Simulating the Unstable Rotation of a Book

Introduction to Computational Science 2011-2012

Ivar Postma
Eamon Nerbonne

Contents

Abstract

Physical experiments show that a book can stably rotate along its shortest and its longest axis. Rotation along the middle axis proves unstable. The rotation of such an object can be simulated in different ways. Here, we show two different approaches to rigid body simulation: a particle based simulation and a simulation based on the conservation of angular momentum.

1 Introduction

Many students consider classical mechanics to be somewhat difficult because it's not always very intuitive. For some students this can lead to frustration when studying for an exam and under stress they might be inclined to throw their classical mechanics book out the window.

There are many different ways to throw your classical mechanics book out of your window. One of the more popular ways is to use it as a frisbee. When a book is thrown like a frisbee it starts rotating around the center of mass of the book. Students that live in one of the higher floors of an apartment building may notice that this rotation is *stable*: until the book hits the ground, its apparent axis of rotation does not change.

Let us now consider a hard cover book (Figure 1). We can define three perpendicular axes. The first axis, x in the figure below, runs through the midpoint of the spine of the book. The second axis, y, runs through the midpoint of the top. The third axis, z, runs through the midpoint of the book's cover.

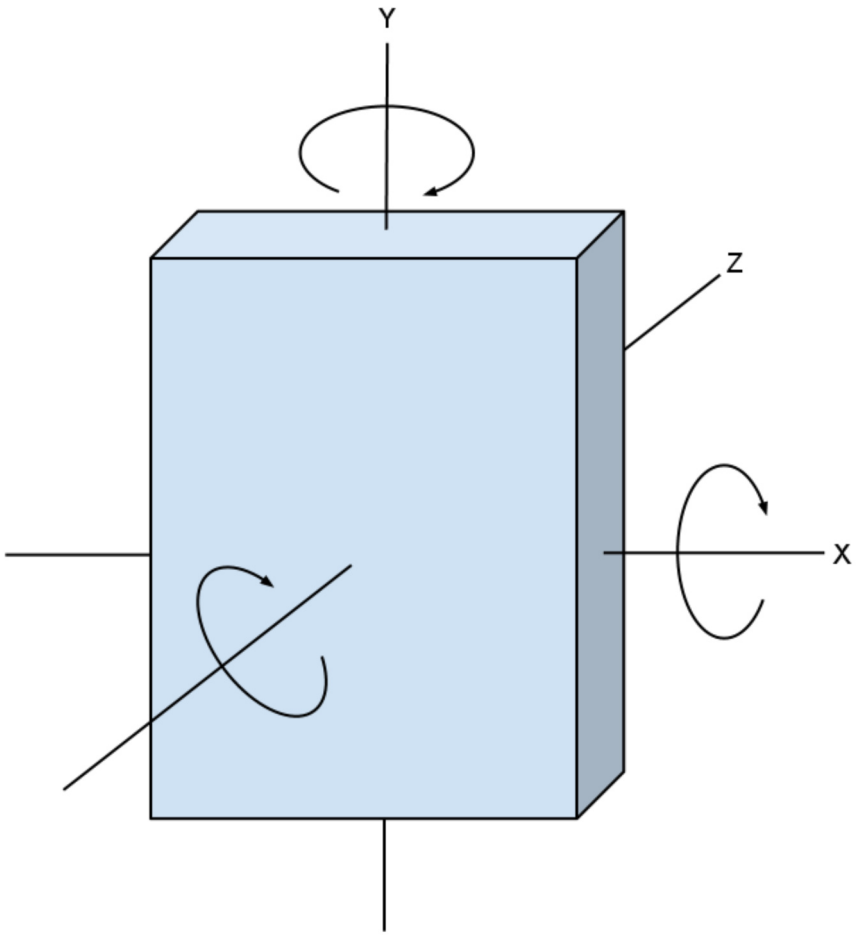

Figure 1: A cuboid with its three main axes.

When thrown as a frisbee, the book rotates along the z axis. This rotation is stable, as is the rotation around the y axis. However, if we throw a book so that it is rotating along the x axis, the rotation is unstable. In space, we can see this unstable rotation more clearly because without gravity linear motion can be eliminated. The video below (Figure 2) demonstrates an experiment done on the international space station.

(Video omitted in print version)

Figure 2: Video of an experiment in space demonstrating both stable and unstable rotation [2].

In this project we will simulate the rotation of a book so that it behaves naturally (as in Figure 2). Since the goal of this project is to simulate the rotation, we will consider a system without external forces. We will look at two different approaches [3]. Firstly, we will simulate the book as a number of particles connected by springs. Rotation, whether stable or not, will be an emergent phenomenon based on the interaction between the particles. The second approach will model the book as a single rigid body in which rotation is explicitly simulated using angular momentum and the book's moment of inertia.

2 Euler method

To implement the simulation, we will define a number of ordinary differential equations (ODEs). These will be solved numerically. Since Euler's method is the simplest method and computationally cheap, it was used to solve the equations arising from the models below.

In Euler integration the value of a function is computed at discrete timesteps t_i where $t_{i+1} = t_i + \Delta t$. The value of a function at the next timestep is computed using a linear extrapolation from its current value and its derivative:

$$g(t_{i+1}) = g(t_i) + \dot{g}(t_i)\Delta t$$

We will use the notation $\dot{g} \equiv \frac{dg}{dt}$ and $\ddot{g} \equiv \frac{d^2g}{dt^2}$ for the first and second time derivatives of a function, as usual in the physics community.

At each time step, Euler integration introduces an error which is (at worst) proportional Δt. Thus, for a fixed time interval (which requires more timesteps as Δt decreases), the overall error is at worst directly proportional to Δt. However, even with a short time step, the error will grow and the calculated value will drift far away from the actual value. To reduce the effect of this drift, it is in general useful to use a higher order numerical integration technique resulting in less error. These (usually) come at the cost of a longer computation time and cannot eliminate the error. As this simulation is not computationally intensive, a CPU can compute very many timesteps in real time, so that even Euler integration is sufficiently accurate to make demonstration of the numerical errors impractical. To highlight the difference between the models, a larger-than-necessary timestep was thus used.

3 Particle Simulation

In the particle simulation a number of particles is defined and their position x is tracked over time. The rate of change of the position is the velocity v of the particle. The velocity of a particle is not necessarily constant, it may also change over time. The rate of change of the velocity is defined by the acceleration a of a particle. Or:

$$v(t) = \dot{x}(t)$$
$$a(t) = \dot{v}(t) = \ddot{x}(t)$$

To find the acceleration of a particle we can use Newton's second law.

$$F = ma$$

Where necessary, we will use subscripts to differentiate between particles. For example, x_i is the position of the i-th particle, F_i the force acting on particle i, and F_{ij} the force acting on particle i due to the spring connected to the j-th particle.

3.1 Modeling the book

To model an object with particles we must define a number of particles, their mass and how they interact with each other. For simplicity, the book is modeled by eight particles located at the eight corner points of a cuboid. Each particle has the same mass and an initial position that depends on the initial orientation of the book. The initial linear velocity of each particle is computed using the initial angular velocity of the book as a whole. To ensure the book maintains its shape, virtual springs are added between each pair of particles. This means that in total there are 28 springs.

The force on i by the spring to j is (by Hooke's law) directly proportional to the displacement from the equilibrium length but in opposite direction. Accounting for the fact that the orientation of the spring depends on the position of the particles it is attached to, this can be expressed as follows:

$$\begin{aligned} \hat{x}_{ij} &\equiv \frac{x_i - x_j}{\|x_i - x_j\|} \qquad \{ \text{ unit-vector shorthand } \} \\ F_{ij} &= -k\big(\|x_i - x_j\| - \mathbf{L}_{ij}\big)\hat{x}_{ij} \end{aligned}$$

Here, k is the the spring constant (defined to be identical for all springs) and $\mathbf{L}_{ij}$ the equilibrium length of the spring between i and j (as determined using the book's shape at rest). The total force on a particle i is simply the sum of the individual forces acting on it.

Although this simple model works, the springs maintaining the book's shape have a tendency to vibrate. Initially, the book's springs are in equilibrium and thus do not provide the necessary centripetal acceleration to maintain the book's shape. As the rotation causes the particles to move further apart spring tensions rise. When the time the outward motion is fully halted by the high spring tension, the spring tension does not cease but instead starts pulling the points closer together again. This process repeats indefinitely, causing vibration. By increasing the spring constant, the amount of distortion to the shape can be minimized; after all, with stiffer springs less displacement is necessary to provide the same centripetal acceleration.

The error caused by Euler integration in this model causes the system to vibrate more and more strongly as time progresses. In effect, by simulating the rotating motion in a piecewise linear fashion, the counteracting pull of the springs comes a short moment after the extension caused by the rotation. Thus, on average, each time step adds a small amount of potential energy to the springs. Eventually, the system diverges. Visually, the book "blows up". By decreasing the spring constant, the amount of potential energy gained each time step is reduced thus delaying the moment of divergence. The choice of spring constant thus represents a trade-off between maintaining the shape as much as possible or delaying divergence as much as possible.

3.2 Normalization

One way to avoid divergence due to energy gain is by normalization. After all, the amount of energy in the system is easily computed, and should remain constant. There are two sources of energy in the model: the elastic potential energy of the springs, and the kinetic energy of the moving particles. The potential energy depends on the displacement $r_{ij} = \|x_i - x_j\| - L_{ij}$ of each spring: $\sum 0.5kr_{ij}^2$. The kinetic energy is $\sum 0.5mv_i^2$. Since altering the amount of elastic potential energy stored in the spring is complicated, we chose instead to simply scale the velocity of all particles equally by constant c. Then, for total known energy E:

$$E = \sum_{i<j} \frac{1}{2}\, k r_{ij}^2 + \sum_i \frac{1}{2}\, m (c v_i)^2$$

$$c = \sqrt{\frac{2E - k \sum r_{ij}^2}{m \sum v_i^2}}$$

Similarly, after very long runs, the center of mass may start drifting. The drift may be avoided by subtracting the mean position from all positions and the mean velocity from all velocities.

Normalization successfully avoids divergence. However, the model nevertheless starts vibrating: the spings still need to provide sufficient centripetal acceleration, and this involves the springs extending then retracting in oscillating fashion as described before. Furthermore, the error caused by the short delay between extension and counteracting pull causes the system to gain potential energy as before — now, however, with a corresponding decrease in kinetic energy. The net effect is that the book's rotation slows as more and more rotational kinetic energy is converted to the spring's vibration.

3.3 Damping

Spings can be prevented from oscillating by damping them: introducing frictional force proportional to the rate of change of the spring's displacement. A single-spring system can be described as follows:

$$\ddot{r} + b\dot{r} + cr = 0$$

This system corresponds to the springs we've defined when $b = 0, c = k/m$. Such a system is critically damped (i.e. returns to rest as quickly as possible) when $b = 2\sqrt{c}$ instead. Assuming critical damping we then have:

$$\ddot{r} = -2\sqrt{\frac{k}{m}}\dot{r} - \frac{k}{m}\, r$$

$$F_{ij} = -\big(2\sqrt{km}\, \hat{x}_{ij} \cdot (v_i - v_j) + k(\|x_i - x_j\| - \mathbf{L}_{ij})\big)\hat{x}_{ij}$$

Damping avoids the unwanted distortion of the books shape. As the system gains energy by extending the springs "for free", damping also somewhat compensates for this by losing energy when they contract again. However, although the system remains stable for much longer than without normalization, a damped system still eventually diverges. The combination of damping and normalization is required to get a reasonable simulation indefinitely.

Damping and normalization prevent divergence, but they do not prevent the actual errors from accumulating. Due to the damping+normalization, the local error is low in the sense that for any given short period of time, the error introduced is small. However, in the long run the book's orientation will still diverge from the true orientation. For many purposes, this doesn't matter: the book's exact state is less important than its behavior.

If global error is to be reduced, smaller timesteps and/or a more accurate numerical integration technique would help. However, a more serious source of global error is likely due to the model error: modeling a book as a bunch of springs does not accurately describe how the book behaves. More accurate simulation will not avoid inherent inaccuracies in the model. A better model would take into account the fact that the book is a rigid object.

3.4 Implementation

The book is visualized using OpenGL to visualize the book. In OpenGL a cuboid is rendered by specifying its eight sides. The position of the eight corner points of the book

define where to draw each side of the book. To be able the distinguish between different sides of the book a texture is rendered on the front, spine and back of the book.

In our implementation the visualization is separated from the calculation of the rotation. For visualization we will need the world coordinates of the corner points of the book. Since the particles are located at these corners their positions can directly be used for visualization.

For the particle simulation, we need to store the positions and the velocities in 3D space of all the particles. Both are stored in a 3 × 8 matrix. The interacting forces and the equilibrium lengths of each spring are stored in an 8 × 8 matrix. The implementation is written in C++, in which matrices are not standard. The Eigen library can be used for linear algebra [1]. This library provides useful data types, such as matrices and vectors, and many common linear algebra operations.

At each time step the positions are updated using the velocity and then the velocity is updated using the computed forces (Newton's third law is exploited to avoid half of the force computations). After calculating the new values, they can be normalized. The initial positions and velocities are computed using the books predefine shape and a predefined angular velocity around an axis of the user's choice. The speed and accuracy of the simulation can be independantly adjusted (within computational limits) by setting the Δt per animation frame and the number of update steps this rate is subdivided into (more, smaller update steps mean a more accurate simulation at higher CPU load).

4 Rigid Body Simulation

In the video in the introduction, one may notice that the only property of the book that changes is its orientation. The orientation can be described with an orientation matrix R. The change in orientation is determined by the angular velocity ω of the object.

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

$$\omega^* \equiv \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \qquad \{ \text{ cross-product as matrix } \}$$

To update the orientation of the object, we need the angular velocity. Initially the angular velocity is known and together with the moment of inertia of the body I, can be used to calculate the angular momentum L:

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

Because we do not consider external forces in our model the angular momentum is constant. This means the angular velocity can be calculated once the moment of inertia is known.

The moment of inertia is the measure of an object's resistance to change in its orientation R. Since the distribution of an objects mass in space changes as its orientation does; so does it's moment of inertia. Under a given rotation (i.e. orientation) matrix the current moment of inertia can be computed as:

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

where I_{body} is the moment of inertia tensor of the object in the bodie's principal axes. If the moment of inertia is known in any coordinate system, by SVD you can find the principal axes. In general, the inertia tensor of an object is defined as:

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

The inertia tensor for a cuboid is known. For a cuboid of mass M and dimensions $a \times b \times c$ the inertia tensor is defined as:

$$I_{body} = \frac{1}{12} \begin{bmatrix} M(b^2 + c^2) & 0 & 0 \\ 0 & M(a^2 + c^2) & 0 \\ 0 & 0 & M(a^2 + b^2) \end{bmatrix}$$

This (body-space) inertia tensor does not change during the simulation and thus it can be precomputed. In particular, it is diagonal and thus has a trivially computed inverse, which is useful in finding the current angular velocity:

$$\begin{aligned} \omega(t) &= I(t)^{-1} L(t) \\ &= \left(R(t)^T\right)^{-1} I_{body}^{-1} R(t)^{-1} L(t) \\ &= R(t) I_{body}^{-1} R(t)^T L(t) \end{aligned}$$

4.1 Modeling the book

Without linear momentum the center of mass of the book is stationary. This center of mass is the origin in our coordinate system. The corner points of the book are defined by body coordinates. Together with the orientation of the body these body coordinates can be used to determine the world coordinates of the corner points. The body coordinates are also used to compute the moment of inertia of the body.

4.2 Normalization

Using Euler integration to numerically solve the ODEs here will produce an error which causes the orientation matrix to contain values that are slightly larger than they should be. Visually, the book will "grow" and distort. To correct for the errors the orientation should be normalized. This is done by singular value decomposition (SVD).

4.3 Implementation

The only thing we need to store is a 3 × 3 matrix describing the orientation, another 3 × 3 matrix for the moment of inertia and vector describing the angular momentum. In each timestep we calculate the angular velocity and we update the orientation. The orientation can then be normalized with SVD using the Eigen library. From the orientation matrix, the world coordinates of the corner points are determined which are used to visualize the book.

4.4 Quaternions

Quaternions, and extension to complex numbers with a further two non-real components, provide a convenient mathematical notation for representing orientations and rotations of objects in three dimensions. Expressing rotations as quaternions offers some advantages over a matrix notation. Concatenating rotations has a low computational cost and it's easier to extract the angle and axis of rotation. In our case, they're useful because normalization of quaternions is computationally much cheaper than normalization of the orientation matrix via SVD.

A quaternion q can be written as $w + xi + yj + zk$ where $i^2 = j^2 = k^2 = ijk = -1$.

In quaternion notation the orientation is denoted as q. The change in orientation can then be written as:

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

(Strictly speaking, the vector ω is reinterpreted as a quaternion $\omega_x i + \omega_y j + \omega_z k$ here). To calculate the moment of inertia we can't use the transpose of the orientation matrix. Instead the quaternion conjugate (just like complex numbers, this means to negate the non-real components) of the orientation quaternion is used:

$$I(t) = q(t) I_{body} \overline{q}(t)$$

Quaternions are supported in the Eigen library thus the implementation of the rigid body method using quaternions is very similar to the one using an orientation matrix. This orientation matrix is replaced by a quaternion and in the calculating of the new orientation we add a factor 0.5. In particular, the library defines a multiplication of a quaternion by a matrix involves conceptually reinterpreting the quaternion as a matrix with coefficients computed to result in an equivalent rotation.

Normalization of quaternions is also supported by Eigen. This normalization is similar to the normalization of a vector. The four values w, x, y and z are simply divided by the magnitude of the quaternion:

$$||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

5 Results

The implementation used to generate the videos below is available in source code form (cross-platform with glut or freeglut dependency) and as a compiled win32 executable: RotateBook-win32.

The implementation shows the visualization of a rotating book. There are three different methods to calculate the position of the corner points of the book, a particle-based method, a rigid method with an orientation matrix and a rigid method with quaternions. The user can reset the visualization to a rotation around each of the three axis. Normalization can be turned on and off.

5.1 Particle Simulation

The particle-based simulation produces a natural looking simulation of the rotation of a book. The rotation along the longest axis, the Z axis in this case is stable (Figure 3), as is rotation about the Y axis which has the lowest moment of inertia (Figure 4). Rotation along the book's X is an instable equilibrium: it initially appears to be stable, but this rotation is not robust. Numerical inaccuries in the simulation suffice to mis-align the rotation from the axis sufficiently for unstable rotation to be visible (Figure 5). The simulation runs sufficiently fast on a standard computer.

(Show Video)

Figure 3: Rotation around the Z axis (the axis with the largest moment of inertia) is stable even when the orientation is jiggled a little.

(Show Video)

Figure 4: Rotation around the Y axis (the axis with the smallest moment of inertia) is stable even when the orientation is jiggled a little.

(Show Video)

Figure 5: Rotation around the X axis (the axis with the median moment of inertia) is unstable; numerical inaccuracies are enough to trigger the unstable rotation.

Damping and normalization are a necessary part of this particle simulation. Without damping, we can observe the rotation being is (quickly) transformed into vibration. We can see this in Figure 6. Without normalization, the total sum of energy increases and the simulation runs faster and faster until it "explodes" (Figure 7). With neither, the simulation vibrates diverges very quickly (Figure 8). Note that all of these simulations were run with relatively large timesteps to intentionally exacerbate the problems; simply running approximately 1000 smaller iterations per frame delays these problems substantially — in particular the damping without normalization simulation can run for a long time before diverging.

(Show Video)
Figure 6: A particle simulation with normalization but without damping: rotational kinetic energy is gradually converted to vibration.

(Show Video)
Figure 7: A particle simulation without normalization but with damping: vibration is suppressed, but the book gradually gains energy and speeds up it "explodes".

(Show Video)
Figure 8: A particle simulation without normalization or dampening. Large timesteps cause instability quickly.

5.2 Rigid Body Simulation

A rigid simulation where the orientation of the body is updated in each time step can be implemented in two different ways: using a 3×3 orientation matrix and using quaternions.

The simulation using an orientation matrix also produces a natural looking result and it also runs sufficiently fast (Figure 9). Without normalization the error produced by Euler integration adds up and we can see the shape of the book start to change noticeably after some time (Figure 10). Eventually, the error is so extreme that the simulation crashes.

(Show Video)
Figure 9: An orientation based simulation showing unstable rotation. With a larger timestep results are also good, but slightly less consistent: (external video).

(Show Video)
Figure 10: An orientation based simulation without normalization eventually "blows up".

The simulation of the rotating book using the quaternion-based implementation appears to be visually the same as the orientation matrix method. However, this changes when normalization is turned off (Figure 11). Like the implementation with the orientation matrix the shape of the book will eventually change when normalization is turned off. It often takes longer for this divergence to occur; however despite many fewer degrees of freedom (than the orientation-matrix based simulation) this is not always the case. The errors caused by unnormalized quaternions are sometimes surprising (Figure 12)

(Show Video)
Figure 11: A quaternion based simulation without normalization. It eventually denormalizes and no longer approximates a rotation.

(Show Video)
Figure 12: A quaternion based simulation without normalization. The denormalization error causes the book to pulsate.

6 Discussion

Normalization is necessary in a stable simulation because of small errors in the integration technique. However, it is not necessary to normalize the calculation in every time step. When to normalize depends on the size of the time step and on the integration method.

Choosing a higher order numerical integration technique, such as fourth-order Runga-Kutta, will produce a smaller error in every time step but this is computationally more expensive. Since we need to normalize anyway, which will eliminate the error made, Euler integration is sufficient for this simulation.

The particle-based simulation and the rigid simulation produce similar results. So can we say which method is better. If we wish to quantify this we could look at the computational load of both methods. However, this would only yield in this particular case where we simulate a rotation cuboid. If we look at the more general problem of rotating objects, the particle-based method has an advantage over the rigid method. In the rigid method we use the definition of the moment of inertia of our object. This means that if the moment of inertia is unknown we have to compute it in advance. For complex shapes this proves to be difficult thus it will take longer to implement. For particle simulation we only need to place the particles and define their interacting forces which is easier.

Rigid simulation does have the advantage that the shape of body is kept intact (apart from numerical errors). This behavior is what you would expect from a solid body. Particles move more freely which causes the shape of the object to change.

7 Conclusion

Simulation of the rotation of a book can be successfully implemented with both a particle-based method and rigid method. In both methods it is important to normalize the calculations because numerical integration techniques introduce errors which will eventually cause the simulation to "explode". Which implementation is more useful depends on the application. For simple shapes such as a cuboid the rigid method is easy to implement and performs well. For more difficult shapes the particle-based method proves to be easier to implement because it does not rely on the moment of inertia of the object to be computed.

References

[1] B. Jacob et al. Eigen, February 2012.

[2] D. Pettit. Solid Body Rotation, February 2012.

[3] D. Baraff. An Introduction to Physically Based Modeling, SIGGRAPH '97, 1997.