Simulating collisions within the Modelica MultiBody library

Andreas Hofmann, Lars Mikelsons, Ines Gubsch, Christian Schubert

Angaben zur Veröffentlichung / Publication details:

Hofmann, Andreas, Lars Mikelsons, Ines Gubsch, and Christian Schubert. 2014. "Simulating collisions within the Modelica MultiBody library." In *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, edited by Hubertus Tummescheit and Karl-Erik Arzén, 949–57. Linköping: Linköping University Electronic Press. https://doi.org/10.3384/ecp14096949.

Simulating Collisions within the Modelica MultiBody Library

Andreas Hofmann[1] Lars Mikelsons[1] Ines Gubsch[2] Christian Schubert[2]
[1]Bosch Rexroth AG, Lohr am Main, Germany
[2]TU Dresden, Chair of Construction Machines and Conveying Technology, Germany

Abstract

In this paper an approach for handling collision within the Modelica MultiBody library is presented. Therefore, a short overview about collision consideration for multibody simulation is given. Different methods for calculating the contact reactions are discussed and their potentials for implementation in a free Modelica library are deliberated. Furthermore the implementation of this collision library, using a penalty-based collision approach and the *Bullet Physics Library* for collision detection is described. The application is demonstrated in examples and limitations are brought up. Although some drawbacks restrict usability, the library can be used to increase the level of detail for multibody simulation models.

Keywords: contact; collision detection; collision library

1 Introduction

1.1 Motivation

Many physical systems cannot be simulated in a feasible manner without the description of collision interaction. Not only the typical applications, like wheel-road-contact, newton's cradle or a bouncing ball need collision consideration, but especially real-life models require contact handling. For example, simulation of typical working processes for construction machines, with lifting rocks can benefit from this. But also machine elements like mechanical springs require collision handling for simulations including dynamic loads.

For the Modelica Library *Modelica.Mechanics.MultiBody* (*M.MB*) several collision handling considerations have been made, with two to be shortly mentioned. In [1] Otter et al. introduced an extension to the *M.MB* library with capabilities of handling collisions. Collision detection using different approaches of surface representation were shown. Engelson [2] described a way of contact implementation using impulse-based and penalty-based methods. However, those approaches have never been available in public.

To offer collision handling to general public, *CollisionLib* – the library presented here – will be freely available. Although the functionality of this very first version has only been tested in Dymola, support for OpenModelica and other Modelica environments are planned for the future.

1.2 Outline

In the following section general information about collision handling is given. The main steps for treating contacts are considered and several methods for handling collisions are described. These methods are compared with respect to their capabilities of straightforward implementation in Modelica.

The next chapter addresses *CollisionLib* - a library for collision handling within Modelica. A summary about requirements and intentions is given and, moreover, the implementation of collision detection and collision response is described.

Examples and limitations of applications are shown in section 4 and the paper is closed with an outlook about further development.

2 Collision handling approaches

2.1 Main aspects of collision handling

When handling collision interaction in multibody simulations two main steps have to be considered.

(1) Collision detection needs to be performed for every possible contact pair of bodies. By testing each combination the effort e_T is $O(n^2)$, see (1), where n is the total number of bodies.

$$e_T = \frac{n \cdot (n-1)}{2} = \frac{n^2 - n}{2} \tag{1}$$

It follows from the foregoing that collision detection might have an impact on simulation time.

In order to accelerate this process different two step methods have been developed. Firstly, in the so called broadphase, a rough estimation which bodies might collide is made. Hence, all body combinations that cannot interact for obvious reasons are rejected. Those couples, that might collide, are tested with a proper distance algorithm, e.g. GJK [11], in the second step, called narrowphase.

By doing so, the overall effort can be reduced considerably. In [3] Baraff describes a broadphase method with $O(n)$. The total resultant effort e_T then is $O(n+k)$, with k being the number of body pairs requiring proper examination.

In addition to finding colliding body pairs, the corresponding contact normal and tangent vectors have to be determined for each of these pairs. For some approaches even more data has to be provided.

(2) Collision reactions have to be computed according to the chosen collision approach. Since collision handling has a long history, going back to Newton, Poisson, Coulomb and their laws and hypotheses, many different approaches for calculating collision responses are available. However, regarding to Mirtich [4] the state-of-the-art methods can be classified into three categories.

(a) The Penalty-based approach is the only one of these three allowing intersection among the bodies. The basic idea is that between the colliding bodies a spring, spring-damper or some other force element is present, which generates separating forces. The forces are related to the penetration of the bodies. This rather simple approach has some disadvantages. Finding right contact parameters is an open problem. Moreover, this parameters cannot be adopted from one simulation to others. Also collision forces have to be big in order to avoid deep intersection between the bodies. This leads to stiff DAE systems which are hard to solve, require small time steps and thus may lead to long simulation times.

(b) Analytical solutions prohibit intersection among the colliding partners. The constraints between the involved bodies are conveyed into a linear complementary problem (LCP), see [5] or [6]. Without consideration of friction the LCP formulation will always lead to contact forces, creating realistic movements of the bodies. However, these contact forces do physically not need to present the right solution since the LCP might find unlimited numbers of solutions. In Figure 1 three possible solutions for the symmetric table under weight force w are presented with only (c) showing correct behavior. If friction is included

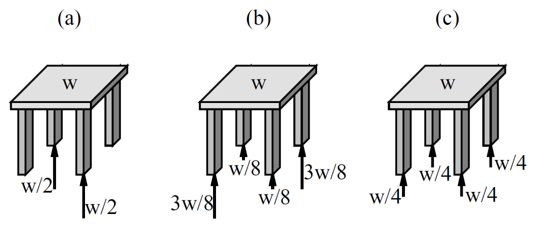


Figure 1: Possible solutions for the a standing table simulation using the LCP approach [4]

into the collision consideration the analytical approach might lead to no or no unique solution. In addition, solving an LCP can become difficult and system information is required, e.g. mass matrix and Jacobian matrix.

(c) Impulse-based methods are the third group of approaches for handling collision reactions. All kinds of constraints are neglected and no constraint forces are calculated. Instead all movements are handled using impulses and correcting impulses. Basics of this rather young idea of simulating movements are described in [7], [4]. The impulse-based methods, like the analytical approaches, prohibit intersection between the bodies. But, in contrast to the solutions of the analytical approach, the contact reactions are always calculated physically correct. Of course, also the impulse-based methods have some drawbacks. Stationary contact, like resting bodies on the ground, need to be solved by a high frequent number of small impulses. Interaction between multiple bodies, like a stack of boxes, can cause problems and corrupt results, cf. [4]. The biggest problem, when working with impulse based methods is, that a special form for the equations of motion is needed and special routines for solving those are required.

2.2 Implementation in Modelica

Because impulse based methods do not use constraint forces, an implementation in Modelica appears expensive – especially since correction impulses need to be calculated in one simulation step and have to be applied in the previous one. Also the interaction with other physical domains seems difficult.

The analytical approach is more applicable since constraint forces are calculated. However, solving the LCP can be difficult and requires knowledge of the mechanical system, like mass matrix of the mechanical system, etc.. Since Modelica is a multi-domain modeling language collecting the equations from all domains, a straightforward implementation in Modelica is, regarding to the author, not possible.

Hence, for this implementation a penalty method is chosen. For the first part of the collision handling, collisions have to be identified, as described above. Different free collision detection software packages, e.g. *Bullet* or *ODE*, are available, calculating the needed data for a penalty approach. Therefore, integration can be done directly without rewriting any contact routine, etc.. However, drawbacks of the penalty based methods may not be ignored and limitations must be accepted.

3 CollisionLib - a contact library for the multibody environment in Modelica

3.1 General aspects

CollisionLib is a library that extends the *Mechanics.MultiBody*-library (*M.MB*) by collision consideration. This expansion implies that existing models do not need to be rebuild. Instead, the mechanical components are connected to special collision objects, as seen in figure 2.

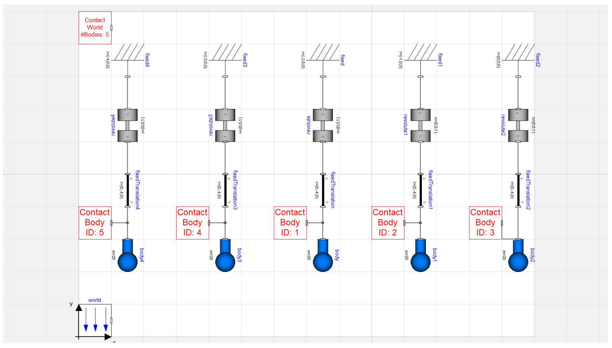


Figure 2: Setup of a mechanical System including collision handling

3.2 Collision detection within Modelica

For solving the issue of collision detection *Bullet Physics Library* (*Bullet*) [8], an external C++-library, is used. It is a free package for simulating mechanical systems, originating from entertainment industry with focus on video games and movies. Since *Bullet* uses a modular conception, see Figure 3, it is possible to take its collision detection only and drop the simulation routines.

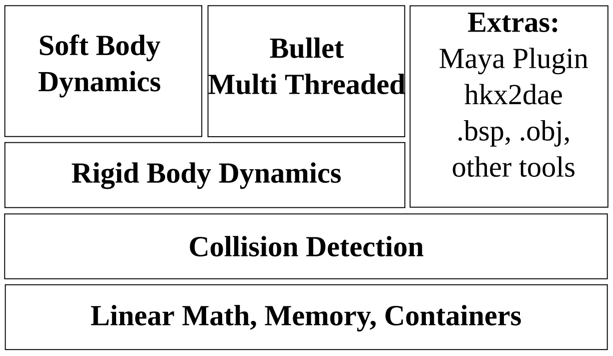

Soft Body Dynamics
Bullet Multi Threaded
Extras: Maya Plugin hkx2dae .bsp, .obj, other tools
Rigid Body Dynamics
Collision Detection
Linear Math, Memory, Containers

Figure 3: The modular concept of the *Bullet Physics Library* [9]

In order to calculate distance between two bodies, their geometry and their position as well as their orientation must be defined. Within the *M.MB* bodies are represented by their mass and inertia and therefore no additional information is present. However, *Bullet* contains a database with the basic geometries – sphere, box, cone, cylinder and capsule –, able to create a representation, if the right information is passed to it. For complex structures surface data, using a polygon mesh, need to be passed. In collaboration of *Bullet* and Modelica this implies that for basic bodies only some identifier and dimensions have to be supplied. The use of complex, mesh based geometries are currently not included, but planned for future versions.

Due to the fact, that all bodies of the *M.MB* are rigid, meaning that their inertia respectively shape will not change by cause of loads, the geometry information is only passed at initialization of the simulation.

In contrast to shape, the pose of each body needs to be updated every simulation step. This is done by transferring position and rotation matrix to *Bullet* each step.

With all the information described above Bullet can perform the check for collisions. From the several available broadphase algorithms within the C++- library an approach using axis aligned boundary box trees (AABB-trees) is used. Detailed information about AABB-trees can be found for example in [12]. For narrowphase intersection tests Bullet provides different algorithms for primitive shapes, triangle meshes, etc..

The coupling between Modelica and *Bullet* is done by expanding the Modelica class *externalObject* into a class called *CollisionWorld* and developing a *Bullet* simulation runtime. Attentive readers might have noticed that *Bullet* is a C++-library and therefore cannot directly be connected to Modelica. To attach the *Bullet Physics Library* to Modelica an additional C-interface is needed. At instantiation of class *CollisionWorld* within a Modelica model, geometry information is passed to the interface. This interface itself, creates a instance of the *Bullet* simulation runtime, passing the position and rotation matrix, and returning a reference to Modelica. This procedure is visualized in Figure 4.

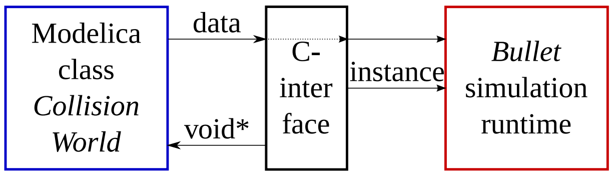


Figure 4: Coupling between Modelica and *Bullet* at system initialization

During each simulation step, the pose of the bodies needs to be updated and collision detection must be performed within the *Bullet* runtime. In Modelica, the function *updatePosition*, of type *externalC*, passes the reference to the *bullet* environment and the pose of the bodies back to the C-interface. This results in an updated *Bullet* simulation, giving back the number of contact body pairs (NoC). Using a second *externalC*-function, named *getCollisionData*, the collision data (coll. data) is given back to Modelica. The complete data flow between Modelica, C-interface and *Bullet* is shown in Figure 5.

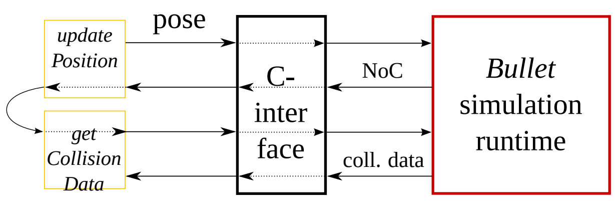


Figure 5: Data flow during simulation

3.3 Collision normal force

As specified before, a penalty based approach for calculation collision reactions is used. The approach included in the *CollisionLib* calculates reaction forces using deepest point penetration. This means the two deepest points between the intersection bodies are calculated and the normal vector based on this two is given by *Bullet*. Another possible penalty method for calculating response, could use consideration of intersecting volume. However, this is not possible using *Bullet*.

There are many different approaches calculating contact normal forces taking geometry, material properties, velocities, mass, etc. into account. A survey about some ideas is given by Machado et al. in [10]. In order to embrace the dimension of methods user-defined collision response laws can easily be implemented.

The basic normal force calculation, available in this library, uses a parallel set of non-linear spring and linear damper. The non-linear spring has its origin in the Hertzian contact theory but is here extended using energy dissipation.

$$F_N = F_c + F_d \tag{2}$$

The part F_c represents contact force due to the spring and F_d is the force due to the damper.

$$F_c = K \cdot \delta^n \tag{3}$$

The parameter K delineates the contact stiffness while the exponent n describes the non-linear force behavior due to penetration δ.

For the damping force some limitations apply. Its magnitude can never exceed the spring part, preventing sticking between the two bodies. Therefore a dummy item F_{d2} is introduced

$$F_{d2} = D \cdot \dot{\delta} \tag{4}$$

$$F_d = \begin{cases} F_c & \text{if } F_{d2} > F_c \\ -F_c & \text{if } F_{d2} < -F_c \\ F_{d2} & \text{else} \end{cases} \tag{5}$$

The factor D describes the contact damping and $\dot{\delta}$ denotes the relative velocity of the contact points.

All parameters (K, n, D) depend on geometry and material of the colliding bodies. For special combinations of bodies different analytical and empirical approaches have been published. The parameters can also be derived by Finite-Element-Analysis or practical test execution. However there are no generalities available.

As mentioned before the contact parameters are individual for each contact. Nevertheless the parameters are globally set for all contacts in the first place. But of course the contact parameters can be individually assigned for each pair of bodies.

3.4 Friction force calculation

In order to calculate friction forces it is necessary to determine two tangent vectors ($\mathbf{tv}_1,\ \mathbf{tv}_2$) from the contact normal vector $\mathbf{nv}$. Since the two tangent vectors and the normal vector have to be orthogonal to each other two components of one tangent vector can be chosen freely. in *CollisionLib* this is done by the following scheme:

- Determination of the biggest component of the normal vector $\mathbf{nv}$
- Assignment of value 1 to the two components of the first tangent vector $\mathbf{tv}_1$, respectively to the two smaller components of $\mathbf{nv}$
- Calculation of the last component of $\mathbf{tv}_1$ using dot product ($\mathbf{nv} \cdot \mathbf{tv}_1 = 0$)
- $\mathbf{tv}_2$ is calculated via cross product ($\mathbf{nv} \times \mathbf{tv}_1$)
- Normalization of both tangent vectors

With this tangent vectors the tangential plane of the contact is determined, in which the friction force vector is located. In order to identify its direction the projection of the relative velocity vector $\mathbf{v}_{rel}$ into the tangential plane is required. The vector $\mathbf{v}_{rel}$ is calculated from the contact pair of deepest penetrating points between the two bodies $\mathbf{K}_a$ and $\mathbf{K}_b$ as described from (6) to (8). All calculations are performed with respect to the initial frame $\{\mathbf{O}^I, \mathbf{e}^I\}$ and refer to Figure 6. To keep calculations as short as possible only the quantities of *bodyB* are derived. The values of *bodyA* are calculated analogical.

First of all, the vector from the body fixed frame $\{\mathbf{O}^B, \mathbf{e}^B\}$ to the contact point $\mathbf{K}_b$ is calculated.

$$\mathbf{r}_b = \mathbf{r}_{Kb} - \mathbf{r}_{b0} \tag{6}$$

Using this vector the velocity of the contact point can be obtained.

$$\mathbf{v}_{Kb} = \frac{\mathrm{d}\mathbf{r}_{Kb}}{\mathrm{d}t} = \dot{\mathbf{r}}_{b0} + \omega_b \times \mathbf{r}_b \tag{7}$$

After determination of $\mathbf{v}_{Ka}$ the relative velocity is computed:

$$\mathbf{v}_{rel} = \mathbf{v}_{Kb} - \mathbf{v}_{Ka}. \tag{8}$$

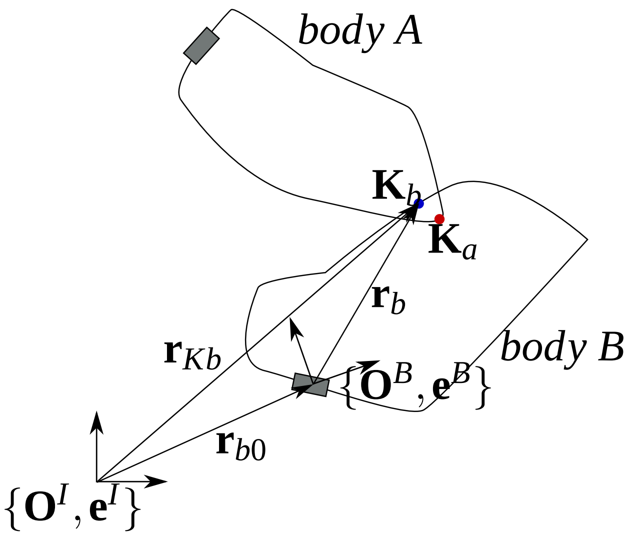


Figure 6: Calculation of the relative velocity during contact

By projection this vector into the tangent plane ($\mathbf{v}_{relP}$) the opposite direction of the friction force vector is derived.

$$\mathbf{v}_{relP} = (\mathbf{tv}_1^T \cdot \mathbf{v}_{rel}) \cdot \mathbf{tv}_1 + (\mathbf{tv}_2^T \cdot \mathbf{v}_{rel}) \cdot \mathbf{tv}_2 \tag{9}$$

The magnitude of $\mathbf{v}_{relP}$ is compared to a limiting velocity v_G at which the coefficient of static friction μ_H is no longer used and sliding friction μ_G is applied.

$$\mu = \begin{cases} \mu_H & \text{if } |\mathbf{v}_{relP}| < v_G \\ \mu_G & \text{if } |\mathbf{v}_{relP}| \geq v_G \end{cases} \tag{10}$$

Along with the magnitude of the contact normal force F_N the magnitude of the Friction force F_{Fmag} is calculated.

$$F_{Fmag} = \mu \cdot F_N \tag{11}$$

However, since the contact normal forces can be enormous a user-defined quantity F_{Fmax} is introduced, allowing to reduce the maximum assignable friction force if wanted, see (12).

$$F_{Fmag} = min(F_{Fmax},\ \mu \cdot F_N) \tag{12}$$

Using the negative normalized vector of the projected relative velocity the friction force vector can be computed.

$$\mathbf{F}_F = -F_{F\,mag} \cdot \frac{\mathbf{v}_{relP}}{|\mathbf{v}_{relP}|} \tag{13}$$

By default the coefficients of friction are equal for all bodies. However, since different body pairs might interact the user can specify frictional coefficient for each contact pair individually.

4 Application of the Contact Library

4.1 General information about application

In order to enable collision handling in the *M.MB* library the components *CollisionWorld* and *CollisonBody* are needed. *CollisionBody* has a multibody connector frame that needs to be connected to a body respectively another multibody frame, see Figure 7.

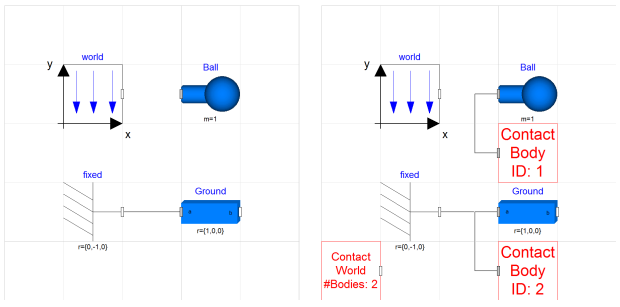
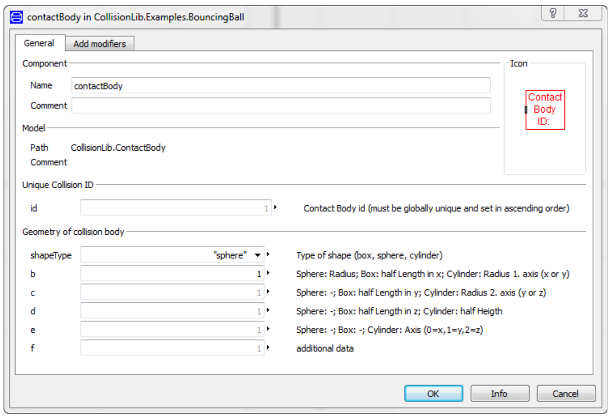


Figure 7: The model *BouncingBall* with and without collision consideration

Within the parameters of this component the user can specify the type of the collision shape and its geometrical parameters, see Figure 8.



Component

Name: contactBody

Comment:

Model

Path: CollisionLib.ContactBody

Comment:

Unique Collision ID

id	1	Contact Body id (must be globally unique and set in ascending order)

Geometry of collision body

shapeType	"sphere"	Type of shape (box, sphere, cylinder)
b	1	Sphere: Radius; Box: half Length in x; Cylinder: Radius 1. axis (x or y)
c	1	Sphere: -; Box: half Length in y; Cylinder: Radius 2. axis (y or z)
d	1	Sphere: -; Box: half Length in z; Cylinder: half Heigth
e	1	Sphere: -; Box: -; Cylinder: Axis (0=x,1=y,2=z)
f	1	additional data

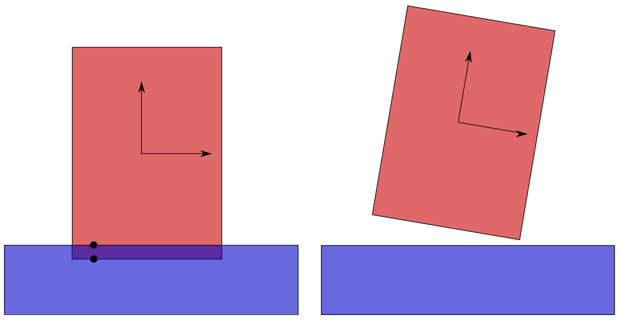
Figure 8: Parameters of the component *contactBody*

Each instance of *CollisionBody* needs a unique ID in ascending order starting at 1 (1,2,3,...). For the *CollisionWorld* the highest ID has to bet set as parameter in order to achieve "automatic" connections to the instances of *CollisionBody*. These connections are build using a *inner*-*outer*-coupling, meaning that an array of frame connectors is created for *CollisionWorld* with size of the passed parameter (maximum ID) and the *CollisionBody* is connected to the corresponding frame, based on the ID. Information about the shape is send from each instance of *CollisionBody* to the *CollisionWorld* via a user-defined connector. This is also done using *inner*-*outer*-coupling.

The problem concerning unique IDs has been discussed by Otter et al. in [1] before. However, planned changes in the Modelica language have not been implemented since then.

One of the bigger problems when handling contacts is the so called ghosting. This means, that one object moves through another object during one time step. This problem can only be solved by reducing the maximum solver step size. Alternatively within the *CollisionWorld* sampling can be activated, i.e. a collision check has to be performed during each sample. However, collision forces are also only calculated during each sample. This can result in wrong behavior, since the forces are constant between the samples.

The last big drawback when using the library *CollisionLib* are problems with collision detection for not strictly convex bodies. The GJK algorithm used by Bullet for many distance calculations, derives exactly one pair of deepest penetrating points, from the infinite number of pairs. This results in more or less unreal behavior, as pointed out in Figure 9. This issue



(a) configuration between cylinder (red with body-frame) and grounded box (blue) during contact with contact point pair (black)

(b) configuration after separation due to contact forces in the contact point pair

Figure 9: Wrong contact reactions due to an incongruous collision point pair

can be fixed by changing from deepest point penetration to intersection volume. However, a solution using the *Bullet Physics Library* is not known.

For some reason a user might want to ignore collision between a specific set of bodies. This feature is implemented by adding contact pairs (IDs) into a table within the component *CollisionWorld*.

4.2 Application 1: The pool table

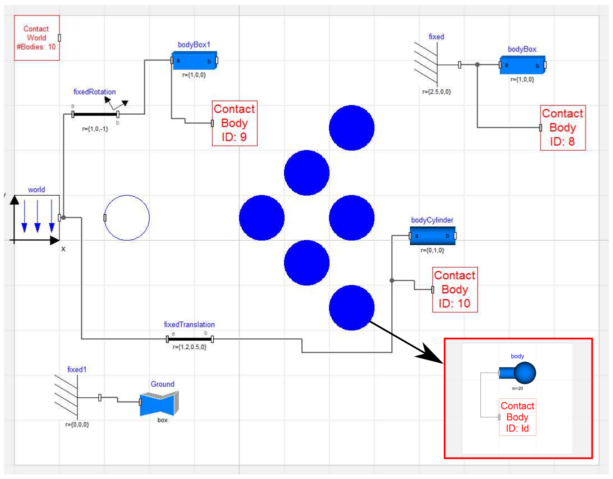


Figure 10: Setup of the model "billiard balls"

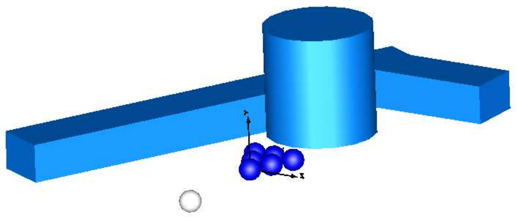
In Figure 10 a setup containing some "billiard balls" and other obstacles can be seen. While the obstacles are attached to the ground, the individual balls are free. Gravity of the system is set to zero. The white ball crashes into the group of balls, causing multiple collisions within the group of balls and the obstacles. In Figure 11 the system at time $t = 0$ s, $t = 0.25$ s and $t = 1$ s is shown.

Note that the conservation of momentum is not fulfilled in the system. This Error occurs due to the chosen penalty approach.

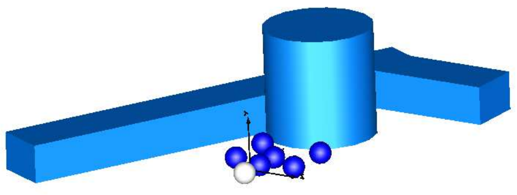
The computation time for this rather simple model is lower than the time simulated, which means for small models real-time simulation is possible.

4.3 Application 2: dynamically loaded compressing spring

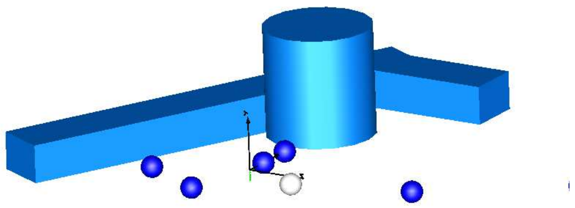
Apart from made-up models, the *CollisionLib* library has been used for the simulation of compressing springs. If simulating springs in high dynamic systems, like hydraulic valves, they can no longer be treated by the relation $F = c \cdot \Delta x$. Effects like coils hitting other coils or coils lifting of the spring cups need



(a) The system at time $t = 0$ s



(b) The system after multiple collisions at time $t = 0.25$ s



(c) The system after multiple collisions at time $t = 1$ s

Figure 11: The model "billiard ball" with multiple collision partners

to be considered. Using collisions this consequences can be handled.

In picture 12 the multibody representation of the spring is shown. It consists of multiple rigid bodies that are connected by spring-damper-elements. There are no joints connecting the rigid spring elements or the single elements to the spring cap.

Due to a force of 200 N acting for 0.01 s on the upper spring cap, the spring is compressed and the coils interact among one another. At time step $t = 0.03$ s the spring has almost reached its block length and starts to expand shortly after. At $t = 0.07$ s the elements moved beyond their starting position during expansion of the spring, see Figure 13.

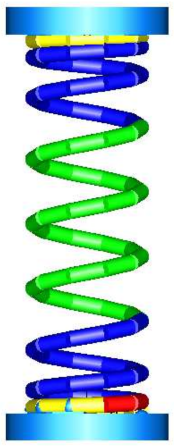
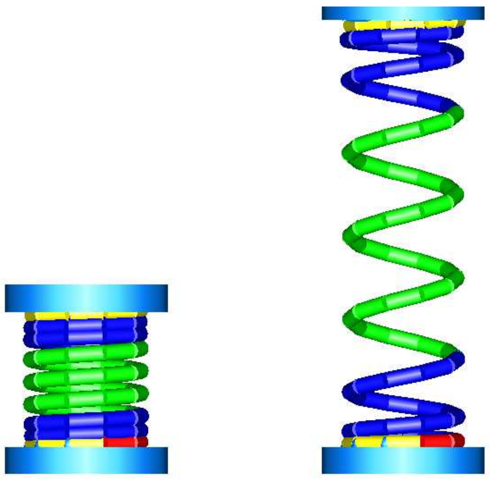


Figure 12: setup of a compressing spring model using the *CollisionLib* (t = 0 s)



(a) The spring at $t = 0.03\ s$ (b) The spring at $t = 0.07\ s$

Figure 13: States of the spring during simulation

Highly detailed modeling of systems, like the spring with a lot of self-interactions, can benefit from collision handling. However, simulation times can become very large. The spring model shown here takes seven hours for the complete simulation. Simulation time will be a subject of further investigation.

5 Outlook

In this paper the development of an add-on library to the *M.MB* for consideration of collision handling was given. The present version can be used to improve simulation, as shown in 4.3. However, there are some greater drawbacks that hinder the full potential of contacts in this early version of *CollisionLib*. Development in order to provide a fully functional collision library continues:

- Support for OpenModelica
- Enable calculation of more than one contact point pair within a contact.
- Finding a suitable solution for the ghosting effect (dynamically change of solver step size)
- Support of mesh representation as geometry information

6 Annotation

Help and cooperation to promote this project are very welcome. Feel free to send your ideas and offers to Andreas.Hofmann7@boschrexroth.de

References

[1] Otter, M. et al. Collision Handling for the Modelica MultiBody Library. In: Proceedings of the 4th Modelica Conference 2002, Hamburg, Germany, Modelica Association, 7-8 March 2005.

[2] Engelson, V. Integration of Collision Detection with Multibody System Library in Modelica. Linköping, Sweden: Thesis, Department of Computer and Information Science, Linköping University, 2000.

[3] Baraff, D. Dynamic Simulation of Non-Penetrating Rigid Bodies. Ithaca, USA: PhD thesis, Department of Computer Science, Cornell University, 1992.

[4] Mirtich, B. V. Impulse-based Dynamic Simulation of Rigid Body Systems. Berkleley, USA: Ph.D. thesis, Graduate Division, UC Berkeley.

[5] Glocker, Ch. Dynamik von Starrkörpersystemen mit Reibung und Stößen. In: VDI-Fortschrittsbereichte Mechanik/Bruchmechanik, 1995

[6] Beitelschmidt, M. Reibstöße in Mehrkörpersystemen. Munich, Germany: Ph.D. thesis, Lehrstuhl B für Mechanik, 1998.

[7] Schmitt, A. A. Dynamische Simulation von gelenkgekopppelten Starrkörpersystemen mit der Impulstechnik. Karlsruhe, Germany: internal report, Department of Computer Science, Karlsruhe Institute of Technology, 2003.

[8] Bullet Physics Library. available at: www.bulletphysics.com.

[9] Coumans, E. Bullet 2.80 Physics SDK Manual. URL: http://www.cs.uu.nl/docs/vakken/mgp/assignment/Bullet%20-%20User%20Manual.pdf, 2012.

[10] Machado, M. et al. Compliant contact force models in multibody dynamics: Evolution of the hertz contact theory. In: Mechanism and Machine Theory 53 (2012), pages 99-121.

[11] Gilbert, E.G., Johnson, D.W. and Keerthi, S.S. A fast procedure for computing the distance between complex objects in three-dimensional space. In: IEEE Journal of Robotics and Automation 4(2), pages 193-203, 1988.

[12] Cohen, J.D. et al. I-Collide: An Interactive and Exact Collision Detection System for Large-Scale Environments. In: Proceedings of the 1995 Symposium on Interactive 3D Graphics 189ff, Monterey, CA, USA, 1995.