Rigid Body Collision Response

Ladislav Kavan*

Faculty of Mathematics and Physics
Charles University
Prague / Czech Republic

Abstract

In many virtual reality applications it is necessary to simulate the interaction among solid objects. One of the basic requirements is to ensure a non-penetration of rigid bodies. We show algorithms for detecting a collision of moving bodies along with the exact time of collision determination. Next we present a method for computation the post-collision velocities and positions of colliding objects. In our approach we exploit the laws of classical mechanics, however the physical accuracy is not the goal. We sacrifice the accuracy in exchange for simple and real-time algorithms.

Keywords: Collision Response, Dynamic Collision Detection, Rigid Body Simulation, Physically Based Simulation

1 Introduction

Although the non-penetration of rigid bodies is quite common in the real world, in virtual environments the contrary is true. The reason is that the dynamics simulation is not an easy task; it involves several different problems. The first is the collision detection, which is considered the bottleneck of the simulation [15]. Moreover, classical collision detection, as we will refer to *static* collision detection, does not count with the objects motion. We demonstrate that considering the motion of the rigid bodies is necessary for correct collision response. After detecting the collision, the dynamic state of the colliding objects must be changed in order to avoid the inter-penetration. The change depends on the type of the collision and on the parameters of the colliding objects.

The very general goal of a rigid body dynamics simulation is to generate a movement of rigid bodies when external forces are given. This requires a numerical method for solving differential equations as presented in [3]. However we are interested only in the non-penetration constraints which we will ensure in rather kinematic way. We do not consider the forces that are responsible for the objects motion[1].

*lkav8604@barbora.ms.mff.cuni.cz

[1] In virtual reality the positions of the objects are usually controlled directly by some kind of input device. It is more intuitive for the user than controlling the forces acting on the objects.

In this paper, we do not discuss neither the static collision detection, nor the simulation of the external forces. We present a method for solving the collision response for only two colliding objects. Although the physical correctness is not our goal, we account for different rigid body's mass properties and the laws of mechanics to achieve more plausible results. In section 2 we present a brief outfit of rigid body mechanics, which we need in next sections; for more detailed explanation see for example [3]. Section 3 describes the problems of static collision detection and proposes a simple algorithm for dynamizing the collision detection. The collision response itself is discussed in section 4. Finally, in section 5 we present an application that implements these methods.

1.1 Related Work and Our Contribution

The static collision detection is well developed, the methods differ mainly by the bounding volumes they use. The Axis-Aligned Bounding Boxes [16] may seem obsolette, but they have certain properties useful for deformable models (quick tree re-computation after deformation). Two modern competing bounding volumes are Discrete Orientation Polytopes [10] and Oriented Bounding Boxes [8]. Dynamic collision detection is studied in [7]. They reduce the dynamic collision detection to a root finding problem, which is solved by numeric methods (namely regula falsi).

Also concerning collision response a lot of work has been done, but no uniform approach is the best in general. See [1] for a survey. Physically based collision response together with dynamics simulation (via numeric solution of differential equations) is explained in excellent tutorial [3, 4]. Besides the colliding contact, it involves also a resting contact treatment, which is one of the most difficult problems. [3, 4] show a physically based solution using static (contact) forces. These forces are computed with quadratic programming. Another method for the contact forces computation is based on the linear-complementarity problem (LCP) [2, 5]. [15] proposes an alternative to static forces by so called micro-collisions – many small impulses that are used instead of static forces.

Well readable articles intended for game developers are [11, 9]. They cover mainly the basic physics and colliding contact. An interesting recent result is described by [13].

They formulate the dynamics simulation as an optimization problem and solve it by quadratic programming.

Our contribution is mainly in simplifying complex approaches. This is the case of the dynamic collision detection (section 3) and resting contact (4.2). The application (5) is of course also original. On the other hand the colliding contact (4.1) as well as the rigid body mechanics (2) have already been described by other authors [3, 4].

2 Rigid Body Mechanics

A *rigid body* is a solid object that can not be deformed in any way – its shape does not change during the simulation. If we denote the body's volume $V \subseteq R^3$ and density function $\rho : V \rightarrow R$ we can define the body's mass

$$m = \int_V \rho(r) dV \tag{1}$$

and the center of mass

$$\frac{\int_V r\rho(r) dV}{m} \tag{2}$$

Vectors relative to the center of mass will be denoted by subscript $_c$. The moments of intertia are defined as follows

$$\begin{aligned}
J_{xx} &= \int_V (r_{c,y}^2 + r_{c,z}^2)\rho(r_c) dV \\
J_{yy} &= \int_V (r_{c,x}^2 + r_{c,z}^2)\rho(r_c) dV \\
J_{zz} &= \int_V (r_{c,x}^2 + r_{c,y}^2)\rho(r_c) dV \\
J_{xy} &= \int_V r_{c,x} r_{c,y} \rho(r_c) dV \\
J_{xz} &= \int_V r_{c,x} r_{c,z} \rho(r_c) dV \\
J_{yz} &= \int_V r_{c,y} r_{c,z} \rho(r_c) dV
\end{aligned}$$

where $r_c = (r_{c,x}, r_{c,y}, r_{c,z})$ is a body space vector relative to the center of mass. The moments form together an inertia tensor

$$J = \begin{pmatrix} J_{xx} & -J_{xy} & -J_{xz} \\ -J_{xy} & J_{yy} & -J_{yz} \\ -J_{xz} & -J_{yz} & J_{zz} \end{pmatrix} \tag{3}$$

All the integrals are computed in the body space relative to the center of mass. In computer graphics we usually work with solids represented by polygonal meshes. The integration over such objects can be efficiently computed using the divergence theorem; the procedure is described in [14].

Position of a rigid body in space is characterized by a translation vector and a rotation. The rotation can be represented by

- orthonormal (rotation) 3×3 matrix
- unitary axis of rotation and an angle of rotation
- unit quaternion

Each of this representations has its advantages and disadvantages. Conversions between them are described in [6]. Any of these representations requires only 3 scalars to store which sums together with the translation vector to 6 degrees of freedom (DOF).

The placement of a rigid body is advantageously described relatively to its center of mass. In a world space coordinate system in time t we define the position of the center of mass $r_c(t)$ and orientation given by rotation matrix $R(t)$. Then the vector r_b in body space maps to the vector $r_w(t)$ in world space by formula

$$r_w(t) = r_c(t) + R(t) r_b \tag{4}$$

The actual inertia tensor $I(t)$ for a rotated body is computed from the body's space inertia tensor by equation

$$I(t) = R(t) J R(t)^T \tag{5}$$

as derived in [3].

The movement of a rigid body can be decomposed to a linear velocity $v_c(t)$ of the center of mass and an angular velocity $\omega(t)$ relative to the center of mass. $\omega(t)$ is the unitary axis of rotation multiplied by the angular velocity. The velocity $v(t)$ of a point that has world space position $r(t)$ is computed as

$$\dot{r}(t) = v(t) = v_c(t) + \omega(t) \times (r(t) - r_c(t)) \tag{6}$$

The linear momentum $p(t)$ is computed from the linear velocity using the mass of the body:

$$p(t) = m v_c(t) \tag{7}$$

By analogy the angular momentum $b(t)$ is computed from the angular velocity using the inertia tensor:

$$b(t) = I\omega(t) \tag{8}$$

To describe the state of a moving body it is recommended to use rather moments than velocities because they are conserved in nature unlike the velocities[2].

The rigid body's linear momentum can be changed by an application of force F acting in the center of mass. The change in linear momentum is described as

$$\frac{\partial p}{\partial t} = F \tag{9}$$

To rotate the object it is necessary to apply a torque τ. The torque is determined by the force F and the point of its application r in the world space relative to the center of mass:

$$\tau = (r - r_c) \times F \tag{10}$$

[2] In a gyroscope $\omega(t)$ can change even if $b(t)$ is constant.

The change in angular momentum is similar to the linear case[3]

$$\frac{\partial b}{\partial t} = \tau \tag{11}$$

From the linear and angular moments it is straightforward to derive the velocities by multiplying equations 7, resp. 8 by inverse mass m^{-1}, resp. inverse inertia tensor I^{-1}. Note that the inertia tensor is a regular matrix if and only if the rigid body's mass is not zero.

2.1 Simulation of Rigid Body Collisions

In nature the rigid bodies do never penetrate each other. When a collision occurs the velocities of the colliding objects are changed in a discontinuous way so that the bodies do not penetrate. The physical explanation for this phenomena is an *impulse*. The impulse of force J_F is

$$J_F = \int_{t_0}^{t_1} F dt \tag{12}$$

if $\langle t_0, t_1 \rangle$ is the period of collision. The impulse of force corresponds to the difference of linear moments

$$\Delta p = p(t_1) - p(t_0) = J_F \tag{13}$$

Consider that a rigid body A with linear momentum p_A collides with a rigid body B with linear momentum p_B. If the change of linear momentum Δp is added to p_A then the opposite impulse $-\Delta p$ must be added to p_B to satisfy the law of conservation.

The impulsive torque of a force F applied in point r in world space is defined as

$$J_\tau = (r - r_c) \times J_F \tag{14}$$

Like the impulse of force changes the linear momentum, the impulsive torque changes the angular momentum

$$\Delta b = b(t_1) - b(t_0) = J_\tau \tag{15}$$

Since the angular momentum must be conserved too, the opposite impulsive torques have to be applied to both rigid bodies as in the linear case.

We are interested only on the impulsive forces and torques that arise during the collision and prevent the rigid bodies from penetration. The impulses are barely handled by the differential equations solver because they introduce discontinuity. However their advantage is that they are instant events – the colliding state usually quickly disappears.

We treat the simulation engine as a black box, no matter if it is a differential equations solver or a VRML manipulator. An input of this black box can by anything from force field to mouse coordinates. Its output is a position and orientation of a rigid body in given time. The important thing we need is that the black box has a feedback mechanism: it can be told the new velocities and positions after collision and use them in consequent simulation. The manipulator can, for example, move the objects itself until they are far enough.

[3]More correctly we should say that τ means the *total external* torque as well as F is the *total external* force because all the contributions of individual forces and torques reflect in moments.

3 Dynamic Collision Detection

Suppose we have already implemented a classic collision detection system based on the hierarchy of some type of bounding volumes. If we give it two triangular meshes it answers whether any two triangles intersect. This is what we call a *static* collision detection, because it does not take care about the movement of the objects. Nonetheless during the simulation of rigid bodies we must assume the solids are moving. The real-time computer simulation usually proceeds in a loop

1. set time counter t to current time t_{act}
2. $\Delta t = t_{act} - t$; $t = t_{act}$
3. simulate the evolution of the system during time Δt[4]
4. output the final state of the system
5. go to step 2

If we check the collisions by a static algorithm during this cycle we may miss a collision that has both arosen and disappeared during Δt. Although it does not lead to visible inter-penetration it may result in unrealistic behavior as illustrated in Fig. 1. The solution of this problem is called a *dynamic* collision detection: it answers whether any two triangles have intersected during time interval $\langle t_0, t_1 \rangle$ and if yes then it gives the exact time $t_c \in \langle t_0, t_1 \rangle$ of contact. More precisely, time interval $\langle t_0, t_c \rangle$ is collision-free but the objects are colliding in time $t_c + \epsilon_t$, where ϵ_t is a given precision. We see that the rest of the simulation in $(t_c, t_1 \rangle$ must be discarded because the system is in incorrect state – the non-penetration constraint has been violated.

We will divide the dynamic collision detection algorithm into two parts:

1. test if the objects collided and if yes then return *any* time t_e during the first collision. (There could have been more collisions during Δt, but the others are of no concern to us.)
2. t_0 and t_e are given such that there is no collision in t_0 but there is a collision in t_e. Find the exact time of contact $t_c \in \langle t_0, t_e \rangle$.

[4]There is an interesting problem connected with this step: for very precise simulation we would have to know how much time step 3 will take during its execution and add this time to Δt. Then it would be possible to draw the results in really *real* time. It is no problem if step 3 takes constant time but if the time depends on the state of the system (which is the case for collision response) it resembles solving a differential equation...

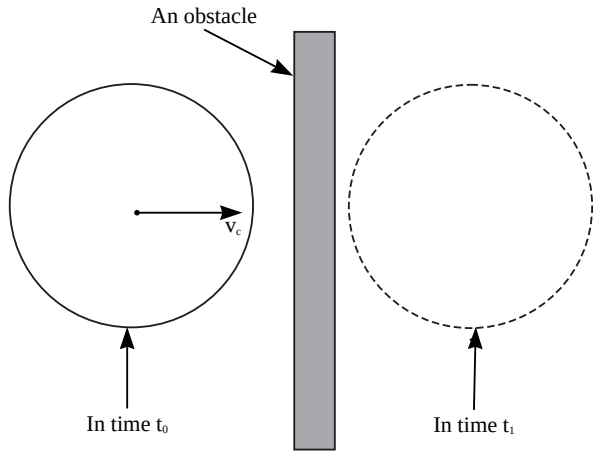

Figure 1: Static collision detection lets the ball pass through the obstacle.

The first point is complex in general. We will show a simple solution in section 3.2. Although it is only an approximation to exact dynamic collision detection, we believe it is quite acceptable for virtual reality applications.

The second point is much easier; it can be quickly solved with arbitrary precision as described in section 3.3.

3.1 Upper Bound of Velocity

Our idea is to avoid large inter-penetration. The inter-penetration magnitude is connected to the velocity of the colliding objects. Therefore we will need an upper bound for the maximal velocity on the surface of a triangular mesh.

Lemma: Let $0, 1, 2$ be vertices of a triangle T and r_c^0, r_c^1, r_c^2 their world space coordinates relative to the center of mass. Assume T rotates around a fixed axis with angular velocity ω and translates with linear velocity v. If r_c^p is any point of the triangle with velocity v^p then

$$\|v^p\| \leq \|v\| + \max_{i=0,1,2} \|\omega \times r_c^i\|$$

Proof: From equation 6 and triangle inequality we have

$$\|v^p\| = \|v + \omega \times r_c^p\| \leq \|v\| + \|\omega \times r_c^p\|$$

It remains to show that

$$\|\omega \times r_c^p\| \leq \max_{i=0,1,2} \|\omega \times r_c^i\|$$

Since r_c^p is a point of the triangle we can write it as a convex combination of vertices

$$r_c^p = r_c^0 s + r_c^1 t + r_c^2 u$$

where $s, t, u \geq 0$ and $s+t+u = 1$. Then by the cross product distributivity, triangle inequality and a trivial property of convex combination

$$\begin{aligned} \|\omega \times r_c^p\| &= \|(\omega \times r_c^0)s + (\omega \times r_c^1)t + (\omega \times r_c^2)u\| \\ &\leq \|\omega \times r_c^0\|s + \|\omega \times r_c^1\|t + \|\omega \times r_c^2\|u \\ &\leq \max_{i=0,1,2} \|\omega \times r_c^i\| \end{aligned}$$

□

Corollary: When looking for an upper bound of a maximal velocity of any point in the triangle mesh it is sufficient to maximize velocities in the vertices.

We did not make any assumptions about the actual object orientation and therefore the upper bound is correct for arbitrary simulation time. Note that for minimal velocity of a point of a triangle this lemma does not hold – minimum may not be in a vertex.

3.2 Collision Search Algorithm

The input data are two objects A and B with shapes defined by triangular meshes, each having a bounding volumes hierarchy. The linear and angular velocities $v_A, \omega_A, v_B, \omega_B$ of both rigid bodies are assumed to be constant during the simulation period $\langle t_0, t_1 \rangle$. Using the results of section 3.1 we determine the upper bound of maximal velocities v_{max}^A, v_{max}^B of any point on object A, resp. B.

As mentioned earlier we confine ourselves to only an approximate solution. The idea of our algorithm is very simple: we perform the simulation during $\langle t_0, t_1 \rangle$ with small enough steps Δt and check for collisions statically[5]. We stop as soon as we encounter a collision and return this time as t_e. It means that we really may miss some instant of collision, but we can ensure it will not be a big one – just a touch. Since no point on the surface of A (resp. B) moves faster than v_{max}^A (resp. v_{max}^B), the maximal possible inter-penetration will not be greater than

$$(v_{max}^A + v_{max}^B)\Delta t \tag{16}$$

If we can admit the maximal inter-penetration depth ϵ then it is sufficient to take

$$\Delta t \leq \frac{\epsilon}{v_{max}^A + v_{max}^B}{}^6 \tag{17}$$

The ϵ should be less than the thickness of the thinnest simulated object. For example for an application presenting a house interior, $\epsilon = 1mm$ will lead to quite precise dynamic collision detection. Nonetheless, the smaller the ϵ, the slower the algorithm will be in the worst case, since the number of steps is

$$\frac{t_1 - t_0}{\epsilon}(v_{max}^A + v_{max}^B) \tag{18}$$

[5] This approach is sometimes called *pseudo-dynamic* collision detection [7].

[6] Obviously if the denominator $v_{max}^A + v_{max}^B = 0$ then there is no need for dynamic collision detection, because both upper bounds are non-negative, thus zero.

We can improve it by introducing *dynamic* bounding volumes.

The dynamic bounding volumes differ from the static ones. Not by its geometry, it is the same (Sphere, AABB, OBB, k-DOP), but they bound all the rigid body positions over a time interval $\langle t_i, t_j \rangle$. Let us denote the rigid body by R (standing for either A or B) and its volume occupied in time t as $R_t \subseteq R^3$. Then the dynamic bounding volume V of rigid body R for time period $\langle t_i, t_j \rangle$ must satisfy

$$V \supseteq \bigcup_{t \in \langle t_i, t_j \rangle} R_t \tag{19}$$

Computation of such a bounding volume is simple for purely linear movement. In this case $\bigcup_{t \in \langle t_i, t_j \rangle} R_t$ is subset of a convex hull of $R_{t_i} \cup R_{t_j}$, thus any ordinary convex bounding volume of both R_{t_i} and R_{t_j} can do it.

The situation for the angular motion is somewhat more complicated as illustrated in Fig. 2. We define the radius R_r of object R as the distance to the most distant vertex from the center of mass. Then any rotation of the rigid body R around its center of mass will be bound by a sphere $S(R)$ with center in the center of mass and radius R_r. Then we can use the previous result for purely linear velocity and bound the volume $S(R_{t_i}) \cup S(R_{t_j})$ by any convex bounding volume.

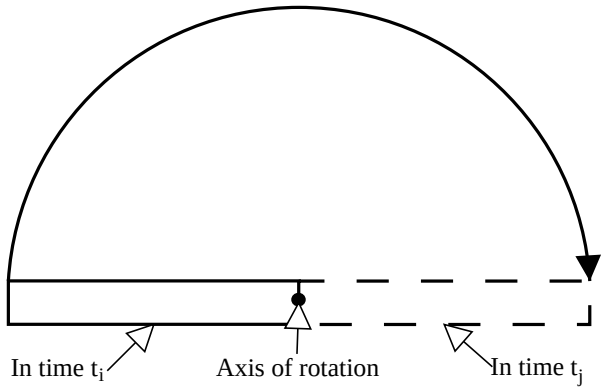

Figure 2: Bounding volume of a rotated box

The sphere boundary is effective only for spherical objects with the center of mass near the center of volume. For objects like the box in Fig. 2, better dynamic bounding volume should be designed, especially when considering the small angle of rotation during time interval $\langle t_i, t_j \rangle$. Such dynamic bounding volumes are object of further research.

3.3 Time of Contact

As described in section 3.2 we have found the first time of collision t_e and we have the time t_0 without collisions as well. Then it is easy to determine the time of contact $t_c \in \langle t_0, t_e \rangle$ by the binary search. It will find t_c up to a given time precision ϵ_t[7]. The algorithm proceeds in iteration

1. $t_s = t_0$; $t_k = t_e$
2. **if** $t_k - t_s < \epsilon_t$ **then return** t_s
3. $t_m = \frac{t_s + t_k}{2}$
4. **if** there is a collision in time t_m **then** $t_k = t_m$
5. **else** $t_s = t_m$
6. go to step 2

The correctness of the algorithm is obvious from the invariant: t_s is always collision-free and in t_k is always present a collision. The time complexity is

$$\mathcal{O}\left(\log \frac{t_e - t_0}{\epsilon_t}\right) \tag{20}$$

Note that the algorithm always returns the time without a collision, when the system is in a correct state.

4 Collision Response

Assume that the algorithms presented in section 3 reported a collision between rigid bodies A and B and computed the exact time of the first contact t_c. Now we shall examine the collision event and methods for its handling as described in [4]. Let $r_A(t_c)$, resp. $r_B(t_c)$ be the point of contact of body A, resp. B in world space with respect to the center of mass of A, resp. B. The points $r_A(t_c)$ and $r_B(t_c)$ coincide in the absolute coordinate system (not relative to the center of mass), but their velocities $\dot{r}_A(t_c)$, resp. $\dot{r}_B(t_c)$ can be quite different. If the rigid body A is moving with linear velocity v_A and angular velocity ω_A in time t_c, and analogously for B, then by equation 6 we have following relations

$$\dot{r}_A(t_c) = v_A(t_c) + \omega_A(t_c) \times r_A(t_c) \tag{21}$$
$$\dot{r}_B(t_c) = v_B(t_c) + \omega_B(t_c) \times r_B(t_c) \tag{22}$$

for the actual velocities of the contact points.

The direction of the collision is described by the unit normal vector $n_B(t_c)$ of the surface of rigid body B[8] in point $r_B(t_c)$. Important fact is that $n_B(t_c)$ points out of the object B's volume. The normal direction depends on the type of contact. There are only two non-singular contact possibilities: vertex-face and edge-edge contact. For vertex-face $n_B(t_c)$ is simply the normal of the face and for edge-edge contact it is the unitized cross-product of the edge directions.

Now we can examine the relative velocity v_{rel} of the two bodies projected to $n_B(t_c)$ direction

$$v_{rel} = n_B(t_c) \cdot (\dot{r}_A(t_c) - \dot{r}_B(t_c)) \tag{23}$$

If v_{rel} is positive (see Fig. 3), then the bodies are moving

[7] It has nothing in common with ϵ in section 3.2

[8] We could choose the body A as well, it is just a convention introduced by [4].

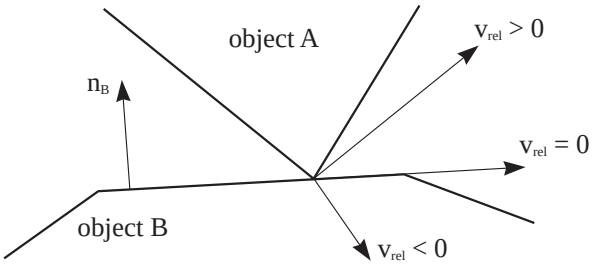

Figure 3: Possible relative velocities of two objects

apart. If we have correctly computed the contact point then this option is theoretically impossible. If v_{rel} is negative, then the bodies are approaching and if we do not change the object velocities immediately, then an inter-penetration occurs. This option is known as *colliding contact* and a method for computing new velocities is presented in section 4.1. If v_{rel} is zero, the bodies are neither receding, nor approaching. This situation is called a *resting contact* which will be discussed in section 4.2.

Consider that we always work up to a certain numeric precision, thus we can not test if $v_{rel} = 0$, since it will practically never be true. This must be replaced with something like $|v_{rel}| \leq \epsilon_r$. It follows that what we classify as a resting contact, can be a small retreating or colliding contact in reality, which introduces certain difficulties, see 4.2. However the colliding contact event is quite clear since the condition is $v_{rel} < -\epsilon_r$.

4.1 Colliding Contact

The physical model for a colliding contact response is an impulse J, see section 2.1. The impulse direction is given by $n_B(t_c)$ and what remains to compute is the impulse magnitude j (a scalar) so that

$$J = jn_B(t_c) \tag{24}$$

Once we have computed J, it is easy to compute the change of linear and angular moment (section 2.1). Recall that we use normal $n_B(t_c)$ outgoing from the rigid body B, pointing towards the rigid body A. Therefore the impulse acts positively on the rigid body A and a negative impulse $-J$ must be applied to B due to the laws of conservation. Then from the new moments we obtain the new linear and angular velocities by inverting equations 7 and 8 as explained in the end of section 2.

In order to compute the impulse magnitude j we denote the pre-impulse quantities by superscript $^-$ and the post-impulse one's by $^+$. The empirical law for frictionless collisions states

$$v_{rel}^+ = -Cv_{rel}^- \tag{25}$$

where C is a *restitution coefficient* satisfying $C \in \langle 0, 1 \rangle$. It is connected with the elasticity of the collision: $C = 1$ means perfectly bouncy collision, where no kinetic energy is lost. On the other hand $C = 0$ means no bounce at all, the maximum of the kinetic energy is used for example on the deformation of the objects.

The post-impulse velocities are connected with the pre-impulse one's by equations

$$v_A^+(t_c) = v_A^-(t_c) + \frac{jn_B(t_c)}{m_A} \tag{26}$$

$$\omega_A^+(t_c) = \omega_A^-(t_c) + I_A^{-1}(t_c)(r_A(t_c) \times jn_B(t_c)) \tag{27}$$

where m_A is the mass of the object A and I_A is its inertia tensor. Recall that I_A depends on the rotation of the object (eq. 5) and therefore on the time t_c. In the following we omit the time variable since it is always t_c.

Plugging equations 26 and 27 into 21 for post- and pre-impulse velocities we obtain

$$\begin{aligned}
\dot{r}_A^+ &= v_A^+ + \omega_A^+ \times r_A \\
&= v_A^- + \frac{jn_B}{m_A} + (\omega_A^- + I_A^{-1}(r_A \times jn_B)) \times r_A \\
&= \dot{r}_A^- + j\left(\frac{n_B}{m_A} + (I_A^{-1}(r_A \times n_B)) \times r_A\right)
\end{aligned}$$

The same can be derived for B considering that B is object of opposite impulse, i.e. of magnitude $-j$

$$\dot{r}_B^+ = \dot{r}_B^- - j\left(\frac{n_B}{m_B} + (I_B^{-1}(r_B \times n_B)) \times r_B\right)$$

Substituting these formulas into the v_{rel}^+ expression according to equation 23 and using the unit length of vector n_B, $n_B \cdot n_B = 1$ we have

$$\begin{aligned}
v_{rel}^+ &= n_B \cdot (\dot{r}_A^+ - \dot{r}_B^+) \\
&= n_B \cdot (\dot{r}_A^- - \dot{r}_B^-) + j(\frac{1}{m_A} + \frac{1}{m_B} + \\
&\quad n_B \cdot (I_A^{-1}(r_A \times n_B)) \times r_A) + \\
&\quad n_B \cdot (I_B^{-1}(r_B \times n_B)) \times r_B)) \\
&= v_{rel}^- + j(\frac{1}{m_A} + \frac{1}{m_B} + \\
&\quad n_B \cdot (I_A^{-1}(r_A \times n_B)) \times r_A) + \\
&\quad n_B \cdot (I_B^{-1}(r_B \times n_B)) \times r_B))
\end{aligned}$$

If we apply the restitution law (eq. 25), we obtain

$$\begin{aligned}
(-1 - C)v_{rel}^- &= j(\frac{1}{m_A} + \frac{1}{m_B} + \\
&\quad n_B \cdot (I_A^{-1}(r_A \times n_B)) \times r_A) + \\
&\quad n_B \cdot (I_B^{-1}(r_B \times n_B)) \times r_B))
\end{aligned} \tag{28}$$

from which we can already derive the impulse magnitude j since all the other variables are known. The inertia tensor inversion can be efficiently computed if we use eq. 5 and realize that

$$I^{-1} = (RJR^T)^{-1} = RJ^{-1}R^T \tag{29}$$

and J^{-1} can be computed off-line. Moreover, the non-inverted inertia tensor will not be needed anywhere in the simulation as well as the non-inverted mass. It allows tricky treatment of objects that should not be moved at all (e.g. walls). If we pose $m^{-1} = 0$ and I^{-1} a zero matrix then it corresponds to objects with an infinite mass. Due to equations 26 and 27 the velocities of these objects will not be changed.

4.2 Resting Contact

As mentioned above, resting contact is a situation where the relative velocity of two bodies is negligible, i.e. $|v_{rel}| \leq \epsilon_r$. Physically based treatment of resting contact is quite difficult: [4] shows a method for the inner forces[9] computation based on the quadratic programming. Another solution is based on the linear complementarity problem [2, 5]. We will considerably simplify this task by not considering the friction and by assumption of only two convex objects[10]. These presumptions enable an intuitive geometric solution based on the idea of pushing the rigid bodies apart.

Recall that in time t_c the objects are very close, but still not colliding. Because of the convexity assumption, we can use a separation theorem from computational geometry. It claims that if two convex sets are disjoint, then there exists a separation plane [12]. The problem is how to find the separation plane. To do this in a mathematically correct way we would need the nearest points from both bodies and construct the separation plane as in the proof of the theorem (see for example [12]). But since the algorithm of this section is rather heuristic, it is sufficient to approximate the separating plane, exploiting the fact that the objects are very close to each other.

We have already computed the normal $n_B(t_c)$ of body B in point $r_B(t_c)$. Assume for a while that the point $r_A(t_c)$ is equal to $r_B(t_c)$ in absolute coordinate system. Then, since the (non-strict) separating plane exists, it must pass through the point $r_B(t_c)$. Because it must separate the bodies, the only choice in general is the tangential plane in point $r_B(t_c)$, i.e. with normal $n_B(t_c)$. This is a good approximation since the points $r_A(t_c)$ and $r_B(t_c)$ can be made arbitrarily close by the algorithm in section 3.3. However, the purists can always compute the two nearest points to obtain the accurate separating plane.

The idea of our resting contact solution is simple: we let the bodies move as if they were not influenced by each other (this is accomplished by the previously mentioned black box) - using the zero friction assumption. A problem may occur when the actual v_{rel} is small but negative. In this case the objects may even collide, which is tested dynamically as described in section 3. Small inter-penetration can be prevented by process we call *separation*: pushing the rigid bodies apart in the direction of the normal of the separation plane. Remember that we need to simulate the time period of length Δt. It would be nice to say that from $|v_{rel}| \leq \epsilon_r$ follows that the inter-penetration is small, less or equal than $\epsilon_r \Delta t$. Unfortunately, this is not true, because the point of contact can move to a position with higher relative velocity than v_{rel}. This change may be arbitrarily high, since the tangential relative velocity does not need to be small. Therefore we propose an algorithm of successive separation. Supposing we simulate time period $\langle t_0, t_1 \rangle$, it works as follows.

1. $t_s = t_0$
2. use the black box to simulate the system during $\langle t_s, t_1 \rangle$ with dynamic collision detection
3. **if** collision reported in time $t_c \in \langle t_s, t_1 \rangle$ **then** stop the simulation in time t_c, separate the objects to distance D and tell the black-box the new positions after separation.
4. **else return**
5. $t_s = t_c + f(D)$
6. **if** $t_s \geq t_1$ **then return**
7. go to step 2

Two things remain to clarify. Firstly, the separation to the distance D means to translate the object A by vector $\frac{m_B}{m_A+m_B} D n_B(t_c)$ and the object B by $-\frac{m_A}{m_A+m_B} D n_B(t_c)$, see Fig. 4. The rigid bodies masses

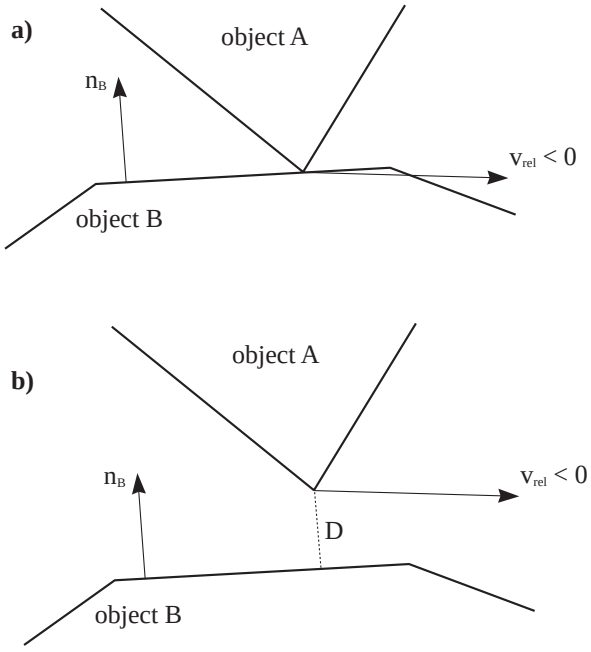

Figure 4: a) The resting contact situation, b) After separation to the distance D

[9] as a consequence of the Newton's law of action and reaction

[10] The case with more convex objects is similar to non-convex objects, since non-convex objects can be decomposed into more convex one's.

m_A and m_B are used to distribute the translation in an intuitive way. How to choose the parameter D is a question of tuning. Higher D leads to faster execution, but more coarse steps. It is similar to the choice of ϵ in section 3.2.

Another subtle point is that we must account some time for separation, otherwise the number of separations each step would not be bounded, resulting in not plausible behavior. This is expressed by function f depending on the separating distance D. A simple linear function can do this job.

Despite the simplicity of this algorithm, the separating plane idea is a basis for one of the most recent rigid body simulators [13].

5 Application

We have tested the presented algorithms in the application for virtual reality simulation of fencing. The typical collision response event in this application is illustrated in Fig. 6. In the picture the fencer on the right is holding its weapon still. The left fencer's weapon is moving with velocities depicted by arrows. The state in the time of contact, Fig. 6b, is not normally drawn in the simulation loop – it is used only for the collision response computations.

Although the weapon is drawn as a textured triangular mesh, the collision response module considers only its approximation by so called *capsule*. A capsule is a point set given by a line segment $\overline{AB}$ and a radius r

$$\left\{x : dist(\overline{AB}, x) \leq r\right\} \tag{30}$$

The point to segment distance $dist$ can be computed quite efficiently. A capsule is similar to cylinder (both are convex), but has certain advantages.

The surface of a capsule is smooth, thus there are no problems with the definition of a normal direction $n_B(t_c)$ in any point of the capsule's boundary. A test if a capsule given by $\overline{A_0B_0}$, r_0 intersects another capsule given by $\overline{A_1B_1}$, r_1 is simply

$$dist(\overline{A_0B_0}, \overline{A_1B_1}) \leq r_0 + r_1 \tag{31}$$

and the segment to segment distance can be computed also very quickly. This allows very fast collision detection, enabling small ϵ for the dynamic collision detection algorithm presented in section 3.2.

Quite different is handling weapon to body collisions. Obviously the human body can not be considered rigid; fortunately the response to weapon-body collisions requires no dynamic simulation, since it simply means the end of the duel. Because the body undergoes many deformations resulting from motion, we use the bounding boxes re-fitting algorithm [16] generalized for k-DOPs.

The weapon, as well as its approximation by a capsule, is controlled by some common type of input device: mouse or joystick. There is a problem that such input devices have only 2 DOF, but rigid body's position and orientation needs 6 DOF. We solved it by defining several keyframe placements of a weapon and interpolating among them according to the current state of the input device. An example of such function mapping $R^2 \rightarrow R^6$ is illustrated in Fig. 5. The input device controls the placement of the weapon directly and the velocities are estimated using the elapsed time Δt. This is our implementation of the black box. It remains to describe its feedback mechanism. The translation resulting from separation is straightforward. Somewhat more tricky is the reaction after colliding contact, because it is generally impossible to feed-back the movement to the input device [11].

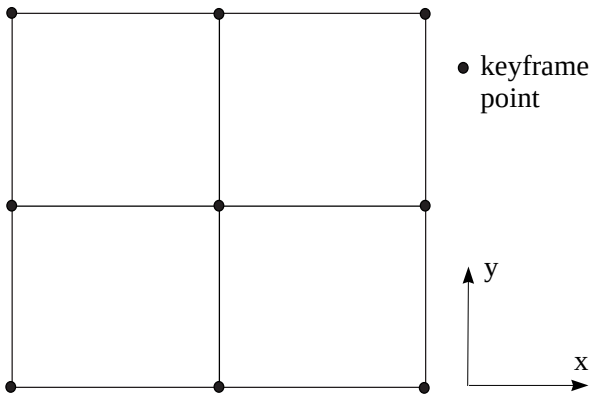

Figure 5: Grid of nine different key positions and orientations of the weapon in the space of an input device.

In fencing, the weapon is grasped by the fencer's hand. When the weapon is hit hardly by the opponents weapon [12], the fencer looses control of his own weapon for a short amount of time (which the opponent may use to attack), because of impulse delivered by the opponents weapon. If the time of the contact is t_a and the time of re-gaining the weapon control is t_b (it depends on the impulse magnitude, strength of the grasp etc.), we simulate the process as follows. In time t_{act} we compute the position and orientation $P(t_{act})$ as if the weapon was not held by the hand and moving only with the post-impulse velocities computed by formulas in section 4.1. Then we interpolate $P(t_{act})$ with the actual position and orientation of the hand with interpolation parameter

$$t = \frac{t_{act} - t_a}{t_b - t_a} \tag{32}$$

which will give the resulting position and orientation. This simulates the process of re-gaining the control ("catching") the weapon realistically: the fencer influences its weapon movement only partially and this influence (t) increases with time.

[11] However certain recent joysticks already support some feed-back effects.

[12] action known as *batuta* aiming to deflect the opponent's weapon

6 Conclusions

We have presented a set of algorithms that can be used together to simulate the response after a collision of two rigid bodies. They are best suited for simple objects, such as the capsules in our application. Colliding contact is handled in a physically correct way, but the resting contact is simulated in rather intuitive way. However, the resulting post-collision motion of the objects is quite plausible as was verified in the application. We did not measure the speed of the algorithms, because it is determined mainly by the speed of the static collision test, which was not discussed. The number of iterations of the presented algorithms is heavily influenced by the ϵ setting, as was shown in the formulas. For a majority of applications the collision response routines are not the bottleneck.

A lot of work can be done in any of the mentioned areas, mainly in the dynamic collision detection and the resting contact handling. Nonetheless, we believe that the simplicity of the referred algorithms has also its advantages, not only for the implementation's sake.

References

[1] David Baraff. Non-penetrating rigid body simulation. *Eurographics 93 State of the Art Reports*, September 1993.

[2] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. *SIGGRAPH*, July 1994.

[3] David Baraff. An introduction to physically based modeling: Rigid body simulation 1 - unconstrained rigid body dynamics. *SIGGRAPH Course Notes*, 1997.

[4] David Baraff. An introduction to physically based modeling: Rigid body simulation 2 - nonpenetration constraints. *SIGGRAPH Course Notes*, 1997.

[5] Matthias Buck and Elmar Schömer. Interactive rigid body manipulation with obstacle contacts. *The Journal of Visualization and Computer Animation*, 9(4):243–257, – 1998.

[6] David Eberly. *3D game engine design: a practical approach to real-time computer graphics*. Morgan Kaufmann Publishers Inc., 2000.

[7] Jens Eckstein and Elmar Schömer. Dynamic collision detection in virtual reality applications. In V. Skala, editor, *WSCG'99 Conference Proceedings*, 1999.

[8] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[9] Chris Hecker. Physics, part 3: Collision response. *Game Developers Magazine*, pages 11–18, March 1997.

[10] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, /1998.

[11] Jeff Lander. Collision response: Bouncy, trouncy, fun. *Game Developers Magazine*, pages 15–19, March 1999.

[12] Jiri Matousek. *Lectures on Discrete Geometry*. Springer, April 2002.

[13] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. *ACM SIGGRAPH*, pages 37–46, August 2001.

[14] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools: JGT*, 1(2):31–50, 1996.

[15] Brian Mirtich and John F. Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, pages 181–188, 217, 1995.

[16] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools: JGT*, 2(4):1–14, 1997.

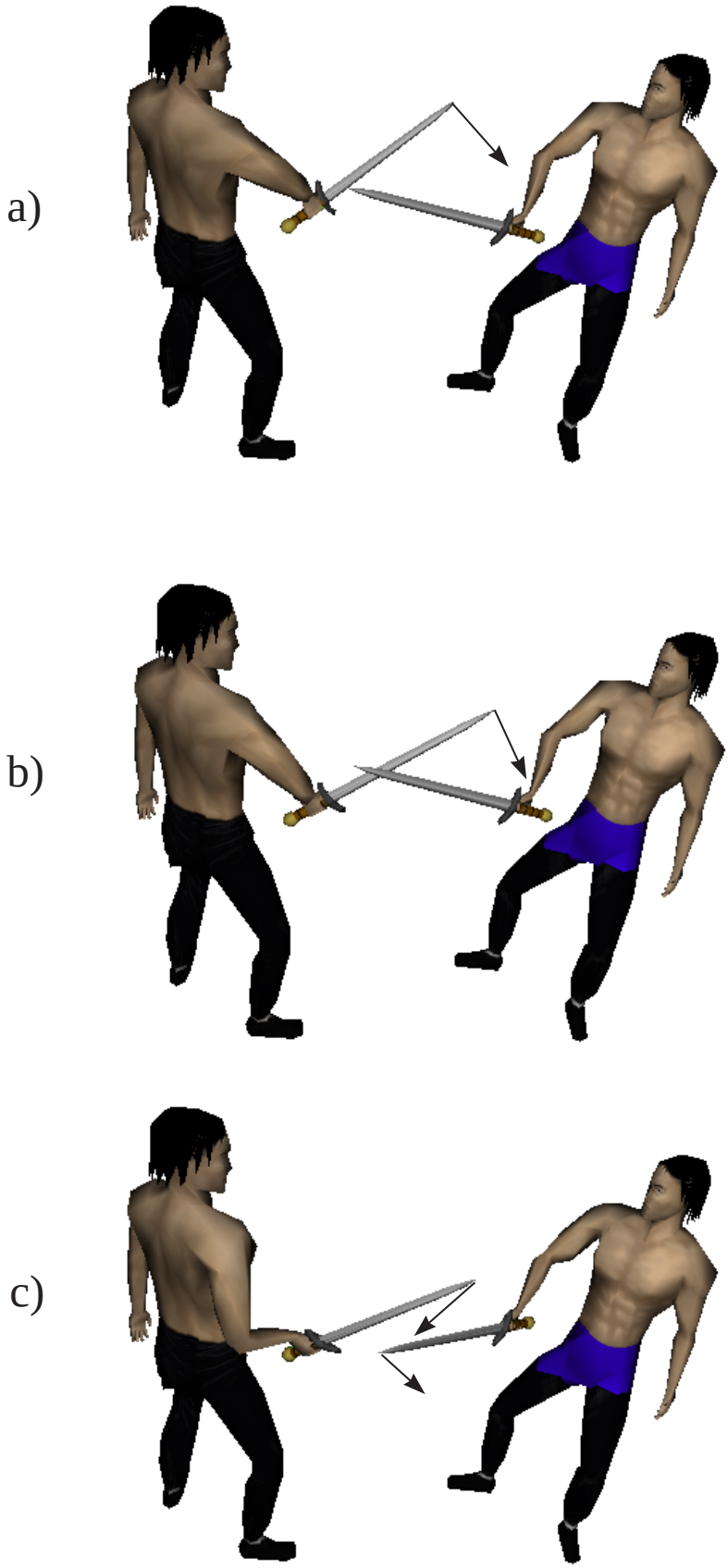

Figure 6: Application: Fencing in Virtual Reality. a) weapons before collision, b) state in the time of contact, c) post-collision movement