Contents

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:

  - 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 contains equations that take into account the hydrodynamic properties of the lubricant that occurs between bodies. Contact forces appear already when bodies have some distance from each other. These forces are caused by compression and decompression of the lubricant.

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 point on the mesh. 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.

However, many simulation applications do not require extreme accuracy. Their goal is to demonstrate qualitative characteristics of the behavior, not numerical ones. Therefore in many cases additional assumptions are taken into account, that may reduce accuracy, but provide high simulation speed. Often different assumptions lead to different computation methods but to the same (or nearly the same) computation results. In that case this it makes no major difference for the application what assumptions and methods were used.

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. As a result, the mechanical parts may penetrate each other freely during simulation. This significantly decrease the realism of mechanical simulations. The simulation results might even be wrong.

The goal of this report 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. Therefore the following basic components of collision simulation models have been identified:

- Mechanical models use certain classes describing physical bodies. Mechanical models including collision response force should extend these classes to describe colliding physical bodies. Such bodies should be distinguished from non-colliding bodies.

- There should be a routine that can detect collisions in the simulated mechanisms and can return detailed information regarding collision parameters, such as collision points and their velocities.

- A special routine should calculate the collision response from collision parameters. The collision response is calculated as force and torque applied to the point of collision at the colliding body (or bodies). Alternately, this routine might change the velocity of the bodies or some other model variables. This component seems to us the most problematic and controversial, since there exist many approaches to collision response computation which require different input information and may produce quite different numerical results.

Each of these three components should have an interface that allows replacing its implementation without doing major redesign of the other components. In particular, the collision detection package as well as the force computation functions should be easily replaceable when necessary.

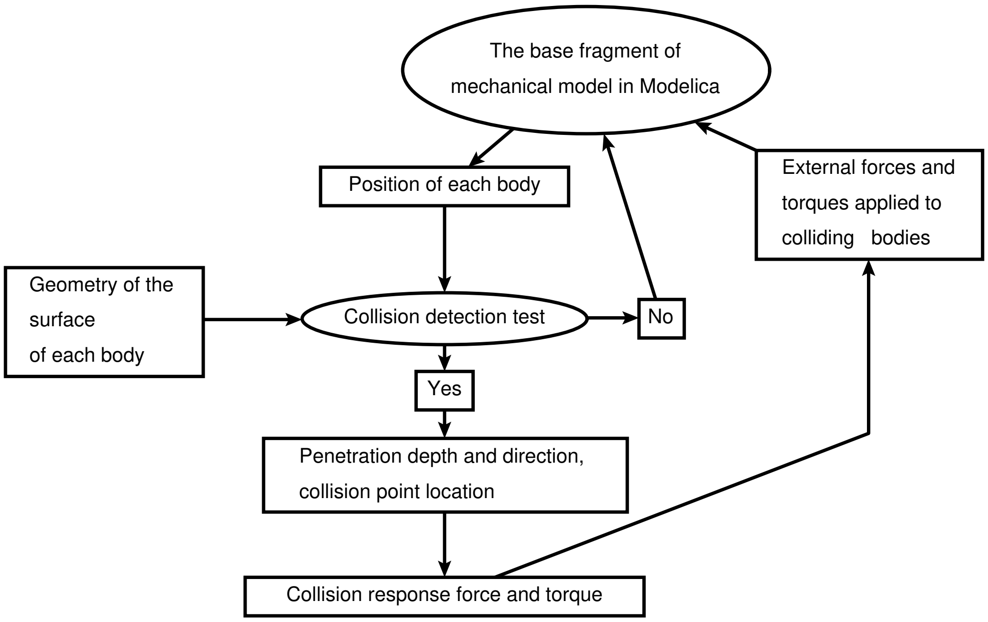

Figure 1: The overall architecture of mechanical simulation models for computing collision detection and response.

The overall architecture of mechanical simulation models with collision detection and response is presented in Figure 1.

This report reflects ongoing research and development. Therefore some fragments of work related to this report will be done in the future. The structure of the report and the relation between its parts is shown in Figure 2.

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. 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 6). Section 7 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.

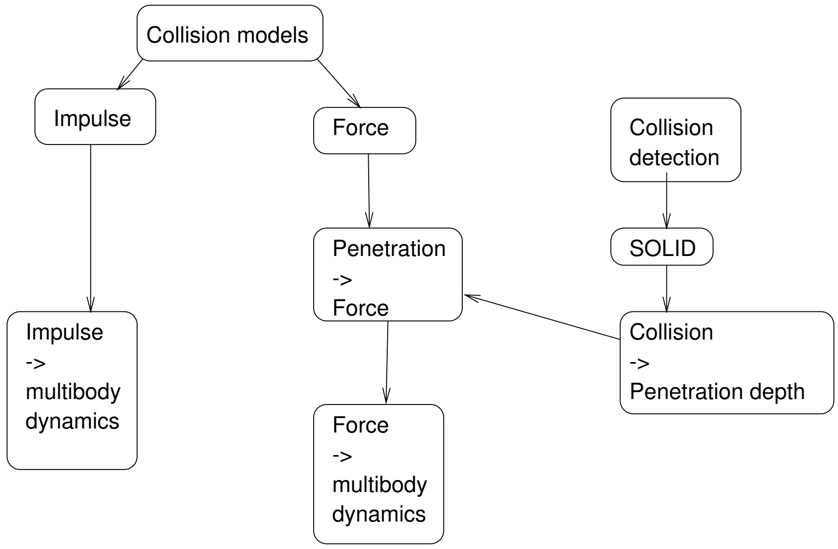

Figure 2: The structure of the report.

2.1 Impulse and Velocity Equations

The following notation is used in the equations below:

- m_a, m_b – masses of bodies A and B;
- v_a, v_b – velocity vectors of bodies A and B before the collision; v_a', v_b' – the velocity vectors of the bodies after the collision;

The movement of a body is described as movement of its center of mass (linear velocity) and rotation of the body around its center of mass (angular velocity). The linear momentum is the mass multiplied by the linear velocity. The total linear momentum of a system consisting of bodies A and B is preserved if there is no external impact. This is also true for colliding point masses and it can be expressed by the equation

$$m_a v_a + m_b v_b = m_a v_a' + m_b v_b'.$$

In order to describe the collision further, we need to build a collision plane and the vector $\vec{n}$ which is normal to the collision plane.

The change of the projections of the velocities of point masses on $\vec{n}$ can be expressed using the restitution coefficient ε:

$$\varepsilon = \frac{v_a' - v_b'}{v_b - v_a}.$$

In the case of absolutely elastic collision $\varepsilon = 1$. In the case of absolutely non-elastic (completely damped) collision $\varepsilon = 0$. Actual physical values of ε always belong to the interval $0 < \varepsilon < 1$. Experimental measurements show that the restitution coefficient ε depends mainly on material properties of bodies A and B.

The object B can be rigidly attached to the inertial system. It can be a static obstacle, such as a ground plane or a wall attached to the ground. In this case ($m_b \to \infty, v_b = 0$), the equations for point masses are simplified, and they yield the following equation:

$$v_a' = -\varepsilon v_a.$$

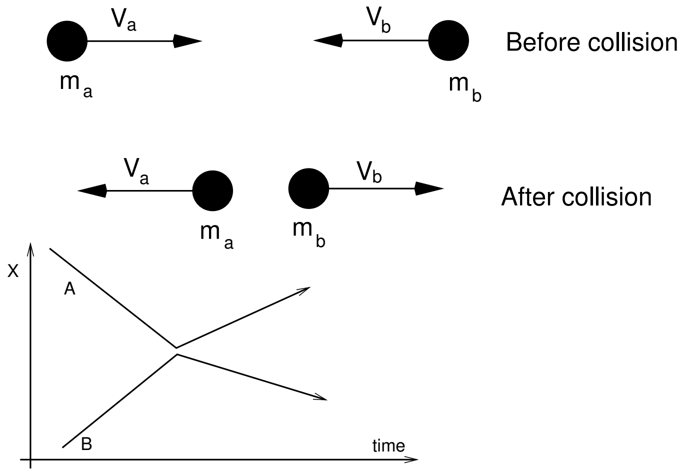

Figure 3: One-dimensional collision of two point masses A and B.

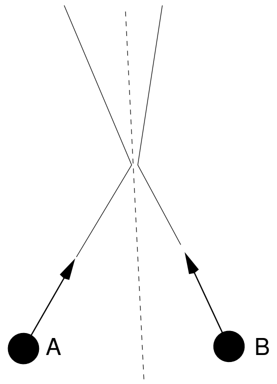

Figure 4: Two-dimensional collision of two point masses A and B. The collision plane is shown as a dashed line.

The simplest case of one-dimensional collision is shown in Figure 3. The cases of two- or three-dimensional collision are similar. The trajectory of collision of point masses in 2D is shown in Figure 4.

However, when bodies collide (see Figure 5), the collision points differ from the center of mass, which should be taken into account.

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.

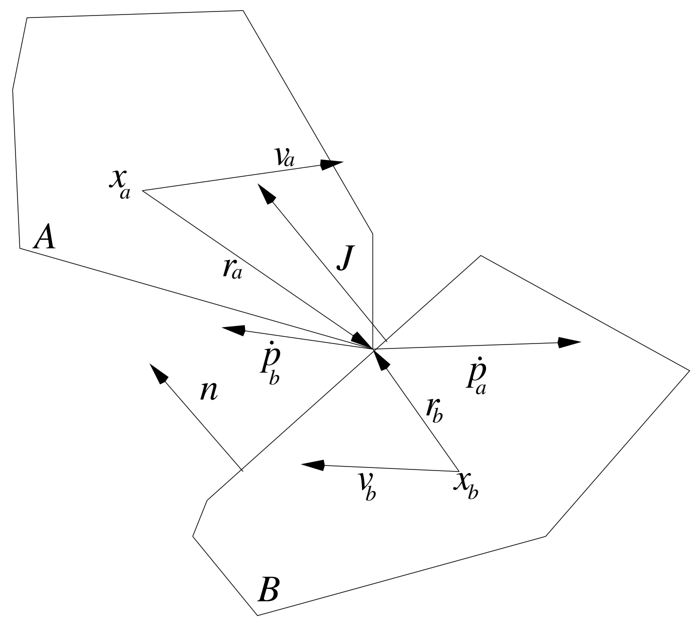

Figure 5: Two dimensional collision of bodies A and B.

- 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.

A derivation of the equations we use here is available in [17]. Assuming that p_a and p_b are points of collisions, x_a and x_b are positions of the centers of mass, and ω is the angular velocity of the body, the velocities of the points can be found as

$$\dot{p_a} = v_a + \omega_a \times (p_a - x_a) \text{ and } \dot{p_b} = v_b + \omega_b \times (p_b - x_b),$$

the relative velocity is defined as $v_{rel} = \vec{n} \cdot (\dot{p_a} - \dot{p_b})$,
and the restitution coefficient is expressed as $\varepsilon = -v'_{rel}/v_{rel}$.
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 I is inertia tensor.

The collision impulse J is the change of momentum. Knowing the velocities of the bodies before the collision, the point of collision, and the restitution coefficient we can find the velocities after the collision. These velocities can for each body be expressed from

$$(v'_a - v_a)m_a = J \text{ and } (v'_b - v_b)m_b = -J$$

$$(\omega'_a - \omega_a)I_a = r_a \times J \text{ and } (\omega'_b - \omega_b)I_b = r_b \times J$$

where $r_a = p_a - x_a$ is the position of the collision point relative to the center of mass of body A.

In order to find J some algebraic manipulations are needed. If we ignore the angular velocities J can be expressed as

$$J = -(1 + \varepsilon)v_{rel}/(m_a^{-1} + m_b^{-1})$$

But if we care about correct angular velocities, the expression becomes more complex [17]:

$$J = \frac{-(1+\varepsilon)v_{rel}}{m_a^{-1} + m_b^{-1} + \vec{n} \cdot (I_a^{-1}(r_a \times \vec{n})) \times r_a + \vec{n} \cdot (I_b^{-1}(r_b \times \vec{n})) \times r_b}.$$

2.2 Simulation Using an Impulse-Based Model

In order to use the impulse-based approach, the computational model should assume that collision takes place if the distance between the closest features of two bodies is less than some threshold T.

Some systems [19] are constructed so that

- 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 time step (Δt) and estimating the maximum speed (v_{max}), so that $T > v_{max}\Delta t$.

Nevertheless, if penetration occurs (because of erroneous estimations) 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 to two, and the 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]).

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.

2.3 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.

In a simple mechanical model, described in Modelica, it is possible to change the velocity variable `v` when some condition triggers an event.

The `when` statement may contain a call to collision detection routine. When collision happens, Modelica event occurs. The angular and linear velocity of bodies can be measured and instantly changed at specific points in time (events) during the simulation. In this case the integration stops, the initial conditions for velocity are changed, but all other variables keep their values as they were before the collision. The continuation of the simulation is completely consistent from the point of view of the MBS library.

We illustrate this possibility with two examples: a bouncing ball and a simplified pendulum.

2.4 Bouncing Ball Example

The bouncing ball example illustrates a model constructed using explicit equations defining 1D movement and collision.

The model below describes a bouncing ball falling down from the height of 10 meters. When the ball reaches the ground, the condition `x<0` becomes true, an event is triggered, the exact instant of this event is automatically found, and the variable `v` gets the new value `-ε*v`. The ball rises from the ground and falls down again. The plot of the height `x` is shown in Figure 6.

```
model BouncingBall "bouncing ball, 1-D model"
  Real v "velocity";
  Real x (start=10) "height";
```

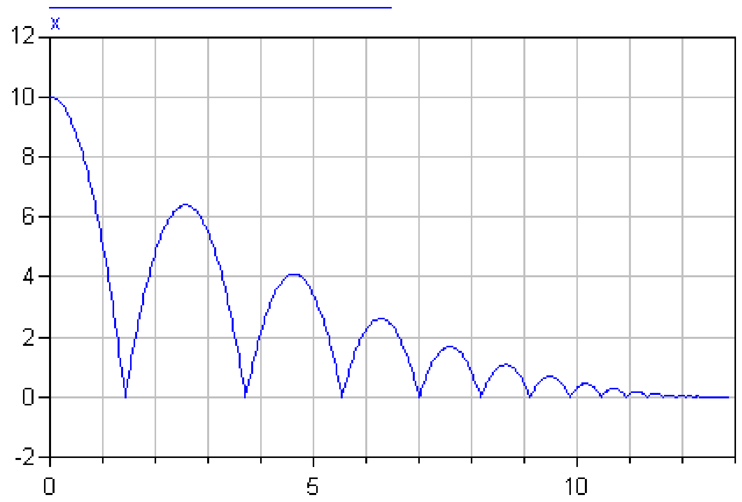

Figure 6: The height of the bouncing ball computed using the impulse model.

```
  parameter Real g=9.8;
  parameter Real eps=0.8 "restitution" ;
equation
  der(v)=-9.8;  // permanent acceleration
  der(x)=v;      // velocity
  when (x<0)  then // when ball crosses the ground level
    reinit(v,-eps*v); // the velocity is re-initialized
  end when;
end  bouncingBall;
```

2.5 Colliding Pendulum Example

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

The pendulum example (see Figure 7) illustrates the use of the MBS library for a model with impulse-based collision response. The state variable q in this model is the angle 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 the angular velocity of the change of this state variable. This assumption is wrong in the more complicated cases.

The trajectory of the pendulum end is depicted in Figure 8. The change of the angle of rotation $R1.q$ is shown in Figure 9.

```
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;
```

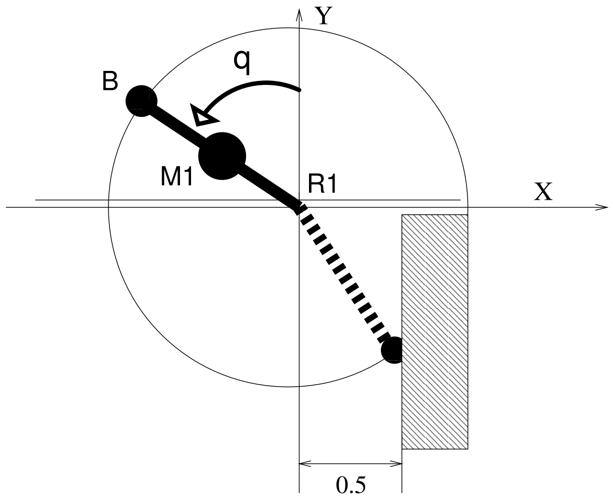

Figure 7: Simple pendulum colliding with an obstacle at point $Obstacle_x = 0.5$.

.

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 ($v \to g(v)$) according to the

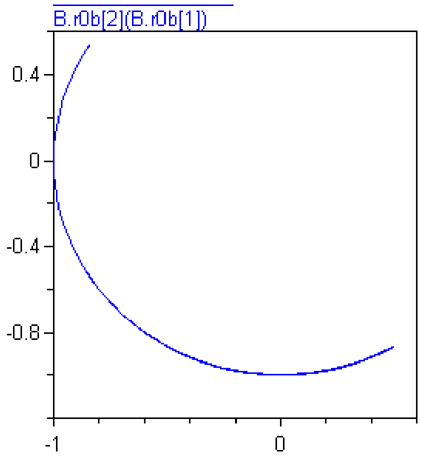

Figure 8: Trajectory of the endpoint of a simple pendulum colliding with an obstacle at point $Obstacle_x = 0.5$

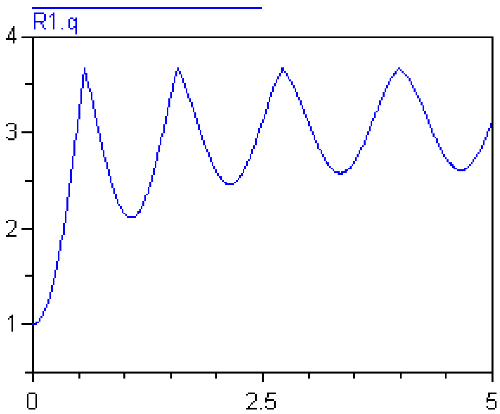

Figure 9: Angle of rotation of a simple pendulum colliding with an obstacle. The restitution coefficient is $\varepsilon = 0.5$. The collision happens when the angle reaches `R1.q`$=\pi + \arcsin(Obstacle_x) \approx 3.66$.

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 10).

We can use kinematic loops, i.e. loops in the structure of the multibody model. In particular, the double pendulum model with 2 revolute joints (containing 2 state variables) (Figure 10(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 10(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 10(b) is given below. The connection diagram of this model is shown in Figure 11.

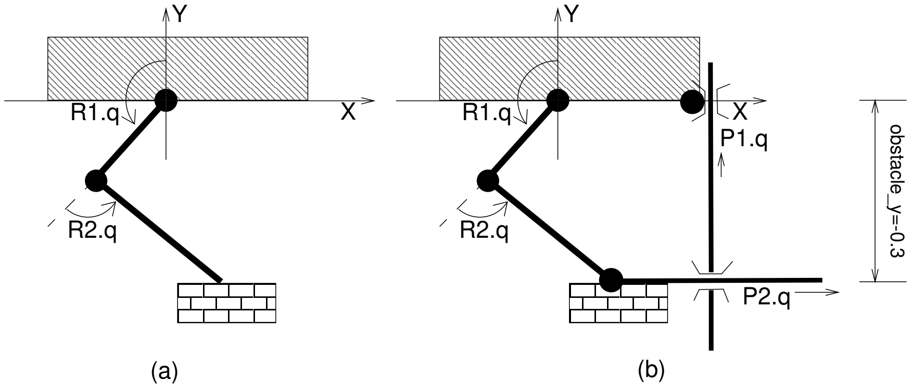

Figure 10: Double pendulum (a) combined with additional joints to form a kinematic loop (b).

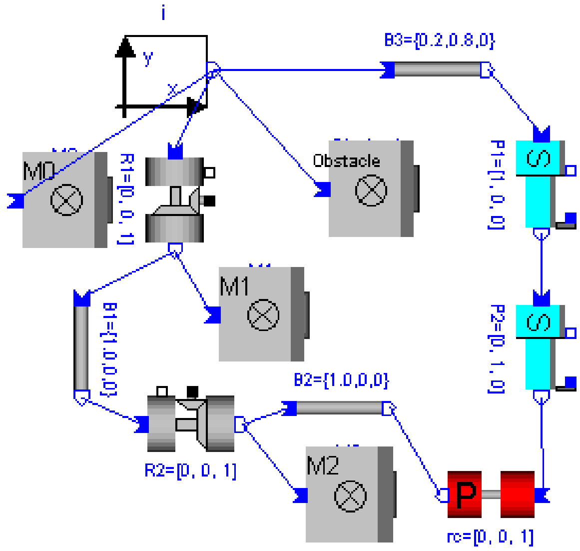

Figure 11: Double pendulum model with kinematic loop, graphical presentation.

```
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 special kinematic loop, which leads to a huge number of additional objects.
- It is difficult to find correct components for the loop, since all potential degrees of freedom should be taken into account and analyzed.
- 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 seemingly be 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, free floating objects, and multibodies (objects consisting of several 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 other bodies in the system is negligible.

For instance, a system of 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.

- 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.

These rules regarding the computation of collision force contain several terms that will be further discussed, clarified and refined.

3.1 Penetration

We assume that A and B are defined using a description of their surfaces, which separate their inner volume from the outer world. Objects with this property are also called also orientable manifolds. In practice such surfaces are described as a set of connected triangles (or arbitrary polygons). In this case the bodies are called *polytopes*. More complex surfaces can be described as spline surfaces, e.g. NURBS.

Formally, two objects A and B penetrate each other if the volume of their intersection is larger than zero. This is equivalent to the statement that at least one point on the surface of body A occurs within body B. However, in practice, the volumes are not computed. The geometrical description of bodies A and B is normally considered by collision detection software as a so-called ”triangle soup”, i.e. a set of arbitrarily placed triangles in 3D. The bodies A and B penetrate if some of their triangles intersect[1]. In some cases the collision detection software assumes that the considered bodies are convex ones (i.e. any line between the points belonging to the body belongs to the body).

3.2 The Bodies and Their Shells

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. When more accurate results are necessary, and a more complicated computation model are used, bodies are subdivided into small fragments and the penetration between these fragments is considered.

The collision methods normally assume that the depth of penetration is negligible in comparison with the size of the body.

We can also consider a body A_{shell} which is A augmented with all the points at a distance less than Δ_{shell} from A (see Figure 12). Impulse based

[1]It might happen that the bodies A and B just touch each other. In this case the volume of intersection is zero and the collision force is zero.

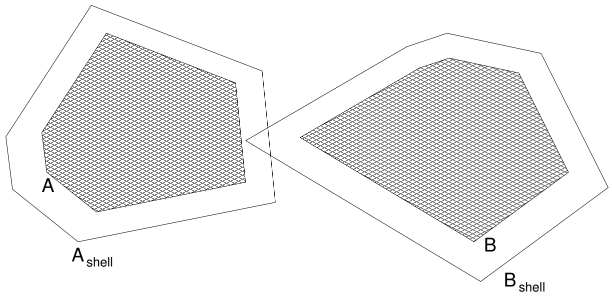

Figure 12: Penetrating shells of objects A and B.

systems take the geometries of A and B into account and consider that collision takes place if A_{shell} and B_{shell} intersect. The bodies A and B are never allowed to intersect in the models using this approach. Here the value Δ_{shell} is used as a threshold.

In our force-based model, the geometries of A_{shell} and B_{shell} can be taken into account instead of A and B. Collision occurs if A_{shell} and B_{shell} intersect. The forces generated due to the collision response are so large that the time when A and B intersect is negligible.

3.3 The Point of Contact

The point of contact can be defined in many different ways. The naive definition states that this is the point where two bodies touch each other the very first time (at the first instant of collision). Such a definition is only good for very short-duration contacts. If the bodies have longer contacts (i.e. 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.

Collision detection software packages usually do not find the point of the first contact. The collision detection functions 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 duration contact between the bodies.

A good integral (average) point of collision would be the center of volume of the intersection between the bodies A and B. The geometry of an intersection can, however, be quite complex. Finding the volume is a computationally intensive task. The volume cannot, in general, be found at all if the body geometry is stored by the collision detection software just as ”triangle soups” (since the soup is composed of a set of triangles with zero thickness, it has no volume, and intersection of two soups has no volume.) Finding the center of the volume does not help in finding the direction of the collision force.

3.4 Direction of Force

The direction of the collision force can be determined in several different ways. For a short contact it would be natural to define the direction of the collision force as opposite to the velocity of the contact point, i.e. the point of the first touch between the bodies. However, for reasons that are similar to the above mentioned approaches, this does not work well for long contacts.

For accurate determination of force direction, a mesh based on colliding surfaces is constructed, and a normal vector to the surface in each mesh point is used as local force direction. The resulting force direction is then found by integrating the vectors of all the forces acting on the contact surface.

3.5 Penetration Prevention Model

In our approach the collision force is computed using the assumption that it prevents further penetration of the bodies. The computations during the contact are not dependent on the duration of the contact. The method can handle both short collision times (the collision force causes termination of collision soon after it starts) and long collision times (the collision force compensates some other forces and therefore the contact can take an indefinitely long time).

If the objects A and B collide, body A forces body B out (pushes it away) from its interior. We assume that the bodies behave as elastic and isotropic media. Therefore the direction of this force should be such that B is pushed out in the shortest possible way. Therefore the *shortest separation vector* should be chosen. The shortest separation displacement is the shortest of all displacements of body B that if applied to B, will cause the bodies to separate.

We formulate the definition in mathematical notation:

Assume that S is a set of all vectors in R^3. If $\vec{m} \in S$, we define $B(\vec{m})$ as the body B being displaced according to the vector $\vec{m}$. The distance between the closest points of bodies A and B is $d(A, B)$; these closest points are $P_{AB} \in A$ and $P_{BA} \in B$. The set of translation vectors that can separate B from A is $S_{AB} = \{\vec{m} \in S \mid A \cap B(\vec{m}) = \emptyset\}$. The shortest separation vector $\vec{c} \in S_{AB}$ is the shortest vector in the set S_{AB}.

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. The ways to evaluate the magnitude of F from $\vec{c}$ are discussed in the next section.

Our method works best if a single vertex (with some surrounding surface fragments) of body B penetrates in the middle of a face of body A. The forces are directed so that this vertex P_{BA} ($P_{BA} \in B$) is pushed away by the force directed as a normal to the face (see Figure 13). Note that the 2D examples are given just for illustration; the actual software works with 3D models.

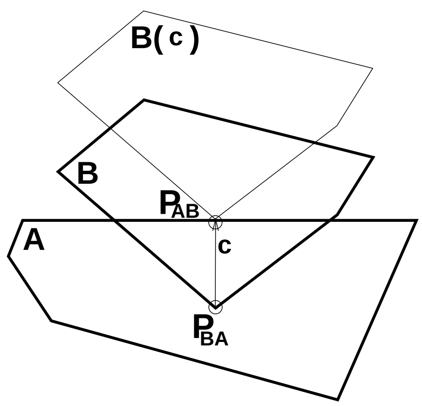

Figure 13: Collision geometry in the simple case: a single vertex of B is located within A.

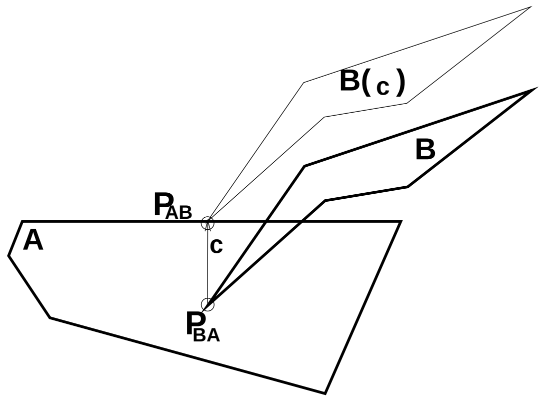

Figure 14: Collision geometry in the case of a sharp angle.

The approach works well even if the center of mass of B is far away from P_{BA} (see Figure 14). In this figure body $B(c)$ is oriented the same way as body B. Note that due to the force being applied to point P_{BA} body B will actually rotate during the collision.

In the case of vertex–to–vertex collisions (see Figure 15) the algorithm computes the shortest direction, c (the vertical direction in this case) and ignores the longer one (c_1). Such collisions, however, are rather rare, especially if the penetration depth is much smaller than the size of the bodies.

In the case of two–vertices–to–plane collision (a box resting on the ground) the force is applied to the deeper vertex of the box (see Figure 16). This force raises the box up, and another vertex of the box becomes the deepest one. After some interaction the box and the platform separate, or the box will be lying on the platform. The collision forces compensate for the force of gravity.

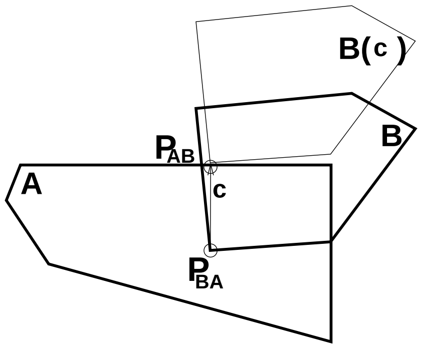

Figure 15: Collision geometry in the case of several vertices penetrating.

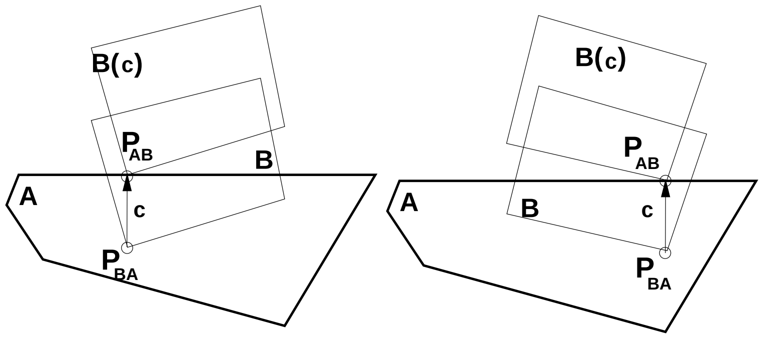

Figure 16: Collision geometry in the case of collision of two vertices and a horizontal platform.

4 Computation of the Force from Penetration Measurement

The source of collision forces is in the following physical phenomenon. Initially, the bodies are compressed with each other and therefore deformed. This deformation causes reaction force and restitution (restoration) of the shapes. The bodies therefore separate from each other. This phenomenon can be modeled in different ways. The traditional approach is to model it as a spring with a damper.

At each instant during the collision the force values should be computed. The collision duration is very small, the velocity of objects changes rapidly during the collision, and the forces needed to change the velocity can be very large.

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.
- An equation for F with known coefficients is available for computation.

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.

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 $v(\tau) = v'$. The acceleration could be computed from the force divided by mass. The acceleration, however, should not be constant during the collision. 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 $\tau = +\infty$.

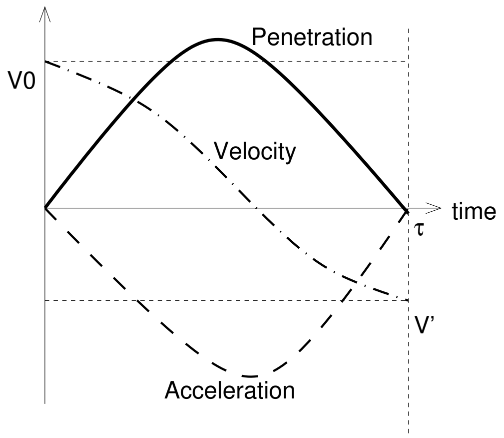

Figure 17: Collision depth, velocity and acceleration in case of elastic collision.

Figure 17 displays what might happen with penetration depth, velocity and acceleration during the collision time in the case of an elastic collision.

Penetration reaches some maximum value and returns to the zero value. The speed gradually changes from v_0 to v'. The acceleration and force either immediately, or gradually, achieve the minimal value, and later return to zero at the time τ.

In the case of a non-elastic collision (Figure 18), the penetration depth, velocity and acceleration become zero after the time τ. If the body has a

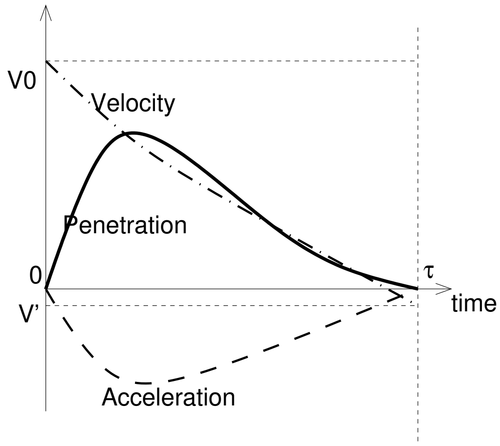

Figure 18: Collision depth, velocity and acceleration in case of non-elastic collision.

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 (constraints C_1)
  - $F(0) = 0$ — the force should be zero at the start of the collision.
  - $x(0) = 0$ — the penetration depth should be zero at the start of the collision.
  - $v(0) = v_0 > 0$ — the actual speed is known at the start of the collision.
- During the collision (C_2)

- $F(t) < 0$ — the force should always push the object away from the obstacle.
- $x(t) > 0$ — the penetration depth is positive (the object penetrates the obstacle).
- $v(t)$ — the speed is reduced to zero and, probably, to negative values.

- If no external force act on the colliding body (or this force is negligible), it behaves exactly according to the impulse law (C_3):
  - The time when collision ends is τ.
  - $F(\tau) = 0$ — the force should be zero at the end of the collision, and after that.
  - $x(\tau) = 0$ — the penetration depth should be zero at the end of the collision.
  - $v(\tau) = v' = v_1 = -\varepsilon v_0 < 0$ — 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 (C_4):
  - $\lim_{t\to\infty} F(t) = -F_{ext}$ — 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.

There can be many definitions for F satisfying these equations and relations. However, in most cases, the difference between these definitions (i.e. difference between the overall effect they cause) is negligible in comparison with the effect caused by the constraints. In practice, the definition for F is sometimes not motivated by physics, but rather by numeric analysis. It is chosen such way that overall result of collision matches physical laws (i.e. preservation of impulse, and preservation of energy). In addition it should be chosen so smooth that numerical methods used in simulation are able to handle with such function.

In the following sections we investigate appropriate variants for the definition of F.

In Section 4.2 we check whether F can be a polynomial of time.

In Section 4.3 we investigate F as a linear function of penetration depth.

In Section 4.4 we prove that if F is a function of penetration depth, it cannot satisfy the condition $v_1 = -\varepsilon v_0$, $\varepsilon < 1$, i.e. cannot model non-elastic collision.

In Section 4.5 we make a conclusion that F should depend on the penetration depth and velocity. We investigate here the coefficients of elasticity and damping for F depending on the penetration depth and velocity. We need to find these coefficients from given restitution coefficient ε.

In Sections 4.6 and 4.7 two methods for finding these coefficients approximately, are given.

4.2 Definition of the Collision Force Using a Polynomial of Time.

The simplest kind of expression that could satisfy the constraints C_1 and C_3 is a polynomial. In this section we demonstrate that $F(t)$ should not be a polynomial of order 2, but it can be a polynomial of order 3.

Initially let us assume that $F(t)$ is a polynomial (of time t) of order 2. In this case $x(t)$ must be a polynomial of order 4. This can be written as

$$x(t) = a_0 + t\,a_1 + t^2\,a_2 + t^3\,a_3 + t^4\,a_4;$$

We invoke the symbolic Mathematica equation solver (`Solve[]`) in order to check whether and under which conditions the constrains can be satisfied.

$$\begin{array}{ll} Solve[& \{ \quad x(0) = 0, x'(0) = v_0, x''(0) = 0, \\ & \quad\;\; x(\tau) = 0, x'(\tau) = v_1, x''(\tau) = 0\}, \\ & \{ \quad v_1, a_4, a_3, a_2, a_1, a_0, \tau\} \\ &] \end{array}$$

The Mathematica solver finds one solution only. In this solution $v_1 = -v_0$.

Now we check whether $F(t)$ can be a polynomial (of time t) of order 3. Therefore $x(t)$ is a polynomial of order 5.

The definition for x is now

$$x[t\_] = a_0 + t\,a_1 + t^2\,a_2 + t^3\,a_3 + t^4\,a_4 + t^5\,a_5;$$

The symbolic Mathematica solver is invoked for the following equation system:

$$\begin{array}{lll} Solve[& \{ & x(0) = 0, x'(0) = v_0, x''(0) = 0, \\ & x & (\tau) = 0, x'(\tau) = v_1, x''(\tau) = 0\}, \\ & \{ & a_5, a_4, a_3, a_2, a_1, a_0\} \\ &] & \end{array}$$

The solution is

$$x(t) = t\,v_0 - \frac{3\,t^5\,(v_0 + v_1)}{\tau^4} - \frac{2\,t^3\,(3\,v_0 + 2\,v_1)}{\tau^2} + \frac{t^4\,(8\,v_0 + 7\,v_1)}{\tau^3}$$

The force can be found from the acceleration expression:

$$F/m = \ddot{x}(t) = \frac{-60\,t^3\,(v_0 + v_1)}{\tau^4} - \frac{12\,t\,(3\,v_0 + 2\,v_1)}{\tau^2} + \frac{12\,t^2\,(8\,v_0 + 7\,v_1)}{\tau^3}$$

Given the collision duration τ as well as the velocity v_0 before the collision and v_1 after the collision, the simulation model can apply the appropriate collision force $F = m\ddot{x}$. However, this approach does not work for bodies resting on the obstacle (condition C_4) since any polynomial $x(t)$ goes to infinity when $t \to \infty$ and it is impossible to have $\lim_{t\to\infty} x(t) = x_{rest}$

4.3 A First Order Linear Collision Force Model (Spring Model)

Traditional penalty methods are based on a linear dependency between the collision force and the penetration depth. Therefore the collision model considered here assumes that the force F depends on $x(t)$ as follows.

$$F(t) = \begin{cases} -Kx(t), & \text{if } x(t) > 0 \\ 0, & \text{if } x(t) \leq 0 \end{cases}$$

where K is a positive constant, called penalty coefficient. Physically this corresponds to a stiff spring, temporarily placed between the objects during the collision. The expression $-Kx(t)$ corresponds to an ideal spring. The expression contains $x(t)$ in the first power only. Therefore this method is a first order linear model.

For brevity of the solution, K might be replaced by another positive constant, q^2m. In this case, taking into account the constraints C_1 and C_3, the system of equations (assuming that no external forces are acting on the colliding bodies) appears as follows:

$$\begin{aligned} F(t) &= -q^2mx(t) \\ F(t) &= m\ddot{x}(t) \\ \dot{x}(0) &= v_0 \\ x(0) &= 0 \end{aligned}$$

This results in the equation

$$\ddot{x} + q^2x = 0$$

which has a solution

$$x(t) = \frac{v_0 \sin qt}{q}$$

The velocity is

$$\dot{x}(t) = \frac{v_0 q \cos qt}{q}$$

When the collision ends (i.e. $x(t) = 0$ giving $qt = \pi$) the velocity $\dot{x}(\pi/q)$ is equal to $-v_0$.

This means that the resulting velocity does not depend on the mass, and does not depend on the penalty coefficient in front of $x(t)$.

But what we need is $\dot{x} = v' = -\varepsilon v_0$, $\varepsilon < 1$.

Therefore the simple penalty-based model above is not good in the general case ($\varepsilon < 1$) and another model should be chosen.

4.4 A Collision Force Model Based on Position

Below we shown that for collision force models that are based on position only, the resulting contact models are purely elastic. Force depending on position is $F(t) = -p(x(t))$. In the same time $F(t) = m\ddot{x}(t)$. Therefore we consider further the equation

$$\ddot{x} + p(x) = 0$$

where p is positive during the collision. We define $\dot{x} = y(x)$. This results in

$$\ddot{x} = \frac{dy}{dt} = \frac{dy}{dx}\frac{dx}{dt} = \dot{x}\frac{dy}{dx} = y\frac{dy}{dx} .$$

The initial equation can be rewritten now as

$$y\frac{dy}{dx} + p(x) = 0 .$$

If all parts of the equation are integrated, the result is (C is an unknown constant)

$$\int y\,dy + \int_0^x p(u)\,du = C .$$

The first component is transformed:

$$\int y\,dy = y^2/2 = \dot{x}^2/2 .$$

We know that at the start of the collision $x(0) = 0$ and $\dot{x}(0) = v_0$. The equation should be valid at $t = 0$, and substitution gives

$$v_0^2/2 + \int_0^0 p(u)\,du = C .$$

Now C is found as $C = v_0^2/2$. The equation with the replaced C is

$$\dot{x}^2/2 + \int_0^{x(t)} p(u)\,du = v_0^2/2 .$$

We know that at the end of the collision $t = \tau$, and $x(\tau) = 0$. Substitution gives:

$$\dot{x}^2(\tau)/2 + \int_0^{x(\tau)} p(u)\,du = \dot{x}^2(\tau)/2 + \int_0^0 p(u)\,du = v_0^2/2 ,$$

or simply $\dot{x}^2(\tau) = v_0^2$. From we can conclude that either $\dot{x}(\tau) = v_0 = \dot{x}(0)$ or $-\dot{x}(\tau) = v_0 = \dot{x}(0)$. But we know that p is positive, therefore $\ddot{x}(t) = -p(x(t)) < 0$. We conclude that $\dot{x}$ decrease and since $\tau > 0$ and $\dot{x}(0) > 0$, the only equation valid is $-\dot{x}(\tau) = v_0 = \dot{x}(0)$. Therefore $v' = -v_0$, which means that the collision was purely elastic ($\varepsilon = 1$).

4.5 Collision Force Model Based on Spring and Damping

Since the definitions for collision force discussed above are not good enough for our collision model, some other factor should be taken into account. The goal is to reduce the magnitude of the velocity at the end of the collision. The velocity should be smaller during the last instant of the collision, and larger during the first instant. Then the magnitude of the velocity at the end the of collision can be reduced. Therefore some factor is needed that gradually changes during the collision. The speed of penetration $\dot{x}$ is an appropriate variable, since it gradually changing from one value to another for the duration of the collision.

First, consider what can be done with the original expression $F = -Kx$. 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 $\varepsilon = 1$. This function H should be equal to 1 in the case of $e = 1$. 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 model can be expressed as a system of equations:

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

The first two equations are reduced to $\ddot{x} = \text{If}[x > 0, -(1+k\dot{x}(t))Kx(t), 0]$.

The function $G(\dot{x}) = k\dot{x}$ is called a damping factor. In practice it is often chosen as a linear function. However, in order to provide differentiability of F other functions are used too. One topic of our future research is to find such a smooth function F that ODE integration algorithms can safely use it as an external function during continuous integration.

The above equation 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 $v(t) = v(\tau)$. Therefore also $v(t_{end}) = v(\tau) = v'$.

When $v_0 = 1$ and $k = 1$ the resulting velocity v' changes in the following way:

If $K = 1$, $v' = -0.593628$;

If $K = 10^{-3}$, $v' = -0.593625$;

If $K = 10^{6}$, $v' = -0.593627$.

Similar effect appears for other values of v_0 and k. Therefore v' 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.

Proof: Assume x(t) is a solution of the system. Consider $y(t) = px(t)$ (where p is a constant). Then $\dot{y} = p\dot{x}$, therefore $\dot{y}(0) = p\dot{x}(0) = pv_0$ and $\dot{y}(\tau) = p\dot{x}(\tau) = pv'$. We compute $\ddot{y}$ from the equation:

$$\ddot{y} = p\ddot{x} = -p(1 + k\dot{x})Kx = -(1 + (k/p)\dot{y})Ky$$

Therefore y is a solution for the equation $\ddot{y} = -(1 + k/p\dot{y})Ky$ with the start condition $\dot{y}(0) = pv_0$. The restitution coefficient is $\varepsilon(v_0, k) = -v'/v_0 = -\dot{x}(\tau)/\dot{x}(0) = -(\dot{x}(\tau)p)/(\dot{x}(0)p) = -\dot{y}(\tau)/\dot{y}(0) = \varepsilon(pv_0, k/p)$.

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 $u = v_0 k$.

Various plots for $\varepsilon(u, 1)$, $10^{-2} < u < 10^2$ have been obtained using Mathematica. We found two ways to approximate this function, using a fraction-based and a polynomial-based approximation.

4.6 Fraction-Based Approximation

By looking at the plots we guessed 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.

4.7 Polynomial-based Approximation

The plots do not resemble a polynomial function. However if an exponential expression is used as the argument, i.e. the function $\varepsilon_1(s) = \varepsilon(e^s, 1)$ is plotted, the plot resembles a cubic polynomial of s. Four "typical" points of the curve have been guessed, and a fitting cubic curve going through these points

$$\varepsilon_1(s) = a_3 s^3 + a_2 s^2 + a_1 s + a_0$$

is found using Mathematica. How to find s? The equation $\varepsilon_1(s) = n$ has three solutions which can be expressed in algebraic form. Some of them are complex numbers with nonzero imaginary part. Some solutions do not belong to the piece of the curve we consider now. The appropriate solution is denoted as $s(n)$. The computations show that s can be chosen as one of the three solutions of the cubic equation above:

$$s(n) = 0.620898 + \frac{-0.163279 + 0.282808\,i}{T^{\frac{1}{3}}} - (15.6309 + 27.0735\,i)\ T^{\frac{1}{3}}$$

where

$$T = -0.000833598 + 0.00193378\,n + 0.00193378\,\sqrt{-0.983167 + n}\,\sqrt{0.121024 + n}$$

Computations show that the imaginary part of the expression is zero. According to the definition of ε_1

$$\varepsilon(v_0, k) = \varepsilon(v_0 k, 1) = \varepsilon_1(\ln(v_0 k))$$

Therefore k can be expressed as

$$k = \frac{1}{v_0} e^{s(\varepsilon(v_0, k))}$$

Again k is inserted into the original differential equation system. When the equation

$$F(t) = -(1 + \frac{1}{v_0} e^{s(\varepsilon_{required})} \dot{x}(t)) K m x(t)$$

is solved for $0.001 < \varepsilon < 0.999$ and the resulting ε is found, the absolute value of the difference between required and resulting restitution is always less than 0.05. Smaller deviation can probably be achieved by choosing another set of four or five points for interpolation.

Polynomials of order 5 and higher (requiring six interpolation points) cannot be used for interpolation in this method because there is no way to express the inverse function (i.e. find the root symbolically).

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 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. There exist efficient algorithms [7] to check whether any bounding boxes in the 3D space intersect.

The bounding boxes are always axis-aligned. Static bounding boxes are computed when the body geometry is specified. They preserve their size and move when the body is moving. Dynamic bounding boxes are determined from the geometry, position and the current rotation angle of the body, and they are more accurate.

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.

Response handling In order to compute reaction forces more detailed geometrical information about the collision is necessary. Often the collision detection package cannot provide the information that is needed, for instance, the penetration depth, collision plane, time of collision, collision volume and area of collision surface.

Sometimes this information is available, but just for certain time steps.

6 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 6.1) and difficulties in obtaining correct collision plane (Section 6.2).
- How geometry of bodies is specified when SOLID is used with Modelica (Section 6.3).
- How the shortest separation vector (defined in Section 3.5) is computed using SOLID callback function (Section 6.4). The shortest separation vector defines direction and magnitude of penetration depth.

The penetration depth is then used for computation of collision force (Section 4).

6.1 Using SOLID interface functions

When SOLID is used the following steps are performed:

- Object shapes are created. 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 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.

- If the current state of the bodies will be used in a future step (for closest feature determination) it should be stored by function call `dtProceed()`.

- The function `dtTest()` is called in order to check all pairs of the bodies and it calls the callback function in case of collision.

- The callback function written by the library user obtains data about the colliding features as parameters.

6.2 Collision Plane Definition Problem

For physics-based simulation the ”smart response” option of SOLID is used. 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. In mathematical notation, assume that the time frame $t+1$ is analyzed and objects $A(t)$[2] and $B(t)$ were disjoint, and objects $A(t+1)$ and $B(t+1)$ penetrate. The closest points $P_{A(t)B(t)}$ and $P_{B(t)A(t)}$ are obtained as parameters by the callback function.

According to the SOLID author recommendations[4], vector defined by these two points can serve as approximation to the normal to the collision

[2] The notation $A(t)$ means body A at time frame t.

plane. The direction of the collision force can be set as equal to the direction of this normal vector.

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.

Assume that the bodies $A(t+i)$ and $B(t+i)$ are disjoint when $i = 0$ and then intersect for $i = 1, ..., m$. The points $P_{A(t+i)B(t+i)}$ cannot be obtained for $i = 1, ..., m$. Therefore we cannot obtain the collision plane. On the other hand, the point $P_{A(t)B(t)}$ cannot be used for finding the collision plane for $i = 2, ..., m$ since the plane could change during the collision.

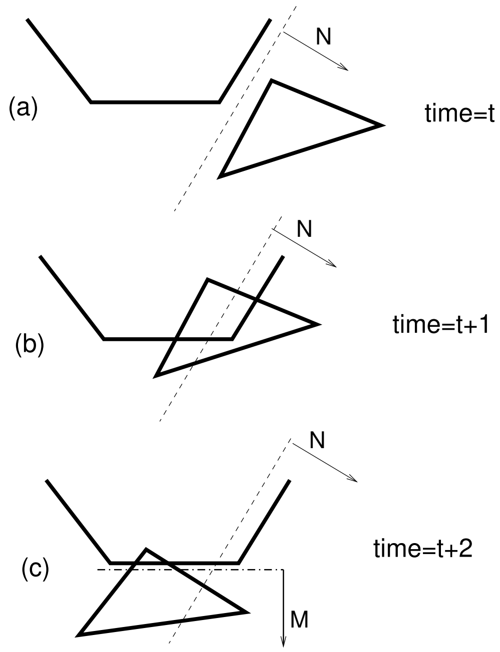

Figure 19: Variations of the collision plane during the collision

Figure 19 demonstrates how the collision plane (that should be perpendicular to the vector of the object separation force) can change during the collision. In Figure 19(a), at time step t the objects are disjoint. In figure 19(b) at time step $t + 1$ the objects start to intersect, and this fact is noticed by SOLID. The package finds the position of the closest features at the previous time frame (t) and determines the collision plane, as well as normal vector to this plane, N . Finally, in Figure 19(c) at time step $t + 2$ the positions of the bodies have changed and the actual collision plane has changed too. The vector M should be used instead of N.

Our solution to this problem is presented in Section 6.4.

6.3 Geometry Specification

In order to transmit geometry specifications used in mechanical modeling in Modelica to SOLID user interface functions, two ways have been chosen. Modelica bodies may have a predefined shape such as box, cone, cylinder, sphere, etc. These shapes correspond to predefined objects in SOLID (box, cone, cylinder and sphere). Alternatively, Modelica bodies may have custom geometry, described using some external format.

When a SolidWorks to Modelica translation is performed, 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.

6.4 Detailed Handling of Collision Response

Our force-based approach to contact handling requires obtaining collision points, separation force direction, and penetration depth from collision detection software.

In our approach we currently consider the worst case assumption. In the worst case the computation of the collision may need many time steps (frames). During such a long collision process the separation force direction may change. It is also possible that the contact never ends.

Assume that the collision between the objects A and B is detected at time frame t. In order to find the shortest separation vector $\vec{c}$ (as defined in Section 3.5) 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[3]. The value of c^i is incremented at each loop iteration, so that $c^i = (1 + \kappa)c^{i-1}$. The value c^0 should be chosen so that it is negligible in comparison with a typical penetration depth. The value κ should be small if high accuracy is required. The reasonable choice interval for κ is $0.1 < \kappa < 0.4$. Smaller values of κ give a slower and more accurate search process.

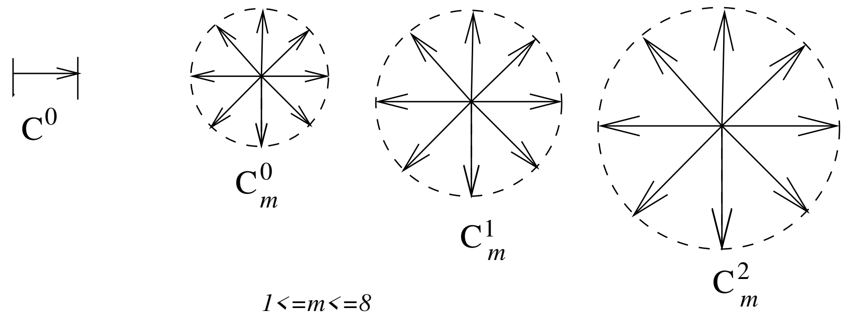

Figure 20: Sequence of trial vectors c^0, c^1, etc.

In the nested internal loop (over variable m) the direction for $\vec{c^i_m}$ is chosen. In order to sweep through all possible directions the 27 vectors $c^i_0, ..., c^i_{26}$ are found. These vectors have components (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 the list. All the direction vectors are normalized and then multiplied

[3]Here and in the rest if Section 6 the superscript should be regarded as an index, not as an exponent representing the *power* operation.

by the current length, i.e. c_i. As a result, the vectors $\vec{c_m^i}, 1 \leq m \leq 26$ are obtained. A 2D variant (8 vectors) is shown in Figure 20.

There is no way to ask SOLID directly about the distance and the closest points of two bodies. However, if the bodies penetrate it is possible to make a request about the closest points in the previous step. We use the following way around this problem.

The algorithm checks if bodies A and $B(\vec{c_m^i})$ (i.e. B displaced by $\vec{c_m^i}$) are disjoint or penetrate. If they are disjoint then routine `dtProceed()` is used in order to store object's position. Again the positions of A and B are passed to the package. The `dtTest` routine finds that they intersect, and therefore the callback routine delivers to the algorithm the closets points between the bodies A and $B(\vec{c_m^i})$. These points can be referred as $P_{A,B(\vec{c_m^i})}$ and $P_{B(\vec{c_m^i}),A}$. 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.

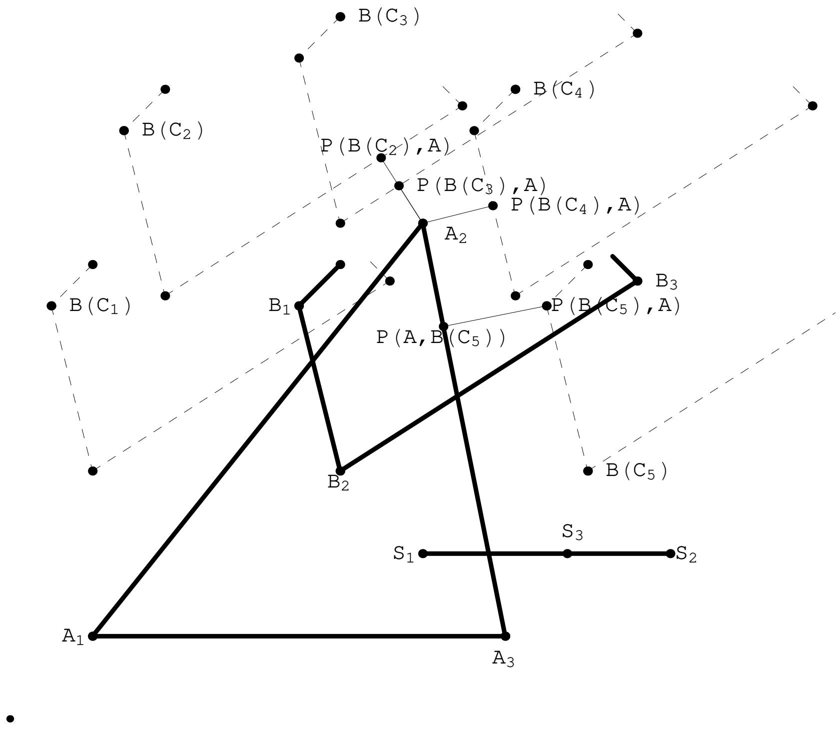

Figure 21: Computation of the shortest separation vector

The inner loop of the algorithm can be demonstrated graphically in the two-dimensional case as in Figure 21. 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. 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 $\vec{c_5} = \overrightarrow{S_1S_2}$. 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.

6.5 Special Cases for Speedup of the Search

The algorithm we consider for finding the penetration depth is rather slow. It can be greatly optimized if we know that all the bodies are convex, that during the collision only one single vertex (P_B) of body B is within A, and for all features of B the face F_A is the closest face. With such an assumption it is enough to move the vertex (together with the body B) to the closest face of A, i.e. F_A.

Finding the closest face can be optimized. The object A consisting of faces $F_A^1, ..., F_A^M$ can be divided during a preprocessing stage to interior regions $R_A^1, ..., R_A^M$ (one region per face) in such a way that for all the points in the region R_A^i the face F_A^i is the closest one (see Figure 22). These regions are sometimes called *pseudo Voronoi regions* [19]. Preprocessing, however, can be a difficult problem.

The Figure 23 demonstrates a case where this assumption is violated. When vertex P_B is moved to the closest face F_A^1, the bodies are still penetrating.

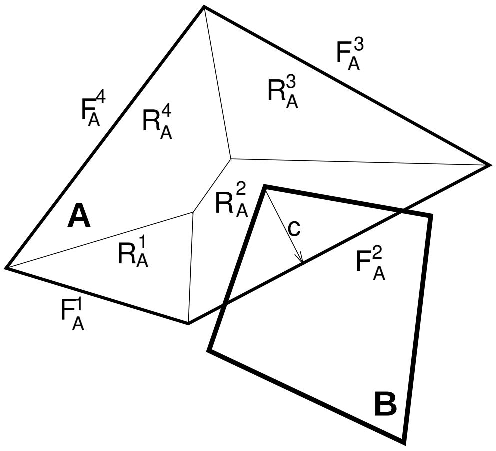

Figure 22: The closest face can be found in advance.

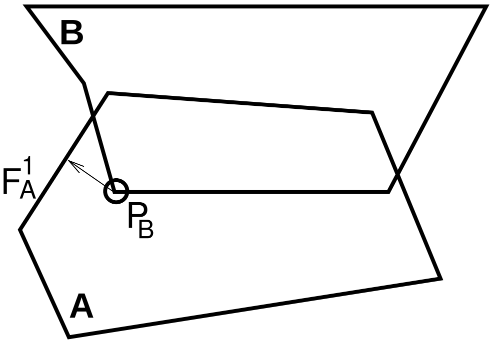

Figure 23: Collision involving several vertices

The algorithm can be made more exact (and slow) by adding more loops for fine-grained search and more steps in the outer loop. Some hints can be given to the algorithm in order to choose the initial approximation.

6.6 Combining Penetration Depth and Distance

The collision response computations based on the impulse model (Section 2) utilize the distances between the bodies in a different way. The bodies are considered as intersecting if the distance between their closest features is less than a certain threshold. It is not very accurate model either, since the bodies start to interact when they are still at some distance from each other. The impulse-based algorithms do not allow the bodies to intersect at all. This is done either by estimating the maximum velocity and choosing an appropriate time step, or by reducing the time steps before the moment of collision. The approach based on a distance threshold can be combined with our force-based model.

In this combined approach we could use both the penetration depth and the distance between the bodies in the computation of forces. The vector $\vec{x}$ used for collision force definition can be defined as described below.

For instance, the vector $\vec{x}$ used for computation of the force (see Figure

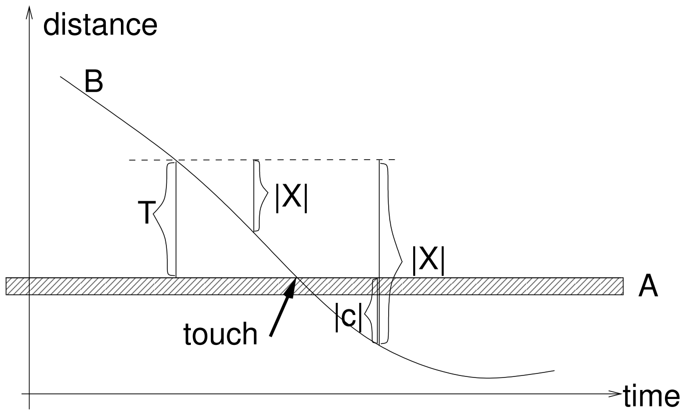

Figure 24: Combined approach to evaluation of $\vec{x}$.

24) can be defined as

- $\vec{x} = 0$ if $d \geq T$,
- $|x| = T - d$ and $\vec{x}$ is directed as $\vec{d}$ if $d < T$ and objects are disjoint,
- $|x| = T$ but its direction is undefined if $d = 0$ (degenerate case which should almost never happen),
- $|x| = T + |c|$ and $\vec{x}$ is directed as $\vec{c}$ if the objects intersect,

where

- d is the current distance between the two closest features of the two disjoint objects.
- $\vec{d}$ is the vector between the closest points on the two disjoint objects.
- T is the threshold, chosen depending on the average velocities and computation time step.
- $\vec{c}$ is the penetration depth and direction.

The SOLID tool does not evaluate the distance between the bodies ($|d|$) if they do not collide during the next time frame. However, an "artificial" collision at additional fake time frame can be added and the distance can be found. In future other collision detection packages such as I_COLLIDE [7] can be used for this purpose.

7 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 last example uses the MBS library, the collision detection tool SOLID, as well as some gluing classes.

7.1 Pendulum Collision with an Obstacle

This pendulum is the same mechanical construct as the one described in Section 2.5, Figure 7. For finding the magnitude of the force it uses the equation derived in Section 4.6. It is possible to choose the penalty coefficient K (discussed in Section 4.6), 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;
```

7.2 Pendulum Resting on an Obstacle after the Collision

This section describes an experiment with the same model as above, but modeling the situation when the pendulum is bouncing and is finally *resting* on the obstacle, see Figure 25.

The only equation replaced in this model is

```
depth = -B.r0b[2]-0.4; // The obstacle equation.
//  B.r0b[2] is the y coordinate of the pendulum end.
```

where `-0.4` is the position of the obstacle.

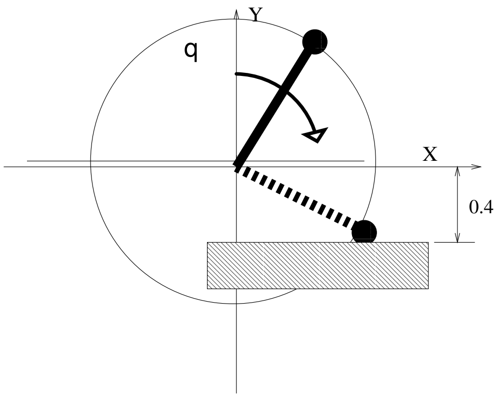

Figure 25: Pendulum resting on an obstacle after the collision.

The result (assuming that $K = 5 \cdot 10^4$) is shown in Figure 26. It demonstrates a collision with small penetration. If the penalty K is set to 10^6 the value of the Y coordinate of the pendulum end practically never becomes less than `-0.4` in this experiment.

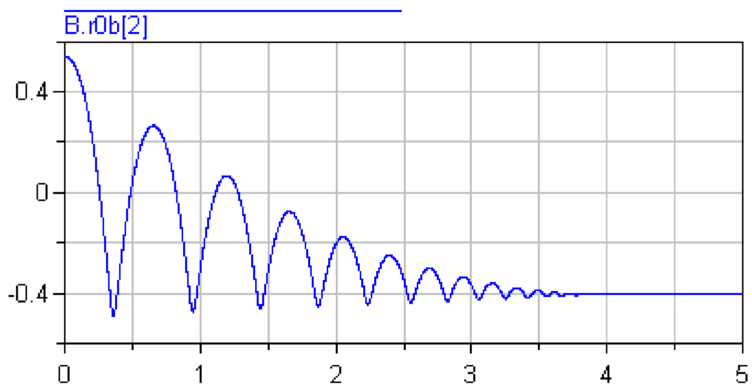

Figure 26: Height of the endpoint of the pendulum resting on an obstacle after the collision.

7.3 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[26] "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=0.133333,I22=0.133333,I33=0.133333,
    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 27 demonstrates dynamic visualization of collision between four free flying cubes. The lines represent the trajectories before and after the collision. Figure 28 contains a plot of forces acting on the first box. The three components (X, Y, Z) of the force are represented by three curves. Two major collisions occur at time 1.0 and 1.3. The trajectories of four bodies (projected on the X axis) are displayed in Figure 29. Initially two boxes move towards the zero point, and all four bodies are separated after the collision occurs.

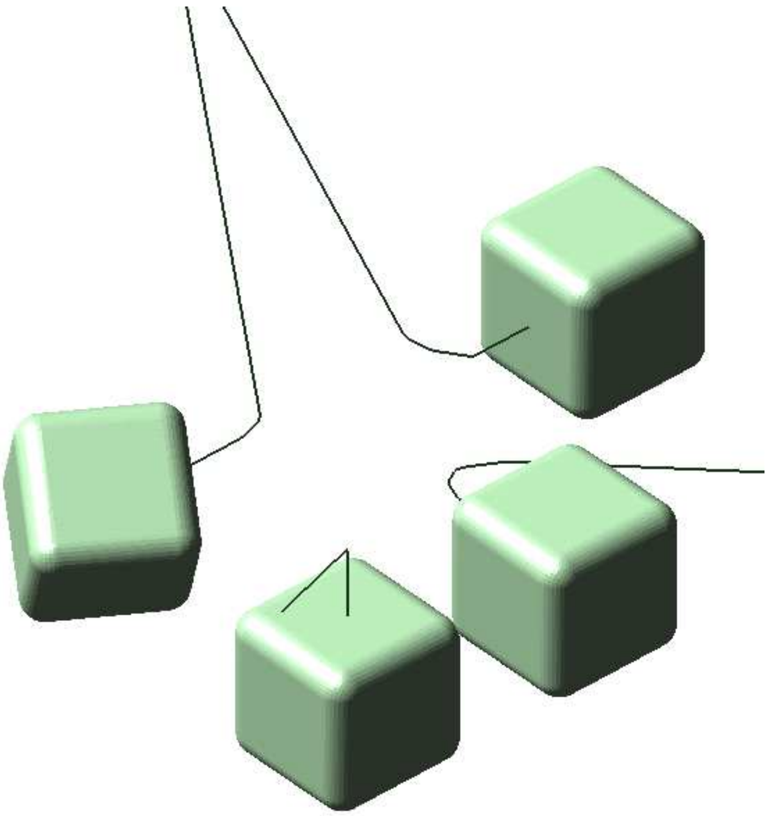

Figure 27: Dynamic visualization of collision between four free flying cubes. The lines represent the trajectories before and after the collision.

8 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. 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.

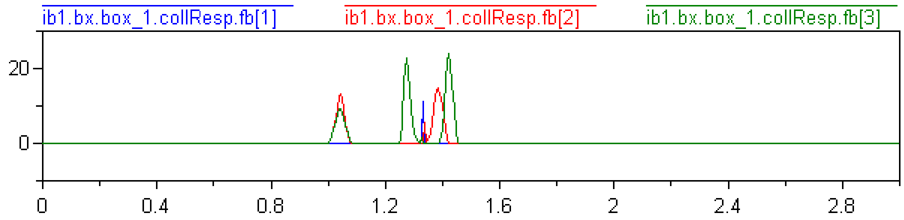

Figure 28: The forces acting on the first box. The three components (X, Y, Z) of the force are represented by three curves.

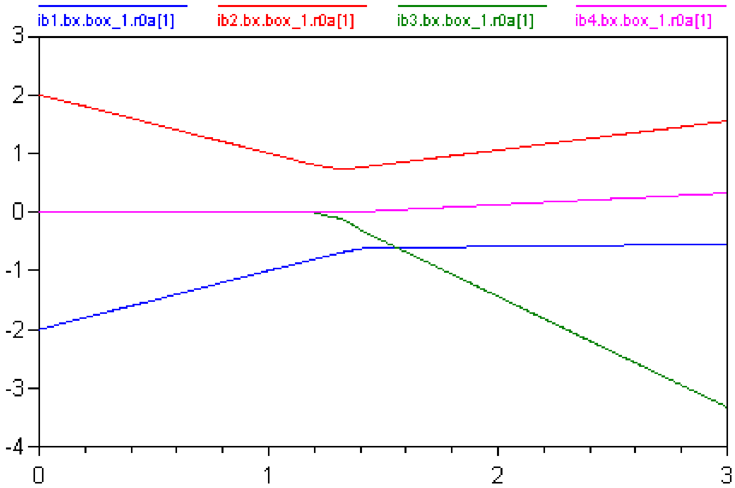

Figure 29: The trajectories of the four bodies projected on the X axis.

9 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 and Jakob Engelson 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/