Simulation of Motion with Collisions Using Modelica

Vadim Engelson, Dag Fritzson, Peter Fritzson,
(vaden,dagfr,petfr)@ida.liu.se
Dept. of Computer and Information Science
Linkp̈ing University, S-58183, Sweden

Abstract

Collision detection and response is one of the most difficult areas in simulation of multibody systems. Two known approaches, the impulse-based method and the force-based (penalty) method, can be applied for multibody simulation in Modelica[13]. The impulse-based method requires instantaneous modification of some variables, but such modification is not always possible in Modelica. The force-based method leads to stiff ODE, which can be handled by solvers used with Modelica. We suggest a new way to express the penalty coefficients. The force-based method, however, requires computation of penetration depth which is time-consuming.

Calling external functions is a preferable method to integrate collision detection algorithms with practical physical models, since body geometry is stored externally. We describe an interface with collision detection tool SOLID.

Keywords: Mechanical modeling, Simulation, Modelica, Collision Detection

1 Introduction

In mechanical systems certain machine elements usually interact with each other. When a mathematical model of such a system is designed, the interactions between the parts can be divided into the two following categories:

- *Mechanical joints* are used for definition of permanent constraints of motion.
- *Mechanical contacts* are almost instantaneous, typically short-time interactions caused by non-penetration contact forces arising between the bodies in the model. The forces occur when the body surfaces touch each other. Two major phenomena occur in mechanical contacts are friction contacts (causing static or dynamic friction forces) and collision contacts (causing collision response forces).

Computation of contact forces is a difficult task. The bodies might move in a complicated way, and they might have a complex geometry. When at some instant they touch each other, penetration of the bodies should be prevented. There is a tradeoff between efficiency and accuracy. One of the goals of Modelica simulations is interactivity, therefore

the computation should have at least the same speed as the processes in the real mechanisms. There exist accurate methods for contact force computation based on finite element methods and other methods using subdivision of bodies into very small fragments. Currently these cannot run at interactive speed.

The most accurate and realistic methods used in mechanical simulations of contacts are developed in the area of mechanical analysis called tribology. This theory takes into account the lubricant properties.

One of the difficulties with the computation of contact forces is the variety of surface geometries. For certain kinds of surfaces (plane, spline of 2nd order) there exist collision checking methods and approaches to calculate the forces that arise. Such systems in general use higher order polynomials to compute forces from geometrical relations. Theory and applications of contact situations are discussed, for instance, in [16].

An accurate method (e.g. a FEM method) for contact force computation requires that the surfaces of two colliding bodies are covered by a mesh, and that the relevant contact force is computed for each node. The resulting force is then found by integrating of all the forces acting on the contact surface. Experiments with contact computation of rolling bearing models [5] show that this method is accurate, but requires tremendous computing resources.

The major advantage of using Modelica[13] is hybrid (both continuous and discrete) modeling in multiple application domains (mechanics, electrical, hydraulic, etc.)

The Multibody System (MBS) library in Modelica[14] is used for mechanical model simulation. Currently this library supports simulation of models with rigid bodies and joints. Friction occurring in the joints can optionally be taken into account in the models. However, collision detection and collision response is not supported.

The goal of this work is to identify the ways to add collision response to mechanical simulation models based on MBS.

Our approach to modeling collisions is based on combining several components. In order to do that, collisions between bodies should be detected, collision response should be evaluated, and this response should in some way affect the simulation.

The structure of the report and the relation between its parts is shown in Figure 1.

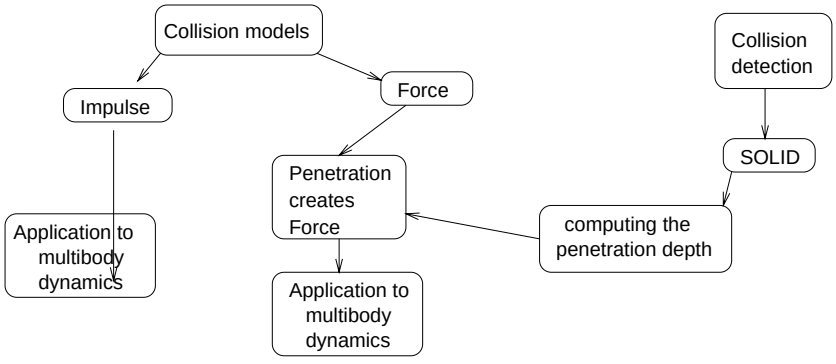

Figure 1: The structure of the report.

The two major collision models used in the simulation are impulse-based and force-based models. Both assume that the bodies are rigid. The impulse-based approach uses collision impulses between the bodies (Section 2). The force-based approach computes a non-penetration force (Section 3). A traditional variant of the force-based approach is the penalty method which assumes that bodies in contact behave like objects connected

by a spring and damper. We consider these methods in application to Modelica. A new method for computing force from penetration depth is given in Section 4. There is a number of common properties of collision detection tools (Section 5). A the particular tool, SOLID, is used for finding the penetration depth (Section 5.2). Section 6 illustrates how the routines for computing the forces are integrated with mechanical models in Modelica.

2 Impulse Model

A standard way of handling collisions in mechanics is based on the linear momentum preservation law.

2.1 Impulse and Velocity Equations

In the equations below m_a, m_b are masses of bodies A and B; v_a, v_b are velocity vectors of bodies A and B before the collision; v'_a, v'_b are the velocity vectors of the bodies after the collision.

The total linear momentum of a system consisting of bodies A and B is preserved if there is no external impact: $m_a v_a + m_b v_b = m_a v'_a + m_b v'_b$.

The change of the projections of the velocities of point masses on $\vec{n}$ (normal to the collision plane) can be expressed using the restitution coefficient $\varepsilon = (v'_a - v'_b)/(v_b - v_a)$, which can vary from absolutely elastic collision ($\varepsilon = 1$) and non-elastic (completely damped) collision ($\varepsilon = 0$), depending mainly on material properties of bodies A and B. If object B is rigidly attached to the inertial system $v'_a = -\varepsilon v_a$.

Also, the angular momentum of two colliding bodies is preserved: $I_a\omega_a + I_b\omega_b = I_a\omega'_a + I_b\omega'_b$, where ω and ω' are angular velocities before and after the collision, I is inertia tensor.

Equations for computations of ω' can be found in [20]. Derivation of the equations we use is available in [17].

Several assumptions (see e.g. [10, 19]) are taken into account when the law of linear momentum preservation is used for physics-based simulation:

— Collision duration is negligible.

— There exists just one point of collision.

— The colliding bodies don't move during the collision.

— No other forces than collision force act on the bodies.

— The impulse gives instantaneous change to the linear and angular velocities of the colliding objects.

2.2 Finding collision time and plane

Some systems [19] are constructed so that collision takes place when the distance between the closest features of two bodies is less than some threshold T. The exact time of collision (if it happens at some instant between two time frames) is not computed. Penetration is avoided by choosing a safe maximal time step (Δt) and estimating the maximum speed (v_{max}), so that $T > v_{max}\Delta t$.

If penetration occurs the time step should be changed and some backtracking in the solution process must be performed. Of course, this requires fine-grained control over the differential equation solver that is used for numerical simulation.

Other systems [17] search for exact time of contact, using the method of bisection.

The time interval normally used in the solver is divided by two, and the time of contact is searched in one of two smaller time intervals. This subdivision continues until some tolerance boundary is reached. This requires even more control over the solver.

The collision plane for disjoint objects is defined as follows. We assume that bodies consist of such features as vertices, edges and faces. If one of the colliding features is a face, this face is used as a collision plane. If vertices or edges are the closest features, the shortest line between them is used as a normal to the collision plane.

The situation in which a body is resting on a surface is treated as a constraint (giving an additional equation) or as series of micro-collisions (as proposed in [10]).

2.3 Simulation

Since the velocity is not continuous in the impulse-based model, it is not very appropriate for use with traditional ODE solvers. Actually, the continuous integration process in the solver should stop at the instant of collision and resume with the new velocity, as it is done, for instance, in Modelica.

An alternative approach is based on writing a system of non-differentiable equations and applying a Newton method specially devised for such equations [15]. This method has successfully been applied for body impact with friction by Johansson and Klarbring [6].

Variations of the same impulse-based model [10] can describe rolling, sliding, resting and bouncing. This mathematical model has been combined by Zhang [19] with collision detection algorithm I_COLLIDE [7] to form an ODE-based simulation tool. This model, however, assumes that between the collisions the objects have ballistic trajectories, i.e. they are not constrained by revolute and translational joints.

2.4 The Impulse-Based Approach and Modelica

Modelica has capabilities for modeling the impulse-based collision response algorithm.

The Modelica language, like some other modeling and simulation tools, has support for instantaneous change of some variable values during simulation. This change might happen at special simulation steps, called *events*. At these events the continuous integration (i.e. the process of solution of the system of differential and algebraic equations) stops, specific equations valid for this event are solved, variables obtain new values, and then the integration continues. This is the way Modelica carries out *hybrid* modeling, where both continuous and discrete behavior of the system are described in the same model.

2.5 Colliding Pendulum Example

It is possible to use the **`when`** and **`reinit`** statements in the MBS library context.

The pendulum example (see Figure 2) illustrates the use of the MBS library for a model with impulse-based collision response. The state variable `qd` $= \dot{q}$ in this model is the angular velocity of the revolute joint of the pendulum, and the model makes an assumption that the velocity (which changes instantaneously at the moment of collision) is proportional to `qd`. This assumption is wrong in the more complicated cases. The change of the angle of rotation $R1.q$ is shown in Figure 2.

```
model Pendulum
  "Impulse-based model of collision of pendulum end with an obstacle"
  Inertial i;
  RevoluteS R1(n={0,0,1},q(start=1));
  BodyBase M1 (m=50,rCM={0,0.5,0},I=[0,0,0;0,0,0;0,0,0]);
  Bar B (r={0,1,0});
  parameter Real restitution=0.5;
  parameter Real obstacle_x = 0.5 " x coordinate of obstacle";
equation
  connect(i.b, R1.a);
  connect(R1.b, M1.a);
  connect(R1.b, B.a);
  when (B.r0b[1]>obstacle_x) then
    reinit(R1.qd,-restitution*R1.qd);
    // A problem is how to propagate this change to velocity of B
  end when;
end Pendulum;
```

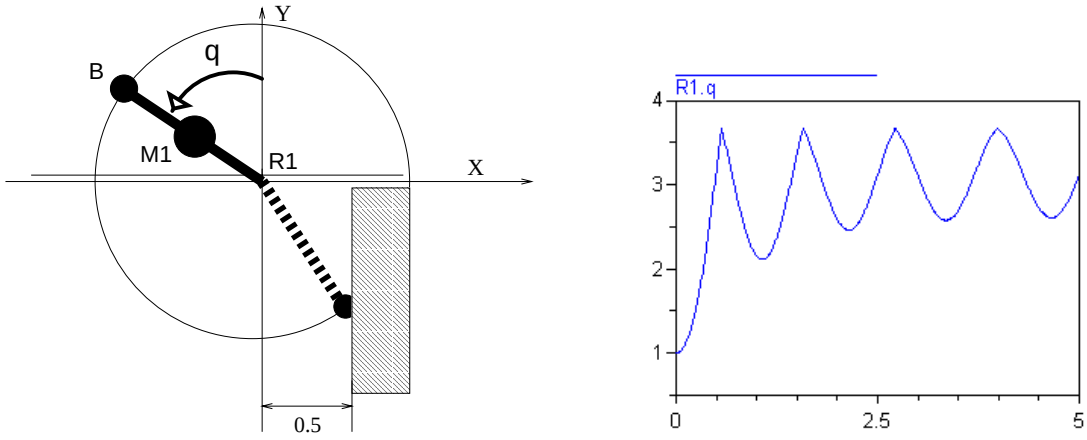

Figure 2: Simple pendulum colliding with an obstacle at point $Obstacle_x = 0.5$. The plot shows the angle of rotation of the pendulum colliding with an obstacle. The restitution coefficient is $\varepsilon = 0.5$. The collision happens when the angle reaches `R1.q`$=\pi + \arcsin(Obstacle_x) \approx 3.66$.

.

2.6 The Problem of Non-State Variables

There is one major difficulty in applying the impulse-based model to mechanisms with rotating parts.

In order to apply the impulse-based approach, the velocity should be changed, according to the rules, $v := g(v)$, where g can be a complicated function depending on collision details. Modelica allows only instantaneous change of *state* variables. In the MBS models, which include rotating bodies, the linear velocity is not a state variable. It is a *dependent* variable that can be computed from state variables. There are only two ways to handle with this difficulty:

- Restructure the model so that velocities become state variables, and apply impulse-based approach to this model.
- Re-initialize the state variables q in such a way ($q := f(q)$) that the corresponding velocities change ($v \rightarrow g(v)$) according to the impulse-based approach. The difficulty is to find the function f from g.

2.7 Restructuring the Model of Colliding Double Pendulum Example

It is possible to use a kinematic loop in order to restructure the double pendulum example (see Figure 3). In particular, the double pendulum model with 2 revolute joints (containing 2 state variables) (Figure 3(a)) can be replaced by a closed kinematic loop with 2 prismatic joints (with state variables), one rotational joint necessary for cutting kinematic loops, and 2 revolute joints (without state variables) (Figure 3(b)) .

Both models (a) and (b) result in the same motion when no collisions occur, but they use different state variables. The Modelica model for the construction in Figure 3(b) is given below. The connection diagram of this model is shown in Figure 4.

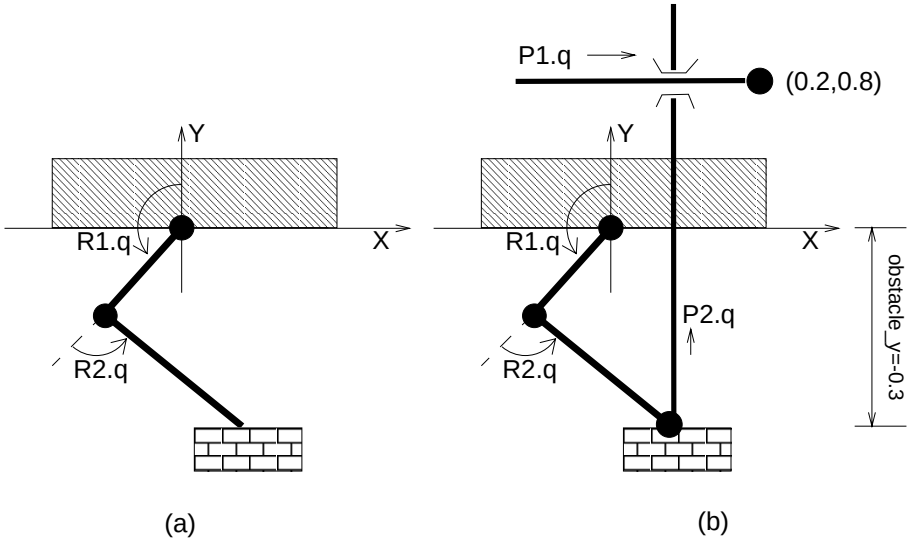

Figure 3: Double pendulum (a) combined with additional joints to form a kinematic loop (b).

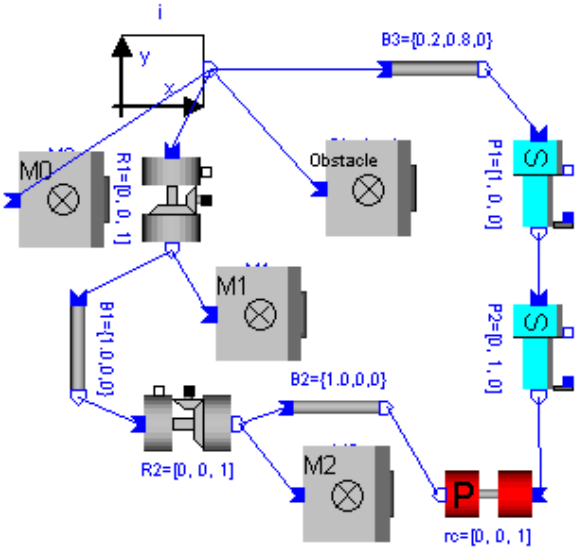

Figure 4: Double pendulum model with kinematic loop, graphical presentation.

```
model Pendulum
  parameter Real obstacle_y=-0.3;
  output Real x;
  output  Real y;
  output  Real dist;
  parameter Real restitution=0.5;
  BodyV M1 (mass=50,r={0.5,0,0},
    Shape="box", Size={1,0.1,0.1});
  BodyV M2 (mass=50,r={0.5,0,0},
    Shape="box", Size={1,0.1,0.1});
  RevoluteCut2D rc;
  // some objects presented in the diagram are omitted here
equation
  x=P1.r0b[1];
  y=P2.r0b[2];
  dist=obstacle_y-y; // positive when collided.
  when (dist>0) then
    reinit( P2.qd,  -restitution * P2.qd);
  // change in linear velocity of the prismatic joint
  end when;
  // connections presented in the diagram are omitted here
end Pendulum;
```

When simulated and visualized, this model demonstrates a correct bouncing behaviour. The difficulties with this approach are:

- Each body in the multibody system requires a specially designed construction with kinematic loop, which leads to a huge number of additional objects.
- Computations with kinematic loops easily reach singularity points where solvers cannot find an appropriate solution (in the case above this happens when the angle `R2.q` crosses 0).

2.8 Using Dependencies Between State Variables and Body Velocities

Computing the dependencies between the state variables and body velocities can be difficult. It should be noted that in a tree-like structure of a multibody system, the collision of a body in one node can cause the change of velocities in all joints between this body and the ”root” of the tree. The technique for this propagation of velocities has been developed in [11] as well as [12] (p. 146). This is a sequential algorithm based on sending three ”test impulses” through the links of the multibody system. It would be rather hard to implement the algorithm in the connection-based MBS model.

2.9 Limitations of the Impulse-Based Method in Modelica Models

MBS-based models might contain static objects, single bodes, and multibodies (objects consisting of more than one single bodies connected by revolute and prismatic joints).

The impulse-based approach can easily be used in MBS-based models if the impact of collision on multibodies bodies in the system is negligible.

For instance, a system of single free-flying bodies, which collide with the walls of some volume, and which may also collide with robot manipulators. The collisions affect the free flying bodies, but they do not affect the robot.

Currently, this restricts applicability of the impulse-based model for dynamic analysis. If an appropriate solution to the problems mentioned in Sections 2.8 and 2.7 is found, collision processing can be added to arbitrary MBS models.

3 The Force Model of Collision

An alternative approach to collision processing in mechanical systems is based on the force and torque model of collision. We assume that colliding bodies penetrate and that a separation force is caused by this penetration. This force tries to prevent further penetration and to separate the colliding bodies.

It is well known that during the collision a relatively large force occurs between the two colliding bodies for a very short period of time. The value and the direction of the force can be approximately computed for each simulation time step. The following properties of the collision force should be taken into account:

- The collision force and collision torque acting on an object is zero if the object does not collide.
- Between the start of collision and the end of collision a force is activated that prevents further penetration.
- If an ideal collision is modeled (collision of points masses), the resulting velocities after the collision are given by the law of preservation of linear momentum.
- A contact force acting on a body resting on a horizontal platform compensates for the gravitational force applied to the body. Therefore such an object does not move (its vertical acceleration and velocity is zero).

In practice it is important that the force is differentiable and the total mechanical energy of the system is consumed (not produced) during the collision. Some energy is transformed to the thermal energy.

In order to balance the required accuracy and available computational power of Modelica simulations, we derived the following rules for collision force computation (some of the terms are discussed in the following sections):

- The collision force acting on a body is zero if the body does not penetrate with any other body.
- Perfectly rigid bodies do not penetrate each other during contacts. However, in practice physical bodies always penetrate a distance which is a small fraction of their size. If body A penetrates body B, collision forces are created to act on the objects A and B and are applied at the point of contact on each body. The force is directed so that A and B are pushed away from each other due to this force. The direction of this force corresponds to the shortest displacement that can separate the bodies.
- The magnitude of the force is proportional to the depth of penetration of A and B. This depth is the length of the shortest displacement that can separate the bodies. This corresponds to the model of spring and damper, inserted between the bodies. The bodies are rigid, but the spring and the damper are not rigid. This also corresponds to the physics of collisions between elastic and homogeneous (isotropic) bodies.

3.1 The Point of Contact and Direction of Collision Force

If the separation force should be computed during several time steps, then the point of the very first contact can differ from the point of contact a few steps later. An example of such a behaviour is two bodies colliding and then keeping sliding contact. In this case the point of the first contact cannot be used for evaluation of contact force for the next steps. The direction of the collision force is usually taken as opposite to the velocity of the contact point.

Collision detection software packages usually do not find the point of the first contact. They just determine a point which belongs to the intersection of the objects A and B if they collide. If the objects do not collide, the closest pair of their points can be determined. Obviously, these points may become irrelevant for further computation in the case of a long contact between the bodies.

3.2 Penetration Prevention Model

In our approach the separation force is applied at the corresponding closest points. The force F in direction $\vec{c}$ is applied to the body B at the point P_{BA}; it pushes out body B from body A. The opposite force $-F$ in direction $-\vec{c}$ is applied to body A at the point P_{AB}, and pushes body A away from B.

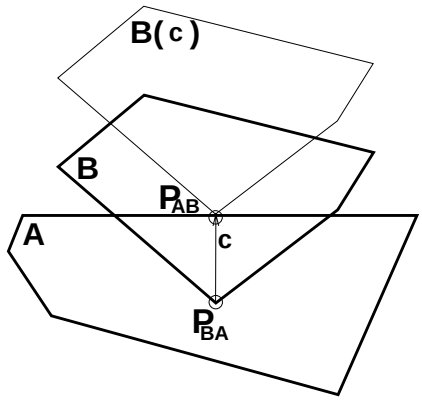

Figure 5: Collision geometry in the simple case: a single vertex of B is located within A.

4 Computation of the Force from Penetration Measurement

The experiments with various approaches to the computation of the collision force F have been done in the following stages:

- An equation for F with some unknown coefficients is chosen.
- A series of simulations is performed to find the dependency between the coefficients, the body velocity before the collision, and the velocity after the collision.
- The inverse dependency between the velocities and the coefficients is computed.
- Since the velocities can be computed using the law of conservation of linear momentum and the restitution coefficient, the coefficients for F can be found.

The formula for F can be found from the laws of dynamics. During the collision ($0 < t < \tau$) the velocity changes from $v(0) = v_0$ to $v(\tau) = v'$. Therefore the actual dependency can be expressed as

$$\int_0^\tau F(t)\,dt = m(v' - v_0)\ .$$

This definition, however, cannot be used for modeling the force in a dynamic simulation since the time τ is not known at the start and during the collision. Furthermore, if some other forces are applied to the body, the collision can go over to a stable contact. In this case $\tau = +\infty$.

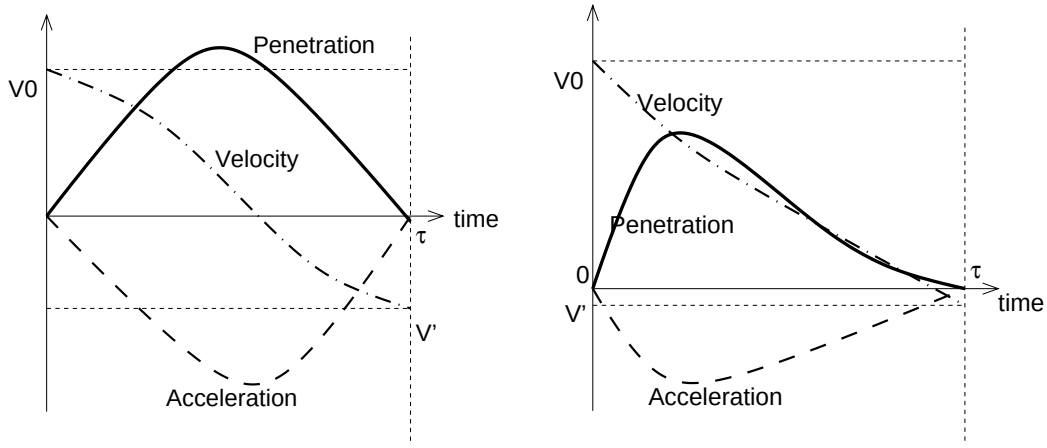

Figure 6: Collision depth, velocity and acceleration in case of elastic(left) and non-elastic (right) collision.

Figure 6(left) displays what might happen with penetration depth, velocity and acceleration during the collision time in the case of an elastic collision. In the case of a non-elastic collision (Figure 6, right), the penetration depth, velocity and acceleration become zero after the time τ. If the body has a non-elastic collision with a horizontal platform, and gravity is present, the penetration (directed downwards) is slightly more than zero, and the collision force (directed upwards) is slightly below zero, which compensates for the gravity force (directed downwards).

The following notation will be used below:

- $F(t)$ — collision force during the collision between a body and a fixed immovable obstacle.
- k, K, q — collision coefficients, they will be used in further computations.
- $x(t)$ — penetration depth.
- $v(t) = \dot{x}(t)$ — speed of change of the penetration depth.
- m — mass of the object penetrating the obstacle.

4.1 Constraints on Force Equations

The following constraints should be used in order to derive the equation for the force magnitude.

- Conditions at the start of the collision are $F(0) = 0$, $x(0) = 0$, $v(0) = v_0 > 0$.
- During the collision $F(t) < 0$, $x(t) > 0$, $v(t)$ changes from positive to negative.
- If no external force act on the colliding body (or this force is negligible), it behaves exactly according to the impulse law. The time when collision ends is τ. Then $F(\tau) = 0$, $x(\tau) = 0$, $v(\tau) = v' = v_1 = -\varepsilon v_0 < 0$ — the speed at the end of the collision can be predicted using the restitution coefficient $0 < \varepsilon < 1$.
- If a constant external force F_{ext} acts on the colliding body, it either behaves as above or, in case F_{ext} is large enough, the body rests on the obstacle, and the collision never ends: $\lim_{t\to\infty} F(t) = -F_{ext}$ — the collision force should compensate the external force; $\lim_{t\to\infty} x(t) = x_{rest} > 0$ — the penetration depth should stabilize at some value; $\lim_{t\to\infty} v(t) = 0$ — the body rests, i.e. it does not move anymore.

4.2 A Collision Force Model Based on Position and Velocity

By integrating the equation system consisting of $F(t) = m\ddot{x}(t)$ and $F(t) = -p(x(t))$ for any function p (that satisfies conditions above) we show in [20] that for collision force models that are based on position only, the resulting contact models are purely elastic. How can we vary collision elasticity ?

First, consider what can be done with the original expression for spring force, $F = -Kx$. We know that $F(0) = F(\tau) = 0$. The velocities differ: $\dot{x}(0) \neq \dot{x}(\tau)$. Therefore the velocity (or any expression based on velocity) cannot be added to $-Kx$.

Another operation that can be applied to $-Kx$ is multiplication. Our hypothesis is that a function H is applied to $\dot{x}$ and the result is multiplied by $-Kx$, i.e. $F(t) = -KH(\dot{x})x$.

The equation $F = -1Kx$ produces a correct F for the case where the restitution coefficient is $\varepsilon = 1$. The function H should be equal to 1 in the case of $\varepsilon = 1$. Therefore it is convenient to represent $H(u)$ as $1 + G(u)$. But H is a function of velocity, therefore (if we choose the simplest alternative) $H = 1 + k\dot{x}$. The complete expression is $F(t) = -(1+k\dot{x})Kx$. Since K is a constant and body mass m is a constant it is possible to replace K by Km (for brevity of the solution). The equation is then $F(t) = -(1+k\dot{x})Kmx$. The parameter K is the penalty coefficient. The parameter k is the damping factor coefficient.

The model can be expressed as a system of equations:

$$
\begin{aligned}
F(t) &= \begin{cases} -(1+k\dot{x}(t))Kmx(t), & \text{if } x(t) > 0 \\ 0, & \text{if } x(t) \leq 0 \end{cases} \\
F(t) &= m\ddot{x}(t) \\
\dot{x}(0) &= v_0 \\
x(0) &= 0
\end{aligned}
$$

This equation system can be solved numerically by Mathematica. The values of v_0, K, k have been chosen and $v(t_{end})$ has been computed, where t_{end} is some instant after the collision is finished.

When $t > \tau$ (after the collision) $F = 0$ and therefore $v(t) = v'$.

Experiments show that the resulting velocity v' is almost independent of K. In the rest of the discussion we assume that $K = 1$.

The resulting restitution coefficient ε can be computed from the solution of the equation as $\varepsilon = -v'/v_0$. The values of v_0 and k affect ε, therefore we will write further ε as a function, $\varepsilon(v_0, k)$.

How can we find an appropriate k depending on the velocity v_0 at the start of the collision and the known restitution coefficient ε of the material ?

First we notice that ε resulting from the solution does not change if we multiply v_0 by some number and divide k by the same number.

Now we can check the properties of $\varepsilon(v_0, k)$ in order to extract $k(v_0, \varepsilon)$. It is hard to do without having an explicit equation relating these values. This equation is too complex to be solved symbolically. Therefore numerical experiments have been done. Since $\varepsilon(v_0, k)$=$\varepsilon(v_0 k, 1)$ (using the above result) it is enough to consider the function $\varepsilon(u, 1)$, where $u = v_0 k$.

It appears that $\varepsilon(u, 1) \approx 1/(u + 1)$. Assuming that $\varepsilon(u, 1) = 1/(u + 1)$ the value of k can be extracted. Since $\varepsilon(v_0 k, 1) = 1/(v_0 k + 1)$, we can extract $k = 1/(v_0 \varepsilon(v_0, k)) - 1/\varepsilon(v_0, k)$. This expression with *required* ε can be inserted into the original equation:

$$F(t) = -(1 + (1/(v_0 \varepsilon_{required}) - 1/\varepsilon_{required})\dot{x}(t))Kmx(t)$$

This equation is solved numerically. It appears that now the resulting ε differs from the required one by at most 0.093, which, we believe, is fairly good approximation. This precision does not depend on v_0, m and K.

5 Collision Detection Software

5.1 General Properties of Collision Detection Software

Collision detection is widely used in simulation of multibody systems, in design of virtual environments, and in general 3D graphics. Given coordinates of two or several bodies, collision detection functions determine whether the objects share common points in space, and if they are close enough, determines the distance between them. Each time when collision is examined, the three steps are performed:

Choice of candidates for collision. On this stage the bodies that cannot collide due to simple geometrical relations between them are rejected. This selection is based on axis-aligned bounding boxes of the bodies. A bounding box is usually a good approximation of body position and size. It is easy to compute the bounding box from body geometry (either statically or dynamically). There exist efficient algorithms [7] to check whether any bounding boxes in the 3D space intersect.

Low-level collision detection and distance determination. When bounding boxes of a pair of bodies intersect, the components (features) of the bodies are checked for intersection. It is easy to check whether triangles intersect, and find the distance between them. There exist methods for more complex features, such as spline surfaces. However it can be difficult to define a body using such surfaces in a consistent way. Also, collision detection for complex surfaces might be time consuming.

If the bodies are disjoint, the low-level collision detection algorithms find the distance between the closest features of the objects. They also determine the closest points on these features.

In the bodies intersect, the algorithms determine a pair of features that intersect.

5.2 Using SOLID for Collision Detection

We chose SOLID [3] as collision detection package for our experiments. In this section we discuss using SOLID interface functions (Section 5.3) and how the shortest separation vector (defined in Section 3.2) is computed using SOLID callback function (Section 5.4). The shortest separation vector defines direction and magnitude of penetration depth. The penetration depth is then used for computation of collision force (Section 4).

5.3 Using SOLID interface functions

When SOLID is used the following steps are performed:

- Object shapes are created, defined by list of coordinates of all their vertices. There exist predefined primitive shapes (box, cone, cylinder and sphere). A complex shape can be constructed from one or several so called polytopes. Each polytope is a point, a line, a triangle, a tetrahedron, a convex polygon or a convex polyhedron. Polytopes are always defined by list of coordinates of all their vertices.

- The bodies are created (instantiated) based on the corresponding shape. There can be several bodies of the same shape. The STL[1] is currently used as a file format for geometry description. This format contains triangles with coordinates of their vertices as well as normal vectors. This representation is translated into a collection of SOLID triangles to form a shape.

- The collision detection is performed for each time frame. In order to describe the state of the bodies at each time frame, their rotation, translation and scaling factor are specified. The tool uses frame coherence when determines collision, i.e. it reuses information about the state of the bodies from the previous time frame.

- Collision details in SOLID are obtained using a *callback* function. Each time when collision between a pair of bodies is detected (i.e. the bodies penetrate already), some user-defined callback function is called. When collision is detected, the closest pair of points of the objects at placements from the previous time frame is reported by the callback function. Vector defined by these two points can serve as approximation to the normal to the collision plane. This approach, however leads to some difficulties when the objects move during the collision and continue to be in the penetration for several time steps.

Vector defined by these two points can serve as approximation to the normal to the collision plane.

5.4 Detailed Handling of Collision Response

Assume that the collision between the objects A and B is detected. In order to find the shortest separation vector $\vec{c}$ (as defined in Section 3.2) we use a search algorithm.

The algorithm is implemented as two nested loops. In the external loop the length for $\vec{c}$ is chosen. Initially a very small estimated c^0 is chosen[1] so that it is negligible in comparison with a typical penetration depth. The value of c^i is incremented at each loop iteration, so that $c^i = (1+\kappa)c^{i-1}$. The value $0.1 < \kappa < 0.4$ should be small if high accuracy is required.

There exist $3 \times 3 \times 3 = 27$ possible direction vectors (x_m, y_m, z_m), where each of the components can be -1, 0 and 1. The vector (0,0,0) is excluded from this set. All the direction vectors $c_0^i, ..., c_{26}^i$ are normalized and then multiplied by the current length, i.e. c_i. As a result, the vectors $\vec{c_m^i}, 1 \leq m \leq 26$ are obtained.

The figure $B(\vec{c_m^i})$ is B displaced by $\vec{c_m^i}$. $P_{A,B(\vec{c_m^i})}$ and $P_{B(\vec{c_m^i}),A}$ are the closest points between the bodies A and $B(\vec{c_m^i})$. The vector between the points is referred as $\vec{d_m^i}$.

If there is at least one vector that makes the bodies disjoint in the set of vectors $\vec{c_m^i}, 1 \leq m \leq 26$, the external loop of the algorithm stops. The internal loop should find the most appropriate vector if several are available. We know that all $\vec{c_m^i}$ that made the bodies disjoint are slightly longer than necessary. All these vectors can be shortened by some distance. This distance is the length of projection of the vector $\vec{d_m^i}$ on $\vec{c_m^i}$. The length of projection is $p^m = \vec{c_m^i}\vec{d_m^i}/|\vec{c_m^i}|$, and the m giving the largest projection should be chosen.

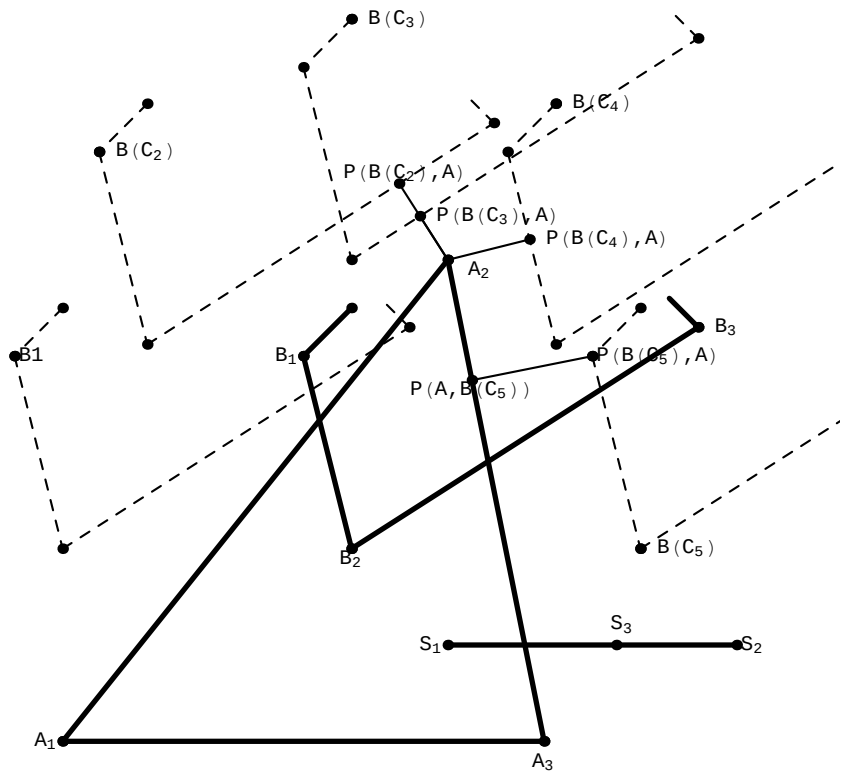

Figure 7: Computation of the shortest separation vector

The inner loop of the algorithm can be demonstrated graphically in the two-dimensional case as in Figure 7. It shows the body A with vertices A_1, A_2, A_3. The body B has vertices B_1, B_2, B_3 as well as some other vertices that are currently ignored. The current separation vector length c^i (in the following we skip the index i) is chosen as $|S_1S_2|$, i.e.

[1]Here and in the rest of this section the superscript should be regarded as an index, not as an exponent representing the *power* operation.

the length of the vector from S_1 to S_2. There are eight vectors $\vec{c_1}, ..., \vec{c_8}$ of the same length. In this demonstration just five of them will be discussed ($\vec{c_1}, ..., \vec{c_5}$). Other vectors are currently ignored. These vectors ($\vec{c_1}, ..., \vec{c_5}$) are not shown in the picture. Instead, the results of the displacements of the body B according to these vectors are shown: $B(c_1), ..., B(c_5)$. These displaced bodies are shown with dashed lines. The collision detection algorithm finds that $B(c_1)$ intersects A, but the other bodies $B(c_2), B(c_3), B(c_4), B(c_5)$ are disjoint with A. For each disjoint object pair the collision detection algorithm finds the closest points:

- For $B(c_2)$ and A the closest points are $P(B(c_2), A)$ and A_2 (vector $\vec{d_2}$).
- For $B(c_3)$ and A the closest points are $P(B(c_3), A)$ and A_2 (vector $\vec{d_3}$).
- For $B(c_4)$ and A the closest points are $P(B(c_4), A)$ and A_2 (vector $\vec{d_4}$).
- For $B(c_5)$ and A the closest points are $P(B(c_5), A)$ and $P(A, B(c_5))$ (vector $\vec{d_5}$).

The vectors $\vec{d_i}$ are shown in the picture as thin solid lines. The longest of the vectors $\vec{d_i}$ is $\vec{d_5}$. Therefore the direction of the separation vector is $\vec{c_5} = \vec{S_1S_2}$. The length of the actual shortest separation vector is shorter than $|S_1S_2|$. The projection of $\vec{d_5}$ on $\vec{c_5}$ is subtracted from $|S_1S_2|$. The result is $|S_1S_3|$. This is the penetration depth used in the computation of collision force.

6 Applications Using the Force-Based Model

This section presents examples of models using the force-based collision model in Modelica.

The first examples use the MBS library but do not use any collision detection package. The second example uses the MBS library, the collision detection tool SOLID, as well as some gluing classes.

6.1 Pendulum Collision with an Obstacle

This pendulum is the same mechanical construct as the one described in Section 2.5, Figure 2. For finding the magnitude of the force it uses the equation derived in Section 4.2. It is possible to choose the penalty coefficient K, which is stored in the variable `penalty` in the code below, arbitrarily large:

— If $K < 10^3$ the collision appears too weak and the penetration is non-realistically large.

— If $10^3 < K < 10^5$ a soft collision occurs.

— If $10^5 < K < 10^{15}$ a hard collision occurs.

— If $K > 10^{15}$ this makes the integrators unstable.

The desired restitution coefficient (variable `e`) almost matches the actual restitution (variable `eres`). The computation error was $(e - eres)/eres < 0.05$.

```
model Pendulum
  Inertial i;
  RevoluteS R1(n={0,0,1},q(start=1));
  BodyBase M1 (m=50,rCM={0,0.5,0});
  Bar B (r={0,1,0});
  ExtForce EF1  "collision force";
  parameter Real penalty=50000;
  parameter Real e=0.5 "desired restitution, 0<e<1";
  parameter Real obstacle_x = 0.5 " x coordinate of obstacle";
  Real depth "penetration depth";
  Real depvel "penetration velocity";
  Real k "velocity coefficient for penalty";
  Real force_magnitude "magnitude of collision force";
  output  Real x "x of pendulum end";
  output  Real y "y of pendulum end";
  output Real eres "resulting restitution coef" ;
  output Real v0(start=9.999) "velocity at start of collision.";
  // Need start value just to avoid zerodivision
  output Real vout "velocity at the end of collision";
equation
  connect(i.b, R1.a);
  connect(R1.b, M1.a);
  connect(R1.b, B.a);
  connect(B.b, EF1.b);
  x=B.r0b[1];
  y=B.r0b[2];
  depth = B.r0b[1]-obstacle_x; // The obstacle equation.
  //  B.r0b[1] is the x coordinate of the pendulum end.
  depvel= -B.vb[1];
  when (depth>0) then v0=depvel;  end when;
  when (depth<0) then vout=depvel; end when;
  eres=vout/v0;
  k=(1-e)/(e*v0); // approximation derived for typical k
  force_magnitude=
   if (depth>0) then (1 + k*depvel)*penalty*depth
   else 0;
  EF1.fb={force_magnitude,0,0};
end Pendulum;
```

6.2 Interfacing to a to Collision Detection Package

The Modelica model interfaces with a collision detection package through function calls. The mechanical simulation based on MBS sends the current positions of all bodies to the collision package (CP) and receives vectors of forces and torques that occur due to collisions.

Since the number of bodies can be large, and the data should be send to the collision package at once, and received at once, all the variables needed are packed into the two-dimensional arrays `collisionInput` and `collisionOutput`.

The function `doCollide` accepts the array `collisionInput` as an argument and returns the array `collisionOutput` as a result.

The example below demonstrates a purely elastic collision, but can also be extended for different restitution coefficient values. Also, the example demonstrates free flying bodies, but it can be extended for arbitrary constructions that use MBS, consisting of bodies, joints and additional external forces and torques.

```
function doCollide "Function to be called to obtain the collision force"
  input Integer bodies  "number of bodies";
  input Real collisionInput[bodies,27] "position and other data";
  output Real collisionOutput[bodies,6] "forces and torques";
external
end doCollide;
```

```
model BodyME "Body that participates in collision and can be visualized"
  extends MbsOneCutA;
  parameter Real id;
  parameter Real r[3]={0, 0, 0} "Vector from cut A to center of mass [m]";
  parameter Real mass=0 "Mass of bar [kg]";
  parameter Real I11=0 "(1,1) element of inertia tensor [kgm^2]";
  // other inertial tensor elements are skipped for brevity
  parameter Real r0[3]={0, 0, 0} "Origin of visual object.";
  parameter Real nx[3]={1, 0, 0} "Vector in x direction.";
  parameter Real ny[3]={0, 1, 0} "Vector in y direction.";
  parameter Real Material[4]={1, 0, 0, 0.5} "Material properties";
  parameter Real parmStlIndex=0 "Index of STL file";
  parameter Real noCollision=0 "0 if the body participates in collisions";
  Real collisionInput[27] "position and other data";
  Real collisionOutput[6] "force and torque";
  output Real StlIndex "Index of STL file";
  BodyME2 ME2(r=r,mass=mass,
    II=[I11, I12, I13; I12, I22, I23; I13, I23, I33])
    "A help class - wrapper for physical body model";
  MbsOneCutB collResp;
  VisualMbsObject B(r0=r0,nx=nx,ny=ny,Material=Material);
equation
  connect(a, ME2.a);
  connect(a, collResp.b);
  connect(a, B.a);
  StlIndex = parmStlIndex;
  B.fa = {0, 0, 0};
  B.ta = {0, 0, 0};
  collisionInput=cat(1, {id, B.shape, B.Form, B.extra, parmStlIndex, noCollision},
      B.size, B.rxvisobj, B.ryvisobj, B.rvisobj, Sa[1,:], Sa[2,:], Sa[3,:]);
  collisionOutput=cat(1, collResp.fb, collResp.tb);
end BodyME;
```

```
model minibox "An MBS construction containing a single body"
// This construction can be extended and contain
// more complex model consisting of bodies and joints
  parameter Real nx[3]={1, 0, 0};
  parameter Real ny[3]={0, 1, 0};
  parameter Real id;
  Real ac[24];  Real th[6];
  MbsCutA a(across=ac, through=th);
  SubInertial I(nx=nx, ny=ny);
  BodyME box_1(r={0, 0, 0},I11=0.13,I22=0.13,I33=0.13,I12=0,I23=0,I13=0,mass=1,
    r0={0, 0, 0},nx={1, 0, 0},ny={0, 1, 0},Material={0.8, 1.0, 0.8, 1.0},
    parmStlIndex=20,id=id);
equation
  connect(a, I.a);
  connect(I.b, box_1.a);
end minibox;
```

```
model IntegratedBox "A glue class representing a free flying element bx"
  extends MbsOneCutA;
  parameter Real id;
  minibox bx(id=id);
  FreeCardan2S free;
equation
  connect(a, free.a);
  connect(free.b, bx.a);
end IntegratedBox;
```

```
model world
  IntegratedBox ib1(id=1) "free flying box no. 1";
  IntegratedBox ib2(id=2) "free flying box no. 2";
  IntegratedBox ib3(id=3) "free flying box no. 3";
  IntegratedBox ib4(id=4) "free flying box no. 4";
  Inertial I(g=0)  "The roor of the MBS tree";
  parameter Integer bodies=4;
  Real wholeInput [bodies,27];
  Real wholeOutput [bodies,6];
equation
  connect(I.b, ib1.a) ;
  connect(I.b, ib2.a) ;
  connect(I.b, ib3.a) ;
  connect(I.b, ib4.a) ;
  ib1.bx.box_1.collisionInput=wholeInput[1,:];
  ib2.bx.box_1.collisionInput=wholeInput[2,:];
  ib3.bx.box_1.collisionInput=wholeInput[3,:];
  ib4.bx.box_1.collisionInput=wholeInput[4,:];
  ib1.bx.box_1.collisionOutput=wholeOutput[1,:];
  ib2.bx.box_1.collisionOutput=wholeOutput[2,:];
  ib3.bx.box_1.collisionOutput=wholeOutput[3,:];
  ib4.bx.box_1.collisionOutput=wholeOutput[4,:];
  wholeOutput=doCollide(bodies,time,wholeInput);
end world;
```

Figure 8 demonstrates dynamic visualization of collision between four free flying cubes. The lines represent the trajectories before and after the collision. Initially two boxes move towards the zero point, and all four bodies are separated after the collision occurs.

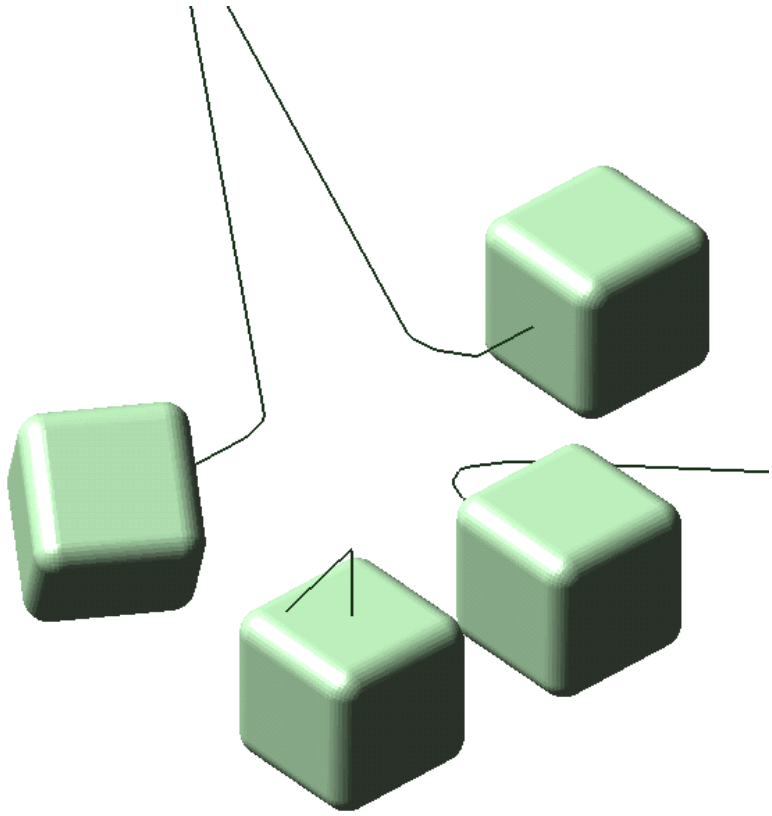

Figure 8: Dynamic visualization of collision between four free flying cubes. The lines represent the trajectories before and after the collision.

7 Conclusion

In this report we attempt to find ways to connect MBS-based models written in Modelica with collision detection and response routines. We also derive the equations and the coefficients that can be used as collision response functions.

As the first step in this direction different collision response models were identified and experiments with Modelica simulations were performed. We demonstrate that an impulse based approach for non-trivial models requires a new library that includes equations for propagation of impulses into joints. This new library can become a topic for new research and experimentation. The force-based (penalty) approach can work with any MBS models, however the performance and stability should be further studied.

The experiments describe a collision of a point mass and a static obstacle. Therefore we do not prove that F is appropriate for collisions in all cases. However it can serve as an approximation of the force in the broader set of cases.

Our currently used force model is continuous, but actually not differentiable at the time of the start and the end of collision. This may cause problems for the ODE solvers because these require that all equations are differentiable. In future models with differentiable force should be derived and used for simulation.

8 Acknowledgments

The Modelica definition has been developed by the Eurosim Technical Committee 1 (Modelica Design Group)[13] under the leadership of Hilding Elmqvist (Dynasim AB, Lund, Sweden) and Martin Otter (DLR, Germany). The Multibody Simulation Library has been developed by Martin Otter. The author thanks Dag Fritzson for useful discussions about the ODE.

References

[1] 3D Systems, *Stereo Lithography Interface Specification*, 3D Systems Inc., Valencia, CA 91355. Available via http://www.vr.clemson.edu/credo/rp.html.

[2] David Baraff, *Dynamic Simulation of Non-Penetrating Rigid Bodies*, Ph.D. thesis, Department of Computer Science, Cornell University, 1992, Available via http://www.cs.cmu.edu/ baraff/

[3] Gino van den Bergen, *SOLID, Software Library for Interference Detection*, http://www.win.tue.nl/cs/tt/gino/solid

[4] Gino van den Bergen, personal communication, November, 1999.

[5] Dag Fritzson, Peter Fritzson, Patrik Nordling, Tommy Persson, Rolling Bearing Simulation on MIMD Computers, *International Journal of Supercomp. Appl. and High Performance Computing*, 11(4), 1997.

[6] Lars Johansson, Anders Klarbring, Study of Frictional Impact Using a Non-smooth Equations Solver, to appear in *Journal of Applied Mechanics*, 2000.

[7] Ming Lin and Dinesh Manocha, *Collision Detection Packages RAPID, PQP, V-COLLIDE, I-COLLIDE*, http://www.cs.unc.edu/ geom/collision_code.html

[8] Ming Lin and Stefan Gottschalk. Collision Detection between Geometric Models: A Survey. *In Proceedings of IMA Conference on Mathematics of Surfaces 1998.* Available via http://www.cs.unc.edu/ dm/collision.html

[9] Brian Mirtich, *V-Clip Collision Detection Library* , available via http://www.merl.com/projects/vclip/

[10] Brian Mirtich. Impulse-based Simulation of Rigid Bodies, In *Symposium on Interactive 3D Graphics*, ACM Press, 1995.

[11] Brian Mirtich. Hybrid Simulation: Combining Constraints and Impulses, in *Proceedings of the 1st Workshop on Simulation and Interaction in Virtual Environments*, ACM Press, July 1995, Available via http://www.merl.com/people/mirtich/

[12] Brian Mirtich, *Impulse-based Dynamic Simulation of Rigid Body Systems*, Ph.D. thesis, University of California, Berkeley, December 1996.

[13] Modelica Design Group, *Modelica WWW site*, http://www.modelica.org

[14] Martin Otter, Hilding Elmqvist and François E. Cellier, Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola, *Nonlinear Dynamics*, 9:91-112, 1996, Kluwer Academic Publishers.

[15] Jianping Pang, Newton's Method for B-Differentiable Equations, *Mathematics of Operation Reserach*, Vol. 15, pp. 311-341.

[16] Friedrich Pfeiffer and Christoph Glocker, *Multibody Dynamics With Unualteral contacts*, Wiley Series in Nonlinear Science, 1996.

[17] Andrew Witkin and David Baraff, Physically Based Modeling: Principles and Practice(Online Siggraph '97 Course notes). Available as http://www.cs.cmu.edu/ baraff/sigcourse.

[18] Wolfram Research, *Mathematica 4.0*, Wolfram Research, 1999.

[19] Peng Zhang, *Physically Realistic Simulation of Rigid Bodies*, Thesis, Department of Computer Science, Tulane University, 1996, Available via http://www.eecs.tulane.edu/www/Zhang/

[20] Vadim Engelson. Integration of Collision Detection with the Multibody System Library in Modelica. In *Vadim Engelson. Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing*, Dissertation No 627. Department of Computer and Information Science, Linkping University, May 2000.