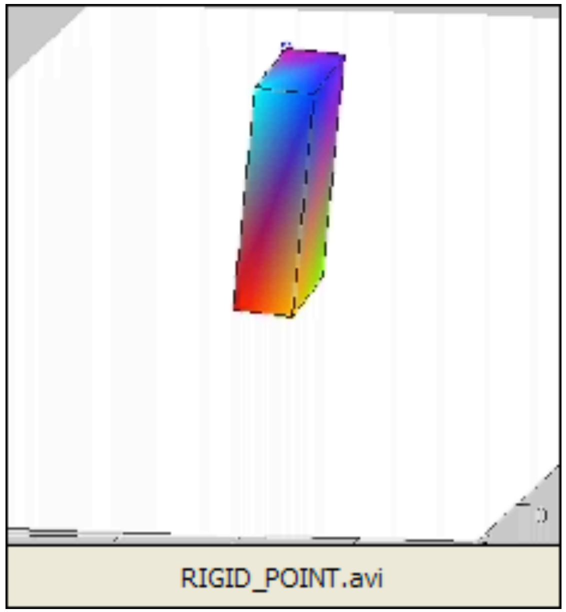
$$0 = \dot{\mathbf{g}} = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r} = \underbrace{\begin{pmatrix} \mathbf{1} & -\mathbf{r}_p^* \end{pmatrix}}_{G} \begin{pmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{pmatrix}$$

$$G = \begin{pmatrix} \mathbf{1} & -\mathbf{r}^* \end{pmatrix} ,$$

3×6 matrix,

$\mathbf{1}$ is the 3×3 identity matrix

$$\mathbf{r}^* = \begin{pmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{pmatrix}$$



$$F_c = \begin{pmatrix} \mathbf{1} \\ -\mathbf{r}^* \end{pmatrix} \begin{pmatrix} \lambda_x \\ \lambda_y \\ \lambda_z \end{pmatrix}$$

Example: point-to-line constraint

$$\mathbf{x}_p = \mathbf{x}_{cm} + \mathbf{r} = \mathbf{x}_{cm} + R\mathbf{r}'$$

$$0 = g_1 \equiv \Delta x_{\perp 1}$$, distance along normal $\mathbf{n}_1$

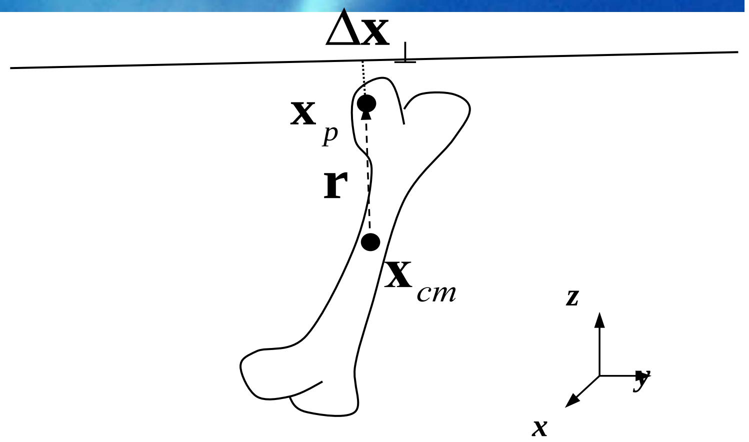
$$0 = g_2 \equiv \Delta x_{\perp 2}$$, distance along normal $\mathbf{n}_2$

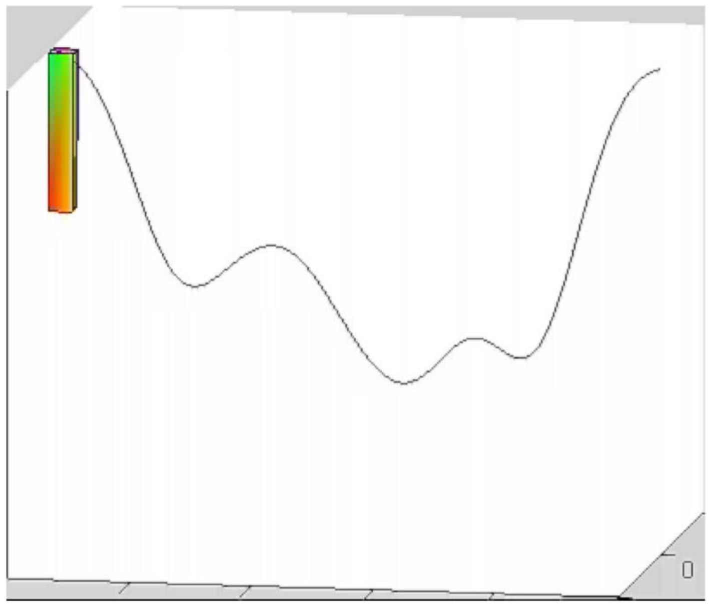
$$0 = \dot{\mathbf{c}} = \begin{pmatrix} [\mathbf{v} + \omega \times \mathbf{r}] \cdot \mathbf{n}_1 \\ [\mathbf{v} + \omega \times \mathbf{r}] \cdot \mathbf{n}_2 \end{pmatrix} = \underbrace{\begin{pmatrix} \mathbf{n}_1^T & (-\mathbf{n}_1 \times \mathbf{r})^T \\ \mathbf{n}_2^T & (-\mathbf{n}_2 \times \mathbf{r})^T \end{pmatrix}}_{G} \begin{pmatrix} \mathbf{v} \\ \omega \end{pmatrix}$$

$$G = \begin{pmatrix} \mathbf{n}_1^T & (-\mathbf{n}_1 \times \mathbf{r})^T \\ \mathbf{n}_2^T & (-\mathbf{n}_2 \times \mathbf{r})^T \end{pmatrix}$$

$$F_{c_1} = \begin{pmatrix} \mathbf{n} \\ (-\mathbf{n}_1 \times \mathbf{r}) \end{pmatrix} \lambda_1$$

$$F_{c_2} = \begin{pmatrix} \mathbf{n} \\ (-\mathbf{n}_2 \times \mathbf{r}) \end{pmatrix} \lambda_2$$





Example: ...

- Constraint

$$0 = \mathbf{g}_1 \equiv \mathbf{x}_1 - \mathbf{p}$$
$$0 = g_4 \equiv |\mathbf{x}_1 - \mathbf{x}_2| - L$$
$$0 = g_5 \equiv |\mathbf{x}_2 - \mathbf{x}_3| - L$$

What does that say?

Example: …

- Constraint

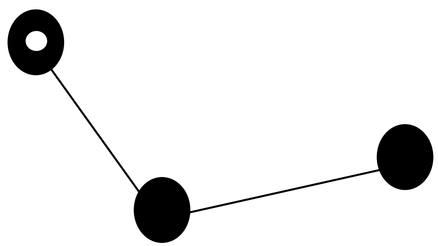
$$0 = \mathbf{g}_1 \equiv \mathbf{x}_1 - \mathbf{p}$$

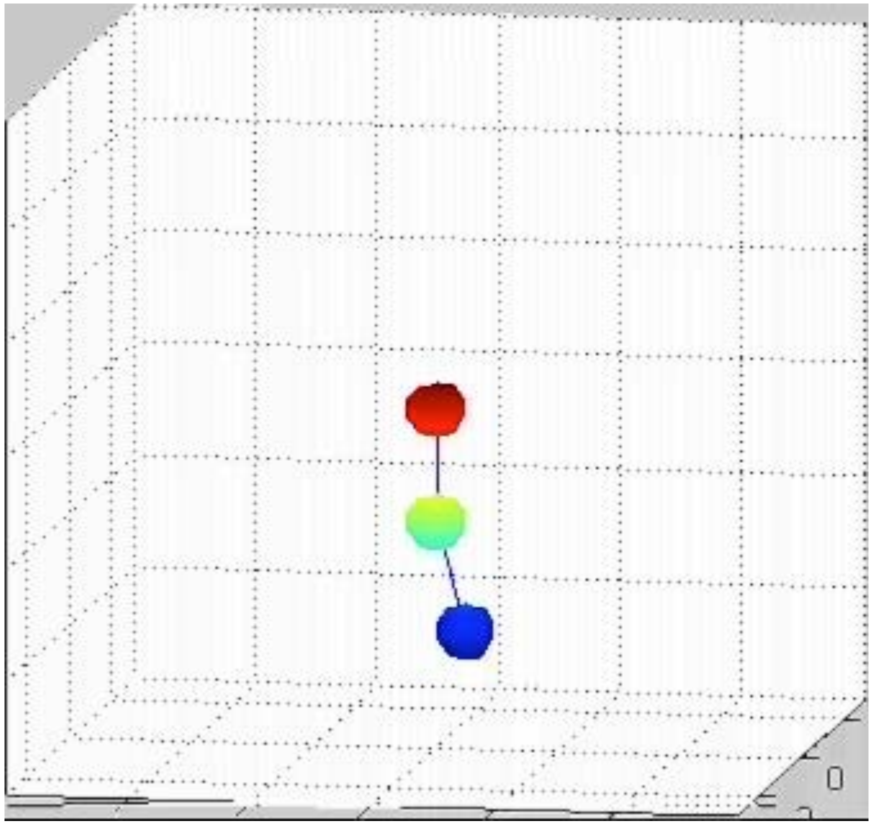
$$0 = g_4 \equiv |\mathbf{x}_1 - \mathbf{x}_2| - L$$

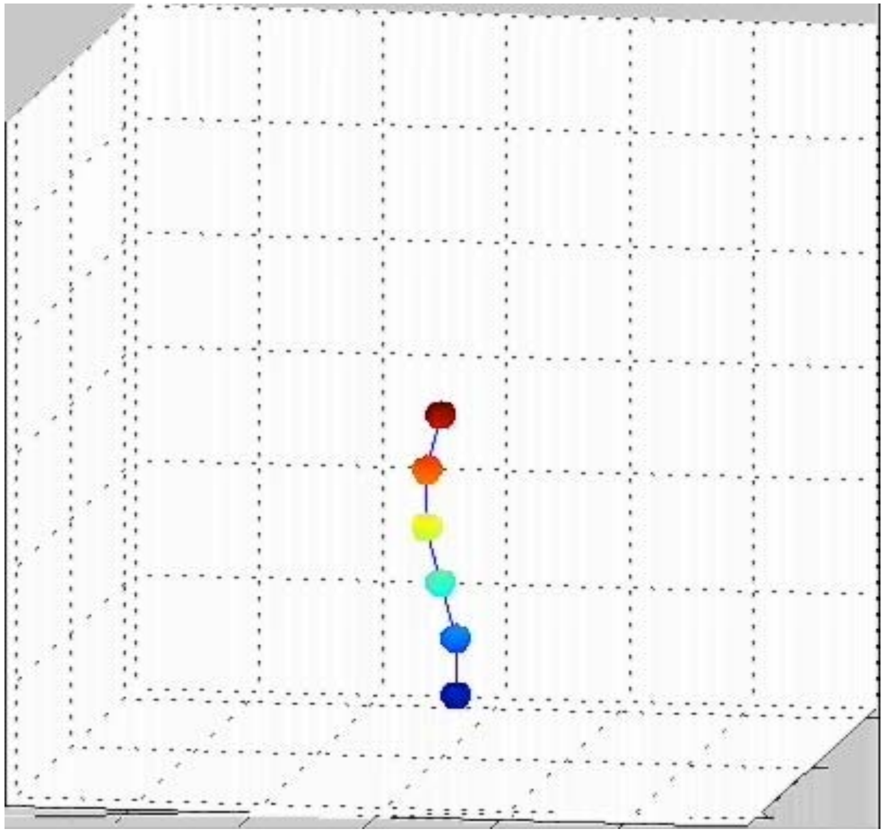
$$0 = g_5 \equiv |\mathbf{x}_2 - \mathbf{x}_3| - L$$

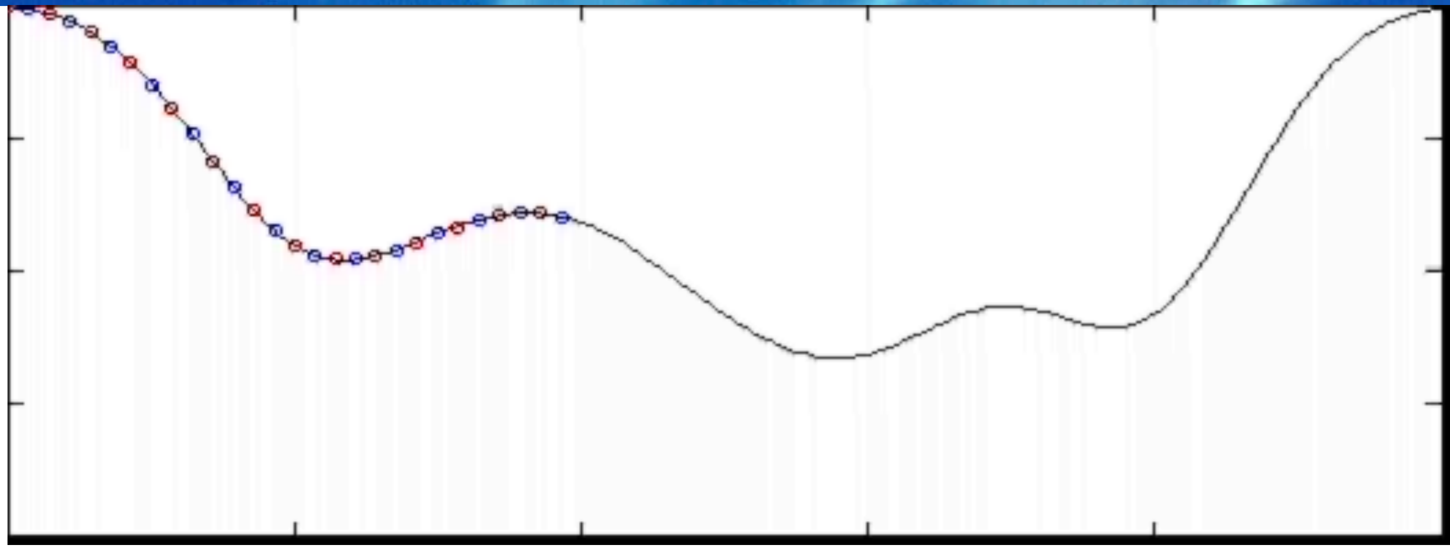
What does that say?

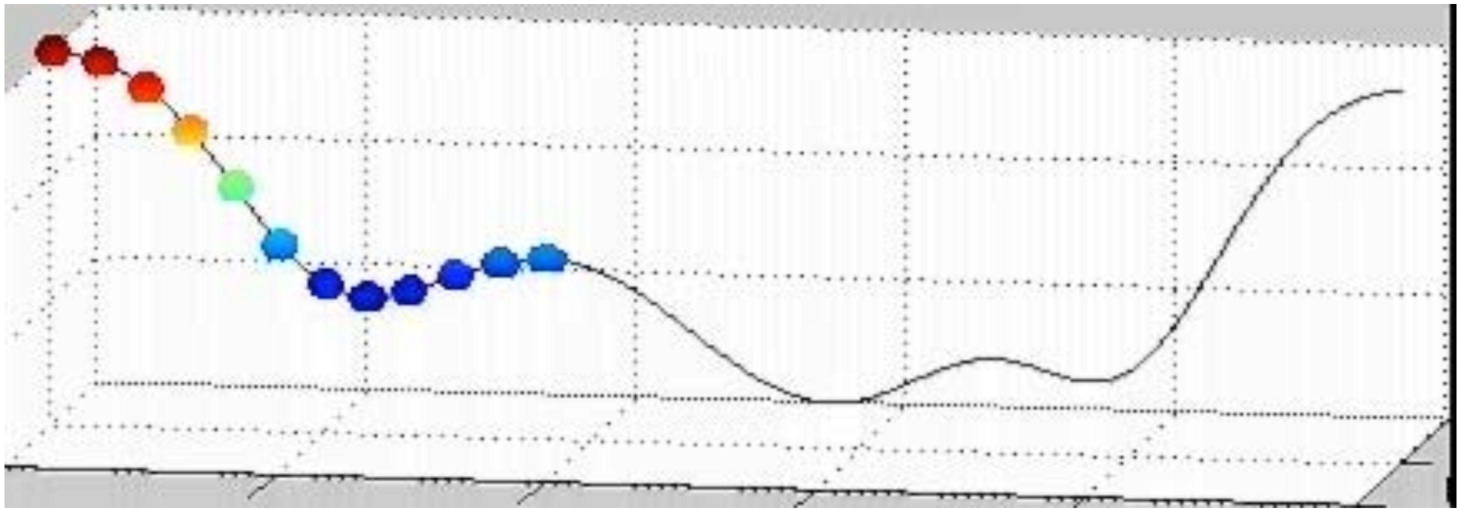
Well,…











Constraint libraries

- In a physics engine the most common constraints are implemente
- Just specify what type of constraint and what position and axis

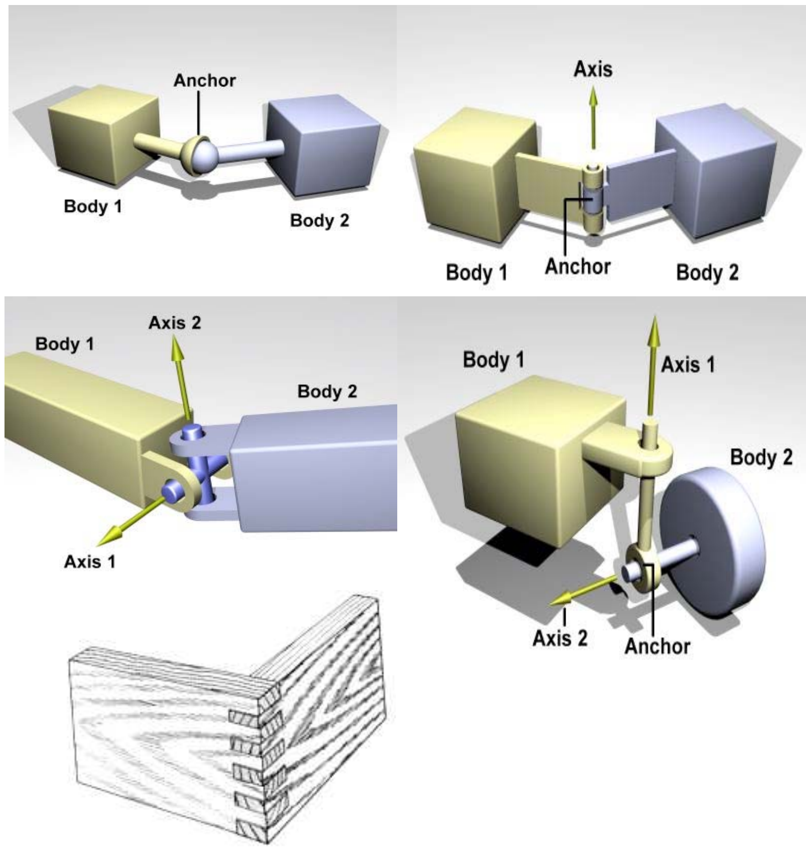
Ball-socket joint (point)

Hinge joint (point + axis)

Universal joint (point + 2 axes)

Carwheel joint (point + 2 axes)

Lock joint (point + 3 axes)



- Joint motors are non-holonomic constraints

The simulation algorithm - constraints

```
definitions
initialization
```

$-\varepsilon\lambda = g(x) \quad G = \frac{\partial g}{\partial x^T}$

```
while (running)
   collision detection and collision response
   compute forces and constraints
```

$$[GM^{-1}G^T + \varepsilon/h]\,\lambda_{n+1} = -hG_nM^{-1}f_n - G_nv_n - h^{-1}g_n$$

$$f_n^c = G_n^T\lambda_{n+1}$$

```
   stepforward:
        update state variables (semi-euler)
```

$$v_{n+1} = v_n + hM^{-1}(f_n + f_n^c)$$

```
                …
                …
        update derived quantities
   simulation I/O
end
post-processing
```