Rigid Body Physics in the context of the application in game physics engines

Benjamin Rüth
Fakulät für Informatik
Technische Universität München
Email: benjamin.rueth@tum.de

***Abstract*— In this paper we are going to describe the most important parts of a game physics engine. The physical model is described in a detailed way, while the implementation of correct collision treatment is only covered as an outlook.**

I. Introduction

In the seminar *Let's play! Simulated Physics for Interactive Games* our goal was to develop an interactive video game with a focus on simulating physics. Hereby one of the main tasks was of course the simulation of rigid bodies. In the following we are going to present the general layout of a game physics engine. Our focus lies on its core, the simulation of the rigid body physics in section II. But since the physically correct modelling of rigid body motion is of course not the only part of a game physics engine, we will draw attention to other important topics in section III.

II. Rigid body physics

The physics of rigid bodies are a very basic and well established topic in nearly all natural and engineering sciences (see [3], [5]). But while in science the emphasis lies on physically correct results[1], in games we care more about appealing pictures. Of course both objectives do not oppose each other, but in games we can often relax some of the strict conditions from science. Anyhow the process of modelling is the same for both science and games. But when it comes to the numerical treatment of the governing equations already in a very easy framework some useful simplifications can be applied.

A. Linear movement

At *time* t a body of *mass* M has the two degrees of freedom *center of mass position* $\vec{x}(t)$ and *velocity* $\vec{v}(t)$ and is subject to the *acceleration* $\vec{a}(t)$. At any time these physical quantities are governed by a system of differential equations

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v}, \quad \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \vec{a}$$

commonly referred to as the *equations of motion*. The acceleration $\vec{a}(t)$ is usually not accessible. But often we know the *forces* $\vec{F}_i(t)$ acting upon the body in their respective *point of load* $\vec{\xi}_i(t)$. Note that for now we restrict ourselves only to forces acting upon the body in such a way, that the *line of action* is running through the center of mass, this means, that $\left(\vec{\xi}_i(t) - \vec{x}(t)\right) \times \vec{F}_i(t) = 0$ (see fig. 2a). The relation of force, mass and acceleration is described by *Newton's second law*

$$\vec{F}_{\text{tot}}(t) = \sum_i \vec{F}_i(t) = M\vec{a}(t)\,.$$

With this set of equations we can already model the movement of any non-rotating rigid body:
Assuming we know the position and velocity (*state*) of a rigid body at initial time t_0 and we can evaluate forces acting on the rigid body at any time using the function $\vec{F}_{\text{tot}}(t)$, we can compute position and velocity at arbitrary time $t = t_0 + \Delta t$:

$$\vec{v}(t) = \vec{v}(t_0 + \Delta t) = \vec{v}(t_0) + \int_{t_0}^{t_0+\Delta t} \frac{1}{M}\vec{F}_{\text{tot}}(\tau)\,\mathrm{d}\tau \qquad (1)$$

$$\vec{x}(t) = \vec{x}(t_0 + \Delta t) = \vec{x}(t_0) + \int_{t_0}^{t_0+\Delta t} \vec{v}(\tau)\,\mathrm{d}\tau. \qquad (2)$$

Using these equations, we can now fully describe the linear movement of a rigid body (see fig. 1). Note that due to the restriction to *linear movement* the velocity of the rigid body is at all parts of the body identical and there is actually no difference between a rigid body and a *point mass*, where the mass is not distributed in a certain volume, but concentrated in a single point.

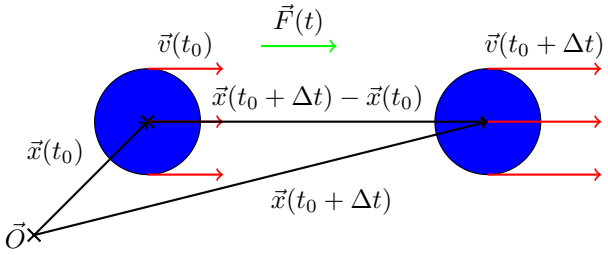


Fig. 1: A rigid body (blue) affected by a force $\vec{F}(t)$ is moved due to the *equations of motion*. $\vec{O}$ denotes the origin of our coordinate system. Black arrows are positions or distances, red arrows velocities. The force $\vec{F}(t)$ acts in the center of mass $\vec{x}(t)$ of the rigid body.

[1] e.g. energy conservation

B. Linear and angular movement

As a next step we drop the restriction to linear movement and allow our body to rotate. As a first consequence we obtain two new degrees of freedom, which are the *rotation angle* $\vec{\varphi}(t)$ and the *angular velocity* $\vec{\omega}(t)$. This time the *angular acceleration* $\vec{\alpha}(t)$ acts on the body. With these new degrees of freedom we have to extend our set of governing equations:

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{v}, \quad \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \vec{a}$$
$$\frac{\mathrm{d}\vec{\varphi}}{\mathrm{d}t} = \vec{\omega}, \quad \frac{\mathrm{d}\vec{\omega}}{\mathrm{d}t} = \vec{\alpha}.$$

But – just like the linear acceleration $\vec{a}(t)$ – the angular acceleration $\vec{\alpha}(t)$ is usually not known. In the end we are – again – going to use Newton's seconds law for computing this quantity, but before we can do this, two important physical qunatities have to be introduced.

1) **Torque:** By also have to extend our model and allowing angular movement, as a consequence we can drop the restriction on the point of load $\vec{\xi}$ from above. This generalization leads to a combination of force $\vec{F}(t)$ and *torque* $\vec{T}(t)$ acting on the body (see fig. 2b. Torque directly evolves from force through

$$\vec{T}(t) = \vec{r}(t) \times \vec{F}(t)\,,$$

where $\vec{r}(t)$ describes the distance from the center of mass $\vec{x}(t)$ to the point of load $\vec{\xi}(t)$. This gives

$$\vec{T}_i(t) = \left(\vec{x}(t) - \vec{\xi}_i(t)\right) \times \vec{F}_i(t)\,.$$

2) **Mass moment of inertia:** The *mass moment of inertia* represents the distribution of mass inside the body and can be computed by the following formula:

$$J_{\alpha\beta} = \int_V \rho\left(\vec{r}\right)\left(r^2\delta_{\alpha\beta} - x_\alpha x_\beta\right),$$

where $\rho\left(\vec{r}\right)$ describes the *density* distribution inside the body and α, β denote different dimensions. This means that in 3D the *mass moment of inertia* is represented by the *tensor*

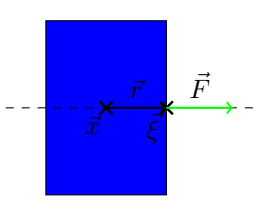
$$J = \begin{pmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{yx} & J_{yy} & J_{yz} \\ J_{zx} & J_{zy} & J_{zz} \end{pmatrix}.$$

In 2D we obtain the scalar quantity

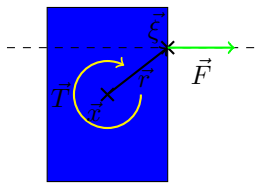
$$J = J_{zz}.$$

Finally we can obtain the angular acceleration from the torque $\vec{T}(t)$ and acting on the body and the mass moment of inertia by applying Newton's second law:

$$\vec{T}_{\text{tot}}(t) = \sum_i \vec{T}_i(t) = J\vec{\alpha}(t)\,.$$



(a) Force $\vec{F}$ with the line of action running through the center of mass. No torque acts upon the body.



(b) Force $\vec{F}$ with the line of action **not** running through the center of mass. Additional to the force also torque acts upon the body.

Like for linear movement, we can now also describe angular movement through the following integral formulas

$$\vec{\omega}(t) = \vec{\omega}(t_0 + \Delta t) = \vec{\omega}(t_0) + \int_{t_0}^{t_0+\Delta t} J^{-1}\vec{T}_{\text{tot}}(\tau)\,\mathrm{d}\tau \tag{3}$$

$$\vec{\varphi}(t) = \vec{\varphi}(t_0 + \Delta t) = \vec{\varphi}(t_0) + \int_{t_0}^{t_0+\Delta t} \vec{\omega}(\tau)\,\mathrm{d}\tau. \tag{4}$$

Note that in 2D $J^{-1}\vec{T}_{\text{tot}}$ represents a simple division $\vec{\alpha} = \frac{1}{J}\vec{T}_{\text{tot}}$, in 3D solving the linear system $J\vec{\alpha} = \vec{T}$ for $\vec{\alpha}$.

C. Numerical treatment

With our model from above we can now – theoretically – fully describe rigid body movement, by analytically evaluating the set of eqs. (1) to (4) at an arbitrary time t. But – practically – this approach is not feasible, because we cannot guarantee, that the corresponding antiderivatives actually exists. Using the set of governing equations and the initial values

$$\left(\vec{x}(t_0)\,, \vec{v}(t_0)\,, \vec{\varphi}(t_0)\,, \vec{\omega}(t_0)\right) = \left(\vec{x}_0, \vec{v}_0, \vec{\varphi}_0, \vec{\omega}_0\right)$$

we can also interpret the problem as the initial value problem

$$\begin{aligned}
\frac{\mathrm{d}\vec{x}(t)}{\mathrm{d}t} &= \vec{v}(t)\,, & \vec{x}(t_0) &= \vec{x}_0 \\
\frac{\mathrm{d}\vec{v}(t)}{\mathrm{d}t} &= M^{-1}\vec{F}_{\text{tot}}(t)\,, & \vec{v}(t_0) &= \vec{v}_0 \\
\frac{\mathrm{d}\vec{\varphi}(t)}{\mathrm{d}t} &= \vec{\omega}(t)\,, & \vec{\varphi}(t_0) &= \vec{\varphi}_0 \\
\frac{\mathrm{d}\vec{\omega}(t)}{\mathrm{d}t} &= J^{-1}\vec{T}_{\text{tot}}(t)\,, & \vec{\omega}(t_0) &= \vec{\omega}_0.
\end{aligned}$$

This initial value problem can be solved by using a numerical scheme for the solution of *ordinary differential equations*. In the following we are going to present representatives of three families of schemes. We are only going to present the schemes for the pair of position $\vec{x}$ and velocity $\vec{v}$, but the same schemes can be applied for angle $\vec{\varphi}$ and angular velocity $\vec{\omega}$ as well:

1) **Explicit Euler:** explicit scheme

$$\vec{x}(t_{k+1}) \approx \vec{x}(t_k) + \vec{v}(t_k)\,\Delta t$$
$$\vec{v}(t_{k+1}) \approx \vec{v}(t_k) + \vec{a}(t_k)\,\Delta t$$

2) **Implicit Euler:** implicit scheme

$$\vec{x}(t_{k+1}) \approx \vec{x}(t_k) + \vec{v}(t_{k+1})\,\Delta t$$
$$\vec{v}(t_{k+1}) \approx \vec{v}(t_k) + \vec{a}(t_{k+1})\,\Delta t$$

3) **Störmer Verlet:** symplectic scheme

$$\vec{x}(t_{k+1}) \approx \vec{x}(t_k) + \vec{v}(t_k)\,\Delta t + \frac{1}{2}\vec{a}(t_k)\,\Delta t^2$$
$$\vec{v}(t_{k+1}) \approx \vec{v}(t_k) + \frac{\Delta t}{2}\left(\vec{a}(t_k) + \vec{a}(t_{k+1})\right)$$

t_k denotes an instant of time with known information, $t_{k+1} = t_k + \Delta t$ the instant of time, for which we want to compute the physical quantities. With given initial conditions for t_0 we can propagate to any instant of time t_k by iteratively applying these schemes. In general bigger timesteps Δt result in a lower number of necessary iterations to reach a certain point in time, but also in lower accuracy of the result. The field of *numerical methods for ordinary differential equations* covers the rich theory and properties of those schemes (see [7]).

By applying one of the methods from above, we can now approximately calculate the position of a rigid body subjected to arbitrary forces at any time t.

III. From the physical model to an engine

The framework from above enables us to simulate the movement of any number of rigid bodies influenced by arbitrary forces in a physical correct way. This can be considered as the core of a game physics engine, but still there are many more aspects, which are essential for most games and should therefore be included in a game physics engine.

One of the most important extensions is rigid body collision, which is so far not covered by our model. Collision treatment basically involves two steps: First we have to detect the collision, then we have to resolve it in a proper way. This section is only providing a brief outlook, for more details see [4], [6].

A. Collision detection

A collision of two bodies occurs as soon as the volumes of both bodies intersect. For two spheres with radii R_i and R_j we can use a very simple check:

$$\text{collision if: } R_i + R_j > \|x_i - x_j\|$$

But already for two planes we have to use more elaborate checks and if we want to detect intersection of two polygons this involves a lot of work and very expensive checks.

For reducing the cost of these checks one uses cheap checks first (*coarse culling*), and if those checks do not provide enough information one uses more expensive checks(*fine culling*).

B. Collision resolution

As soon as a collision is detected, we also have to resolve it in a physical correct way. The straightforward approach would be by applying the following force field on each pair of rigid bodies i and j:

$$\vec{F}_{ij}\left(\vec{r}_{ij}\right) = \begin{cases} 0 & \text{if bodies do not intersect} \\ -\infty \frac{\vec{r}_{ij}}{\|\vec{r}_{ij}\|} & \text{if bodies do intersect,} \end{cases}$$

where $\vec{r}_{ij} = \vec{x}_j - \vec{x}_i$ denotes the vector connecting the centre of mass of both bodies. This results in a pair of forces repelling both bodies from each other, if they intersect. While this approach makes perfectly sense from a physical point of view, it results in numerical problems, if we want to do the time propagation using one of the schemes introduced above.

Therefore a different approach has to be used, where we first resolve the collision by moving the bodies apart from each other such that they do not intersect anymore, then we compute the new velocities of the bodies using a *impulse based method*.

C. Many, many rigid bodies

Due to the $\mathcal{O}(n^2)$ complexity of collision detection, where n denotes the number of rigid bodies we want to model, the simulation of very many rigid bodies is going to cause problems if we implement it in a naïve way.

The solution to this problem is a *boundary volume hierarchy* for our set of rigid bodies. By using this technique, we can reduce the complexity to $\mathcal{O}(n \log(n))$ and it also becomes feasible to simulate a very high number of objects.

D. The linear complementarity problem formulation

The resolution of collisions might lead to problems as soon as more than two bodies are involved in one collision. Using the *linear complementary problem formulation* we can find the *optimal* way of resolving a collision of multiple bodies. This means we shift the bodies from a set of intersecting bodies by a minimal distance, while still resolving the collision in such a way that no more intersections occur.

E. The Bullet physics engine

Since implementing all the functionality, needed to correctly model rigid body behaviour, involves a lot of work, there exist many open-source and proprietary physics engines. One example of such an implementation is the *Bullet* physics engine [1], which implements all of the described methods and much more. The *Bullet* engine is also used in computer games (*GTA IV*) or movies (*Hancock*) [2].

References

[1] bulletphysics.org. `http://bulletphysics.org`. Accessed: 2015-11-22.

[2] Wikipedia: Bullet (software). `https://en.wikipedia.org/wiki/Bullet{\_}{\%}28software{\%}29`. Accessed: 2015-11-22.

[3] Wolfgang Demtröder. *Experimentalphysik 1: Mechanik und Wärme*. 5., neu be edition, 2008.

[4] Christer Ericson. *Real-time collision detection*, volume 1. 2005.

[5] Dietmar Gross, Werner Hauger, Jörg Schröder, and Wolfgang A Wall. *Technische Mechanik 3 : Kinetik*. Springer-Lehrbuch. Springer, Berlin, Heidelberg, 13., {ü}be edition, 2015.

[6] Ian Millington. *Game Physics Engine Development*.

[7] Gilbert Strang. *Computational science and engineering*. Wellesley-Cambridge Press, Wellesley, Ma., 2007.