Collision Handling with Elastic Response Calculation and Zero-Crossing Functions

Andrea Neumayr; Martin Otter

Collision Handling with Elastic Response Calculation and Zero-Crossing Functions

Andrea Neumayr
Martin Otter
andrea.neumayr@dlr.de
martin.otter@dlr.de
DLR Institute of System Dynamics and Control
Oberpfaffenhofen, Germany

ABSTRACT

The prototype Modia3D is used to test and evaluate ideas for modeling and simulating larger and more complex 3-dimensional systems than it is possible with a pure equation-based modeling system such as current Modelica. Collision handling in Modia3D is performed on convex geometries with elastic response calculation using an improved formulation of the contact forces and torques. The computed penetration depths and Euclidean distances are utilized in a novel way as zero-crossing functions. The resulting differential algebraic equations are solved with a variable-step solver.

KEYWORDS

collision handling, Julia, Modia, Modia3D, Modelica, Minkowski Portal Refinement algorithm, elastic force law, response calculation, zero-crossing function, hysteresis

ACM Reference Format:
Andrea Neumayr and Martin Otter. 2019. Collision Handling with Elastic Response Calculation and Zero-Crossing Functions. In *9th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT '19), November 5, 2019, Berlin, Germany.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3365984.3365986

1 INTRODUCTION

The Modelica standard library[1] supports the modeling of 3-dimensional multi-body systems with its sub library Modelica.Mechanics.-MultiBody [29]. There have been several attempts to improve this library with regards to visualization, collision handling or support of larger models, for example [2, 8, 16, 19, 28]. Over the years it was recognized that the technology of current Modelica has some natural limitations.

Therefore, the open source Modia[2] [9, 10] project was launched as a domain-specific extension of the Julia programming language[3] [5]. The equation-based modeling language is called Modia as well as the experimental modeling environment consisting of several Julia packages[4]. The intention is to utilize the results of this prototyping in the design of the next Modelica language generation. Julia allows to implement numerical algorithms conveniently on a high level. It supports modern data structures, multiple dispatch, metaprogramming, has a just-in-time-compiler and benchmarks demonstrate that a similar performance can be reached as with C.

One part of the Modia project is Modia3D[5] [23–26]. Initially, it is an experimental modeling and simulation environment for 3D mechanical systems, but it shall be expanded into other domains in the future.

Ideas from modern computer game engines are used to achieve a highly flexible setup of mechanical systems including collision handling. Other features are utilized from multi-body programs, such as hierarchical structuring, support for closed kinematic loops, and elastic response calculation. The underlying mathematical formulation are hybrid Differential Algebraic Equations (DAEs) that are solved with the variable-step solver IDA [17] via the Sundials.jl Julia package [31]. The emphasis is on variable-step solvers to combine Modia3D with equation-based modeling in the future, using Modia3D assemblies as components within one high level programming environment (for example a joint of a Modia3D system is driven by a Modia model of an electrical motor and gearbox).

Modia3D provides a generic interface to visualize simulation results with different 3D renderers. The free community edition as well as the professional edition[6] of the *DLR Visualization* library[7] [3, 14] are supported. Currently, another team is developing a free 2D/3D web-based authoring tool that includes result visualization.

Collision handling with elastic response calculation and error controlled integration is challenging. A survey of collision detection methods for convex and concave rigid bodies as well as for deformable shapes is for example given in [1, 21]. In [30], there is an overview of response calculation methods based on impulses. Whenever a collision with elastic response calculation occurs, the model response is drastically changed. Variable-step integrators usually do not perform well if a change is simply applied discontinuously. Instead, both the efficiency and the precision of the result are improved if state events are generated at the start and end of a contact, provided contact situations only occur from time to time, see e.g. [28].

[1]https://github.com/modelica/ModelicaStandardLibrary
[2]https://github.com/ModiaSim/Modia.jl
[3]https://julialang.org
[4]https://github.com/ModiaSim/
[5]https://github.com/ModiaSim/Modia3D.jl
[6]https://visualization.ltx.de/
[7]http://www.systemcontrolinnovationlab.de/the-dlr-visualization-library/

Contact handling in Modia3D uses variable-step solvers where the penetration depths and Euclidean distances computed with the improved Minkowski Portal Refinement (MPR) algorithm [23] are utilized to construct appropriate zero-crossing functions. The user's view of Modia3D is introduced in [24] showing the very flexible definition of 3D systems. A user's guide of defining models with collision handling is presented in [26]. Some key algorithms are discussed in [25].

This article gives an overview to the collision handling of Modia3D (see section 2) and more insight on how existing elastic response formulations are combined and enhanced (see section 3). The formulation of zero-crossing functions as introduced in [23] is considerably simplified and improved (see section 4) and some examples are shown in section 5. Note, Modia3D end-users do not need to bother about the zero-crossing concepts of this article, since this is a pure internal issue that improves the reliability and efficiency of simulations.

2 COLLISION HANDLING

A Modia3D model is mathematically defined by the hybrid DAE system (1), where $\boldsymbol{x} = \boldsymbol{x}(t)$ and the Jacobian $\boldsymbol{J}$ (1c) is regular:

$$\mathbf{0} = \begin{bmatrix} f_d(\dot{\boldsymbol{x}}, \boldsymbol{x}, t, z_i > 0) \\ f_c(\boldsymbol{x}, t, z_i > 0) \end{bmatrix} \quad (a) \qquad \boldsymbol{J} = \begin{bmatrix} \dfrac{\partial f_d}{\partial \dot{\boldsymbol{x}}} \\ \dfrac{\partial f_c}{\partial \boldsymbol{x}} \end{bmatrix} \quad (c) \qquad (1)$$
$$z = f_z(\boldsymbol{x}, t) \quad (b)$$

When differentiating f_c once and using the regularity of $\boldsymbol{J}$, it is (conceptually) possible to solve (1a) and $\dot{f}_c$ for $\dot{\boldsymbol{x}}$. Therefore, (1a) is an index 1 DAE. (1b) defines zero-crossing functions $\boldsymbol{z}(t)$. Whenever a z_i crosses zero, an event is triggered, simulation is halted, functions f_d, f_c can be changed, and simulation is restarted. (1) is numerically solved with the variable-step DAE integrator IDA of the Sundials suite [17] via the Sundials.jl [31] Julia package.

The model behavior changes drastically depending on whether shapes are in contact and contact forces and torques are applied or not in contact. The distances between convex shapes, in contact or not, are used as zero-crossing functions $z_i(t)$ in [23]. The zero-crossings of the distances trigger events, at the start and end of a contact phase, to increase the reliability of a simulation. In section 4 this approach is significantly improved. Furthermore, available proposals for elastic response calculations are combined and enhanced. The resulting formulation needs the initial relative velocity when a contact starts to compute an appropriate damping factor from the coefficient of restitution and this velocity. Therefore, an event at the start of a contact is mandatory for this approach.

It is standard to perform collision handling in the following way

1. *Broad phase*
   The n shapes where contact can occur are approximated by other shapes where collision can be very cheaply determined. Furthermore, the approximated shapes can be placed in a hierarchy so that all direct and indirect children of a node cannot penetrate, if a collision is not possible for the node. Typically, $O(n\,log(n))$ collision tests are being made in this phase, instead of $O(n^2)$ tests.
2. *Narrow phase*
   The signed distances are computed for the potentially colliding shape pairs that have been identified in the broad phase.
3. *Response calculation*
   If two shapes are penetrated, a finite force and/or torque is applied at the contact point as function of the pentration depth and its derivative, such as a spring/damper force element. Alternatively, a force/torque impulse is applied that leads to discontinuous changes of the velocities/angular velocities of the shapes. In this paper, the first approach is used.

In the following, details of the collision handling are given, as implemented in Modia3D:

2.1 Preprocessing

A preprocessing of a mechanical structure is executed to reduce the number of possible collision pairs to n_{pp} and to speed up the broad phase. This leads to $n_{pp} \leq O(n^2)$ tests [24, 25]. There are two preprocessing rules:

1. Rigidly attached shapes cannot collide with each other.
2. Shapes connected by a joint cannot collide with each other if the joint specific option `canCollide` is set to `false` by the user.

2.2 Broad Phase

To determine in a cheap way whether two shapes might intersect, shapes are approximated in Modia3D by Axis Aligned Bounding Boxes (AABB's), see e.g. [4, 13]. The narrow phase is executed only if the AABB's are overlapping. Otherwise, the shapes cannot penetrate and the Euclidean distance δ_{AABB} between two AABB's is calculated instead.

2.3 Narrow Phase

Based on the Minkowski difference, Snethen [33] proposed the Minkowski Portal Refinement (MPR) algorithm to detect whether two convex shapes penetrate and if this is the case compute an approximation of the penetration depth. The MPR algorithm in 3D is much simpler than the often used GJK/EPA algorithms [4, 13] because it operates with simpler geometries. The drawback is that MPR may only compute an approximation of the penetration depth in some situations. In [27] GJK/EPA and MPR are compared with several millions randomized benchmarks with various shape types. In [20] a compact pseudo-code for the MPR algorithm is given.

The MPR algorithm can be used to compute a lower and upper bound on the closest Euclidean distance of two non-penetrating convex shapes [23]. In Modia3D improvements of the MPR-algorithm are utilized that have been proposed in [20, 23], in particular to compute the distances of shapes that are not in contact, treating special collision situations properly and introducing a new termination condition to speed up the algorithm in some situations.

The signed distance δ_{mpr} between two shapes is calculated with the MPR algorithm only in the narrow phase if the AABB's in the broad phase are overlapping. If two shapes are not in contact to each other, $\delta_{mpr} > 0$ is an approximation of the Euclidean distance. If two shapes are penetrating each other, $\delta_{mpr} < 0$ is an approximation of the penetration depth.

In Modia3D, all non-smooth shapes (like meshes or boxes) are smoothed with a small sphere that is moved over all surface points, according to [4]. The effect is that all sharp edges and vertices of a shape get "rounded" and therefore a unique normal is always defined at every surface point. Furthermore, if a surface point is changing continuously, also its unique normal is changing continuously. This feature is important, because otherwise functions f_d, f_c are changing discontinuously if two shapes are in contact at their edges and vertices and a variable-step integration method might fail. Another effect is that the penetration depth between two shapes A and B computed by the MPR algorithm is the exact penetration depth, $\delta_{mpr}(A, B) = \delta_{exact}(A, B)$, if $|\delta_{mpr}| < r_A + r_B$, where r_A, r_B are the radii of the smoothing spheres of shape A and B [23].

3 ELASTIC RESPONSE CALCULATION

The user's view of the Modia3D elastic response calculation and how to define models with collision detection has been presented in [26]. Below, the details of the mathematical description are provided. Hereby, the following definition of the penetration depth δ is used:

Definition 3.1. **(Penetration depth).**
When two shapes A and B are intersecting, so $\delta_{mpr} \leq 0$, the penetration depth δ is defined as:

$$\delta := |\delta_{mpr}| \geq 0. \tag{2}$$

Assume two shapes penetrate each other as shown in Figure 1. The intuition is that there is a contact area with a certain pressure distribution in normal and a stress distribution in tangential direction and that the response characteristics provides an approximation of the resultant normal force $\vec{f}_n$, resultant tangential force $\vec{f}_t$, and resultant contact torque $\vec{\tau}_\omega$. The MPR algorithm calculates an approximation of the contact point, of the signed distance δ_{mpr} and of a unit vector $\vec{e}_n$ that is orthogonal to the contacting surfaces. Figure 1 shows the relation between the forces $\vec{f}_n, \vec{f}_t$, torque $\vec{\tau}_\omega$ and unit vectors $\vec{e}_n, \vec{e}_t, \vec{e}_\omega$ in direction of the respective relative movement.

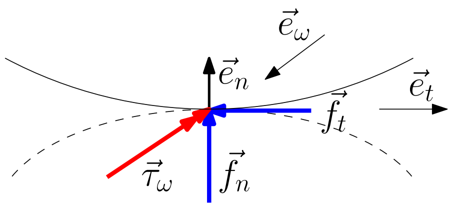

Figure 1: Contact normal force $\vec{f}_n$, contact tangential force $\vec{f}_t$ (= sliding friction force) and contact torque $\vec{\tau}_\omega$ between two penetrating objects. $\vec{e}_n, \vec{e}_t, \vec{e}_\omega$ are unit vectors in direction of the respective relative movement [26].

3.1 Force Law

The novel response characteristic is based on [11, 28, 32]:

$$f_n = k_{red} \max\left(0, c_{res}\, c_{geo}\, \delta^{n_{geo}} \left(1 + d\, \dot{\delta}\right)\right) \tag{3a}$$

$$\vec{f}_n = f_n \vec{e}_n \tag{3b}$$

$$\vec{f}_t = -\mu_k f_n \vec{e}_{t,reg} \tag{3c}$$

$$\vec{\tau}_\omega = -\mu_r \mu_{geo} f_n \vec{e}_{\omega,reg} \tag{3d}$$

$\dot{\delta}$	Penetration velocity between two objects, see (5b).
$\vec{e}_n$	Unit vector normal to the contacting surfaces.
$\vec{e}_{t,reg}, \vec{e}_{\omega,reg}$	Regularized unit vectors in direction of the relative tangential and relative angular velocity.
μ_k, μ_r	Kinetic/sliding friction force and rotational resistance torque coefficient.
c_{res}	Resultant spring constant in normal direction.
d	Damping coefficient.
$c_{geo}, n_{geo}, \mu_{geo}$	Geometry dependent coefficients.
k_{red}	Elastic contact reduction factor.

The normal force value f_n (3a) is defined with the max(..) function to guarantee always a compressive force and avoid the nonphysical behavior of a pulling force.

3.2 Solid Material and Contact Pair Material Constants

To describe the physical behavior in contact situations solid material constants and constants which describe the behavior between two objects need to be defined by the user:

Solid material constants.

E	Young's modulus of contact material in $[N/m^2]$.
ν	Poisson's ratio of contact material ($0 < \nu < 1$).

Contact pair material constants.

cor	Coefficient of restitution ($0 \leq cor \leq 1$). An ideal inelastic collision is defined with $cor = 0$ and an ideal elastic collision with $cor = 1$.
μ_k	Kinetic/sliding friction force coefficient ($\mu_k \geq 0$).
μ_r	Rotational resistance torque coefficient ($\mu_r \geq 0$). Its effect is that torque $\vec{\tau}_\omega$ is computed to reduce the relative angular velocity $\vec{\omega}_{rel}$ between the two objects until $\vec{\omega}_{rel} = 0$. μ_r can be interpreted as the rolling resistance coefficient if a sphere is rolling on a plane.

For each shape, where contact can occur, a contact material like "steel" or "dry wood" needs to be assigned. Each material respectively defines the contact parameters E and ν. Furthermore, for every contact material pair potentially occurring during a simulation (for example "steel, steel" and "steel, dry wood"), the contact pair material constants (cor, μ_k, μ_r) need to be defined as well.

3.3 Regularization

The unit vector $\vec{e}_t$ is not defined for vanishing tangential relative velocity and unit vector $\vec{e}_\omega$ is not defined for vanishing relative angular velocity. In order to treat these cases appropriately, the following function is introduced which is a continuous and smooth regularization of an absolute value:

$$\operatorname{reg}(v, v_{min}) = \begin{cases} |v|, & |v| \geq v_{min} > 0 \\ \frac{|v|^2}{v_{min}}\left(1 - \frac{|v|}{3v_{min}}\right) + \frac{v_{min}}{3}, & else \end{cases}, \tag{4}$$

where $v \in \mathbb{R}, v_{min} \in \mathbb{R}^+$. Function reg(..) returns $|v|$ if $|v|$ is not close to zero and otherwise defines a third order polynomial such that its minimum is at $\frac{v_{min}}{3}$ for $v = 0$ and has smooth first and second derivatives at $|v| = v_{min}$. A plot of the function is shown in Figure 2 for $v_{min} = 0.01$.

Regularized Unit Vectors. Regularized unit vectors are defined with function (4), whereby the absolute value of a vector $|\cdot|$ is the length of $\vec{v} \in \mathbb{R}^3$:

$$\vec{v}_{rel} = \vec{v}_2 - \vec{v}_1 \tag{5a}$$
$$\dot{\delta} = \vec{v}_{rel} \cdot \vec{e}_n \tag{5b}$$
$$\vec{v}_t = \vec{v}_{rel} - \dot{\delta}\,\vec{e}_n \tag{5c}$$
$$\vec{e}_{t,reg} = \frac{\vec{v}_t}{\text{reg}(|\vec{v}_t|, v_{min})} \tag{5d}$$
$$\vec{\omega}_{rel} = \vec{\omega}_2 - \vec{\omega}_1 \tag{5e}$$
$$\vec{e}_{\omega,reg} = \frac{\vec{\omega}_{rel}}{\text{reg}(|\vec{\omega}_{rel}|, \omega_{min})} \tag{5f}$$

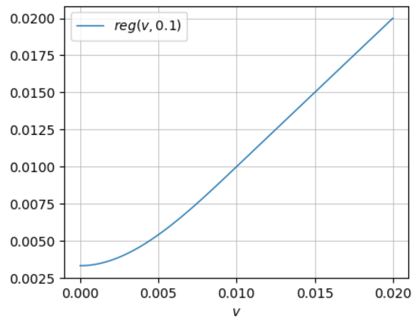

Figure 2: Regularization for $v_{min} = 0.01$.

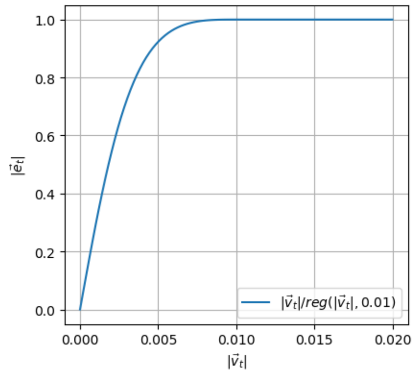

Figure 3: A regularized unit vector.

Figure 3 shows the absolute value of a regularized unit vector $\vec{e}_{t,reg}$ with $v_{min} = 0.01$. If $|\vec{v}_t| < v_{min}$ the regularization is used and $|\vec{e}_{t,reg}| = 0$ if $|\vec{v}_t| = 0$. Otherwise, if $|\vec{v}_t| \geq v_{min}$ then $|\vec{e}_{t,reg}| = 1$ and $\vec{e}_{t,reg} = \vec{e}_t$.

Regularized Contact Start Velocity. In section 3.4, the damping coefficient d is computed from the velocity at contact start $\dot{\delta}^-$. To avoid a division by zero for vanishing $\dot{\delta}^-$ (e.g. if two shapes start in touching position), function (4) is used to regularize $\dot{\delta}^-$:

$$\dot{\delta}^-_{reg} = \text{reg}(\dot{\delta}^-, v_{min}) > 0. \tag{6}$$

Regularized Coefficient of Restitution. In section 3.4, the damping coefficient d is computed from the coefficient of restitution cor. If $cor > 0$, two objects would bounce infinitely often in finite time until coming to rest, this is also known as the Zeno effect. This is nonphysical, because in reality all bouncing objects come to rest after a finite number of bounces. For this reason cor is reduced when the velocity at contact start $\dot{\delta}^-$ becomes small. Furthermore, the regularized coefficient of restitution is restricted to a minimum value cor_{min} (default = 0.001) to avoid a division by zero when computing d. The regularized coefficient of restitution cor_{reg} is defined as:

$$cor_{reg} = cor + (cor_{min} - cor)e^{\log(0.01)\frac{|\dot{\delta}^-|}{v_{min}}}. \tag{7}$$

Figure 4 shows this characteristic for several different coefficients of restitution.

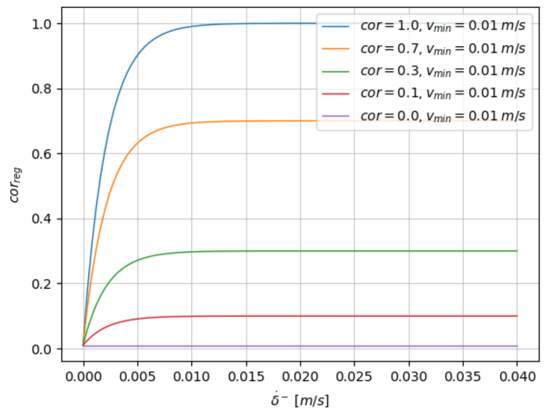

Figure 4: Characteristics for several regularized coefficients of restitution.

3.4 Spring and Damping Constants

Spring Constant. With the Hertz' law [15] for two penetrating bodies, the spring constant

$$c_i = \frac{E_i}{1 - \nu_i^2}, \quad i = 1, 2 \tag{8}$$

of an object i is based on its Young's modulus E_i and its Poisson's ratio ν_i (see section 3.2). The resultant spring constant of two penetrating objects 1 and 2 is computed as

$$c_{res} = \frac{1}{\frac{1}{c_1} + \frac{1}{c_2}}. \tag{9}$$

Damping Constant. There are several proposals to compute the damping coefficient as a function of the coefficient of restitution *cor* and the normal velocity $\dot{\delta}^-$ when contact starts. For a comparison of the different formulations, see [32]. In Modia3D, basically the formulation of [11] is used, because it gives similar results with respect to a response calculation with impulses for a wide range of *cor* values for several experiments performed in [32]:

$$d = min\left(d_{max}, \frac{8(1 - cor_{reg})}{5cor_{reg}\dot{\delta}^-_{reg}}\right). \tag{10}$$

(10) has the following improvements with respect to [11]:

- The regularized coefficient of restitution cor_{reg} (7) is used instead of *cor* to avoid a division by zero for $cor = 0$.
- The regularized initial velocity $\dot{\delta}^-_{reg}$ (6) is used instead of $\dot{\delta}^-$ to avoid a division by zero.
- The damping coefficient is limited to $d_{max} = 1000$ to avoid a nonphysical strong creeping effect for collisions with small cor_{reg} values.

The damping coefficient d is shown as function of $\dot{\delta}^-$ for several *cor* values in Figures 5 and 6.

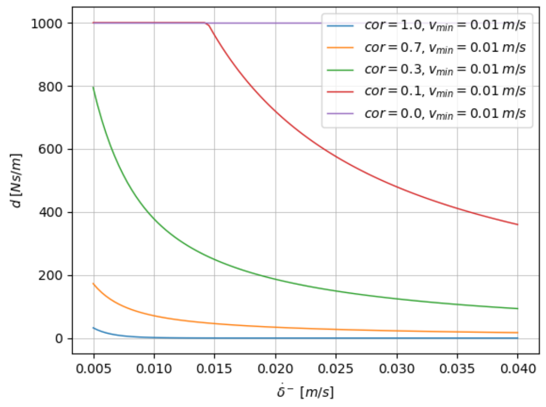

Figure 5: Damping coefficient as function of $\dot{\delta}^-$ and *cor*.

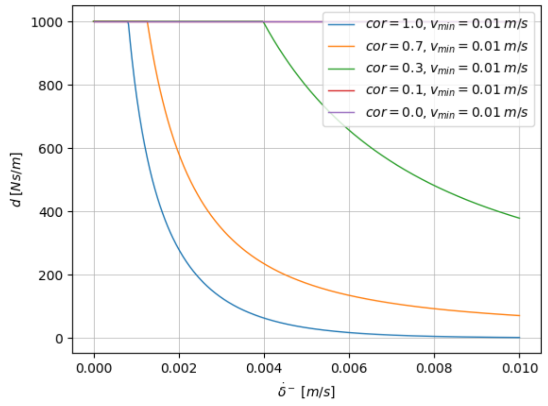

Figure 6: Damping coefficient as function of $\dot{\delta}^-$ and *cor*.

3.5 Geometry Dependent Coefficients

The geometries of two objects that are in contact influence the physical behavior reflected in the coefficients $c_{geo}, n_{geo}, \mu_{geo}$. If at least one of the shapes is a sphere, Hertz' law [15] is assumed. If not enough information is available, these factors are set to one (see Table 1).

object 1	object 2	μ_{geo}	c_{geo}	n_{geo}
sphere	sphere	$\frac{1}{\frac{1}{r_1}+\frac{1}{r_2}}$	$\frac{4}{3}\sqrt{\mu_{geo}}$	$\frac{3}{2}$
sphere	no sphere	r_1	$\frac{4}{3}\sqrt{r_1}$	$\frac{3}{2}$
no sphere	no sphere	1	1	1

Table 1: Geometry dependent coefficients.

3.6 Elastic Contact Reduction Factor

Applying the elastic response calculation (3) on hard materials such as steel, typically results in penetration depths in the order of $10^{-5}..10^{-6}$ m. A penetration depth is implicitly computed by the difference of the absolute positions of the objects in contact. Further, these absolute positions are typically error-controlled variables of the integrator. This in turn means that typically at least a relative tolerance of 10^{-8} needs to be used for the integration, in order that the penetration depth is computed with 2 or 3 significant digits.

To improve simulation speed, factor k_{red} is introduced in (3a). This reduces the stiffness of the contact and therefore enlarges the penetration depth. If k_{red} is for example set to 10^{-4}, the penetration depth might be in the order of 10^{-3} m and then a relative tolerance of 10^{-5} might be sufficient. In many cases, the essential response characteristic is not changed, except that the penetration depth is larger, but simulation speed is significantly improved.

4 ZERO-CROSSING FUNCTIONS

As proposed in [23] the distance computed with the improved MPR algorithm can be used as zero-crossing function for collision handling with variable-step solvers.

The following properties of the improved MPR algorithm are used for collision handling with variable-step integrators. Hereby, A and B are convex shapes and δ_{mpr} is the signed distance returned by the MPR algorithm under the assumption that computations are performed with infinite precision and δ_{AABB} is the Euclidean distance between non overlapping AABB's. The following Theorem 4.1 is a slight extension of [23, Theorem 4.1]

Theorem 4.1. ***(Contact detection).****[23, Theorem 4.1]*
The following holds:

1. $\delta_{AABB} > 0$ *or* $\delta_{mpr} > 0$*: A and B are not in contact to each other.*
2. $\delta_{mpr} = 0$*: A and B are touching each other.*
3. $\delta_{mpr} < 0$*: A and B are penetrating each other.*

Proof. [23, Theorem 4.1] These properties are proven in [23, Theorem 4.1], with exception of $\delta_{AABB} > 0$. The latter is obvious, because two shapes cannot be in contact to each other if their AABB's are not in contact to each other. □

Basically, the following zero-crossing function z_{AB} is used between two shapes A and B:

$$z_{AB} = \begin{cases} \delta_{AABB} & \textbf{if } \delta_{AABB} > 0 \\ \delta_{mpr} & \textbf{if } \delta_{AABB} \le 0 \end{cases} . \tag{11}$$

Theorem 4.1 is sufficient in order that (11) can be used as zero-crossing function for a variable-step solver: It is standard for variable-step solvers with zero-crossing support to evaluate a zero-crossing function $z_{AB}(t)$ at time instant $t_j + h$ of a completed integrator step with step-size h. If $z_{AB}(t_j) \cdot z_{AB}(t_j + h) \le 0$ an interval $[t_j, t_j + h]$ is determined in which z_{AB} crosses zero. Several algorithms with guaranteed convergence are known to reduce the interval in which the zero-crossing takes place until a prescribed tolerance for the final interval is met. For example, the integrators of the Sundials suite use a modified secant method [18, Sec. 2.3], whereby the integrator DASSL [6] uses the method of Brent [7, pp. 58–59]. In both cases the interval is successively reduced in every iteration. Hereby, the only needed property is the sign of z_{AB} (which is provided by (11) according to theorem 4.1). If $z_{AB}(t)$ is additionally smooth, convergence is faster (for example, in case of the method of Brent [7], convergence is super-linear).

4.1 Simplified Zero-Crossing Functions

In [23], the signed distances of every shape pair are computed and the z_{max} smallest values are used as zero-crossing functions (otherwise $O(n^2)$ zero-crossing functions would be needed, if n shapes can potentially collide with each other). This approach has the drawbacks that an upper bound z_{max} for the number of zero-crossing functions has to be defined by the user and the algorithm for keeping track of the z_{max} smallest values is complicated.

The approach from [23] is significantly simplified by using only two zero-crossing functions

$$z_1 = \max(z_{AB} \;:\; \text{in contact at last event}) \tag{12a}$$

$$z_2 = \min(z_{AB} \;:\; \text{not in contact at last event}). \tag{12b}$$

Hereby, $\max(z_{AB}$: in contact at last event) is the maximum over all zero-crossing functions (11) of shape pairs which have been in contact at the last event instant. Therefore, a zero-crossing of z_1 takes place, if at least one shape pair, that has been in contact at the last event, changes from negative to positive values.

Additionally, $\min(z_{AB}$: not in contact at last event) is the minimum over all zero-crossing functions (11) of shape pairs which have not been in contact at the last event instant. Therefore, a zero-crossing of z_2 takes place, if at least one shape pair, that has not been in contact at the last event, crosses from positive to negative values.

The implementation in Modia3D uses basically one dictionary (called `contactDict`) to hold the shape pairs which have been in contact at the last event instant in order to decide whether a computed zero-crossing (11) of two shapes A and B shall be utilized in z_1 or in z_2. Furthermore, the following three internal functions are used[8]:

- `selectContactPairsAtEvent!(..)`
  This function is called at every event instant and computes (11) for all potentially colliding shape pairs. The actual penetrating shape pairs are detected and stored in `contactDict`. This selection is kept until `selectContactPairsAtEvent!` is called again.
- `selectContactPairsNoEvent!(..)`
  This function is called whenever the solver requests a new zero-crossing function evaluation and computes (11) for all potentially colliding shape pairs. Furthermore, the contact points and the contact normals of the collision pairs stored in `contactDict` are updated.
- `getDistances!(..)`
  This function is called at communication points and updates (11), contact points and contact normals for all collision pairs stored in `contactDict`.

The difference between these three functions is also pointed out in Julia pseudo code-snippet[9] lines 1 - 8.

```
1  hasEvent = isEvent(...)

2  if hasEvent
3     selectContactPairsAtEvent!(...)
4  elseif isZeroCrossing(...)
5     selectContactPairsNoEvent!(...)
6  else
7     getDistances!(...)
8  end
```

Partitioning into two Sets. The possible contact pairs are partitioned into two sets. One where all information (contact points on each object, the contact normal, the two penetrating objects, and the distance with hysteresis `distanceWithHysteresis` (lines 14 - 19)) about penetrating objects is stored and updated in `contactDict` (lines 9 - 21 and lines 22 - 25). The other set, where the objects are not in contact, is not stored. But, their distances are compared and the minimum value is stored in `noContactMinVal` (lines 26 - 29).

```
9  function updateContactPair!(pair::ContactPair,
10           contactPoint1::Vector, contactPoint2::Vector,
11           contactNormal::Vector,
12           obj1::Object3D, obj2::Object3D,
13           distanceWithHysteresis::Float64)
14    pair.contactPoint1 = contactPoint1
15    pair.contactPoint2 = contactPoint2
16    pair.contactNormal = contactNormal
17    pair.obj1          = obj1
18    pair.obj2          = obj2
19    pair.distanceWithHysteresis = distanceWithHysteresis
20    ...
21 end

22 if contact
23   updateContactPair!(contactDict[pairID],
24          contactPoint1, contactPoint2, contactNormal,
25          obj1, obj2, distanceWithHysteresis)
26 else
27   if noContactMinVal > distanceWithHysteresis
28     noContactMinVal = distanceWithHysteresis
29 end; end
```

Hysteresis for Zero-Crossing Functions. A restart of the integration after an event requires that no zero-crossing function is identical to zero, otherwise a zero-crossing cannot be detected anymore and solvers trigger an error. However, the zero-crossing function (11) might be identically to zero at an event instant (for example two objects are touching each other at initialization). For this reason, a

[8]As usual in Julia, function names with a ! at the end indicate that one or more of the input arguments are changed by the function call.

[9]For better reference every code-snippet is marked with a unique line number on the left-hand side.

hysteresis is added to (11) in a similar way as in [12] (see Figure 7 and code lines 30 - 33). There are three ranges: no contact $[0, \infty)$, contact $(-2\varepsilon, -\infty)$, and hysteresis area $[0, -2\varepsilon)$ where the pairs might be in contact or not. At an event instant, each pair with $distance < -\varepsilon$ is identified (line 32), marked as a pair that is in contact, and its original distance is stored in `distanceWithHysteresis`. Otherwise, it is not in contact and 2ε is added to the original distance (line 33).

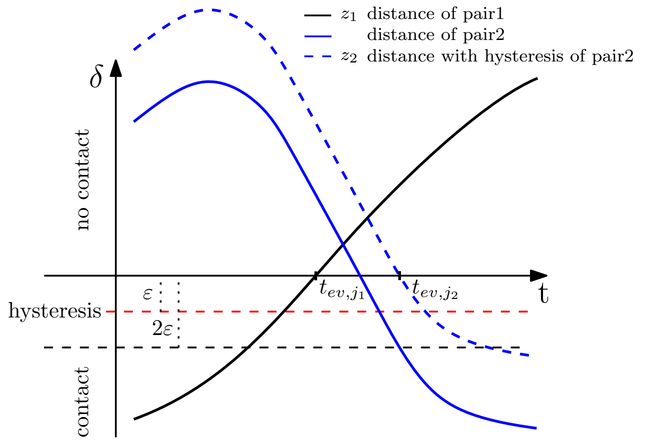

Figure 7: Hysteresis for zero-crossing functions.

Each possible contact pair is identified with a unique ID, in this case (Figure 7): `pairID` $\in \{1, 2\}$. At t_{ev,j_1} an event is triggered by z_1, since pair 2 is not in contact, the distance is modified and therefore, the blue graph is shifted with a hysteresis of 2ε. This leads to the dashed blue graph z_2. The solver needs to deal with two zero-crossing functions. z_1 is the original distance of pair 1 and z_2 is the distance with hysteresis of pair 2. The next time event t_{ev,j_2} is of zero-crossing function z_2.

```
30 const zEps  = 1.e-8

31 hasContact = haskey(contactDict, pairID)
32 contact    = hasEvent ? distance < -zEps : hasContact
33 distanceWithHysteresis = contact  ? distance : distance + 2*zEps
```

Zero-Crossing Functions. As defined in (12), two zero-crossing functions are used:

z_1 indicates if at least one zero-crossing from penetrating to non-penetrating takes place. Therefore, the maximum value over all distances of shape pairs of `contactDict` is stored in z_1. If the contact set is empty z_1 gets a negative dummy value. If z_1 is positive a zero-crossing takes place (lines 36 - 41). This is a zero-crossing from negative to positive.

z_2 indicates if at least one zero-crossing from non-penetrating to penetrating takes place. The minimum value over all other distances of shapes (lines 26 - 29) which are not in `contactDict` is stored in z_2 (line 42). This is a zero-crossing from positive to negative.

Both zero-crossing functions are updated in function `updateZeroCrossing!` (lines 34 - 44).

```
34 function updateZeroCrossing!(...)
35   ...
36   if isempty(contactDict)
37     z[1] = -42.0
38   else
39     (pair, key) = findmax(contactDict)
40     z[1] = pair.distanceWithHysteresis
41   end
42   z[2] = noContactMinVal
43   ...
44 end
```

5 EXAMPLES: BILLIARDS

The billiards game is used to test the elastic response law (3) and to reproduce effects like sliding and rolling of balls, impacts between balls and between ball and cushion.

The observation of sliding and rolling balls, and the collision between balls with the elastic response law (3) is described in more detail in [26].

Sliding and Rolling Ball. The billiard ball is a free flying object with 6 degrees-of-freedom where the rotation is described with quaternions (in total, the state of this object is defined with 13 variables). At initialization, the billiard ball is placed in touching position with the table (penetration depth $\delta = 0$, see Theorem 4.1) with an initial velocity. The ball subsides immediately, because of gravity in z-direction. Therefore, the ball and the table are colliding, a collision event is triggered, and the two objects are penetrating each other.

At the beginning the ball is sliding and due to sliding friction ($\mu_k = 0.6$) the relative velocity is reduced. At the same time, the sliding friction force acts as a torque around the ball center and forces a rotation of the ball in y-direction. At a certain time the relative velocity in tangential direction v_t is zero, therefore sliding friction force $\vec{f}_t$ is zero (3c) as well and ideal rolling of the ball takes place. On the other hand, the rotational resistance torque $\vec{\tau}_\omega$ ($\mu_r = 0.02$) acts as a rolling resistance that continuously reduces the angular velocity and the ball comes to rest. The effect of sliding and rolling can be seen in Figure 8 in the first 0.23 s. The first 0.11 s is the transition between sliding and rolling, and afterwards it is rolling only.

The elastic response law (3) is able to reproduce the effect of a sliding and rolling ball. However, for simplicity of the formulation, velocity dependency of the coefficients cor, μ_k, μ_r is neglected. For further information and graphics about sliding and rolling see [26].

Collision of two Balls. Two billiard balls are positioned on a billiard table with a gap between them. The cue ball has initial velocity, the effects of sliding and rolling occur as analyzed before, and it hits the other resting ball after some time.

Since the coefficient of restitution between these two billiard balls is one, a fully elastic collision takes place. In this case the cue ball transfers most of its kinetic energy to the resting ball which starts moving with the velocity of the cue ball. However, since the cue ball is rolling, the angular momentum is greater zero. This momentum is conserved. Therefore, the cue ball continues rolling and velocity rises from zero again. Due to the impact the relative velocity is no longer zero, again a friction force $\vec{f}_t$ is acting that introduces a counter torque at the balls axis which quickly reduces the angular velocity until the relative velocity is zero again. Both balls are ideally rolling again. Due to the rotational resistance torque, the angular velocities are slowly reduced until both balls come to rest. For further information and graphics about two colliding balls see [26].

Ball colliding with Cushion. Corresponding to [22] the cushion is arranged and shaped in ratio to the billiard balls radius r. The height of the contact point at the rail is $h = \frac{7}{5}r$ (see Figure 9). Figure 10 shows the used simplification of [22] and Figure 9. The billiards cushion has very good rebound properties $cor = 0.98$.

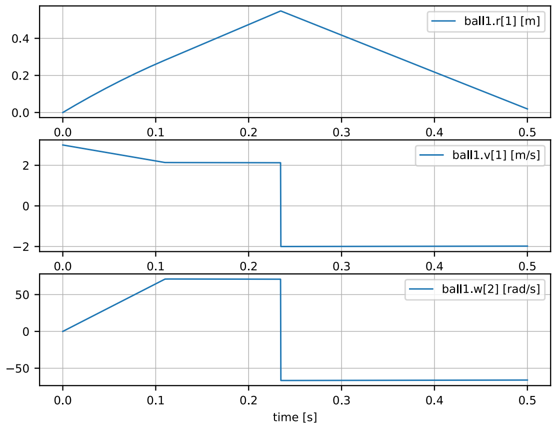

Figure 8: Position of a billiard ball colliding with a cushion, velocity in x-direction, and angular velocity in y-direction.

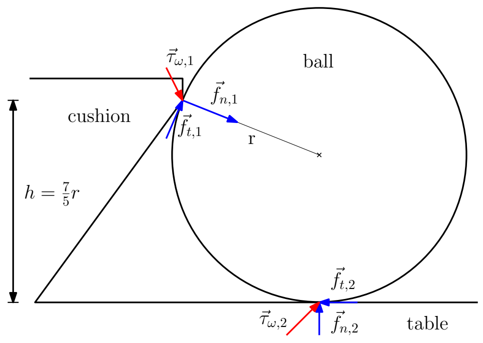

Figure 9: Relation between ball and cushion.

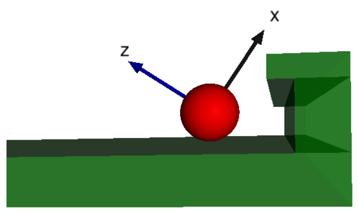

Figure 10: Ball collides with cushion.

A billiard ball is positioned on a billiard table. The ball has initial velocity, the effects of sliding and rolling occur as analyzed before in the first 0.2 s (see Figure 8). At $time = 0.23$ s the ball impacts on the cushion rail. The corresponding force points towards the table (Figure 9). Therefore, the ball changes sign of velocity and angular velocity immediately and rolls backwards.

Billiards. All described effects (sliding, rolling, colliding) act together in a simulation of a billiard game. It consist of 16 billiard balls, the table and cushions (Figure 11). The cue ball has an initial velocity pointing to the right and hits the the center of the red rack (15 other balls) exactly after a short time interval. This results in a symmetric evolution of the balls, as one would expect (Figure 11, bottom). The hybrid DAE system (1) has $\dim(\boldsymbol{x}) = 13 \cdot 16 = 208$ and there are about 200 possible collision pairs. Simulation on a standard PC needs about 20 min for 5 s of simulation time and a tolerance of 10^{-5}. The computation time speeds up to about 13 min if the elastic contact reduction factor is set $k_{red} = 10^{-4}$. At the moment, the Modia3D code is implemented for functionality and not optimized for efficiency, so an even better speed-up is expected in the future.

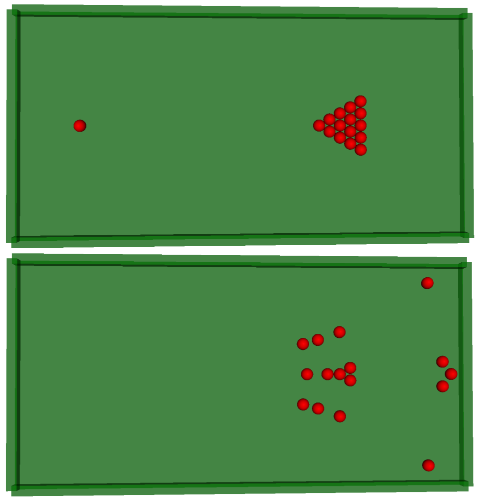

Figure 11: Initial setting of 16 billiard balls (top). Billiard balls after a few seconds (bottom).

6 CONCLUSION

In this article existing response formulations are combined and enhanced to migrate a novel force and torque formulation. In particular, collision handling with variable-step solvers has been sketched and the signed distance (= closest distance if not in contact and penetration depth if in contact) between two convex shapes is used as zero-crossing function. Furthermore, a new method is proposed to reduce the total number of zero-crossing functions to two, even if many shapes are present that can potentially collide. To avoid numerical issues a hysteresis is introduced.

Modia3D combines ideas from different communities. The architecture with component-oriented modeling is inspired by game engines so that 3D models can be setup in a very flexible way, as well as several elements for collision handling. Other features are from multi-body programs, like hierarchical structuring, support of closed kinematic loops, and algorithms to compute results close to real physics.

Modia3D is still a prototype implementation and several important parts are under development. Especially, the integration with Modia is missing at the moment. Furthermore, the code was currently mainly developed for its functionality and is not yet tuned for efficiency. For these reasons, benchmarks and comparisons with other programs with respect to simulation efficiency have not yet been performed.

REFERENCES

[1] Q. Avril, V. Gouranton, and B. Arnaldi. 2009. *New trends in collision detection performance.* Technical Report. INRIA. https://hal.archives-ouvertes.fr/hal-00412870

[2] G. Bardaro, L. Bascetta, F. Casella, and M. Matteucci. 2017. Using Modelica for advanced Multi-Body modelling in 3D graphical robotic simulators. In *Proc. of the 12th International Modelica Conference*, J. Kofranek and F. Casella (Eds.). LiU Electronic Press. http://www.ep.liu.se/ecp/132/097/ecp17132887.pdf

[3] T. Bellmann. 2009. Interactive Simulations and advanced Visualization with Modelica. In *Proc. of the 7th International Modelica Conference*, Francesco Casella (Ed.). LiU Electronic Press. http://www.ep.liu.se/ecp/043/062/ecp09430056.pdf

[4] G.v.d. Bergen. 2003. *Collision Detection in Interactive 3D Environments.* Morgan Kaufmann Publishers.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM review* 59, 1 (2017), 65–98.

[6] K. E. Brenan, S. L. Campbell, and L. R. Petzold. 1996. *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations.* SIAM.

[7] R.P. Brent. 1973. *Algorithms for Minimization without derivatives.* Prentice Hall. http://maths-people.anu.edu.au/~brent/pub/pub011.html

[8] H. Elmqvist, A. Goteman, V. Roxling, and T. Ghandriz. 2015. Generic Modelica Framework for MultiBody Contacts and Discrete Element Method. In *Proc. of the 11th International Modelica Conference*, Peter Fritzson and Hilding Elmqvist (Eds.). LiU Electronic Press. http://www.ep.liu.se/ecp/118/046/ecp15118427.pdf

[9] H. Elmqvist, T. Henningsson, and M. Otter. 2016. Systems Modeling and Programming in a Unified Environment based on Julia. In *Proc. of ISoLA Conference.* Springer. https://doi.org/10.1007/978-3-319-47169-3$_1$5

[10] H. Elmqvist, T. Henningsson, and M. Otter. 2017. Innovations for Future Modelica. In *Proc. of the 12th International Modelica Conference*, J. Kofranek and F. Casella (Eds.). LiU Electronic Press. http://www.ep.liu.se/ecp/132/076/ecp17132693.pdf

[11] P. Flores, M. Machado, M. Silva, and J. Martins. 2011. On the continuous contact force models for soft materials in multibody dynamics. *Multibody system dynamics* 25, 3 (2011), 357–375.

[12] Modelica Association FMI Project. 2017. *Functional Mock-up Interface for Model Exchange.* https://fmi-standard.org

[13] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. 1988. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation* 4, 2 (1988), 193–203. https://graphics.stanford.edu/courses/cs448b-00-winter/papers/gilbert.pdf

[14] M. Hellerer, T. Bellmann, and F. Schlegel. 2014. The DLR Visualization Library - Recent development and applications. In *Proc. of the 10th International Modelica Conference*, Hubertus Tummescheit and Karl-Erik Arzen (Eds.). LiU Electronic Press. http://www.ep.liu.se/ecp/096/094/ecp14096094.pdf

[15] H. Hertz. 1896. On the contact of solids - on the contact of rigid elastic solids and on hardness. *Miscellaneous papers* (1896), 146–183.

[16] C. Höger, A. Mehlhase, C. Nytsch-Geusen, K. Isakovic, and R. Kubiak. 2012. Modelica3D - Platform Independent Simulation Visualization. In *Proc. of the 9th International Modelica Conference*, M. Otter and D. Zimmer (Eds.). http://www.ep.liu.se/ecp/076/049/ecp12076049.pdf

[17] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. 2005. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Trans. Math. Software* 31, 3 (Sept. 2005), 363–396.

[18] A.C. Hindmarsh, R. Serban, and A. Collier. 2015. *User Documentation for IDA v2.8.2.* Technical Report UCRL-SM-208112. Lawrence Livermore National Laboratory.

[19] A. Hofmann, L. Mikelsons, I. Gubsch, and C. Schubert. 2014. Simulating Collisions within the Modelica MultiBody Library. In *Proc. of the 10th International Modelica Conference*, Hubertus Tummescheit and Karl-Erik Arzen (Eds.). LiU Electronic Press. http://www.ep.liu.se/ecp/096/099/ecp14096099.pdf

[20] B. Kenwright. 2015. *Generic Convex Collision Detection using Support Mapping.* Technical Report. https://www.semanticscholar.org/paper/Generic-Convex-Collision-Detection-using-Support-Kenwright/4f0f2d95375db7cfdbfaa345847418789d8cb970

[21] D. Mainzer. 2015. *New Geometric Algorithms and Data Structures for Collision Detection of Dynamically Deforming Objects.* Ph.D. Dissertation. Clausthal University of Technology. http://cgvr.cs.uni-bremen.de/papers/diss$_w$eller2012/DissertationWeller.pdf

[22] S. Mathavan, M. R. Jackson, and R. M. Parkin. 2010. A theoretical analysis of billiard ball dynamics under cushion impacts. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 224, 9 (2010), 1863–1873.

[23] A. Neumayr and M. Otter. 2017. Collision Handling with Variable-step Integrators. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT'17).* ACM, 9–18. https://modiasim.github.io/Modia3D.jl/resources/documentation/CollisionHandling$_N$eumayr$_O$tter$_2$017.pdf

[24] A. Neumayr and M. Otter. 2018. Component-Based 3D Modeling of Dynamic Systems. In *Proceedings of the American Modelica Conference*, M. Tiller, H. Tummescheit, and L. Vanfretti (Eds.). https://elib.dlr.de/124126/1/2018$_M$odelica$_M$odia3D.pdf

[25] A. Neumayr and M. Otter. 2019. Algorithms for Component-Based 3D Modeling. In *Proceedings of the 13th International Modelica Conference.* LiU Electronic Press.

[26] A. Neumayr and M. Otter. 2019. Modia3D: Modeling and Simulation of 3D-Systems in Julia. Under Review.

[27] L. Olvang. 2010. *Real-time Collision Detection with Implicit Objects.* Technical Report IT 10 009. Department of Information Technology, Uppsala Universitet, Sweden. http://uu.diva-portal.org/smash/get/diva2:343820/FULLTEXT01.pdf

[28] M. Otter, H. Elmqvist, and J. D. López. 2005. Collision Handling for the Modelica MultiBody Library. In *Proceedings of the 4th International Modelica Conference.*

[29] M. Otter, H. Elmqvist, and S. E. Mattsson. 2003. The New Modelica MultiBody Library. In *Proc. of the 3rd International Modelica Conference*, P. Fritzson (Ed.). https://www.modelica.org/events/Conference2003/papers/h37$_O$tter$_m$ultibody.pdf

[30] T. Pfeiffer. 2012. On non-smooth multibody dynamics. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 226, 2 (2012), 147–177.

[31] C. Rackauckas and Q. Nie. 2017. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software* (2017). https://doi.org/10.5334/jors.151

[32] L. Skrinjar, J. Slavič, and M. Boltežar. 2018. A review of continuous contact-force models in multibody dynamics. *International Journal of Mechanical Sciences* 145 (2018), 171–187.

[33] G. Snethen. 2008. Xenocollide: Complex collision made simple. In *Game Programming Gems 7*, Scott Jacobs (Ed.). Charles River Media, 165–178.