Proceedings
of the 4th International Modelica Conference,
Hamburg, March 7-8, 2005,
Gerhard Schmitz (editor)

M. Otter, H. Elmqvist, J. Díaz López
Dynasim AB, Sweden; DLR Oberpfaffenhofen, Germany

Collision Handling for the Modelica MultiBody Library

pp. 45-53

Collision Handling for the Modelica MultiBody Library

Martin Otter[2], Hilding Elmqvist[1], José Díaz López[1]
[1]Dynasim AB, Lund, Sweden, {Elmqvist, Jose.Diaz}@Dynasim.se
[2]DLR Institute of Robotics and Mechatronics, Oberpfaffenhofen, Germany, Martin.Otter@DLR.de

Abstract

The Modelica MultiBody library is extended with collision handling. It is demonstrated how to use this new feature. Different implementations are explained based on parametric surfaces, on surfaces described by algebraic constraints, and on surface descriptions by primitives and triangles using the collision package SOLID 3.5. Furthermore, the response calculation by a resultant contact force and torque is discussed.

1 Introduction

Modeling of contacts between mechanical objects is important in many disciplines such as wheel handling for vehicle dynamics, robot gripping, CAM modeling, etc. The Modelica.Mechanics.MultiBody library [13] is extended with support for collision handling. The user interface and the implementation variants are discussed in the next sections.

Describing collisions between mechanical bodies is still a difficult topic. The solution can be divided into two main steps:

(1) Collision detection of surfaces, determining features such as shortest distances, penetration depths and contact normal vectors. Several software systems are available for this task, e.g., SWIFT [6], ODE [16] or SOLID [2][3]. This is also an important part of CAD and FEM systems. Fast real-time solutions are mainly driven by the game industry due to their particular needs [5][16].

(2) Calculation of the contact response. Several quite different approaches are in use:

(2a) The response is computed in an idealized way using impulses based on an impact law such as Poisson’s hypothesis: relating the impulses of the compression and decompression phase of an impact to each other, see, e.g., [15][10][5]. The main advantage is that only few constants are needed to describe the impact law and that the integrator step size is not influenced by the response calculation because it is performed in an infinitely small time instant. The disadvantages are that such idealized impact laws are only valid for stiff collisions and that the constants of the impact law cannot be computed by material properties of the colliding bodies, i.e., they must be determined by measurements. Furthermore, it is quite involved to compute the new initial conditions after an impact in a robust way, especially if *several* surface *contacts* are present at the same time instant. In the latter case either no or infinitely many solutions may exist using impulse descriptions. For a physically meaningful response there are cases where multiple impacts have to be applied one after each other whereas also cases are present where they must be applied altogether (for a more thorough discussion, see [5], pp. 256 – 264).

(2b) The response is computed by a simple elastic spring/damper element. E.g., the spring force is just proportional to the penetration depth. The advantages of this approach are its simplicity and that it can be used for stiff and soft contacts. This approach works also reasonably well if several contact points are present at the same time instant. The disadvantage is that the integrator step size is reduced significantly in the contact phase in order to catch the rapidly changing contact forces and torques. A necessary and harder task is to determine experimentally the spring and damper constants. Those are in consequence only valid in situations close to the experimental conditions. The main reason is that the contact force is not only proportional to the penetration depth but rather to the contact area and the contact volume.

(2c) The response is computed by taking into account the contact area and the contact volume. This might be performed by a discretization of the contact area or the contact volume, see, e.g., [8][9]. Contrary to (2b), the force and torque computation will be more precise and the material properties, such as the E-modul and the contraction number ν, can be used to calculate the spring constants. Furthermore, the contact torque can be calculated in a reasonable way. This torque is particularly important for gripping operations.

(2d) The response is computed for special situations, e.g., for wheel/road contact [14]. Solutions that are specialized to a particular contact problem are usually more precise and practically applicable as the generic solutions of (2a,b,c).

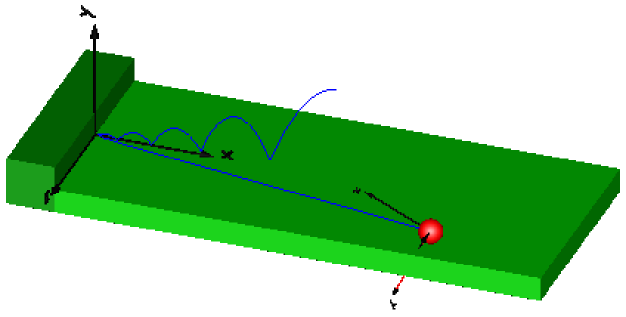
At the moment, it is not possible to implement the solution with impulses (2a) in a *generic* way in Modelica. For special cases it can be implemented with the reinit(..) operator. The reason is that an appropriate Modelica language element is missing as well as the needed symbolic algorithms for the most general cases. In the European project "RealSim" basic research was carried out to handle models with *varying index* and with *dirac impulses*. The latter might be implicitly occurring at switching points where the number of states, and therefore also the DAE index, is changing. For certain classes of systems a reasonable solution method was developed [11]. Still, the algorithms are in a research stage.

For this reason, in this article only elastic response actions according to (2b) and (2c) are taken into account. A specialized solution for wheel/road contact is already available in the Modelica VehicleDynamics library [1].

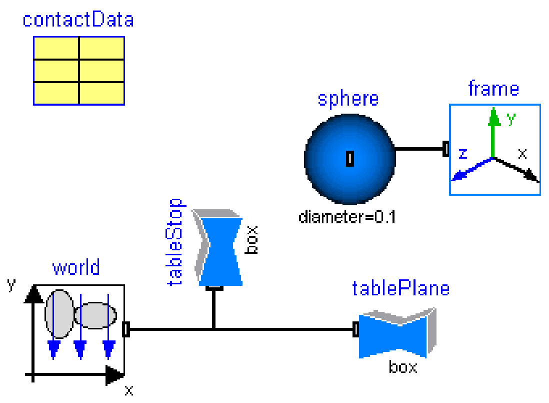
2 Users View

In this section the user's view of the library is shortly sketched. This view is independent of the implementation variants discussed in subsequent sections.

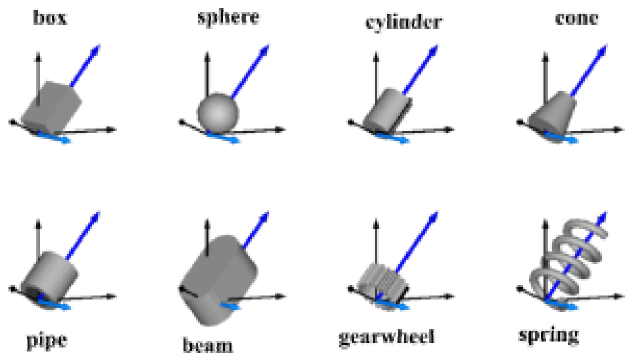
Components are provided to handle collisions between bodies using elastic force/torque laws at contact points. An example is shown in the following figure where a ball is thrown on a table. The ball first bounces on the table, then into the wall and finally rolls on the table.



This example is defined by the Modelica model of the next figure. In brief, there is a modified MultiBody.**World** component with a new subcomponent named "collisionHandling". This new object performs collision detection and contact response calculations. The table is defined by two boxes of the component "MultiBody.Visualizers.FixedShape".

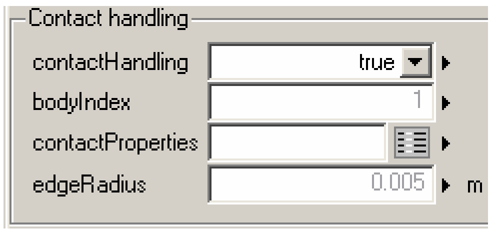


The ball is described by a sphere of the component "FixedShape" together with body properties, such as mass and inertia. The record "contactData" contains material constants that are used in the table and sphere components. No special collision handling objects are needed. Instead, the existing "MultiBody.Visualizers.**FixedShape**" component has been modified to optionally detect and treat collisions for the supported visual shapes shown in the next figure:



For collision detection, shapes "pipe", "gearwheel" and "spring" are treated as full cylinders. There are currently limitations for shapes using "*.dxf" files (AutoCAD R12 descriptions): Only sets of triangles are supported and only one contact point between two surfaces is taken into account, although more contact points might be present for non-convex objects. Note, all objects can be scaled in the 3 coordinate axes by providing length, height and width of the shape. E.g., ellipsoids are also supported by defining shapeType="sphere" and appropriate length, height, and width scaling.

The following new parameters are present in a FixedShape object:



Contact handling

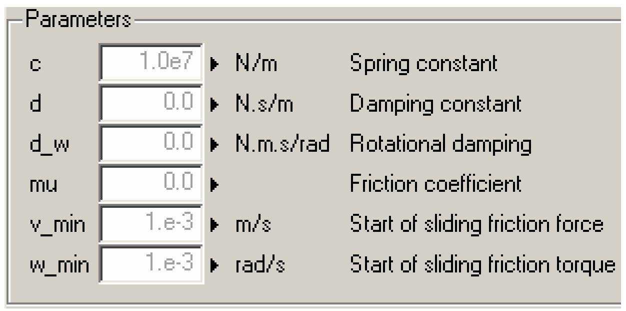
contactHandling	true	
bodyIndex	1	
contactProperties		
edgeRadius	0.005	m

The collision handling has to be explicitly activated by setting "**contactHandling** = **true**". The effect in this case is that the distance between this object and all other objects that have contactHandling = true is continuously computed and monitored. When the distance between two objects becomes zero, an event is triggered and a contact response is applied.

If two FixedShape objects are rigidly attached to each other (see, e.g., the two boxes representing the table in the example above), a contact would permanently be present. To avoid this, all objects are reduced in size by a factor of "1 - 1.e-9" for the collision detection. As a consequence, shapes that are fixed together, do not lead to an unnecessary contact response computation.

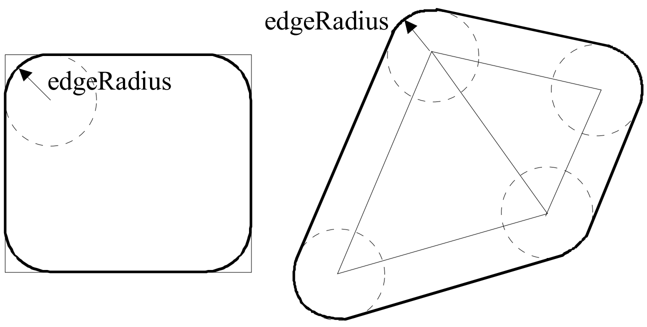
Parameter "**bodyIndex**" is a unique Integer identifier for each "FixedShape" object. For instance, if 4 FixedShape objects with contactHandling = true are present, they must have bodyIndex = 1, 2, 3, and 4. Additionally, in the "World" object, parameter "nContactBodies" has to be set to the number of FixedShape objects with contactHandling = true. In the example above, nContactBodies = 4 is required. There is currently a Modelica language enhancement under development, in order that this user input is no longer required since it can be automatically deduced by a Modelica translator.

The data for the response calculation are provided via parameter record "**contactProperties**", see next figure. It defines material data of the corresponding surface. The type of response calculation used for all collisions is defined in the World object: If parameter simpleResponse = true, a linear spring and a linear damper force acts in contact normal direction. Additionally, linear rotational damping proportional to the relative angular velocity is present in contact normal direction, and a sliding friction force acts in opposite direction to the tangential sliding velocity at the contact point. If simpleResponse = false, the contact area is discretized and a resultant force and torque is computed by summation of appropriate forces over the contact area. The latter option is currently under development. More details of the force/torque calculations are given in section 4.5.

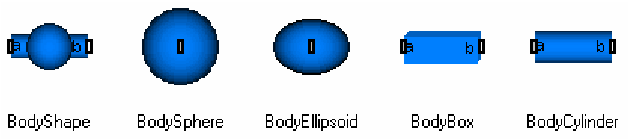

Parameters

c	1.0e7	N/m	Spring constant
d	0.0	N.s/m	Damping constant
d_w	0.0	N.m.s/rad	Rotational damping
mu	0.0		Friction coefficient
v_min	1.e-3	m/s	Start of sliding friction force
w_min	1.e-3	rad/s	Start of sliding friction torque

Finally, parameter "**edgeRadius**" defines how much the edges of primitive shapes, such as boxes, cylinders etc., are "rounded" with spheres, see left part of figure below. For "*.dxf" files, a *layer* of spheres with radius "edgeRadius" is put on the surfaces to get a smooth surface description, too, see right part of figure below. The edge rounding and the "layer of spheres" is used for collision detection and response calculation. It is currently **not** shown in the rendering (animation). It is recommended to use a non-zero edgeRadius because the collision detection will be usually faster and more robust. Still, it is possible to set edgeRadius=0. The technique of smoothing the surfaces with spheres is from [3][2].



In sublibrary MultiBody.Parts the available body components have now also optional collision handling support. Furthermore, new body types have been added, as shown in the next figure:



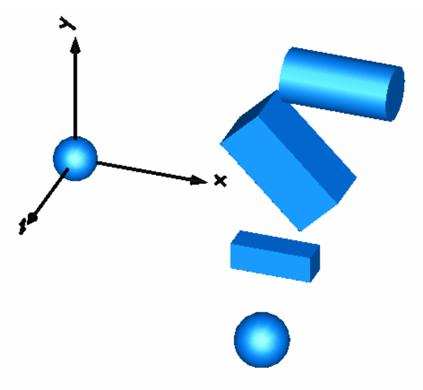
For example, "BodyEllipsoid" is a part that defines an ellipsoid by length, width, height and material properties. From this information, the body properties (mass, center of mass, inertia tensor) are computed and the rendering and collision handling is deduced.

3 Applications

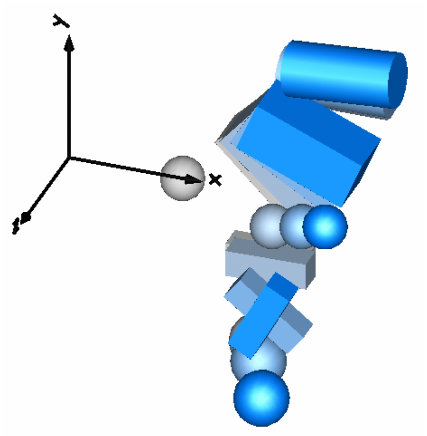
In this section some applications of the library are shown.

3.1 Free Flying Objects

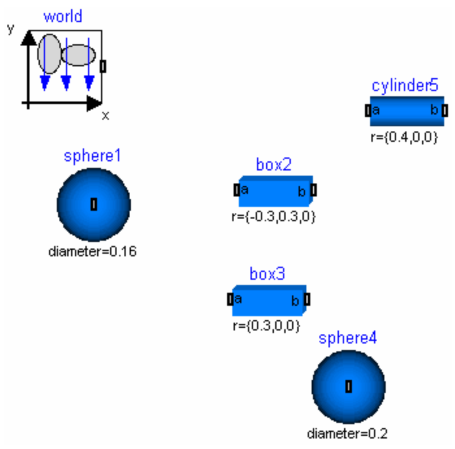
Five different free flying objects are colliding with each other. The start configuration is shown in the next figure. The 4 objects on the right are in rest at the beginning and the sphere at the left side is flying in the direction of the other objects.



Several collisions between all objects occur after a few seconds:

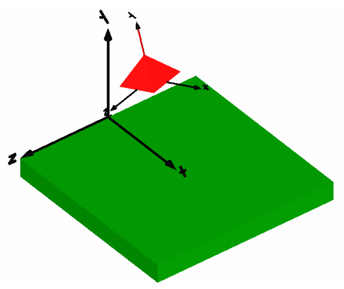


This system is defined with the following Modelica model:

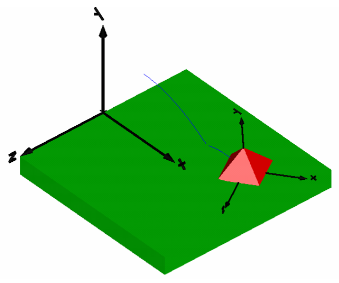


3.2 Collision of triangularized surface with a table

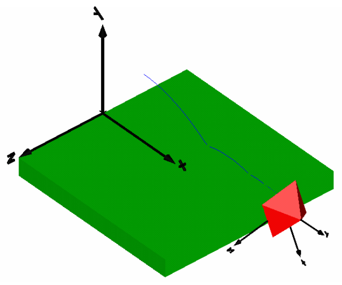
A simple application of AutoCAD files in the library is shown in the following example: An AutoCAD generated pyramid is rotating around his main axis and falls to a table.



After some time we observe how the trajectory of the pyramid evolves. Due to friction, the velocity and angular velocity of the pyramid is permanently reduced.



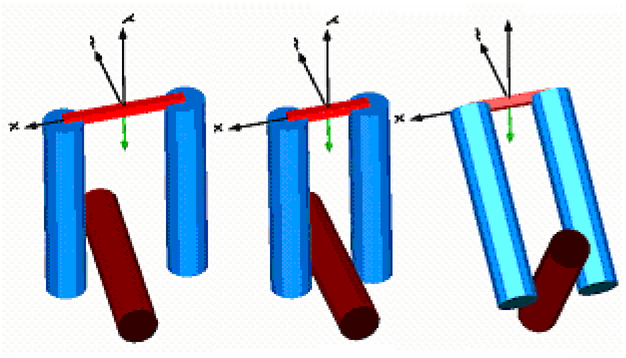
The final position after gliding over the surface is shown in the next figure



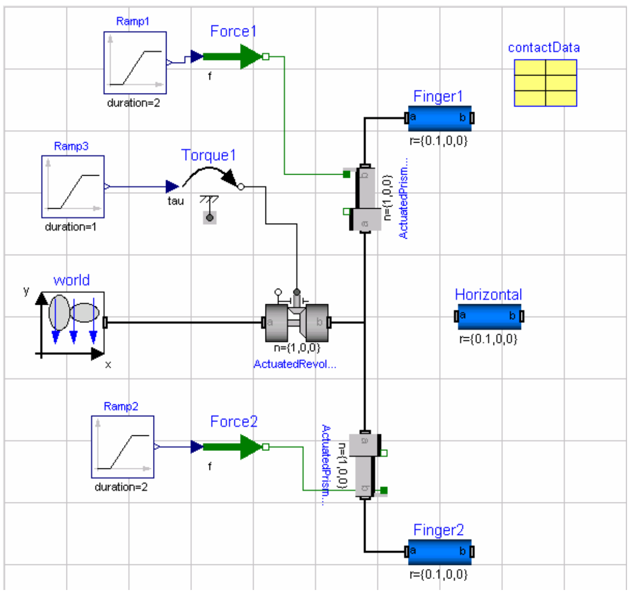
as the pyramid is falling over the edge of the table with a velocity that is almost zero.

3.3 Gripping

The sequence of images below shows two blue fingers gripping a lying red object (all cylindrical dxf-defined objects). The fingers are attached with prismatic joints to a revolute joint. The lying object is gripped since it is squeezed between the fingers. Due to the friction torque between the surfaces, the red cylinder is elevated after gripping it and rotating the revolute joint.



The model of this experiment setup is shown below. “Horizontal” is the lying red cylinder and “Finger1” and “Finger2” are the two fingers. This example shows the important role of the friction torque. If this feature would not be present in the model, then the red cylinder would rotate after gripping.



4 Implementation

In this section implementation details and variants for the collision handling are discussed.

4.1 Central Collision Handling

Since distances and contact response calculations are needed between any two collision objects, a central collision handling is present in the modified World component. In order that this is possible, every collision object needs a unique Integer identifier. The surface data, position, orientation, forces and torques are copied in appropriate arrays using the corresponding Integer identifier as index in these arrays. Currently, this identifier has to be provided manually by the user as shown in section 2.

As already mentioned, a Modelica language enhancement is under development to get rid of this unnecessary user input. The current plan is to introduce the new dimension qualifier “each” and the new operator “uniqueElement(..)” to automatically provide a unique array index for objects. Examples:

```
Real vec[each,5,3];
Real subvec[5,3] = uniqueElement(vec);
Real x[each];
Real xv = uniqueElement(x);
```

The following rules apply:

(1) If a public array component, A, is declared using the subscript [each], e.g. Real A[each], it has the same access restriction as though it were protected except the "uniqueElement" operator can be applied to the array. This is the only allowed use of the uniqueElement operator and the only allowed use of the array name outside the declared scope.

(2) The size of all array components with declared size [each] starts at zero and is then increased as follows. This is performed before size() of the array can be determined (e.g. to determine the size of other arrays).

(3) For each use of the uniqueElement(..) operator the size of the array component is increased by one and a unique element of the array is referenced.

These new language elements will allow an implementation where the unique collision object indices are automatically derived without requiring them from the user. This feature will be also useful for other purposes.

4.2 Variant 1: Parametric surfaces

The first implementation variant for the collision handling uses parametric surfaces. That is, the absolute position vector **r** to each surface point is described as a vector valued function of two parameters, called α and β.

$$\mathbf{r} = \mathbf{r}(\alpha, \beta)$$

Two tangent vectors $\mathbf{e}_\alpha$ and $\mathbf{e}_\beta$ are defined by partial derivatives from which the normal **n** to the surface can be computed:

$$\mathbf{e}_\alpha = \frac{\partial \mathbf{r}}{\partial \alpha} = \mathbf{r}_\alpha(\alpha, \beta)$$

$$\mathbf{e}_\beta = \frac{\partial \mathbf{r}}{\partial \beta} = \mathbf{r}_\beta(\alpha, \beta)$$

$$\mathbf{n} = \mathbf{e}_\alpha \times \mathbf{e}_\beta = \mathbf{n}(\alpha, \beta)$$

Both position and orientation of the surface patch close to a possible contact point are defined as functions of α and β. The tangential planes of two surfaces that are potentially colliding are constrained to be parallel, i.e., their normal vectors are parallel (quantities belonging to surface “a” are denoted by superscript “a”, e.g., x^a):

$$\mathbf{n}^a(\alpha^a, \beta^a) + \lambda_1 \cdot \mathbf{n}^b(\alpha^b, \beta^b) = 0$$

Furthermore, the relative position vector of the contact point candidates is constrained to be parallel to the surface normals:

$$\mathbf{r}^b(\alpha^b, \beta^b) - \mathbf{r}^a(\alpha^a, \beta^a) = \lambda_2 \cdot \mathbf{n}^a(\alpha^a, \beta^a)$$

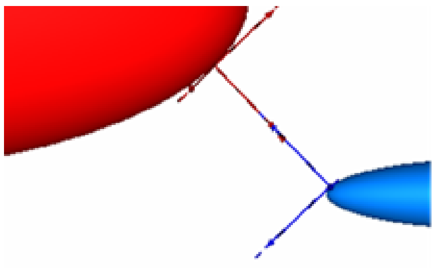
These constraints constitute 6 scalar equations in the unknown variables $\alpha^a, \beta^a, \alpha^b, \beta^b, \lambda_1, \lambda_2$. These equations are in general nonlinear and have to be solved per each potential collision point pair. We may encounter computational problems since multiple solutions may exist. For closed surfaces, at least 4 different solutions exist (closest-closest, closest-farthest, farthest-closest, farthest-farthest). A feasible solution must have a positive scalar product between the surface normal and the relative distance:

$$\mathbf{n}^a(\alpha^a, \beta^a)\cdot(\mathbf{r}^b - \mathbf{r}^a) > 0$$

For convex surfaces, the solution of the above nonlinear system of equations, taking into account the inequality constraints, gives the two closest points when the bodies are apart. It is possible to track the correct closest points also for non-convex surfaces provided good starting values are given for the unknown variables.

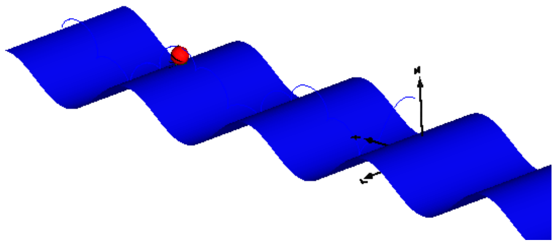
The local coordinate systems at these points as well as the relative position vector are illustrated below as rendered by Dymola during animation

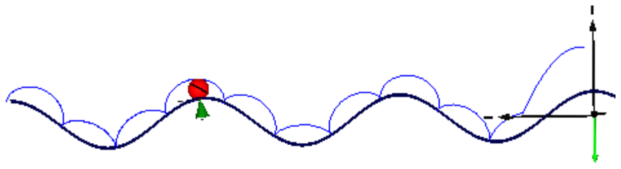
.



Certain special surfaces such as ellipsoid, plane and parabola have been implemented as parameterized surfaces. It is easy to add other surfaces by defining corresponding parametric functions.

An example for a non convex parameterized surface is shown below where a ball is thrown towards left on a “cosine” surface.





The side view shows the trajectory of one point on the ball. The ball oscillates forth and back in the leftmost valley with the fixed point following the same path.

The major drawback of closed parameterized surfaces is the occurrences of singular points. For example, every closed surface has at least one singular point where the calculation of the normal vector fails because at least one of the tangent vectors becomes zero. For a sphere, these are the “north” and “south” pole of the sphere. Therefore, in general it is not possible to get a robust solution of the non-linear system of equations. For special cases where it is guaranteed that the singular points are outside of the operation region, a parametric surface might be used without re-parameterization, see, e.g., [12] that demonstrates collision handling of a CAM. However, for a generic collision handler, parametric surfaces are difficult to treat, since singular points require a re-parameterization of the surface description.

4.3 Variant 2: Algebraic constraint surfaces

An alternative is to describe especially closed surfaces with constraint equations

$$0 = h(\mathbf{r})$$

For example, a sphere can be defined by

$$h(\mathbf{r}) = x^2 + y^2 + z^2 - 1$$

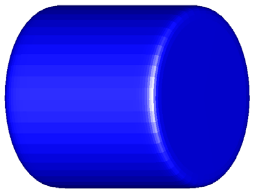
An approximate representation of a box can be made by

$$h(\mathbf{r}) = x^{20} + y^{20} + z^{20} - 1$$

and an approximation to a cylinder can be made by

$$h(\mathbf{r}) = x^{20} + (y^2 + z^2)^{10} - 1$$

as shown below.



By choosing higher values of the exponent, the edges get sharper. It is possible to define cones and pyramids as well by such closed formulas. However, it is difficult to find the closed formula for arbitrary surfaces.

This representation allows the smooth normal to be calculated as the gradient

$$\mathbf{n}(\mathbf{r}) = grad\left(h(\mathbf{r})\right) = \left\{\frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}, \frac{\partial h}{\partial z}\right\}$$

It should be noted that there are no singular points when using this surface representation. Inserting the definition of the normals into the same constraint equations as used in variant 1, and considering the constraint equations for each surface, 8 scalar equations in the unknown variables $\mathbf{r}^a, \mathbf{r}^b, \lambda_1, \lambda_2$ are obtained. Note, that start values are easier to give in this representation since the position vectors themselves are unknowns.

This approach has the advantage to get smooth surface descriptions. The drawback are the highly nonlinear equations closed to the edges, especially for high exponents.

In both variant 1 and 2, partial derivatives of functions defining the surfaces are required in the model code. This can be achieved by a special operator in the Modelica code and automatic differentiation. For details see, [12].

4.4 Variant 3: Collision handling with SOLID

As a third variant, the collision detection of shapes and the computation of the penetration depth between shapes is performed with the software system SOLID 3.5 [3][2]. The SOLID software is free for non-commercial purposes. Commercial use requires a license. The SOLID software supports collisions between primitives such as spheres and cylinders, as well as between complex convex and non-convex objects described by a set of polytopes (point, line segment, triangle, tetrahedron, convex polygons, and convex polyhedrons).

The software provides a good interface to define "response functions" that are called when contact happens. In these response functions, contact forces and torques could be programmed. The disadvantage of this interface is that the integrator does not have information about occurred collisions and reduces the step-size around a collision only due to the sharp changes in contact response. Experiments showed that it is difficult to get a robust solution. In general, the integrator may stop for a corrector failure. The reason is that integrators require that the equations describing the system are continuous with a smooth first or higher derivative. At a contact point, these assumptions are not fulfilled and since the changes in the contact forces and torques are so drastic, it is understandable that an integrator may fail. This is also the reason why slight changes in the tolerances of the integrator or the tolerances of the contact detection may change the simulation time very significantly.

For this reason, another interface of SOLID is used to explicitly compute either the distance of two objects or the penetration depth of two colliding objects. This allows to compute indicator functions for the root finder of an integrator, in order that an event is generated when contact occurs. The solution is more robust and usually more efficient than the solution with "response functions".

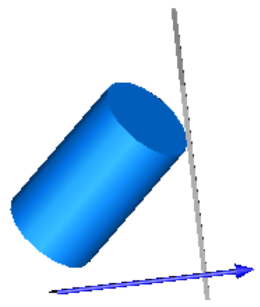
The main disadvantage is that the calling environment has to perform all distance function calls by its own. In the current implementation a brute force method is used by computing the distances between all defined objects. The "broad phase" present in the "response function" interface of SOLID to reduce the number of distance tests significantly (based on "axis-aligned-bounding-boxes" approximations of the objects), is not present with the chosen "root-finding" approach. This will be improved in the future.

The SOLID package uses a generalized version of the GJK algorithm [7] to compute the distances between convex polytopes in a finite number of steps and for other convex surfaces converges globally with a fast convergence rate. For the penetration depth calculation an algorithm is used that is based on similar principles as the GJK algorithm. Details are described in the book [2].

In order that the two algorithms can be applied, a "support mapping" of the corresponding surface is needed. This is a function s_A that maps a vector **n** to a point on a convex shape A according to:

$s_A(\mathbf{n})$ returns a point on the surface of A such that "$\mathbf{n}\cdot s_A(\mathbf{n}) = \max(\mathbf{n}\cdot\mathbf{r}$ for all $\mathbf{r}$ in A)"

This definition is visualized in the next figure for a cylinder:



The arrow in this figure is vector "**n**" of the support mapping. This vector is proposed and changed by the distance and penetration depth algorithm. The "grey" shape in the figure above is a plane that is perpendicular to "**n**" and is moved to the cylinder such that the plane touches the cylinder. The support mapping function has to return the coordinates of this touching point. If this is not unique, one of the points is returned. For a smooth surface this just means that a point **r** on the surface is defined as a function of its normal **n**: $\mathbf{r} = \mathbf{r}(\mathbf{n})$.

Due to this simple basic definition of a convex object via a support mapping, a user can introduce additional base shapes in a simple way.

The "penetration depth" algorithm adds in every iteration an additional edge to a simplex that defines the penetration volume. From this simplex, two points are selected in such a way that both points are on the surface of the respective shape and the distance between these two points is as small as possible. When moving the two collided objects along the connection line of these two points, until the two points coincide, then the two shapes are in touching contact. These two points are reported as result of the penetration depth calculation. The penetration depth is then the distance between these two points and the contact normal is on the connection line along these two points.

The SOLID interface functions are used in the World.collisionHandler component that has the following basic structure:

```
equation
  // Compute signed distances
  (signedDistance, ...) =
                     surfaceDistances(..);

  // Generate event when distance is zero
  for i in 1:nContactPairs loop
    contact[i] = signedDistance[i] < 0.0;
  end for;

  // Contact response calculation
  (frame_a_f, ...) =
              contactForces(contact, ...);
```

Function "**surfaceDistances**(...)" returns vector "signedDistance". An element of this vector signals the shortest distance of two objects that are not yet in contact by a positive value. A negative element characterizes the penetration depth of two objects that are in contact. The for loop in the code fragment above triggers events whenever two objects get in contact and whenever two objects are separating. Finally, the function "**contactForces**(...)" is used to perform the response calculation. It returns the resultant forces and torques acting at appropriate reference frames of the corresponding surfaces.

4.5 Response calculation

Contact forces and torques are applied when the relative distance along the normal vector is negative, signaling an interpenetration. As already shortly discussed in section 2, two different response calculations are provided: The first one uses simple spring/damper elements. The second one discretizes the contact area and a resultant response force and torque is computed by summation of appropriate forces over the contact area. This more precise calculation is currently under development. In the following, the first option is discussed in some more detail:

The response is computed according to the following equations:

$$f_n = \min(c_m \cdot s + d_m \cdot v_n,\ 0)$$

$$f_t = \mu_m \cdot |f_n| \cdot \begin{cases} 1 & \textbf{if}\ |v_t| > v_{min,m} \\ |v_t| / v_{min,m} & \textbf{else} \end{cases}$$

$$\tau_n = d_{w,m} \cdot |f_n| \cdot \begin{cases} -1 & \textbf{if}\ \omega_n < -\omega_{min,m} \\ 1 & \textbf{if}\ \omega_n > \omega_{min,m} \\ \omega_n / \omega_{min,m} & \textbf{else} \end{cases}$$

$$\mathbf{f} = f_t \cdot \mathbf{e}_t + f_n \cdot \mathbf{e}_n$$

$$\boldsymbol{\tau} = \tau_n \cdot \mathbf{e}_n$$

where

f_n	contact force in normal direction
f_t	contact force in tangential direction
τ_n	contact torque in normal direction
$\mathbf{f}$	resultant contact force
$\mathbf{t}$	resultant contact torque
s	penetration depth (≤ 0)
v_n	relative velocity in normal direction
v_t	relative velocity in tangential direction
ω_n	relative angular velocity in normal direction
$\mathbf{e}_n$	unit vector in normal direction
$\mathbf{e}_t$	unit vector in tangential direction

In other words, a linear spring/damper element is used to compute the force in contact normal direction. In tangential direction a sliding friction force is taken into account, if the tangential velocity is larger as v_{min}. Below v_{min}, the friction force is reduced so that it is zero, when the tangential velocity becomes zero. Sticking is currently not implemented. For gripping operations, it is important to take into account the friction torque. This is accomplished by a linear rotational damper that is proportional to the normal force and the relative angular velocity. Finally, all force and torque parts are summed up resulting in the contact force and torque. Note, if the normal force would become positive since the damping part is too large, it is reduced to zero, since a positive normal force is physically not correct.

For the equations above material constants are needed, e.g., for the spring and the dampers. However, only material data for the respective surfaces are provided. The correct solution would be to apply, say, (1) a spring/damper element on surface A using the material constants of surface A, (2) a spring/damper element on surface B using the material constants of surface B and (3) connect the spring/damper elements of surfaces A and B in series. Due to the linear dampers, this would result in additional differential equations depending on the number of contact points. To avoid complications and to enhance efficiency, the following approximation is used: A resultant spring constant is computed from the surface data under the assumption of a series connections of two springs. For all other data, mean values are used:

$$\begin{aligned}
c_m &= \frac{c_A \cdot c_B}{c_A + c_B} \\
d_m &= \tfrac{1}{2}\cdot\left(d_A + d_B\right) \\
d_{w,m} &= \tfrac{1}{2}\cdot\left(d_{w,A} + d_{w,B}\right) \\
\mu_m &= \tfrac{1}{2}\cdot\left(\mu_A + \mu_B\right) \\
v_{min,m} &= \tfrac{1}{2}\cdot\left(v_{min,A} + v_{min,B}\right) \\
\omega_{min,m} &= \tfrac{1}{2}\cdot\left(\omega_{min,A} + \omega_{min,B}\right)
\end{aligned}$$

5 Outlook

An overview was given, in which way the Modelica MultiBody library is extended with collision handling. The current stage is already useful for applications. Development continues to improve the collision handling:

- Using the "broad-phase" of SOLID to reduce the number of collision tests significantly.
- Support more than one contact point between two surfaces. This is important for non-convex surfaces.
- Optionally, provide a more detailed response calculation by discretization of the contact area.
- Reduce the limitations of "*.dxf" files.

Acknowledgements

Partial financial support of this work by the Toyota Motor Corporation is gratefully acknowledged.

References

[1] Andreasson J. (2003): **Vehicle Dynamics library**. Proceedings of Modelica'2003, ed. P. Fritzson, pp. 11-18. Download: http://www.Modelica.org/-Conference2003/papers.shtml/ h28_vehicle_Andreasson.pdf

[2] Bergen G. van den (2004): **Collision Detection in Interactive 3D Environments**. Elsevier and Morgan Kaufmann Publishers.

[3] Bergen G. van den (2003): **SOLID 3.5 - User's Guide to the SOLID Collision Detection Library**. On the CD of the book of G. van den Bergen [2].

[4] Dynasim (2005): **Dymola – Users Manual** (http://www.dynasim.com)

[5] Eberly D.H., and Shoemake K. (2004): **Game Physics**. Elsevier and Morgan Kaufmann Publishers.

[6] Ehmann S. H. (2000): **SWIFT - Speedy Walking via Improved Feature Testing**. Version 1.0. Download: http://www.cs.unc.edu/~geom/SWIFT/

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

[8] Hasegawa S., and Sato M. (2004): **Real-time Rigid Body Simulation for Haptic Interactions Based on Contact Volume of Polygonal Objects**. EUROGRAPHICS 2004, ed. M.-P. Cani and M. Slater, Volume 23, Number 3. Download: http://eg04.inrialpes.fr/Programme/Papers/PDF/paper1209.pdf

[9] Hippmann G. (2003): **An Algorithm for Compliant Contact between complexly shaped Surfaces in Multibody Dynamics**. Multibody Dynamics, Jorge A.C. Ambr´osio (Ed.), IDMEC/IST, Lisbon, Portugal, July 14. Download: http://www.pcm.hippmann.org/-doc/eccomas03_hippmann.pdf

[10]Leine R.I., and Glocker C. (2003): **A set-valued force law for spatial Coulomb–Contensou friction**. European Journal of Mechanics A/Solids 22, pp. 193–216.

[11]Mattsson S.E., Olsson H., and Elmqvist H. (2001): **Methods and Algorithms for Varying Structure Hybrid DAE Simulation**. EC IST Project RealSim under contract IST-1999-11979, Internal Report 2.2.

[12]H. Olsson, H. Tummescheit, H. Elmqvist (2005): **Modeling with Partial Derivatives of Modelica Functions and Automatic Differentiation**. Modelica'2005 conference, Hamburg, March 7-8.

[13]Otter M., Elmqvist H., Mattsson S.E. (2004): **The New Modelica MultiBody Library**. Proceedings of Modelica'2003, ed. P. Fritzson, pp. 311-330. Download: http://www.Modelica.org/Conference-2003/papers.shtml/h37_Otter_multibody.pdf

[14]Pacejka H.B. (2002): **Tyre and Vehicle Dynamics**. Butterworth-Heinemann.

[15]Pfeiffer F., and Glocker C. (1996): **Multibody Dynamics with Unilateral Contacts**. John Wiley & Sons.

[16]Smith R. (2004): **Open Dynamics Engine – V0.5 Users Guide**. Download from http://ode.org.

[17]Wriggers P. (2002): **Computational Contact Mechanics**. John Wiley & Sons.