Constraint based physics solver

Marijn Tamis, Giuseppe Maggiore

June 15, 2015 (v1.02)

Abstract

Rigid body simulations are complex systems and literature describing them is not easily accessible to programmers new to the subject. This is unfortunate as these simulations play an important role in modern games. This paper aims to make the subject more accessible by giving a more complete reference without assuming an extensive mathematical background.

1 Introduction

Rigid body physics is an important part of many modern games. It is used to realistically simulate interactions within the game world, including both game play elements and environmental effects. The visible correct simulation of these interactions enhance the players immersion and contribute to the overall experience. It also opens up new possibilities for game mechanics.

Developing a physics engine is a complex task and nearly impossible without understanding the mathematics behind such a system. The simulations used for games need to be fast and stable but not necessarily accurate. Most of the literature on simulations that fit this description is not easily accessible for programmers without the necessary mathematics background. This paper will give a more complete review of the basic technique described in [2] making it more accessible for programmers aiming to understand it while keeping need to consult external resources minimal. We show in great detail how the equations are set up and show the intermediate results. We also explain many aspects of the simulation with examples, showing why things are setup in a certain way.

Aside from reviewing existing work we also present two new aspects. We analyze the dimensions of the system equations, showing what units are used and that the equations provide meaningful results. We also present an improvement in the behavior of violated constraint resolution utilizing a push factor and combine this with restitution properties.

It is assumed that readers of this document have knowledge of 3D mathematics, linear and angular physics, and basic calculus. A table containing all symbols used in this paper is included at the end of this document for quick reference.

- In Section 2 we define the notational conventions used throughout this document.
- In Section 3 we define our rigid bodies and how we integrate them.

- In Section 4 we briefly mention the tasks of the collision detection system.
- In Section 5 we discuss the constraints we use in the simulation.
- In Section 6 we discuss the system equation and how to solve it.
- In Section 7 we discuss the iterative method used to solve the system equation.
- In Section 8 we discuss more uses for the push factor, including the improved technique.
- In Section 9 we analyze the dimensions of the system equation.
- In Section 10 we discuss the results of the described system using example simulations.
- In Section 12 we included some useful information used in the rest of the document.

2 Notational conventions

Here we define the notational conventions used throughout this paper.

Element	Typesetting	Examples
Scalars	lowercase italics	m, ω, Ω
Vectors	lowercase fat upright	$\mathbf{x}, \boldsymbol{\theta}$
Quaternions	lowercase fat upright	$\mathbf{o}, \boldsymbol{\omega}$
Matrices	uppercase fat upright	$\mathbf{I}, \mathbf{R}$
Units	upright	$\mathrm{kg\,m\,s^{-1}}$
Functions	upright	$\mathrm{c}(\ldots)$

3 Rigid body and integrator

In Subsection 3.1 we define the properties of the rigid bodies in our simulation. In Subsection 3.2 we define how we integrate when advancing to the next time step.

3.1 Rigid body

We limit the scope of the simulation to only handle rigid bodies. A rigid body is an undeformable solid body. We divide the bodies on our simulation in two categories:

1. Movable bodies
2. Static bodies

Static bodies have a fixed location in world space and can not be moved. A static body has infinite mass. This is defined as inverse mass $m^{-1} = 0$ and inverse inertia tensor $\mathbf{I}^{-1} = \mathbf{0}$.

Movable rigid bodies have a mass of $m \in (0, +\infty)$ and an inertia tensor with at least a non-zero diagonal. Mass m is given in kg and the inertia tensor $\mathbf{I}$

is given in $\mathrm{kg\,m^2}$. It is assumed rigid bodies are movable in the rest of this document.

The position of the center of mass of a rigid body in world space is defined by a vector $\mathbf{x} = [x_x, x_y, x_z]$ in m. Rigid bodies also have an orientation in world space defined by a rotation $\mathbf{o} = o_w + o_x i + o_y j + o_z k$ about the center of mass. The quaternion is dimensionless as it is a rotation in $\mathrm{rad} = \frac{\mathrm{m}}{\mathrm{m}}$ and a (dimensionless) axis combined with Equation 12.4.1.

Rigid bodies keep track of the linear and angular momentum instead of velocities. This is both because of the better numerical stability of the integration and because angular velocity is not constant in torque-free motion[3][6]. Linear momentum is defined as a vector $\mathbf{p}$ in $\mathrm{kg\,ms^{-1}}$ or $\mathrm{N\,s}$. Angular momentum is defined as a vector $\boldsymbol{\ell}$ in $\mathrm{kg\,m^2\,s^{-1}}$ or $\mathrm{N\,m\,s}$ (as $[\boldsymbol{\ell}] = [\mathbf{p}] \times [\mathbf{r}]$ where $\mathbf{r}$ is the offset from the center of mass in m).

The rigid body also keeps track of all the forces applied to it in a frame. Linear force applied to the center of mass is stored in vector $\mathbf{f}$ in $\mathrm{kg\,m\,s^{-2}}$ or N. Torque about the center of mass is stored in vector $\boldsymbol{\tau}$ in $\mathrm{kg\,m^2\,s^{-2}}$ or $\mathrm{N\,m}$. The forces are stored as force, and not as impulse. This means that the integrator still needs to multiply with the delta time. We choose to do this so that other parts of the game code do not need to know what the time step is each frame to deliver a constant force like gravity.

3.2 Integrator

Momenta are integrated as follows:

$$\mathbf{p}(t + \Delta t) = \mathbf{p}(t) + \mathbf{f}(t)\Delta t \tag{3.2.1}$$

$$\boldsymbol{\ell}(t + \Delta t) = \boldsymbol{\ell}(t) + \boldsymbol{\tau}(t)\Delta t \tag{3.2.2}$$

Where t is the current time and Δt is the time step.

Position is integrated as follows:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \frac{\mathbf{p}(t + \Delta t)}{m}\Delta t \tag{3.2.3}$$

Orientation is integrated as follows:

$$\mathbf{o}(t + \Delta t) = \mathbf{o}(t) + \left[0, \frac{1}{2}\mathbf{I}^{-1}\boldsymbol{\ell}(t + \Delta t)\right]\mathbf{o}(t)\Delta t \tag{3.2.4}$$

It is good practice to normalize $\mathbf{o}$ after such an integration. Note that Equation 3.2.3 and Equation 3.2.4 are almost in the same form. Section 12.5 derives this Equation. Note that for large time steps or high angular velocities 3.2.4 may be too imprecise and 12.4.1 should be used instead.

These equations describe a (semi-implicit) Euler integrator and they can easily be extended to an RK4 integrator. However, the constraint solver assumes an Euler integrator is used, so using a different integrator when constraints are acting on the body might result in poor constraint resolution. Using RK4 for ballistic motion, and switching to Euler on collision might give the best of both worlds, as the error for a single Euler step would be negligible.

It is also important to keep in mind that some continuous collision detection algorithms only work with Euler steps[8].

4 Collision detection

The collision detection system has two jobs:

1. Detect if rigid bodies collide
2. Generate a contact manifold from the collision information

For step 1 with convex polyhedra a separating axis test is used. Step 2 is carried out by clipping the most penetrating face of the first polyhedron to the volume below the corresponding anti-parallel face on the second polyhedron.

It is beyond the scope of this document to give an in-depth explanation on collision detection and manifold generation.

5 Constraints

In this section we introduce the constraints used to restrict movement. Our simulation will only deal with pair wise constraints (constraints acting on two bodies) to keep the solver simple and make some optimization possible.

- In Subsection 5.1 we discuss the general properties of the constraints.
- In Subsection 5.2 we describe the distance constraint and we elaborate on Subsection 5.1 by using it as an example.
- In Subsection 5.3 we describe the contact constraint.
- In Subsection 5.4 we describe the friction constraints.

5.1 General definition

A position constraint c acting on bodies a and b can be expressed as a function of the positions and orientations of a and b:

$$\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = 0 \tag{5.1.1}$$

If $\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) \neq 0$ it means that the constraint is broken. $\mathrm{c}(\ldots)$ is a monotonic function depending on length in m.

The constraint solver we employ can not solve position constraints, but only velocity constraints. To get the velocity constraint we get the derivative of c:

$$\frac{d}{dt}\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \dot{\mathrm{c}}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = 0$$

Now the scalar result indicates how fast the constraint is being violated, so $\dot{\mathrm{c}}(\ldots)$ has a monotonic dependency on speed in $\mathrm{m\,s^{-1}}$. However, we lose the information about how much the (position) constraint was violated to begin with. This would make it impossible for the solver to correct constraints that are already broken. To get around this problem we introduce a push factor υ[I]:

$$\dot{\mathrm{c}}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \upsilon \tag{5.1.2}$$

[I]This is usually called Baumgarte stabilization, but we use υ to indicate other biases (e.g. for restitution) as well.

Now we can use υ to bias $\dot{\mathrm{c}}$ so that the constraint can add a velocity to the system in order to fix already broken constraints. Susbsection 5.2 will go into more detail with an example.

Some constraints have a limit on the force they can generate, or can only apply force in one direction. We store this information in $c_{\min}$ and $c_{\max}$. $c_{\min}$ and $c_{\max}$ can be set to $-\infty$ and $+\infty$ respectively for constraints without a limit. Sitting either $c_{\min}$ or $c_{\max}$ to 0 makes the constraint only able to act in one direction.

The solver expects the constraints to be in the following format:

$$\dot{\mathrm{c}}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \mathbf{j}_c \mathbf{v}_{ab} = \upsilon \tag{5.1.3}$$

Where $\mathbf{v}_{ab}$ is a vector containing the velocities of bodies a and b:

$$\mathbf{v}_{ab} = \begin{bmatrix} \mathbf{v}_a \\ \boldsymbol{\omega}_a \\ \mathbf{v}_b \\ \boldsymbol{\omega}_b \end{bmatrix}$$

We obtain $\mathbf{j}_c$ by taking $\dot{\mathrm{c}}$ and factoring out $\mathbf{v}_{ab}$. Subsection 5.2 gives an elaborate example of this.

5.2 Distance constraint

The distance constraint ensures that point $\mathbf{p}_a$[II] from rigid body a and $\mathbf{p}_b$ from rigid body b remain at fixed distance l from each other, see Figure 1. We define $\mathbf{p}_{ab}$ for convenience:

$$\mathbf{p}_{ab} = \mathbf{p}_b - \mathbf{p}_a$$

We setup the constraint like this:

$$\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \frac{1}{2}\left(|\mathbf{p}_{ab}|^2 - l^2\right) = 0$$

The $\frac{1}{2}$ and the squares are added to keep the derivative simple. We derive in order to obtain the velocity constraint:

$$\begin{aligned} \tfrac{d}{dt}\,\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) &= \tfrac{d}{dt}\left(\tfrac{1}{2}|\mathbf{p}_{ab}|^2\right) \\ &= \mathbf{p}_{ab}\tfrac{d}{dt}\mathbf{p}_{ab} \\ &= \left(\mathbf{p}_b - \mathbf{p}_a\right)\cdot\left(\tfrac{d}{dt}\mathbf{p}_b - \tfrac{d}{dt}\mathbf{p}_a\right) \end{aligned}$$

$\frac{d}{dt}\mathbf{p}_i$ is easily obtained from the body velocities:

$$\frac{d}{dt}\mathbf{p}_i = \mathbf{v}_i + \boldsymbol{\omega}_i \times \mathbf{r}_i$$

Where $\mathbf{r}_i = \mathbf{p}_i - \mathbf{x}_i$ is the position relative to center of mass of body i. Now we can expand to the full velocity constraint formula:

$$\frac{d}{dt}\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \left(\mathbf{p}_b - \mathbf{p}_a\right)\cdot\left(\mathbf{v}_b + \boldsymbol{\omega}_b \times \mathbf{r}_b - \mathbf{v}_a - \boldsymbol{\omega}_a \times \mathbf{r}_a\right)$$

[II] In this section $\mathbf{p}$ is used to indicate a point, not linear momentum.

Note how $\mathbf{p}_b - \mathbf{p}_a$ is the direction in which the constraint can apply force, also called the constraint axis. Also note that $\frac{d}{dt}\mathbf{p}_b - \frac{d}{dt}\mathbf{p}_a$ is how fast the constraint points are moving relative to each other. We project this velocity on to the constraint axis to see how fast the constraint is becoming broken (see Figure 2). After solving the constraint this should be 0 as the points may not move away or towards each other along the constraint axis (see Figure 3).

We now need to factor out velocities $\mathbf{v}_{ab}$ from the constraint definition to obtain the constraint in the form of equation 5.1.3. The remaining factor will become the constraint axis $\mathbf{j}_c$. Constraint violation (how fast points $\mathbf{p}_a$ and $\mathbf{p}_b$ move towards/away from each other) will then come from $\mathbf{j}_c\mathbf{v}_{ab}$ (projecting the velocities on the constraint axis).

$$\begin{aligned}
&\frac{d}{dt}\mathrm{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) \\
&= \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \left(\mathbf{v}_b + \boldsymbol{\omega}_b \times \mathbf{r}_b - \mathbf{v}_a - \boldsymbol{\omega}_a \times \mathbf{r}_a\right) \\
&= \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \left(\mathbf{v}_b + \boldsymbol{\omega}_b \times \mathbf{r}_b\right) - \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \left(\mathbf{v}_a + \boldsymbol{\omega}_a \times \mathbf{r}_a\right) \\
&= \begin{matrix} \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \mathbf{v}_b + \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \left(\boldsymbol{\omega}_b \times \mathbf{r}_b\right) \\ - \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \mathbf{v}_a - \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \left(\boldsymbol{\omega}_a \times \mathbf{r}_a\right) \end{matrix} \\
&= \begin{matrix} \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \mathbf{v}_b + \left(\mathbf{r}_b \times \left(\mathbf{p}_b - \mathbf{p}_a\right)\right) \cdot \boldsymbol{\omega}_b \\ - \left(\mathbf{p}_b - \mathbf{p}_a\right) \cdot \mathbf{v}_a - \left(\mathbf{r}_a \times \left(\mathbf{p}_b - \mathbf{p}_a\right)\right) \cdot \boldsymbol{\omega}_a \end{matrix} \\
&= \begin{bmatrix} - \left(\mathbf{p}_b - \mathbf{p}_a\right) \\ - \left(\mathbf{r}_a \times \left(\mathbf{p}_b - \mathbf{p}_a\right)\right) \\ \left(\mathbf{p}_b - \mathbf{p}_a\right) \\ \left(\mathbf{r}_b \times \left(\mathbf{p}_b - \mathbf{p}_a\right)\right) \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \mathbf{v}_a \\ \boldsymbol{\omega}_a \\ \mathbf{v}_b \\ \boldsymbol{\omega}_b \end{bmatrix} \\
&= \mathbf{j}_c \mathbf{v}_{ab}
\end{aligned} \tag{5.2.1}$$

Note that to get from $(\mathbf{p}_b - \mathbf{p}_a) \cdot (\boldsymbol{\omega}_b \times \mathbf{r}_b)$ to $(\mathbf{r}_a \times (\mathbf{p}_b - \mathbf{p}_a)) \cdot \boldsymbol{\omega}_b$ the properties of the scalar triple product are used.

The constraint will still be broken after solving for $\frac{d}{dt}\mathrm{c}(...) = 0$ if the position constraint was broken before, as we only eliminated the velocity along the constraint axis. Points $\mathbf{p}_a$ and $\mathbf{p}_b$ need to be moved (along the constraint axis) to make the constraint satisfied again, which means that the velocity along $\mathbf{j}_c$ should not be zero. Ideally the constraint violation will be corrected after one time step. To do this the constraint should give points $\mathbf{p}_a$ and $\mathbf{p}_b$ a velocity so that after Δt time they are at distance l from each other:

$$v = \frac{|\mathbf{p}_{ab}| - l}{\Delta t} \tag{5.2.2}$$

Now the relative velocity of $\mathbf{p}_a$ and $\mathbf{p}_b$ along the constraint axis will be equal to v after solving the system. $|\mathbf{p}_{ab}| - l$ is the distance by which the constraint is broken. This means that the bodies will travel a distance of $|\mathbf{p}_{ab}| - l$ per frame, satisfying the constraint after one frame. However, $\mathbf{p}_a$ and $\mathbf{p}_b$ will keep moving at that velocity making the constraint broken again the next frame. This might cause oscillation for this constraint or unpredictable bouncing for contact constraints that can only push the two points apart. To alleviate this problem we introduce a constant β to lower v:

$$v = \frac{|\mathbf{p}_{ab}| - l}{\Delta t}\beta \tag{5.2.3}$$

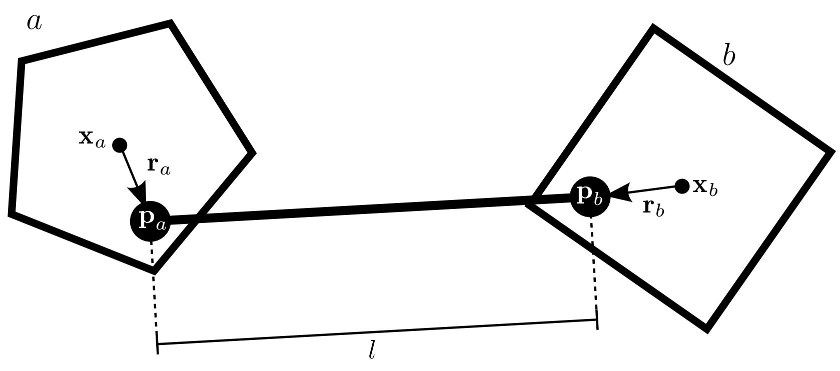


Figure 1: Distance constraint.

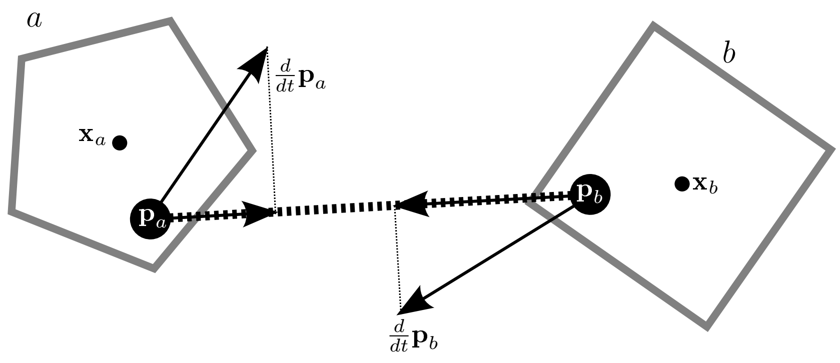


Figure 2: Distance constraint. Pojecting the velocities of $\mathbf{p}_i$ on the constraint axis.

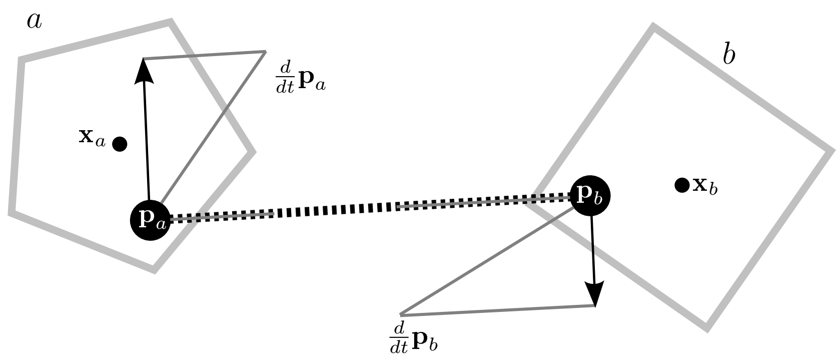


Figure 3: Desired velocities for $\dot{c}(...) = 0$.

Where $\beta \in [0,1]$. The higher β the faster constraint violations will be corrected and the more erroneous velocity will be added to the system. Using the position constraint to correct the drift in its derivative is known as Baumgarte stabilization.

The distance constraint force is not clamped since the constraint can both push and pull the two points.

$$c_{\text{max}} = +\infty$$
$$c_{\text{min}} = -\infty$$

5.3 Contact constraint

This constraint is added to the system for each contact point in the manifold generated by the collision detection stage. $\mathbf{n}$ is the contact point normal. $\mathbf{r}_i$ is the offset from the center of mass of rigid body i to the contact point. The contact constraint is setup in the same way as the distance constraint except that $\mathbf{p}_a$ and $\mathbf{p}_b$ are the same point, see figure 4. This makes it impossible to calculate the constraint axis, so we use $\mathbf{n}$ instead. This way the velocity constraint will eliminate all movement in the direction of the contact normal, and the push factor can push both bodies away from each other along the normal.

Taking $\mathbf{j}_c$ from Equation 5.2.1 and replacing $\mathbf{p}_b - \mathbf{p}_a$ with $\mathbf{n}$:

$$\mathbf{j}_c = \begin{pmatrix} -\mathbf{n} \\ -\mathbf{r}_a \times \mathbf{n} \\ \mathbf{n} \\ \mathbf{r}_b \times \mathbf{n} \end{pmatrix}^{\mathsf{T}}$$

The constraint force is clamped so that it can only push the bodies apart:

$$\mathbf{c}_{\text{max}} = +\infty$$
$$\mathbf{c}_{\text{min}} = 0$$

v is also calculated in the same way as Equation 5.2.3:

$$v = \frac{d}{\Delta t}\beta \qquad (5.3.1)$$

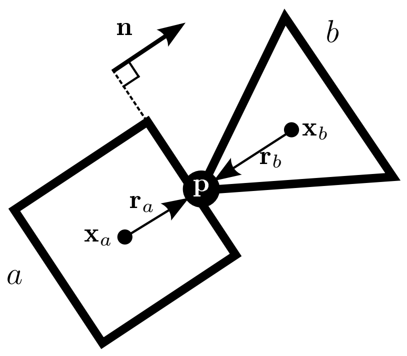


Figure 4: Contact constraint collision diagram.

Where d is the penetration depth of the contact point.

The constraint can be simplified if rigid body b is static:

$$\mathbf{j}_c = \begin{pmatrix} -\mathbf{n} \\ -\mathbf{r}_a \times \mathbf{n} \\ 0 \\ 0 \end{pmatrix}^{\mathsf{T}}$$

5.4 Friction constraint

The friction constraint uses exactly the same formula for $\mathbf{j}_c$ as in Section 5.3, but instead of using the contact normal, a perpendicular direction is used. This way two friction constraints (both with a perpendicular direction to $\mathbf{n}$ and perpendicular to each other) can slow down the movement along the contact plane of the bodies.

It is usually enough to use a v of 0, even when the maximum friction force is bigger than the constraint force, to keep objects from sliding down slopes.

The maximum friction force is calculated with a simplified model like this:

$$-c_{\text{min}} = c_{\text{max}} = \mu m \mathbf{g}\mathbf{n}$$

Where m is the combined mass of both bodies divided over the contact points, $\mathbf{g}$ is gravity and $\mathbf{n}$ is the contact normal. The mass of the movable body is used when one of the bodies is static. This model is correct when a movable body is resting/sliding on top of a static body. This model is incorrect when multiple movable bodies are stacked on top of each other as $m\mathbf{g}\mathbf{n}$ is not a good approximation for the normal force in this case.

6 Constraint solver

In this section we show how the solver works.

- In Subsection 6.1 we show how we present the information about the entire system to the solver.
- In Subsection 6.2 we derive the system equation, and show how to solve the entire system with an iterative method.

6.1 Variable definitions

We want the constraint solver to solve an entire system. To make this possible we need to describe the whole system in a single equation, and then solve that equation. To do this we need to represent the information about the rigid bodies in such a way that the constraints can connect them. Remember we defined the constraint derivative to be split up like $\dot{\mathbf{c}}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \mathbf{j}_c \mathbf{v}_{ab}$ in Subsection 5.1, but we want to have all constraints of the system in one equation. To do this we put $\mathbf{j}_c$ for each constraint as a row into one big matrix $\mathbf{J}$, and we put the velocities of all the bodies in $\mathbf{v}$. We do this in such a way that $\mathbf{J}\mathbf{v}$ calculates $\mathbf{j}_c \mathbf{v}_{ab}$ for every constraint.

6.1.1 System velocity

We define $\mathbf{v}$ as an $\mathbb{R}^{6n}$ column vector for n bodies:

$$\mathbf{v} = [\mathbf{v}_{1x}, \mathbf{v}_{1y}, \mathbf{v}_{1z}, \boldsymbol{\omega}_{1x}, \boldsymbol{\omega}_{1y}, \boldsymbol{\omega}_{1z}, \ldots, \mathbf{v}_{nx}, \mathbf{v}_{ny}, \mathbf{v}_{nz}, \boldsymbol{\omega}_{nx}, \boldsymbol{\omega}_{ny}, \boldsymbol{\omega}_{nz}]^{\mathsf{T}}$$

Such vectors will be shortened as:

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \boldsymbol{\omega}_1 \\ \vdots \\ \mathbf{v}_n \\ \boldsymbol{\omega}_n \end{bmatrix} \tag{6.1.1}$$

Where each element represents 3 scalars.

6.1.2 System constraints

We define $\mathbf{J}$ as a $6n \times c$ matrix where n is the number of bodies in the system, and c is the number of constraints in the system. Each row of $\mathbf{J}$ looks like this:

$$\mathbf{J}_c = \begin{bmatrix} \ldots & \mathbf{j}_c^{1\ldots6} & \ldots & \mathbf{j}_c^{7\ldots12} & \ldots \end{bmatrix}$$

$\mathbf{j}_c$ is split in two parts. The first part $(1\ldots6)$ is the constraint axis for body a, and the second part $(7\ldots12)$ is the axis for body b. The dots represent zero filled padding used to align the first and second parts with $\mathbf{v}$ so that $\mathbf{Jv}$ performs a series of dot products between the correct velocities and the constraint matrix like this:

$$\mathbf{J}_c\mathbf{v} = \begin{bmatrix} \ldots & \mathbf{j}_c^{0\ldots6} & \ldots & \mathbf{j}_c^{7\ldots12} & \ldots \end{bmatrix} \begin{bmatrix} \vdots \\ \mathbf{v}_a \\ \boldsymbol{\omega}_a \\ \vdots \\ \mathbf{v}_b \\ \boldsymbol{\omega}_b \\ \vdots \end{bmatrix} = \mathbf{j}_c\mathbf{v}_{ab}$$

For a single row this computes the sum of the projected velocities along the constraint directions. This sum represents the total violation along this constraint axis. Our goal is to calculate the constraint force $\mathbf{f}_c$ that will change $\mathbf{v}$ so that the next frame the violation is $\mathbf{Jv} = 0$.

We store the constraint limits for all constraints in vectors $\mathbf{c}_{\text{min}}$ and $\mathbf{c}_{\text{max}}$.

6.1.3 System mass

We introduce the system inverse mass which is a convenient way to convert forces into accelerations for all bodies at the same time:

$$\mathbf{M}^{-1} = \begin{bmatrix} [m_1^{-1}\mathbf{D}] & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_1^{-1} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & [m_n^{-1}\mathbf{D}] & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{I}_n^{-1} \end{bmatrix} \tag{6.1.2}$$

Where each element in the above matrix is a 3×3 matrix. $\mathbf{0}$ is a matrix containing only zeroes and $\mathbf{D}$ is the identity matrix[III]. This is a convenient notation as we can, for example, get $\mathbf{v}_n m_n^{-1}$ and $\boldsymbol{\omega}_n \mathbf{I}_n^{-1}$ for the entire system like this: $\mathbf{M}^{-1}\mathbf{v}$.

Given a vector $\mathbf{f}$ of forces, describing a force and torque for each body:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \boldsymbol{\tau}_1 \\ \vdots \\ \mathbf{f}_n \\ \boldsymbol{\tau}_n \end{bmatrix} \tag{6.1.3}$$

We can easily get acceleration $\mathbf{a}_n = \mathbf{f}_n m_n^{-1}$ and angular acceleration $\boldsymbol{\alpha}_n = \boldsymbol{\tau}_n \mathbf{I}_n^{-1}$ for the entire system like this: $\mathbf{M}^{-1}\mathbf{f}$.

6.1.4 Constraint push factor

In Section 5.1 we showed that solving $\mathbf{j}_c \mathbf{v}_{ab} = 0$ will not correct broken constraints. This is the same for $\mathbf{Jv} = 0$. We define $\boldsymbol{\upsilon}$ as an $\mathbb{R}^c$ column vector for c constraints containing υ for every constraint:

$$\boldsymbol{\upsilon} = \begin{bmatrix} \upsilon_1 \\ \vdots \\ \upsilon_c \end{bmatrix} \tag{6.1.4}$$

Now we can solve for $\mathbf{Jv} = \boldsymbol{\upsilon}$ to also correct broken constraints for the entire system.

6.1.5 Equations of motion

Now we show how we can use the variables defined in this subsection to extend the basic equations of motion in order to describe a whole system with one equation.

We start with the basic equations of motion for both linear and angular physics:

$$\mathbf{F} = m\mathbf{a} = m\dot{\mathbf{v}} = \mathbf{f}_c + \mathbf{f}_{\text{ext}} \tag{6.1.5}$$

$$\boldsymbol{\tau} = \mathbf{I}\boldsymbol{\alpha} = \mathbf{I}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau}_c + \boldsymbol{\tau}_{\text{ext}}{}^{\text{IV}} \tag{6.1.6}$$

$\mathbf{f}_{\text{ext}}$ and $\boldsymbol{\tau}_{\text{ext}}$ are external forces we can add to the system without adding constraints. This is useful to add gravity for example.

We substitute m and $\mathbf{I}$ with $\mathbf{M}$, and we substitute $\mathbf{v}$ and $\boldsymbol{\omega}$ with $\mathbf{v}$ from Equation 6.1.1:

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{f}_c + \mathbf{f}_{\text{ext}} \tag{6.1.7}$$

Where $\mathbf{f}_c$ ($\mathbf{f}_c$ and $\boldsymbol{\tau}_c$ combined from the basic equations of motion with Equation 6.1.3) is the constraint reaction force which we will discuss in detail in the next subsection, and $\mathbf{f}_{\text{ext}}$ is $\mathbf{f}_{\text{ext}}$ and $\boldsymbol{\tau}_{\text{ext}}$ also combined with Equation 6.1.3.

[III] $\mathbf{D}$ is used for the identity matrix as $\mathbf{I}$ is already used for the inertia tensor.

[IV] We ignore Coriolis forces for simplicity.

6.2 Setting up the system equation

In this subsection we will show how the solver is setup.

- In Subsection 6.2.1 we show how the reaction forces are calculated.
- In Subsection 6.2.2 we discuss the requirements for the unknown from Subsection 6.2.1, and we show how to setup a system of linear equations used to solve the unknown from Subsection 6.2.1.
- In Subsection 6.2.3 we show how to solve the system of equations from Section 12.3.1.

6.2.1 Constraint forces

We defined the constraints so that $\dot{c}\left(\mathbf{x}_a, \mathbf{q}_a, \mathbf{x}_b, \mathbf{q}_b\right) = \mathbf{J}_c\mathbf{v}$. We want to stop any movement that will break the constraints. So we want to have $\mathbf{J}\mathbf{v} = 0$ after constraint resolution. Constraint forces do not perform work as they ensure a velocity of 0 (relative to the constraint itself) in the direction the constraint is acting. This means that constraint forces only eliminate all movement in the direction of the constraint. So after constraint resolution $\mathbf{f}_c\mathbf{v} = 0$.

Since all the constraint axes in $\mathbf{J}$ are orthogonal to the velocities in $\mathbf{v}$ and the reaction forces $\mathbf{f}_c$ are also orthogonal to $\mathbf{v}$ and constraints can only apply forces along their axis, $\mathbf{f}_c$ can be expressed as a linear combination of the constraint axes in $\mathbf{J}$:

$$\mathbf{f}_c = \mathbf{J}^\mathsf{T}\boldsymbol{\lambda} \tag{6.2.1}$$

Where $\boldsymbol{\lambda}$ is an $\mathbb{R}^c$ vector for c constraints that holds one scalar for each constraint, indicating how much force that constraint needs to apply in order to keep it satisfied. This is the unknown we are going to solve for in the next subsection. $\mathbf{f}_c$ is an $\mathbb{R}^{6n}$ vector for n bodies in the system that holds the force and torque that needs to be applied for every body to keep the constraints satisfied.

6.2.2 Constraints for lambda

We know that

$$\mathbf{f}_c\mathbf{v} = 0$$

after constraint resolution. Substituting Equation 6.2.1 for $\mathbf{f}_c$ we get:

$$\mathbf{J}^\mathsf{T}\boldsymbol{\lambda}\mathbf{v} = 0 \tag{6.2.2}$$

This is the first constraint for $\boldsymbol{\lambda}$, meaning we only want any $\boldsymbol{\lambda}$ for which Equation 6.2.2 holds. We can use the properties of matrix transpose while treating both $\boldsymbol{\lambda}$ and $\mathbf{v}$ as column matrices to prove that any $\boldsymbol{\lambda}$ satisfies Equation 6.2.2:

$$(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda})^\mathsf{T}\mathbf{v} = (\boldsymbol{\lambda}^\mathsf{T}\mathbf{J})\mathbf{v} = \boldsymbol{\lambda}^\mathsf{T}(\mathbf{J}\mathbf{v}) = \boldsymbol{\lambda}^\mathsf{T}(0) = 0 \tag{6.2.3}$$

This means that we cannot use 6.2.2 to solve for $\boldsymbol{\lambda}$. However, $\boldsymbol{\lambda}$ in Equation 6.2.1 also has to generate a force that will change $\mathbf{v}$ so that $\mathbf{J}\mathbf{v}$ will become 0 after constraint resolution, which is our second constraint for $\boldsymbol{\lambda}$.

We approximate $\dot{\mathbf{v}}$ (the change of $\mathbf{v}$) with a single Euler step:

$$\dot{\mathbf{v}} \approx \frac{\mathbf{v}_2 - \mathbf{v}_1}{\Delta t} \tag{6.2.4}$$

Where $\mathbf{v}_1$ is the current system velocity and $\mathbf{v}_2$ is the next system velocity (after Δt from now).

Now we can write our second constraint as follows:

$$\mathbf{J}\mathbf{v}_2 = 0 \tag{6.2.5}$$

And using Equation 6.2.4 we can make $\dot{\mathbf{v}}$ the unknown:

$$\begin{aligned} \mathbf{J}(\mathbf{v}_1 + \dot{\mathbf{v}}\Delta t) &= 0 \\ \mathbf{J}\mathbf{v}_1 + \mathbf{J}\dot{\mathbf{v}}\Delta t &= 0 \end{aligned} \tag{6.2.6}$$

Now we want to express $\dot{\mathbf{v}}\Delta t$ with only $\boldsymbol{\lambda}$ as unknown. We take Equation 6.1.7 and substitute $\mathbf{f}_c$ with Equation 6.2.1:

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{f}_{\text{ext}} \tag{6.2.7}$$

Which we solve for $\dot{\mathbf{v}}$:

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$$

Now we combine this with Equation 6.2.6:

$$\mathbf{J}\mathbf{v}_1 + \mathbf{J}\left(\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right)\Delta t = 0$$

And we solve for $\boldsymbol{\lambda}$ as far as we can:

$$\begin{aligned} \mathbf{J}\mathbf{v}_1 + \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda}\Delta t + \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\Delta t &= 0 \\ \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda}\Delta t &= -\mathbf{J}\mathbf{v}_1 - \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\Delta t \\ \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} &= -\tfrac{1}{\Delta t}\mathbf{J}\mathbf{v}_1 - \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_{\text{ext}} \\ \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} &= \mathbf{J}\left(-\tfrac{1}{\Delta t}\mathbf{v}_1 - \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right) \\ \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} &= -\mathbf{J}\left(\tfrac{1}{\Delta t}\mathbf{v}_1 + \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right) \end{aligned} \tag{6.2.8}$$

If we want the constraint to perform work (to enforce the correction of violated constraints such as compenetration) we can use the bias vector $\boldsymbol{\upsilon}$ such that $\mathbf{J}\mathbf{v} = \boldsymbol{\upsilon}$ and repeat the same steps we took to get to Equation 6.2.8. This results in the following:

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda} = \frac{1}{\Delta t}\boldsymbol{\upsilon} - \mathbf{J}\left(\frac{1}{\Delta t}\mathbf{v}_1 + \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right) \tag{6.2.9}$$

We define some constants:

$$\begin{aligned} \mathbf{A} &= \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^{\mathsf{T}} \\ \mathbf{b} &= \tfrac{1}{\Delta t}\boldsymbol{\upsilon} - \mathbf{J}\left(\tfrac{1}{\Delta t}\mathbf{v}_1 + \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right) \end{aligned}$$

So that Equation 6.2.9 can be written as:

$$\mathbf{A}\boldsymbol{\lambda} = \mathbf{b}$$

Now $\boldsymbol{\lambda}$ can be obtained by solving this linear system. Knowing $\boldsymbol{\lambda}$, Equation 6.2.1 can be used to calculate the reaction forces.

We show another way to get to Equation 6.2.9 in Subsection 12.3.1.

6.2.3 Iterative solution

To solve $\mathbf{b} = \mathbf{A}\boldsymbol{\lambda}$ for $\boldsymbol{\lambda}$ an iterative methods like the Jacobi or Projected Gauss–Seidel method can be used. We clamp $\boldsymbol{\lambda}$ each iteration to the limit given by $\mathbf{c}_{\text{min}}$ and $\mathbf{c}_{\text{max}}$ to ensure that the constraint reaction force will not be bigger than this limit.

These iterative methods use an initial guess for $\boldsymbol{\lambda}$. This can be $\mathbf{0}$, but the value from previous the frame can be used by caching the constraint information [2].

The algorithm can be terminated on several conditions. We have found that for solving simple systems a fixed number of iterations yields good results and predictable timings.

7 Projected Gauss-Seidel

In this section we show how to use projected Gauss-Seidel to solve a linear system structured as $\mathbf{b} = \mathbf{A}\boldsymbol{\lambda}$ where $\mathbf{A}$ and $\mathbf{b}$ are known and $\mathbf{A}$ is a square matrix with a non zero diagonal[5].

7.1 Deriving the method

We rewrite

$$\mathbf{b} = \mathbf{A}\boldsymbol{\lambda}$$

for the purpose of an example as:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ d_1 & d_2 & d_3 \\ e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix}$$

Where $\mathbf{b}$ can be expanded as:

$$\mathbf{b} = \begin{bmatrix} c_1\lambda_1 + c_2\lambda_2 + c_3\lambda_3 \\ d_1\lambda_1 + d_2\lambda_2 + d_3\lambda_3 \\ e_1\lambda_1 + e_2\lambda_2 + e_3\lambda_3 \end{bmatrix}$$

Now we rearrange each row of $\mathbf{b}$ to give a different scalar from $\boldsymbol{\lambda}$:

$$\begin{aligned} b_1 &= c_1\lambda_1 + c_2\lambda_2 + c_3\lambda_3 \\ b_1 - c_2\lambda_2 - c_3\lambda_3 &= c_1\lambda_1 \\ (b_1 - c_2\lambda_2 - c_3\lambda_3)\, c_1^{-1} &= \lambda_1 \end{aligned}$$

So we end up with:

$$\begin{aligned} (b_1 - c_2\lambda_2 - c_3\lambda_3)\, c_1^{-1} &= \lambda_1 \\ (b_2 - d_1\lambda_1 - d_3\lambda_3)\, d_2^{-1} &= \lambda_2 \\ (b_3 - e_1\lambda_1 - e_2\lambda_2)\, e_3^{-1} &= \lambda_3 \end{aligned}$$

or more general[5]:

$$\boldsymbol{\lambda}_j = \left(\mathbf{b}_j - \sum_{i=1}^{n} (\mathbf{A}_{ji}\boldsymbol{\lambda}_i) + \mathbf{A}_{jj}\boldsymbol{\lambda}_j \right) \mathbf{A}_{jj}^{-1} \tag{7.1.1}$$

Now we have a formula for every λ_n which is depending on every other λ_n. All we have to do is use a guess/approximation for all the other λ's to get a better approximation. This better approximation can be used to solve for an even better approximation. More iterations gives a better result.

The Jacobi method uses the newly calculated λ's only after the iteration is finished. Gauss-Seidel method uses the new λ's immediately while the iteration is running to converge faster.

Projected Gauss-Seidel clamps λ to the range given by $c_{\min}$ and $c_{\max}$ in each iteration.

7.2 Projected Gauss-Seidel algorithm

```
BigVector SolveProjectedGaussSeidel(A, b, cmin, cmax,
                                    iterations)
  let lambda be a new vector with b.size elements

  //Set initial guess to 0
  lambda.ZeroAll();

  //Do the algorithm k iterations
  for k = 1 to iterations
    //Solve every row
    for j = 0 to A.height-1
      //calculate lambda[j] = (b[j] - sum) / A[j,j]
      lambda[j] = b[j]

      //subtract sum
      for int i = 0 to lambda.size-1
        //same as adding A[j,j]*lambda[j]
        if(i==j)
          continue

        lambda[j] -= A[i,j]*lambda[i]

      //divide
      lambda[j] /= A[j,j]

      //project
      lambda[j] = MIN(lambda[j],cmax[j])
      lambda[j] = MAX(lambda[j],cmin[j])
  return lambda
```

Where `A` is the matrix **A**, and `b` is the vector **b** from last subsection. This algorithm assumes zero-based numbering. `A[i,j]` Selects the element located at the `i`th column and `j`th row.

8 The push factor

In this section we discuss other uses for the push factor υ.

- In Subsection 8.1 we discuss a way to improve the resolution of broken constraints using an additional constraint resolution step.

- In Subsection 8.2 we discuss how to use the push factor to add elastic collisions to the simulation.

8.1 Single frame push factor

In Subsection 5.2 we discussed how to use υ to correct violated constraints by introducing a velocity. We also showed that this erroneous velocity can be problematic for the simulation as it will still affect the body after the constraint is corrected.

In the case of contact constraints bodies might bounce unpredictable. This happens if discrete collision detection is used. The bounce will depend on the penetration depth and k from Equation 5.2.3. Figure 5 illustrates what would happen when $k = 1$. In frame 2 collision is detected after the box has penetrated the floor. υ generates a velocity so that the box is on top of the floor again in frame 3. The box, however, still has this velocity in frame 4, causing it to bounce up again.

Reducing k will make sure that the bounce will be lower, but it also causes the box to be inside the floor longer. What we really want is that the velocity introduced by υ in frame 2 is removed again in frame 3, so that the box will settle on the floor immediately.

To do this we need to solve the system twice. First we solve the system like usual, and store the constraint forces in $\mathbf{f}_{c\upsilon}$. Then we solve the system with $\upsilon = 0$, and store the forces in $\mathbf{f}_{c0}$. Now we have two sets of forces, $\mathbf{f}_{c\upsilon}$ is the force that will correct violated constraints. $\mathbf{f}_{c0}$ is the force that will stop any movement along the constraint axes. We use $\mathbf{f}_{c\upsilon}$ to calculate the translation for one frame, and update the body's position accordingly. While we use $\mathbf{f}_{c0}$ to calculate the body's velocity. This way we have a translation that fixes the constraint, and the correct velocity as if $k = 0$.

It is still a good idea to use a value for k that is less than 1 so that bodies will always stay in contact with each other. Otherwise the collision detection might not generate the contact points for the resting bodies each frame.

Note that solving the system for $\mathbf{f}_{c0}$ requires υ to exclude constraint correcting push factors. If υ is used for other purposes that will add velocities to the simulation (like restitution or constant speed motors) those addends should be left in, otherwise these values will not cause an acceleration.

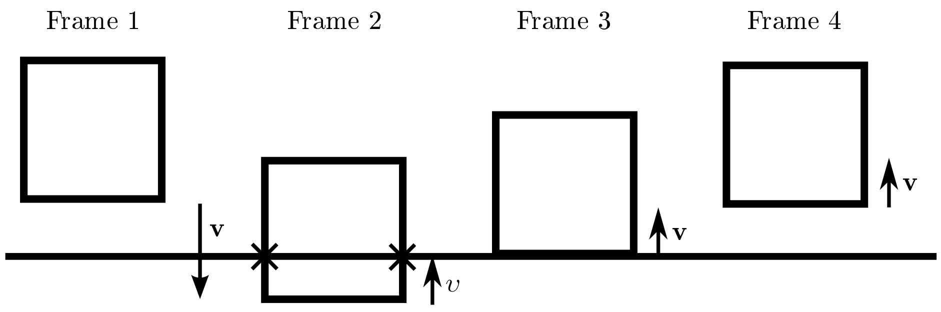


Figure 5: Error caused by the push factor.

8.2 Elastic collisions

υ can also be used to make collisions elastic. In Section 5 we derived constraints assuming that the velocity along the constraint axis would be 0 after constraint resolution. Then we stated that, to fix violated constraints, it would not be 0 as we needed to cause a translation to fix the constraint. An elastic collision is also a case where the velocity along the constraint axis is not 0 after resolution.

We know that a collision with a coefficient of restitution of 1 will have $\mathbf{j}_c\mathbf{v}_1 = -\mathbf{j}_c\mathbf{v}_2$ as the coefficient of restitution is the ratio of relative speed before and after the collision[4]. In this case $\upsilon_r = -\mathbf{j}_c\mathbf{v}_1$ where υ_r is υ excluding the addends dealing with correcting violated constraints. If we want to use a different coefficient of restitution we can use:

$$\upsilon_r = -\mathbf{j}_c\mathbf{v}_1\epsilon \tag{8.2.1}$$

where ϵ is the coefficient of restitution.

This can be combined with the push factor described in Section 5 by adding Equations 8.2.1 and 5.2.3.

Note that the addend based on Equation 8.2.1 should be included in both systems when using the technique from Subsection 8.1 otherwise all collisions will be completely inelastic.

9 Dimensional analysis

The equations we derived in Subsection 6.2 are mathematically correct. However, it is good practice to verify if the derived equations are plausible by checking if the units/dimensions on the left and right side are the same. It is also useful to know with what quantities the equations work, so that real life measurements can be used as values for the simulation. And finally it may help us better understand how the equations work.

- In Subsection 9.1 we discuss the dimensions of the variables defined in Subsection 6.1
- In Subsection 9.2 we show that the dimensions of the equations described in Subsection 6.2 are correct.

9.1 Dimensions of the system variables

$\mathbf{v}$ as defined by Equation 6.1.1 has a combination of two units. For the linear part the unit is $\mathrm{m\,s^{-1}}$. For the angular part $\mathrm{rad\,s^{-1}}$ which is the same as $\mathrm{s^{-1}}$ because radians are the ratio between the distance along the circumference of a circle and the radius of a circle, or $\mathrm{rad} = \frac{\mathrm{m}}{\mathrm{m}}$. rad will be omitted in the rest of this section.

The dimension of the constraint axis matrix $\mathbf{J}$ can be derived from the fact that constraints do not perform work:

$$\mathbf{Jv} = 0$$

We know the units of work and we know the units of $\mathbf{v}$ both for the linear part:

$$\begin{array}{r|l} \mathbf{J}_{\text{lin}}\mathbf{v}_{\text{lin}} = 0 & \mathrm{kg\,m^2\,s^{-2}} \\ \frac{\mathbf{J}_{\text{lin}}\mathbf{v}_{\text{lin}}}{\mathbf{v}_{\text{lin}}} & \frac{\mathrm{kg\,m^2\,s^{-2}}}{\mathrm{m\,s^{-1}}} \\ \mathbf{J}_{\text{lin}} & \mathrm{kg\,m\,s^{-1}} \end{array}$$

And the angular part:

$$\begin{array}{r|l} \mathbf{J}_{\text{ang}}\mathbf{v}_{\text{ang}} = 0 & \mathrm{kg\,m^2\,s^{-2}} \\ \frac{\mathbf{J}_{\text{ang}}\mathbf{v}_{\text{ang}}}{\mathbf{v}_{\text{ang}}} & \frac{\mathrm{kg\,m^2\,s^{-2}}}{\mathrm{s^{-1}}} \\ \mathbf{J}_{\text{ang}} & \mathrm{kg\,m^2\,s^{-1}} \end{array}$$

Note that the units of angular work can be derived from $w = \tau\phi$ where w is work and ϕ is the displacement angle[7].

The dimension of $\mathbf{M}$ as defined by Equation 6.1.2 is kg for the linear part and $\mathrm{kg\,m^2}$ for the angular part.

The dimension of $\mathbf{f}$ as defined by Equation 6.1.3 is $\mathrm{kg\,m\,s^{-2}}$ for the linear part and $\mathrm{kg\,m^2\,s^{-2}}$ for the angular part.

The units of $\boldsymbol{\lambda}$ can be derived from Equation 6.2.1:

$$\begin{array}{r|l} \mathbf{f}_{\text{lin}} = \mathbf{J}_{\text{lin}}^{\mathsf{T}}\boldsymbol{\lambda} & \mathrm{kg\,m\,s^{-2}} = \mathrm{kg\,m\,s^{-1}} \cdot \mathrm{s^{-1}} \\ \boldsymbol{\lambda} & \mathrm{s^{-1}} \end{array}$$

And for the angular part:

$$\begin{array}{r|l} \mathbf{f}_{\text{ang}} = \mathbf{J}_{\text{ang}}^{\mathsf{T}}\boldsymbol{\lambda} & \mathrm{kg\,m^2\,s^{-2}} = \mathrm{kg\,m^2\,s^{-1}} \cdot \mathrm{s^{-1}} \\ \boldsymbol{\lambda} & \mathrm{s^{-1}} \end{array}$$

The units are the same for both parts. This is correct as $\boldsymbol{\lambda}$ has only one scalar per constraint, forcing it to use one unit for both the linear part and the angular part.

The dimensions of $\boldsymbol{\upsilon}$ can be derived from Equation 12.3.2:

$$\begin{array}{r|l} \mathbf{J}_{\text{lin}}\mathbf{v}_{\text{lin}} = \boldsymbol{\upsilon} & \mathrm{kg\,m\,s^{-1}} \cdot \mathrm{m\,s^{-1}} \\ \boldsymbol{\upsilon} & \mathrm{kg\,m^2\,s^{-2}} \end{array}$$

And for the angular part:

$$\begin{array}{r|l} \mathbf{J}_{\text{ang}}\mathbf{v}_{\text{ang}} = \boldsymbol{\upsilon} & \mathrm{kg\,m^2\,s^{-1}} \cdot \mathrm{s^{-1}} \\ \boldsymbol{\upsilon} & \mathrm{kg\,m^2\,s^{-2}} \end{array}$$

The units are again the same for both parts as $\boldsymbol{\upsilon}$ also only has one scalar per constraint.

9.2 Validating the system formula

Now we show that the equations from Subsection 6.2 have the same units on the left and the right side. The original equation is written on the first line, the corresponding units for the linear part on the second line, and the units for the angular part on the third line. Combining all the factors shows that the units match on both sides, and thus showing that the equations produce meaningful results.

We start with validating the equations of motion, beginning with Equation 6.2.7:

$$\begin{array}{ccccccc} \mathbf{M} & \dot{\mathbf{v}} & = & \mathbf{J}^{\mathsf{T}} & \boldsymbol{\lambda} & + & \mathbf{f}_{\text{ext}} \\ \mathrm{kg} & \mathrm{m\,s^{-2}} & = & \mathrm{kg\,m\,s^{-1}} & \mathrm{s^{-1}} & + & \mathrm{kg\,m\,s^{-2}} \\ \mathrm{kg\,m^2} & \mathrm{s^{-2}} & = & \mathrm{kg\,m^2\,s^{-1}} & \mathrm{s^{-1}} & + & \mathrm{kg\,m^2\,s^{-2}} \end{array}$$

Equation 12.3.1:

$$\begin{array}{ccccccccc}
\mathbf{v}_2 & = & \Delta t & \mathbf{M}^{-1} & \Big(\mathbf{J}^{\mathsf{T}} & \boldsymbol{\lambda} & + & \mathbf{f}_{ext}\Big) & + & \mathbf{v}_1 \\
\mathrm{m\,s^{-1}} & = & \mathrm{s} & \mathrm{kg^{-1}} & \Big(\mathrm{kg\,m\,s^{-1}} & \mathrm{s^{-1}} & + & \mathrm{kg\,m\,s^{-2}}\Big) & + & \mathrm{m\,s^{-1}} \\
\mathrm{s^{-1}} & = & \mathrm{s} & \mathrm{kg^{-1}\,m^{-2}} & \Big(\mathrm{kg\,m^2\,s^{-1}} & \mathrm{s^{-1}} & + & \mathrm{kg\,m^2\,s^{-2}}\Big) & + & \mathrm{s^{-1}}
\end{array}$$

Equation 12.3.3 in two parts. First the left side:

$$\begin{array}{cccccccc}
\frac{1}{\Delta t} & \boldsymbol{\upsilon} & - & \mathbf{J} & \Big(\frac{1}{\Delta t} & \mathbf{v}_1 & - & \mathbf{M}^{-1} & \mathbf{f}_{\mathrm{ext}}\Big) \\
\mathrm{s^{-1}} & \mathrm{kg\,m^2\,s^{-2}} & - & \mathrm{kg\,m\,s^{-1}} & \Big(\mathrm{s^{-1}} & \mathrm{m\,s^{-1}} & - & \mathrm{kg^{-1}} & \mathrm{kg\,m\,s^{-2}}\Big) \\
\mathrm{s^{-1}} & \mathrm{kg\,m^2\,s^{-2}} & - & \mathrm{kg\,m^2\,s^{-1}} & \Big(\mathrm{s^{-1}} & \mathrm{s^{-1}} & - & \mathrm{kg^{-1}\,m^{-2}} & \mathrm{kg\,m^2\,s^{-2}}\Big)
\end{array}$$

Next the right side:

$$\begin{array}{cccc}
\mathbf{J} & \mathbf{M}^{-1} & \mathbf{J}^{\mathsf{T}} & \boldsymbol{\lambda} \\
\mathrm{kg\,m\,s^{-1}} & \mathrm{kg^{-1}} & \mathrm{kg\,m\,s^{-1}} & \mathrm{s^{-1}} \\
\mathrm{kg\,m^2\,s^{-1}} & \mathrm{kg^{-1}\,m^{-2}} & \mathrm{kg\,m^2\,s^{-1}} & \mathrm{s^{-1}}
\end{array}$$

Both sides have $\mathrm{kg\,m^2\,s^{-3}}$ as unit for the linear part, and $\mathrm{kg\,m^2\,s^{-3}}$ for the angular part.

The constraint formulas from Section 5 are not complying with the units of $\mathbf{J}$. This is not a problem in reality as the solver will take care of it as long as each whole row of $\mathbf{J}$ uses consistent units for the corresponding linear and angular parts. This is because the solver will take care of the incorrect scaling when calculating $\boldsymbol{\lambda}$. However care must be taken as each element in $\boldsymbol{\upsilon}$ needs to follow the units of each row of $\mathbf{J}$ so that $\boldsymbol{\upsilon}$ is in the same units as $\mathbf{Jv}$. As long as this is true units can be exchanged between $\mathbf{J}$ and $\boldsymbol{\lambda}$. Following these rules we can make it so that $\mathbf{J}$ has no linear unit and m as angular unit, and $\boldsymbol{\upsilon}$ has the same units as $\mathbf{v}$:

Equation 12.3.3 left side:

$$\begin{array}{ccccccccc}
\frac{1}{\Delta t} & \boldsymbol{\upsilon} & - & \mathbf{J} & \Big(\frac{1}{\Delta t} & \mathbf{v}_1 & - & \mathbf{M}^{-1} & \mathbf{f}_{\mathrm{ext}}\Big) \\
\mathrm{s^{-1}} & \mathrm{m\,s^{-1}} & - & & \Big(\mathrm{s^{-1}} & \mathrm{m\,s^{-1}} & - & \mathrm{kg^{-1}} & \mathrm{kg\,m\,s^{-2}}\Big) \\
\mathrm{s^{-1}} & \mathrm{m\,s^{-1}} & - & \mathrm{m} & \Big(\mathrm{s^{-1}} & \mathrm{s^{-1}} & - & \mathrm{kg^{-1}\,m^{-2}} & \mathrm{kg\,m^2\,s^{-2}}\Big)
\end{array}$$

Right side:

$$\begin{array}{cccc}
\mathbf{J} & \mathbf{M}^{-1} & \mathbf{J}^{\mathsf{T}} & \boldsymbol{\lambda} \\
 & \mathrm{kg^{-1}} & & \mathrm{m\,kg\,s^{-2}} \\
\mathrm{m} & \mathrm{kg^{-1}\,m^{-2}} & \mathrm{m} & \mathrm{m\,kg\,s^{-2}}
\end{array}$$

Resulting in both sides having $\mathrm{m\,s^{-2}}$ as linear unit and angular unit. Checking back if $\mathbf{J}^{\mathsf{T}}\boldsymbol{\lambda}$ still produces a force and torque:

$$\begin{array}{cccc}
\mathbf{f} & = & \mathbf{J}^{\mathsf{T}} & \boldsymbol{\lambda} \\
\mathrm{kg\,m\,s^{-2}} & = & & \mathrm{m\,kg\,s^{-2}} \\
\mathrm{kg\,m^2\,s^{-2}} & = & \mathrm{m} & \mathrm{m\,kg\,s^{-2}}
\end{array}$$

The units of the constraints from Section 5 are correct in this case. This also matches the explanation of $\mathbf{J}$ being an axis better, as axes do not have units, and $\boldsymbol{\upsilon}$ is now in $\mathrm{m\,s^{-1}}$ which is intuitive for the push factor explanation from Equation 5.2.3.

10 Results

In this section we show and discuss results from a couple of simulations made with the described physics solver.

10.1 Stack test

The stacking test is a common test for physics engines. The time it takes before a stack falls over without using a sleeping mechanism for inactive bodies tells us how precise the physics engine is. In Figure 6 can be seen how the implementation written for this paper behaves with a stack of 5 cubes with size 2.4m and mass 1.2kg under influence of a gravitational acceleration of $-10\mathrm{m\,s}^{-2}$. The solver uses 25 iterations without contact caching, and $k = 0.2$ for Equation 5.2.3. We run the simulation at a fixed time step for 60Hz. At the start the cubes' center of mass are separated by 3m leaving a space of 0.6m between them, and the bottom cube's center of mass is 2m above the ground.

It takes about 85 seconds for the stack to fall over, however incorrect movement is already quite noticeable from the start.

10.2 Intersecting stack test

The intersecting stack tests uses 5 cubes in an intersecting begin situation to test the improvement described in Subsection 8.1. We start with 5 cubes of size 2.4m with the centers of mass separated by 2m, creating an overlap of 0.4m, while the bottom cube intersects the floor plane with an overlap of 1.2m (see Figure 7a).

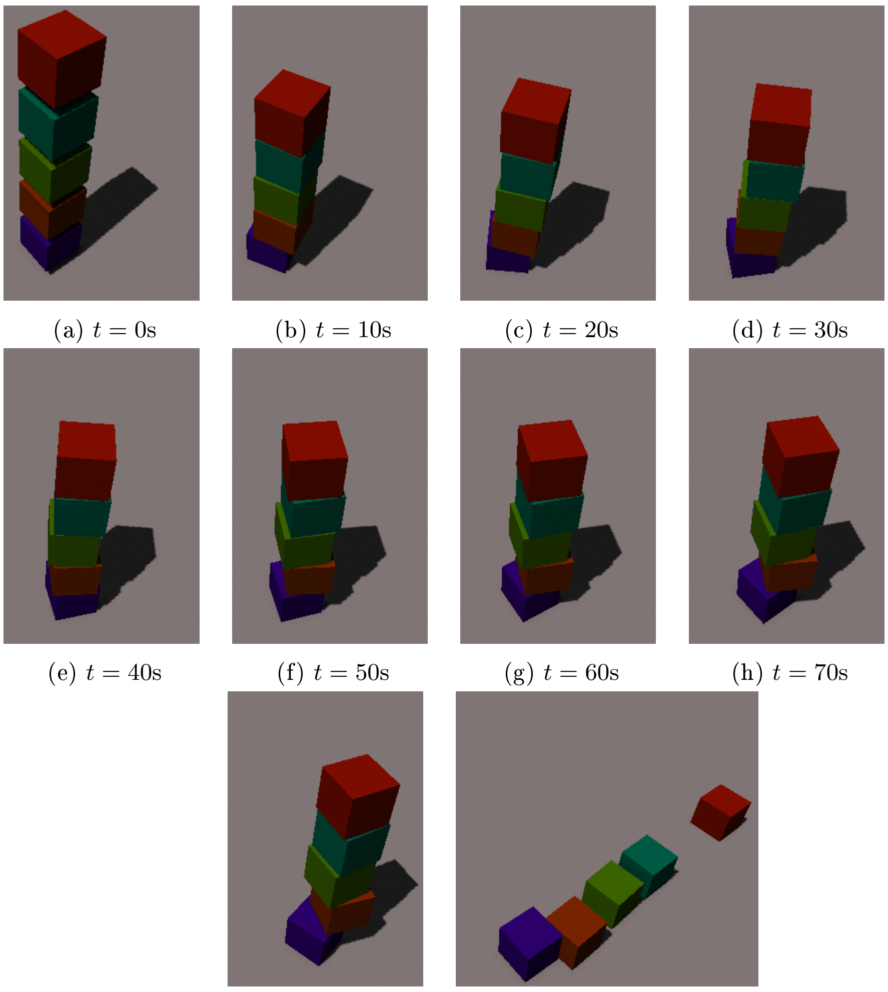
Because the cubes start at an intersecting configuration $\boldsymbol{v}$ will cause the solver to generate a correcting force. The original simulation (as described in [2]) keeps the resulting velocity of this force after the intersection overlap is eliminated. This causes the cubes to keep separating after all contact points are gone. Figure 7b shows the original simulation after $\frac{1}{2}$s. The cubes (especially the higher ones) have gained a very high velocity, and are still separating at this time.

Figure 7c shows the improved simulation after $\frac{1}{2}$s. Here we see that the cubes stop moving after the overlap is eliminated. This produces more natural results as the velocities generated by the correcting force stop affecting the simulation when a correct state is achieved.

10.3 Various other tests

The described system is easy to extend with more constraints and primitives. Here we show a couple of simulations using a combination of primitives and constraints to illustrate this.

- Figure 8 shows 20 balls in a pyramid funnel created out of 4 infinite planes. This configuration becomes reasonably stable for an implementation without contact caching.
- Figure 10 shows 50 balls with the top of each ball attached to a fixed point in the world with a distance constraint. The varied length generates the wave like procession just like its real world counterpart, a pendulum wave.



(a) $t = 0$s (b) $t = 10$s (c) $t = 20$s (d) $t = 30$s

(e) $t = 40$s (f) $t = 50$s (g) $t = 60$s (h) $t = 70$s

(i) $t = 80$s (j) $t = 90$s

Figure 6: Stack test

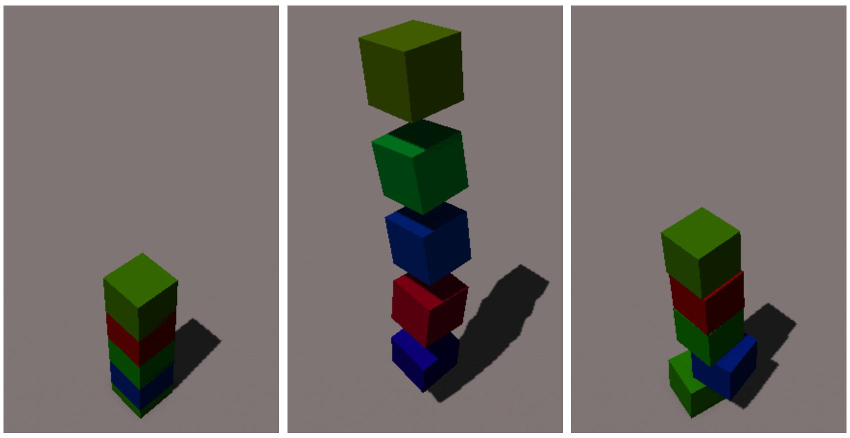

(a) Begin situation (b) Original (c) Improved

Figure 7: Intersecting stack test

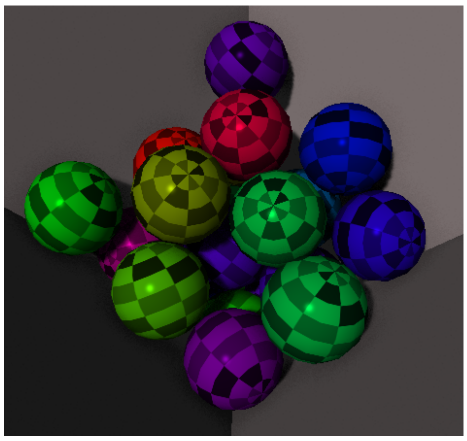


Figure 8: Funnel stack

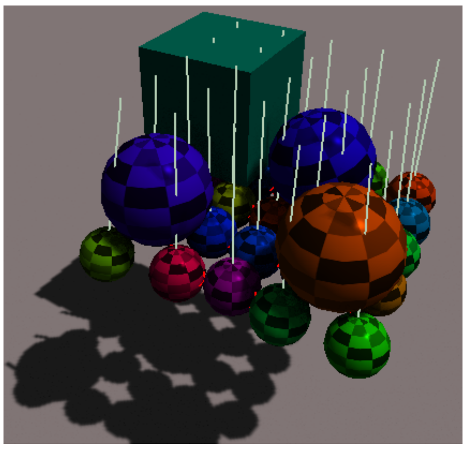


Figure 9: Pendulum grid

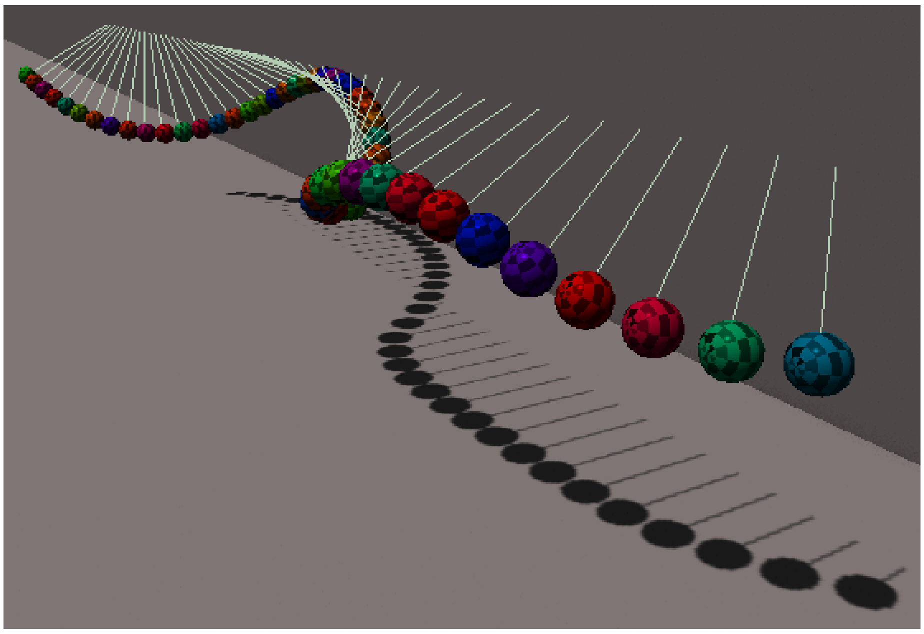


Figure 10: Pendulum wave

- Figure 9 shows a grid of 5 by 5 pendulums. This configuration is stable enough to hold various other shapes. Note that each pendulum is attached only to one point in space by a (non intractable) bar, and not to each other. If enough force is applied to one of the spheres resting on top it will push the pendulums to the side and fall through.

11 Conclusion

We described in depth a system to simulate rigid body physics with constraints suitable for usage in games. We have seen how to utilize larger vectors and matrices to describe an entire physics system, and how to solve this system with an iterative method. We showed how to extend and improve the simulation described in [2] by using the push factor $\boldsymbol{\upsilon}$. And finally, aside from the algebraic derivation of this system, we also presented the dimensional analysis, showing that the formulas provide viable results.

12 Appendix

12.1 Derivative of the length of a vector

Here we show how to obtain the derivative of the squared length of a vector $\mathbf{v}$:

$$\begin{aligned} \mathbf{v}(t) &= [x(t), y(t)] \\ |\mathbf{v}(t)|^2 &= x(t)^2 + y(t)^2 \end{aligned}$$

$$\begin{aligned} \frac{d}{dt}|\mathbf{v}(t)|^2 &= 2x(t)\dot{x}(t) + 2y(t)\dot{y}(t) \\ &= 2\mathbf{v}(t) \cdot \dot{\mathbf{v}}(t) \end{aligned}$$

12.2 Multiplying with J

For quick reference we show a simplified example on multiplying $\mathbf{J}$ illustrating the meaning. This example only considers a single movable rigid body and no rotation. We have a sphere resting in a static corner (Figure 11) resulting in two contact constraints. This results in a $\mathbf{J}$ matrix like the following:

$$\mathbf{J} = \begin{bmatrix} \mathbf{a}_x & \mathbf{a}_y & \mathbf{a}_z \\ \mathbf{b}_x & \mathbf{b}_y & \mathbf{b}_z \end{bmatrix}$$

Multiply $\mathbf{J}$ with $\mathbf{v}$ gives us a vector where each component indicates the speed along an axis of a constraint:

$$\mathbf{Jv} = \begin{bmatrix} \mathbf{a}_x & \mathbf{a}_y & \mathbf{a}_z \\ \mathbf{b}_x & \mathbf{b}_y & \mathbf{b}_z \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} = \begin{bmatrix} \mathbf{a}_x\mathbf{v}_x + \mathbf{a}_y\mathbf{v}_y + \mathbf{a}_z\mathbf{v}_z \\ \mathbf{b}_x\mathbf{v}_x + \mathbf{b}_y\mathbf{v}_y + \mathbf{b}_z\mathbf{v}_z \end{bmatrix} = \begin{bmatrix} \mathbf{av} \\ \mathbf{bv} \end{bmatrix}$$

Or more general:

$$(\mathbf{Jv})_i = \mathbf{J}_i \cdot \mathbf{v} \tag{12.2.1}$$

Multiplying $\mathbf{J}^\mathsf{T}$ with $\boldsymbol{\lambda}$ gives us a combination of $\mathbf{a}$ and $\mathbf{b}$ scaled by the corresponding elements in $\boldsymbol{\lambda}$:

$$\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} = \begin{bmatrix} \mathbf{a}_x & \mathbf{b}_x \\ \mathbf{a}_y & \mathbf{b}_y \\ \mathbf{a}_z & \mathbf{b}_z \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\lambda}_1\mathbf{a}_x + \boldsymbol{\lambda}_2\mathbf{b}_x \\ \boldsymbol{\lambda}_1\mathbf{a}_y + \boldsymbol{\lambda}_2\mathbf{b}_y \\ \boldsymbol{\lambda}_1\mathbf{a}_z + \boldsymbol{\lambda}_2\mathbf{b}_z \end{bmatrix} = \mathbf{a}\boldsymbol{\lambda}_1 + \mathbf{b}\boldsymbol{\lambda}_2$$

Or more general:

$$\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} = \sum_i (\mathbf{J}_i\boldsymbol{\lambda}_i) \tag{12.2.2}$$

12.3 Alternate prove for Equation 6.2.2

We have proven with Equation 6.2.3 that any $\boldsymbol{\lambda}$ satisfies Equation 6.2.2 by using the properties of matrix transpose. In this subsection we show that we can prove the same by using sum notation.

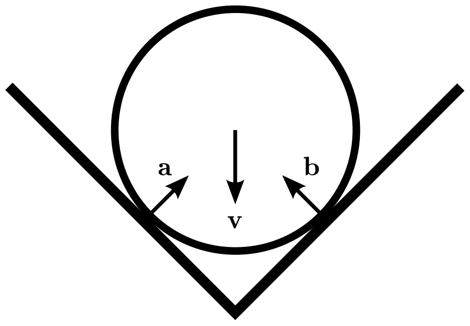


Figure 11: Sphere resting in a V shaped corner. Two contact points with normal's **a** and **b**.

Expanding Equation 6.2.2 with Equation 12.2.2 we get:

$$\left(\sum_i \mathbf{J}_i \boldsymbol{\lambda}_i\right)\mathbf{v} = \sum_j \left(\left(\sum_i \mathbf{J}_i \boldsymbol{\lambda}_i\right)_j \mathbf{v}_j\right) = 0$$

And since

$$\left(\sum_i \mathbf{J}_i \boldsymbol{\lambda}_i\right)_j = \sum_i \mathbf{J}_{ij} \boldsymbol{\lambda}_i$$

we can substitute:

$$\sum_j \left(\left(\sum_i \mathbf{J}_i \boldsymbol{\lambda}_i\right)_j \mathbf{v}_j\right) = \sum_j \left(\left(\sum_i \mathbf{J}_{ij} \boldsymbol{\lambda}_i\right) \mathbf{v}_j\right) = \sum_j \sum_i \left(\ \mathbf{J}_{ij} \boldsymbol{\lambda}_i \mathbf{v}_j\right)$$
$$= \sum_i \sum_j \left(\ \mathbf{J}_{ij} \boldsymbol{\lambda}_i \mathbf{v}_j\right) = \sum_i \left(\sum_j \left(\ \mathbf{J}_{ij} \mathbf{v}_j\right) \boldsymbol{\lambda}_i\right) = \sum_i \left(\sum_j \left(\ (\mathbf{J}^\mathsf{T})_j \mathbf{v}_j\right)_i \boldsymbol{\lambda}_i\right)$$
$$= \sum_i \left((\mathbf{J}\mathbf{v})_i \, \boldsymbol{\lambda}_i\right) = (\mathbf{J}\mathbf{v})\, \boldsymbol{\lambda} = (0)\boldsymbol{\lambda} = 0$$

12.3.1 Deriving the solver

Here we show how to derive the solver in a similar way as described in [2].

We start with Equation 6.2.7 and substitute $\dot{\mathbf{v}}$ with Equation 6.2.4 and solve for $\mathbf{v}_2$:

$$\begin{aligned}
\mathbf{M}\tfrac{\mathbf{v}_2-\mathbf{v}_1}{\Delta t} &= \mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{ext} \\
\mathbf{M}\left(\mathbf{v}_2-\mathbf{v}_1\right) &= \Delta t\left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{ext}\right) \\
\mathbf{M}\left(\mathbf{v}_2\right)-\mathbf{M}\left(\mathbf{v}_1\right) &= \Delta t\left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{ext}\right) \\
\mathbf{M}\left(\mathbf{v}_2\right) &= \Delta t\left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{ext}\right) + \mathbf{M}\left(\mathbf{v}_1\right) \\
\mathbf{v}_2 &= \Delta t\mathbf{M}^{-1}\left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{ext}\right) + \mathbf{v}_1
\end{aligned} \tag{12.3.1}$$

We know partially what the velocities at the next time step should be:

$$\begin{aligned}
\mathbf{J}\mathbf{v}_2 = &\ \boldsymbol{\upsilon} \\
\mathbf{v}_2 = &\ \mathbf{J}^{-1}\boldsymbol{\upsilon}
\end{aligned} \tag{12.3.2}$$

Now we can combine Equation 12.3.1 and Equation 12.3.2 and solve for $\boldsymbol{\lambda}$:

$$\begin{aligned}
\mathbf{J}^{-1}\boldsymbol{\upsilon} &= \Delta t\mathbf{M}^{-1}\left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{\text{ext}}\right) + \mathbf{v}_1 \\
\tfrac{1}{\Delta t}\left(\mathbf{J}^{-1}\boldsymbol{\upsilon} - \mathbf{v}_1\right) &= \mathbf{M}^{-1}\left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{\text{ext}}\right) \\
\tfrac{1}{\Delta t}\mathbf{M}\left(\mathbf{J}^{-1}\boldsymbol{\upsilon} - \mathbf{v}_1\right) &= \left(\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} + \mathbf{f}_{\text{ext}}\right) \\
\tfrac{1}{\Delta t}\mathbf{M}\left(\mathbf{J}^{-1}\boldsymbol{\upsilon} - \mathbf{v}_1\right) - \mathbf{f}_{\text{ext}} &= \mathbf{J}^\mathsf{T}\boldsymbol{\lambda} \\
\tfrac{1}{\Delta t}\left(\mathbf{J}^{-1}\boldsymbol{\upsilon} - \mathbf{v}_1\right) - \mathbf{M}^{-1}\mathbf{f}_{\text{ext}} &= \mathbf{M}^{-1}\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} \\
\tfrac{1}{\Delta t}\mathbf{J}^{-1}\boldsymbol{\upsilon} - \tfrac{1}{\Delta t}\mathbf{v}_1 - \mathbf{M}^{-1}\mathbf{f}_{\text{ext}} &= \mathbf{M}^{-1}\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} \\
\mathbf{J}\left(\tfrac{1}{\Delta t}\mathbf{J}^{-1}\boldsymbol{\upsilon} - \tfrac{1}{\Delta t}\mathbf{v}_1 - \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right) &= \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} \\
\tfrac{1}{\Delta t}\boldsymbol{\upsilon} - \mathbf{J}\left(\tfrac{1}{\Delta t}\mathbf{v}_1 + \mathbf{M}^{-1}\mathbf{f}_{\text{ext}}\right) &= \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\mathsf{T}\boldsymbol{\lambda} \\
\mathbf{b} &= \mathbf{A}\boldsymbol{\lambda}
\end{aligned} \tag{12.3.3}$$

And we arrive at equation 6.2.9.

12.4 Quaternion axis angle formula

Here we show the formula to construct a quaternion that represents a rotation of a radians about (unit length) axis $\mathbf{b}$.

$$\mathbf{q} = \cos\left(\frac{1}{2}a\right) + \left(\sin\left(\frac{1}{2}a\right)\mathbf{b}\right)[i, j, k]$$

Or without dot product notation:

$$\mathbf{q} = \left[\cos\left(\frac{1}{2}a\right), \sin\left(\frac{1}{2}a\right)\mathbf{b}\right] \tag{12.4.1}$$

Or in alternate form:

$$\begin{aligned}\mathbf{q} = \; & \cos\left(\tfrac{1}{2}a\right) \\ & +i\sin\left(\tfrac{1}{2}a\right)b_x \\ & +j\sin\left(\tfrac{1}{2}a\right)b_y \\ & +k\sin\left(\tfrac{1}{2}a\right)b_z\end{aligned}$$

12.5 Derivative of a quaternion

Here we show how to obtain the derivative of quaternion $\mathbf{q}(t)$ when we know the angular velocity $\boldsymbol{\omega}(t)$ (and assume it is constant for our entire time step).

We use formula 12.4.1 to calculate $\mathbf{q}(t_0+\Delta t)$ by filling in $a = |\boldsymbol{\omega}(t_0)|\Delta t$ and $\mathbf{b} = \frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\Delta t$ [1]:

$$\mathbf{q}(t_0+\Delta t) = \left[\cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right), \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right]\mathbf{q}(t_0)$$

Substituting $t = t_0 + \Delta t$:

$$\mathbf{q}(t) = \left[\cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right), \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right]\mathbf{q}(t_0)$$

We do not differentiate $\boldsymbol{\omega}(t_0)$ as we assume it is constant. We also do not differentiate $\mathbf{q}(t_0)$ as it is constant. Differentiating the first part at $t = t_0$ (or $\Delta t = 0$):

$$\frac{d}{dt}\left[\cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|0\right), \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|0\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right]$$

$$\begin{gathered}\tfrac{d}{dt}\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right) = \tfrac{d}{dt}\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|t - \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|t_0\right) = \\ -\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right) = 0\end{gathered}$$

$$\begin{aligned} & \tfrac{d}{dt}\left(\sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right) \\ = \; & \tfrac{d}{dt}\left(\sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|t - \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|t_0\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right) \\ = \; & \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|t - \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|t_0\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|} \\ = \; & \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|} \\ = \; & \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\cos(0)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|} \\ = \; & \tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|} \\ = \; & \tfrac{1}{2}\boldsymbol{\omega}(t_0)\end{aligned}$$

giving:

$$\begin{aligned} \tfrac{d}{dt}\mathbf{q}(t) = & \left[\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right), \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|(t-t_0)\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right]\mathbf{q}(t_0) \\ = & \left[0, \tfrac{1}{2}\boldsymbol{\omega}(t_0)\right]\mathbf{q}(t_0) \end{aligned} \tag{12.5.1}$$

We can also prove this with a limit:

$$\dot{\mathbf{q}}(t_0) = \lim_{\Delta t \to 0} \frac{\mathbf{q}(t_0+\Delta t) - \mathbf{q}(t_0)}{\Delta t}$$

We can calculate $\mathbf{q}(t_0+\Delta t)$ with Equation 12.4.1:

$$\begin{aligned} \mathbf{q}(t_0+\Delta t) = & \left[\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right), \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right]\mathbf{q}(t_0) \\ = & \left[\cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right), \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}\right][w,x,y,z] \\ = & \left[a,\mathbf{b}\right][w,x,y,z] \\ = & \begin{bmatrix} aw - \mathbf{b}_x x - \mathbf{b}_y y - \mathbf{b}_z z \\ ax + \mathbf{b}_x w + \mathbf{b}_y z - \mathbf{b}_z y \\ ay - \mathbf{b}_x z + \mathbf{b}_y w + \mathbf{b}_z x \\ az + \mathbf{b}_x y - \mathbf{b}_y x + \mathbf{b}_z w \end{bmatrix} \\ = & \begin{bmatrix} aw - \mathbf{b}\,[x,y,z] \\ ax + \mathbf{b}\,[w,z,-y] \\ ay + \mathbf{b}\,[-z,w,x] \\ az + \mathbf{b}\,[y,-x,w] \end{bmatrix} \end{aligned}$$

$$\mathbf{q}(t_0+\Delta t) - \mathbf{q}(t_0) = \begin{bmatrix} aw - \mathbf{b}\,[x,y,z] \\ ax + \mathbf{b}\,[w,z,-y] \\ ay + \mathbf{b}\,[-z,w,x] \\ az + \mathbf{b}\,[y,-x,w] \end{bmatrix} - \mathbf{q}(t_0)$$

$$\mathbf{q}(t_0+\Delta t) - \mathbf{q}(t_0) = \begin{bmatrix} aw - \mathbf{b}\,[x,y,z] - w \\ ax + \mathbf{b}\,[w,z,-y] - x \\ ay + \mathbf{b}\,[-z,w,x] - y \\ az + \mathbf{b}\,[y,-x,w] - z \end{bmatrix}$$

$$\mathbf{q}(t_0+\Delta t) - \mathbf{q}(t_0) = \begin{bmatrix} \cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)w - \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[x,y,z] - w \\ \cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)x + \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[w,z,-y] - x \\ \cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)y + \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[-z,w,x] - y \\ \cos\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)z + \sin\left(\tfrac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\tfrac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[y,-x,w] - z \end{bmatrix}$$

Filling in $\Delta t = 0$:

$$\mathbf{q}(t_0+\Delta t)-\mathbf{q}(t_0) = \begin{bmatrix} \cos(0)w - \sin(0)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[x,y,z] - w \\ \cos(0)x + \sin(0)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[w,z,-y] - x \\ \cos(0)y + \sin(0)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[-z,w,x] - y \\ \cos(0)z + \sin(0)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[y,-x,w] - z \end{bmatrix}$$

$$\mathbf{q}(t_0+\Delta t)-\mathbf{q}(t_0) = \begin{bmatrix} 1w - 0\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[x,y,z] - w \\ 1x + 0\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[w,z,-y] - x \\ 1y + 0\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[-z,w,x] - y \\ 1z + 0\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[y,-x,w] - z \end{bmatrix}$$

$$\mathbf{q}(t_0+\Delta t)-\mathbf{q}(t_0) = \begin{bmatrix} w-w \\ x-x \\ y-y \\ z-z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This gives us an indeterminate form, so we can apply L'Hôpital's rule:

$$\dot{\mathbf{q}}(t_0) = \lim_{\Delta t\to 0}\frac{\mathbf{q}(t_0+\Delta t)-\mathbf{q}(t_0)}{\Delta t} = \lim_{\Delta t\to 0}\frac{\frac{d}{d\Delta t}\left(\mathbf{q}(t_0+\Delta t)-\mathbf{q}(t_0)\right)}{\frac{d}{d\Delta t}\Delta t}$$

$$\frac{d}{d\Delta t}\left(\mathbf{q}(t_0+\Delta t)-\mathbf{q}(t_0)\right)$$

$$= \frac{d}{d\Delta t}\begin{bmatrix} \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)w - \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[x,y,z] - w \\ \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)x + \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[w,z,-y] - x \\ \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)y + \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[-z,w,x] - y \\ \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)z + \sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[y,-x,w] - z \end{bmatrix}$$

$$= \begin{bmatrix} -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)w\frac{1}{2}|\boldsymbol{\omega}(t_0)| - \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}|\boldsymbol{\omega}(t_0)|\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[x,y,z] \\ -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)x\frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}|\boldsymbol{\omega}(t_0)|\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[w,z,-y] \\ -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)y\frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}|\boldsymbol{\omega}(t_0)|\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[-z,w,x] \\ -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)z\frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}|\boldsymbol{\omega}(t_0)|\frac{\boldsymbol{\omega}(t_0)}{|\boldsymbol{\omega}(t_0)|}[y,-x,w] \end{bmatrix}$$

$$= \begin{bmatrix} -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)w\frac{1}{2}|\boldsymbol{\omega}(t_0)| - \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}\boldsymbol{\omega}(t_0)[x,y,z] \\ -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)x\frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}\boldsymbol{\omega}(t_0)[w,z,-y] \\ -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)y\frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}\boldsymbol{\omega}(t_0)[-z,w,x] \\ -\sin\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)z\frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos\left(\frac{1}{2}|\boldsymbol{\omega}(t_0)|\Delta t\right)\frac{1}{2}\boldsymbol{\omega}(t_0)[y,-x,w] \end{bmatrix}$$

Filling in $\Delta t = 0$:

$$\begin{bmatrix} -\sin(0) w \frac{1}{2}|\boldsymbol{\omega}(t_0)| - \cos(0)\frac{1}{2}\boldsymbol{\omega}(t_0)\,[x,y,z] \\ -\sin(0) x \frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos(0)\frac{1}{2}\boldsymbol{\omega}(t_0)\,[w,z,-y] \\ -\sin(0) y \frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos(0)\frac{1}{2}\boldsymbol{\omega}(t_0)\,[-z,w,x] \\ -\sin(0) z \frac{1}{2}|\boldsymbol{\omega}(t_0)| + \cos(0)\frac{1}{2}\boldsymbol{\omega}(t_0)\,[y,-x,w] \end{bmatrix}$$

$$= \begin{bmatrix} 0 - 1 \cdot \frac{1}{2}\boldsymbol{\omega}(t_0)\,[x,y,z] \\ 0 + 1 \cdot \frac{1}{2}\boldsymbol{\omega}(t_0)\,[w,z,-y] \\ 0 + 1 \cdot \frac{1}{2}\boldsymbol{\omega}(t_0)\,[-z,w,x] \\ 0 + 1 \cdot \frac{1}{2}\boldsymbol{\omega}(t_0)\,[y,-x,w] \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{1}{2}\boldsymbol{\omega}(t_0)\,[x,y,z] \\ \frac{1}{2}\boldsymbol{\omega}(t_0)\,[w,z,-y] \\ \frac{1}{2}\boldsymbol{\omega}(t_0)\,[-z,w,x] \\ \frac{1}{2}\boldsymbol{\omega}(t_0)\,[y,-x,w] \end{bmatrix}$$

$$= \begin{bmatrix} 0w - \frac{1}{2}\boldsymbol{\omega}(t_0)\,[x,y,z] \\ 0x + \frac{1}{2}\boldsymbol{\omega}(t_0)\,[w,z,-y] \\ 0y + \frac{1}{2}\boldsymbol{\omega}(t_0)\,[-z,w,x] \\ 0z + \frac{1}{2}\boldsymbol{\omega}(t_0)\,[y,-x,w] \end{bmatrix}$$

$$= \left[0, \tfrac{1}{2}\boldsymbol{\omega}(t_0)\right][w,x,y,z]$$

$$= \left[0, \tfrac{1}{2}\boldsymbol{\omega}(t_0)\right]\mathbf{q}(t_0)$$

So $\dot{\mathbf{q}}(t_0) = \left[0, \frac{1}{2}\boldsymbol{\omega}(t_0)\right]\mathbf{q}(t_0)$.

References

[1] David Baraff. An introduction to physically based modeling: Rigid body simulation II-nonpenetration constraints. *SIGGRAPH course notes*, 1997.

[2] E. Catto. Iterative Dynamics with Temporal Coherence. *Game Developer Conference*, January 2005.

[3] Hugh Hunt. Instability of a rigid body spinning freely in space. `http://www2.eng.cam.ac.uk/~hemh/rigid_body.htm`. Accessed: 2014-4-1.

[4] Ian Millington. *Game physics engine development.* CRC Press, 2007.

[5] Eric W. Weisstein. Jacobi method. From MathWorld—A Wolfram Web Resource. Last visited on 14-4-14.

[6] Wikipedia. Poinsot's ellipsoid. `http://en.wikipedia.org/wiki/Poinsot's_ellipsoid`. Accessed: 2014-4-1.

[7] Wikipedia. Work. `http://en.wikipedia.org/wiki/Work_(physics)`. Accessed: 2014-5-1.

[8] Xinyu Zhang, Minkyoung Lee, and Young J Kim. Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer*, 22(9-11):749–760, 2006.

Symbol	Type	Unit	Section
	Description		
m	Scalar	kg	3.1
	Mass		
$\mathbf{M}$	6n x 6n matrix	kg and $\mathrm{kg\,m^2}$	6.1.3
	System mass containing m and $\mathbf{I}$ for all n bodies.		
$\mathbf{I}$	3x3 matrix	$\mathrm{kg\,m^2}$	3.1
	Inertia tensor		
$\mathbf{x}$	$\mathbb{R}^3$ vector	m	3.1
	Position of the center of mass		
$\mathbf{o}$	Quaternion	rad	3.1
	Orientation		
$\mathbf{p}$	$\mathbb{R}^3$ vector	$\mathrm{kg\,ms^{-1}}$ or $\mathrm{N\,s}$	3.1
	Linear momentum		
$\mathbf{p}$	$\mathbb{R}^3$ vector	m	5
	Used in section 5 to indicate a point		
$\boldsymbol{\ell}$	$\mathbb{R}^3$ vector	$\mathrm{kg\,m^2\,s^{-1}}$ or $\mathrm{N\,m\,s}$	3.1
	Angular momentum		
$\mathbf{f}$	$\mathbb{R}^3$ vector	$\mathrm{kg\,m\,s^{-2}}$ or N	3.1
	Force		
$\mathbf{f}_c$	$\mathbb{R}^{12n}$ vector	$\mathrm{kg\,m\,s^{-2}}$ and $\mathrm{kg\,m^2\,s^{-2}}$	6.2.1
	Constraint reaction forces for all n bodies		
$\mathbf{f}_{\text{ext}}$	$\mathbb{R}^{12n}$ vector	$\mathrm{kg\,m\,s^{-2}}$ and $\mathrm{kg\,m^2\,s^{-2}}$	6.1.5
	External forces for all n bodies		
$\boldsymbol{\tau}$	$\mathbb{R}^3$ vector	$\mathrm{kg\,m^2\,s^{-2}}$ or $\mathrm{N\,m}$	3.1
	Torque		
$\mathbf{v}$	$\mathbb{R}^3$ vector	$\mathrm{m\,s^{-1}}$	
	Linear velocity		
$\mathbf{v}$	$\mathbb{R}^{12n}$ vector	$\mathrm{m\,s^{-1}}$ and $\mathrm{rad\,s^{-1}}$	6.1.1
	System velocity, containing $\mathbf{v}$ and $\boldsymbol{\omega}$ for all n bodies		
$\boldsymbol{\omega}$	$\mathbb{R}^3$ vector	$\mathrm{rad\,s^{-1}}$	
	Angular velocity		
t	Scalar	s	3.2
	(Current) Time		
Δt	Scalar	s	3.2
	Time step		

Symbol	Type	Unit	Section
	Description		
$\mathrm{c}(\ldots)$	Scalar function	N/A	5.1
	Position constraint		
$\dot{\mathrm{c}}(\ldots)$	Scalar function	$\mathrm{m\,s^{-1}}$	5.1
	Velocity constraint		
υ	Scalar	$\mathrm{m\,s^{-1}}$	5.1
	Velocity constraint bias / push factor		
$\boldsymbol{\upsilon}$	$\mathbb{R}^c$ vector	$\mathrm{m\,s^{-1}}$	6.1.4
	Vector containing υ for every constraint c		
β	Scalar	None	5.2
	Push factor constant		
c_{min} c_{max}	Scalar	$\mathrm{kg\,m\,s^{-2}}$ or N	5.1
	Constraint c's max/minimum forces		
$\mathbf{c}_{\mathrm{min}}$ $\mathbf{c}_{\mathrm{max}}$	$\mathbb{R}^c$ vector	$\mathrm{kg\,m\,s^{-2}}$ or N	6.1.2
	c_{min} and c_{max} for all c constraints		
$\mathbf{j}_c$	$\mathbb{R}^{12}$ vector	None and m	5.1
	Constraint c's axis		
$\mathbf{J}$	$12n \times c$ Matrix	None and m	6.1.2
	All constraint axis in the system.		
$\mathbf{v}_{ab}$	$\mathbb{R}^{12}$ vector	$\mathrm{m\,s^{-2}}$ and $\mathrm{rad\,s^{-2}}$	5.1
	Velocities of bodies a and b		
$\mathbf{r}_i$	$\mathbb{R}^3$ vector	m	5.2
	Offset between point i and center of mass of body i. $\mathbf{r}_i = \mathbf{p}_i - \mathbf{x}_i$		
$\boldsymbol{\lambda}$	$\mathbb{R}^c$ vector	$\mathrm{m\,kg\,s^{-2}}$	6.2.1
	Vector containing a multiplier for all c constraints		