Modia3D: Modeling and Simulation of 3D-Systems in Julia

Andrea Neumayr[1] and Martin Otter[1]

[1]DLR, Institute of System Dynamics and Control, Germany

ABSTRACT

Modia3D is an experimental Julia package to model and simulate 3D mechanical systems. Ideas from modern game engines are used to achieve a highly flexible setup and features of multi-body programs are used to get a rigid mathematical formulation and support, for example, of closed kinematic loops. Collision handling is performed on convex geometries with elastic response calculation. A Modia3D model is solved with a variable-step integrator. This is important to combine Modia3D with the equation-based modeling system Modia in the future.

Keywords

Julia, Modia, Modia3D, Modelica, collision handling, Minkowski Portal Refinement algorithm, component-based modeling, elastic force law, response calculation

1. Introduction

Modia3D[1] is an experimental modeling and simulation environment for 3D mechanical systems. It is based on the Julia programming language [3]. Ideas from modern 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 of closed kinematic loops and elastic response calculation. The underlying mathematical formulation are hybrid DAEs (Differential Algebraic Equations) that are solved with the variable-step solver IDA via the Sundials.jl Julia package [11, 18]. Emphasis is on variable-step solvers because Modia3D shall be combined in the future with equation-based modeling by using Modia3D assemblies as components in the Julia package Modia[2] [5, 6] (for example a joint of a Modia3D system is driven by a Modia model of an electrical motor and gearbox). Collision handling with elastic response calculation and error controlled integration is challenging and this article discusses some of the difficulties and how they are solved.

Modia3D provides a generic interface to visualize simulation results with different 3D renderers. Currently, the free community edition as well as the professional edition[3] of the *DLR Visualization* library[4] [1, 10] are supported. Currently, another team is developing a free 2D/3D web-based authoring tool that will include result visualization.

The user's view of Modia3D was introduced in [14] showing the very flexible definition of 3D systems. Some key algorithms are discussed in [13, 15]. This article gives an overview of Modia3D from a user's perspective, and in particular how collisions between objects are defined. Furthermore, existing elastic response formulations are combined and enhanced so that a minimum number of material data has to be provided.

2. Flexible Definition of 3D Systems

Modia3D follows the approach of modern game engines to provide a coordinate system as a primitive that is located in 3D and has a *container with optional components* (such as geometry, visualization, dynamics, collision properties, light, camera, sound, etc.), see for example [16][5].[6] Such types of objects are called *GameObject*[7] in Unity, *Actor*[8] in Unreal Engine, and *Object3D*[9] in Three.js. In Modia3D the name *Object3D* is used. This very flexible approach allows to define many optional components and variants and treat them in a modular way. The Julia programming language is particularly suited for this *component-oriented* programming pattern and therefore key-concepts of Julia, such as multiple dispatch, are heavily used in Modia3D.

Hierarchical structuring for grouping and aggregation is performed with the Modia3D macro @assembly. Julia macros are metaprogramming[10] language elements and a macro name starts with @. It generates an abstract syntax tree (AST) of Julia code which is automatically compiled and executed at the line where the macro is called. Fig. 1 shows a bar that is constructed from several Object3D elements and is defined with the following Julia code:

```
@assembly Bar(;Lx=0.1,Ly=Lx/5,Lz=Ly) begin
    obj0 = Object3D(Solid(SolidBeam(Lx,Ly,Lz),
             "Aluminium", Material(color="Blue"))
    obj1 = Object3D(obj0,r=[-Lx/2,0.0,0.0])
    obj2 = Object3D(obj0,r=[ Lx/2,0.0,0.0])
end
```

The first Object3D is a `SolidBeam` geometry that is made of Aluminium and is visualized in blue color. Mass, center of mass and inertia matrix are computed internally from this information. The next two objects place coordinate systems on the first object defining locations for revolute joints. Three instances of the `Bar` assembly together with a ground object (= the fourth bar) are used to build

[1]https://github.com/ModiaSim/Modia3D.jl
[2]https://github.com/ModiaSim/Modia.jl
[3]https://visualization.ltx.de/
[4]http://www.systemcontrolinnovationlab.de/the-dlr-visualization-library/

[5]http://gameprogrammingpatterns.com/component.html
[6]This section utilizes some descriptions, figures and Julia code from [14].
[7]https://docs.unity3d.com/Manual/GameObjects.html
[8]https://docs.unrealengine.com/en-us/Engine/Components
[9]https://threejs.org/docs/index.html#api/core/Object3D
[10]https://docs.julialang.org/en/v1/manual/metaprogramming

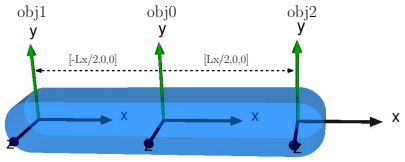

Fig. 1. A solid bar with two additional Object3Ds.

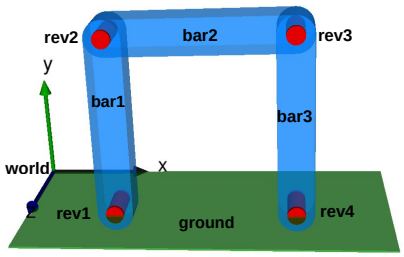

Fig. 2. Planar fourbar mechanism.

up the *fourbar* mechanism[11] shown in Fig. 2. The Julia code of this mechanism is shown in the next listing:

```
@assembly Fourbar(;Lx=0.1) begin
    world   = Object3D(CoordinateSystem(0.6))
    pos1    = Object3D(world,r=[Lx/2,0.0,Lx/2)
    pos2    = Object3D(pos1, r=[Lx,0.0,0.0])
    ground  = Object3D(world,Box(..),..)
    bar1    = Bar(Lx=Lx)
    bar2    = Bar(Lx=Lx)
    bar3    = Bar(Lx=Lx)
    rev1    = Revolute(pos1,bar1.obj1,
                  phi_start= pi/2)
    rev2    = Revolute(bar1.obj2,bar2.obj1,
                  phi_start=-pi/2)
    rev3    = Revolute(bar3.obj2,bar2.obj2,
                  phi_start=-pi/2)
    rev4    = Revolute(pos2,bar3.obj1,
                  phi_start= pi/2)
    ...
end
```

The angle of revolute joint `rev1` is moved kinematically via a signal. The resulting system is simulated by calling function `simulate!` (for more details of the Modia3D elements, see [14]):

```
@assembly MoveFourBar(Lx=0.1) begin
    fourbar = Fourbar(Lx=Lx)
    sine    = Sine(...)
    sig     = SignalToFlangeAngle(sine.y)
    connect(sig, fourbar.rev1)
end

model  = SimulationModel(MoveFourBar(Lx=0.2))
result = simulate!(model, stopTime=3.0)
```

3. Collision Handling

Modeling and simulating the collisions between geometrical objects is difficult and there are many methods dedicated to particular purposes. For example, in a game it is important that collisions of *many* objects are supported in *real-time* and that a simulation looks reasonably realistic. Game engines typically make compromises regarding physics (see for example [7], where it is pointed out that NVIDIA PhysX and Havok neglect Coriolis forces). When designing a product, it is important that a simulation of the collision between objects is *validated by measurements*. Therefore, the simulation results must be much closer to reality as in a game.
In Modia3D an algorithm computes the closest distance and/or the penetration depth between two *convex* objects. If two objects are penetrating each other, two contact forces and a contact torque are computed from the penetration depth and the relative movement and are applied at one point of the contact area (see Fig. 4). The advantages of this approach compared to a response calculation with impulses is that (a) simulation results are closer to reality and (b) it is easier to treat complex contact situations correctly. For example, if several objects have contact at the same time instant, as in the billiard table benchmark of section 4, it is difficult to solve a model with an impulsive response calculation. The drawback of an elastic response calculation is that simulation is often much slower.

3.1 User's View

Objects only take part in collision handling, if a `contactMaterial` is associated with them. Modia3D uses two sets of material data:

(1) *Solid material constants*
In a dictionary, the name of *one* material is used as a key, for example `"Steel"`. The values are typical material constants of a solid. For elastic response calculation, Young's modules E and Poisson's ratio ν are used. From this data of two contacting objects, a spring constant is computed to describe the elastic deformation in the contact area.

(2) *Contact pair material constants*
In another dictionary, the names of *two* materials are used as a key, for example `"Steel"` and `"DryWood"`. The values are the *coefficient of restitution cor*, the *kinetic/sliding friction force coefficient* μ_k and the *rotational resistance torque coefficient* μ_r between these two contacting objects of the respective materials, see section 3.3.

A simple example is shown in Fig. 3, where a steel ball is bouncing on a wooden table. This model is defined with the following Julia program (`r` is the position vector from the parent – defined with the first argument – to the reference frame of the object and `fixed` defines whether the object is fixed or not fixed in space):

```
@assembly BouncingBall begin
  world = Object3D()
  ball  = Object3D(world, Solid(SolidSphere(0.02),
             ...;contactMaterial = "Steel");
             r=[0,0,1.01], fixed=false)
  table = Object3D(world, Solid(SolidBox(1,1,0.1),
             ...;contactMaterial = "DryWood");
             fixed=true)
end

bounce = BouncingBall(sceneOptions=
          SceneOptions(enableContactDetection=true))
model  = SimulationModel(bounce)
result = simulate!(model; stopTime=3.0)
```

During simulation, the `contactMaterials` defined for `ball` and for `table` are used as keys in the two dictionaries sketched above. When the two objects penetrate each other, an elastic response is computed with the help of the dictionary values (provided, collision handling was explicitly enabled in the scene options).

[11] https://en.wikipedia.org/wiki/Four-bar_linkage

Fig. 3. Ball falling on a table.

3.2 Numerical Solution

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

$$\begin{aligned} \mathbf{0} &= \begin{bmatrix} \boldsymbol{f}_d(\dot{\boldsymbol{x}}, \boldsymbol{x}, t, z_i > 0) \\ \boldsymbol{f}_c(\boldsymbol{x}, t, z_i > 0) \end{bmatrix} \quad (a) \\ \boldsymbol{z} &= \boldsymbol{f}_z(\boldsymbol{x}, t) \quad (b) \end{aligned} \qquad \boldsymbol{J} = \begin{bmatrix} \frac{\partial \boldsymbol{f}_d}{\partial \dot{\boldsymbol{x}}} \\ \frac{\partial \boldsymbol{f}_c}{\partial \boldsymbol{x}} \end{bmatrix} \quad (c) \qquad (1)$$

When differentiating $\boldsymbol{f}_c$ once, it is (conceptually) possible to solve (1c) for $\dot{\boldsymbol{x}}$ because $\boldsymbol{J}$ is regular. 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 $\boldsymbol{f}_d, \boldsymbol{f}_c$ can be changed, and simulation is restarted. (1) is numerically solved with the variable-step DAE integrator IDA of the Sundials suite [11] via the Sundials.jl [18] Julia package. Since contact forces and torque lead to extreme changes of $\boldsymbol{f}_d, \boldsymbol{f}_c$, the distances between convex shapes are used as zero-crossing functions $z_i(t)$ so that the start and the end of a contact phase triggers an event. It is expected that this approach improves the reliability of a simulation. Furthermore, the elastic response calculation of section 3.3 needs the relative velocity when a contact starts, to compute an appropriate damping factor d. Therefore, an event at the start of a contact is mandatory.

Distances between convex shapes, as well as penetration depths are computed with an improved version of the Minkowski Portal Refinement algorithm (MPR-algorithm) [20]. The MPR-algorithm is much simpler to implement and has less numerical problems than the often used GJK/EPA-standard algorithms [2, 9], because it only works with triangles and not with tetrahedrons. In the original version of the MPR-algorithm [20] only penetration depths are determined. In Modia3D improvements of the MPR-algorithm are utilized that have been proposed in [12, 13], in particular to compute the distances of shapes that are not in contact and treating special collision situations properly.

Collisions of n potentially colliding shapes are handled in the following (mostly standard) way:

1. *Broad Phase*
   The shapes are approximated by *Axis Aligned Bounding Boxes*, see e.g. [2], where potential collisions and approximate distances can be very cheaply determined resulting in $O(n^2)$ cheap tests. When using special data structures (such as octrees or kd-trees), it is possible to reduce the number of cheap tests to $O(n \log(n))$.
2. *Narrow Phase*
   For the potentially colliding shape pairs as identified in the broad phase, the signed distances are computed with the improved MPR-algorithm [13].
3. *Response Calculation*
   If two shapes are penetrating, a normal and a tangential force, as well as a torque are applied at the contact point. For details, see section 3.3 and Fig. 4.

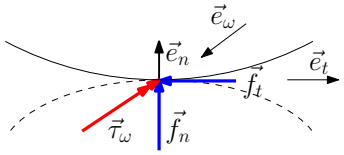

Fig. 4. Contact normal force $\vec{f}_n$, contact tangential force $\vec{f}_t$ (= sliding friction force) and contact torque $\vec{\tau}_\omega$ between two penetrating objects. $\vec{e}_n, \vec{e}_t, \vec{e}_\omega$ are unit vectors in direction of the respective relative movement.

3.3 Elastic Response Calculation

A contact normal force $\vec{f}_n$, a contact tangential force $\vec{f}_t$ (= sliding friction force), and a contact torque $\vec{\tau}_\omega$ are calculated, when two 3D objects penetrate each other with a penetration depth $\delta \geq 0$, as shown in Fig. 4. 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 force and torque of these distributions.

The MPR-algorithm computes the contact point, δ, and a unit vector $\vec{e}_n$ that is orthogonal to the contacting surfaces. The novel response characteristic of (2) utilizes ideas from [8, 17, 19] with some extensions and corrections. Variables with index $_{geo}$ are computed from the contacting geometries and vectors with index $_{reg}$ are regularized with smooth characteristics to avoid divisions by zero (for example, $|\vec{e}_{t,reg}| = 1$ with exception of a small region around, $|\vec{v}_{rel,t}| < v_{small}$ where $|\vec{e}_{t,reg}| = 0$ at $|\vec{v}_{rel,t}| = 0$ and it smoothly approaches 1 at $|\vec{v}_{rel,t}| = v_{small}$):

$$f_n = max\left(0, c_{res}\, c_{geo}\, \delta^{n_{geo}} \left(1 + d\,\dot{\delta}\right)\right) \qquad (2a)$$

$$\vec{f}_n = f_n \vec{e}_n \qquad (2b)$$

$$\vec{f}_t = -\mu_k f_n \vec{e}_{t,reg} \qquad (2c)$$

$$\vec{\tau}_\omega = -\mu_r \mu_{geo} f_n \vec{e}_{\omega,reg} \qquad (2d)$$

The symbols of (2) have the following meaning:

$\vec{e}_n$ Unit vector normal to the contacting surfaces.

$\vec{e}_t$ Unit vector in direction of the relative tangential velocity.

$\vec{e}_\omega$ Unit vector in direction of the relative angular velocity.

$c_{res}(E_1, E_2, \nu_1, \nu_2)$ Resultant spring constant in normal direction. $c_{res} = 1/(1/c_1 + 1/c_2)$; $c_i = E_i/(1 - \nu_i^2)$.

$d(cor_{reg}, \dot{\delta}^-)$ Damping coefficient as function of the regularized coefficient of restitution cor_{reg} and $\dot{\delta}$ when contact starts ($\dot{\delta}^- \geq 0$). Variable cor_{reg} is computed according to [8] and regularized so that $cor_{reg} = 0.001$ if the normal relative velocity $v_{rel,n} = 0$ and approaches cor when $v_{rel,n} = v_{small}$. One reason for this is that otherwise [8] would result in a division by zero if $cor = 0$. The other reason is that a bouncing object stays at rest if $v_{rel,n}$ becomes small enough and therefore cor_{reg} must be drastically reduced for small $v_{rel,n}$ (this effect is typically not taken into account in other contact laws, such as in [19]).

μ_k Kinetic/sliding friction force coefficient (≥ 0).

μ_r Rotational resistance torque coefficient (≥ 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 $\vec{\omega}_{rel} = 0$. For a ball, μ_r is the (standard) rolling resistance coefficient and μ_{geo} is the ball radius.

$c_{geo}, n_{geo}, \mu_{geo}$ depend on the geometries of the two objects in contact. If not enough information is available, these factors are set to one. c_{geo}, n_{geo} take the contact volume into account, under the assumption of Hertz' pressure (e.g. $n_{geo} = 3/2$ if at least one of the two contacting objects is a sphere).
The response characteristics (2) shall be clarified with a few special experiments:

Comparison between elastic and impulsive contact response. In Fig. 5, the height of the bouncing ball (see model of Fig. 3) is shown as a function of time. The red curve is the reaction when

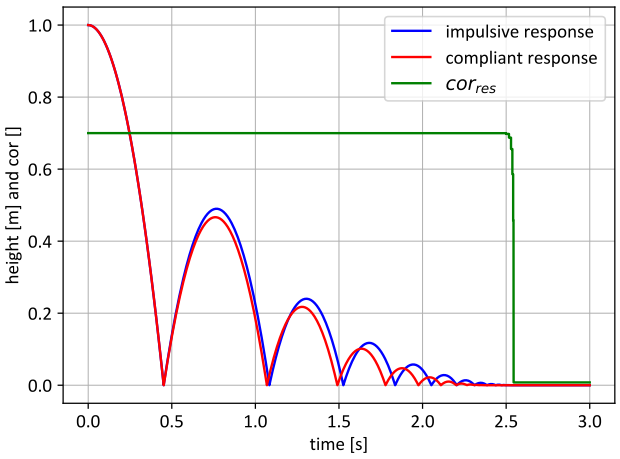

Fig. 5. Bouncing ball (see Fig. 3) with impulsive contact and the elastic response characteristic of (2).

using the elastic response calculation of (2) with $cor = 0.7$ and the solid material constants of steel and of dry wood. The blue curve is the result when the contact force is computed with an impulse for the same cor value. The green curve is cor_{reg} (for small velocities it becomes small). This shows that f_n in (2) leads to a similar reaction if compared to an impulsive response. Note, this was the intention for the development of this force law in [8]. A systematic comparison was made in [19].

Sliding and rolling ball. In Fig. 6 an animation of a billiard ball is shown that is sliding and rolling on a billiard table. In Fig. 7, time plots of some of the variables are shown. 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). It starts at its initial position $r_s = 0.2\ m$ with an initial velocity $v_s = 3\ m/s$. At initialization, the billiard ball is placed in touching position with the table (penetration depth $\delta = 0$). 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. The material data is shown in table 1.

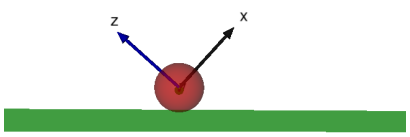

Fig. 6. Billiard ball sliding and rolling on a billiard table.

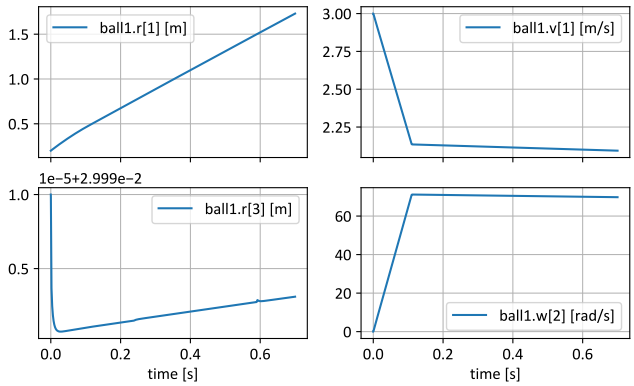

Fig. 7. Position of a sliding and rolling billiard ball in x- and z-direction, velocity in x-direction, and angular velocity in y-direction.

The upper left plot in Fig, 7 shows the position of the billiard ball in x-direction. The lower left plot of Fig. 7 shows the subsidence of the ball into the table which is in the order of $10^{-5}\ m$. The upper right plot shows the velocity of the ball in x-direction and the lower right plot displays the angular velocity in y-direction. The first $0.15\ s$ the ball is sliding. 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 around the y-direction. At $time = 0.15\ s$, the relative velocity in tangential direction $v_{rel,t}$ is zero

$$v_{rel,t} = v_{ball,1} - \omega_{ball,2} r_{ball} = 2.125 - 70 \cdot 0.03 \approx 0 \qquad (3)$$

and *ideal rolling* of the ball takes place. Since $v_{rel,t} = 0$, the sliding friction force $\vec{f}_t = 0$, because $\vec{e}_{t,reg} = 0$. Therefore, the ball would roll forever, contrary to reality. On the other hand, the rotational resistance torque $\vec{\tau}_\omega$ ($\mu_r = 0.02$) acts as a rolling resistance that reduces the angular velocity $\omega_{ball,2}$ starting at $time = 0.15\ s$ and the ball comes to rest at some point in the future outside of the plot area (see right upper and lower plot of Fig. 7). The result is that (2) is able to reproduce the effect of a sliding and rolling ball. However, for simplicity of the formulation, in this simplistic elastic response law, velocity dependency of the coefficients cor, μ_k, μ_r is neglected.

Collision of two balls. Two billiard balls are positioned on a billiard table (see Fig. 8. One has an initial velocity and hits the other resting ball after some time. Simulation results are shown in Fig. 9). Before the first ball hits the second one, the effects of sliding and rolling occur as analyzed before. At $time = 0.55\ s$, the first ball hits the second ball. Since the coefficient of restitution between the two balls is one, a fully elastic collision takes place. It is well-known that in this case the first ball stops and the second ball continues with the velocity of the first ball. This "exchange" of velocity can be seen in the middle plot of Fig. 9 at $time = 0.55\ s$. However, since the first ball was rolling, the angular momentum was greater

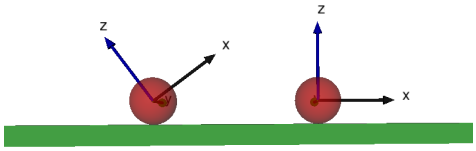

Fig. 8. Billiard ball colliding with another billiard ball.

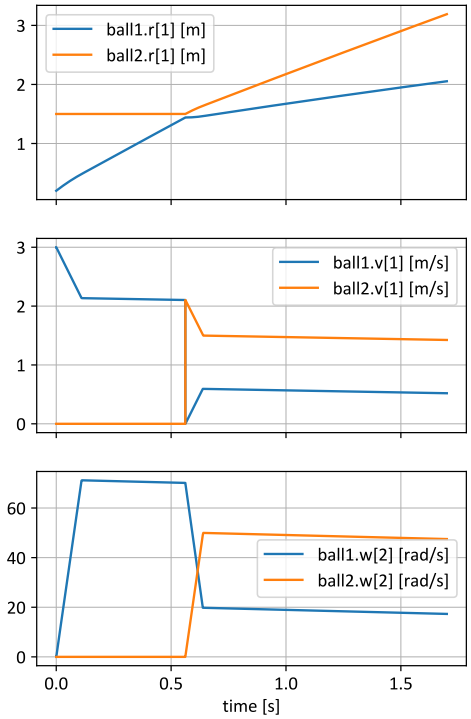

Fig. 9. Position of two colliding billiard balls, velocity in x-direction, and angular velocity in y-direction.

zero. This momentum is conserved. Therefore, the first ball continues rolling and velocity $v_{ball,1}$ starts from zero and becomes larger. Since the relative velocity is no longer zero due to the impact, again a friction force $\vec{f}_t$ is acting that introduces again a counter torque at the balls axis which quickly reduces the angular velocity until again the relative velocity is zero around $time = 0.65s$. Both balls are again ideally rolling and due to the rotational resistance torque, the angular velocities are slowly reduced until both balls come to rest (this is not shown in the plots).

3.4 Regularizing the Contact Area

A DAE integrator such as IDA, solves a nonlinear algebraic equation system at every time instant. If $\boldsymbol{f}_d, \boldsymbol{f}_c$ in (1) are not smooth or even not continuous, it is highly probable that no solution of this algebraic equation system is found and *simulation stops with an error message*. This situation easily occurs, for example, if the edges or vertices of two boxes collide on each other, because very small changes of the positions of the boxes can change the contact situation drastically. To improve the reliability of the simulation, Modia3D uses the approach proposed in [2], and is smoothing every shape with a small sphere (default radius $= 1\ mm$) that is (conceptually) moved over all surfaces. Practically, this smoothing can be very easily and cheaply incorporated in to the convex collision handling.

A box placed on a table is a standard configuration, as well as a box falling on a table and the box faces that come into contact are completely parallel to each other (see Fig. 10, left). Every method that computes only contact points on two colliding objects and the corresponding penetration depth, has the large disadvantage that easily an infinite number of solutions exists, as in this case: small changes in the configuration can result in very different contact

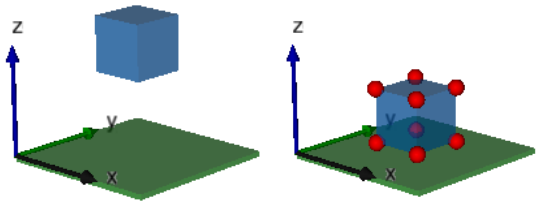

Fig. 10. Bouncing box. Left figure: vertex spheres in original size. Right figure: vertex spheres enlarged by a factor of 30.

points in the contact area. It is then highly probable that simulation with a variable-step integrator fails. Furthermore, the contact force and torque depend heavily on the size of the contact area. The penetration depth alone provides not enough information to calculate reasonable values that are in accordance with physics. For both cases, it would be necessary to take the complete contact area and volume into account, as, for example done in [4]. This is much more costly and probably only works reasonably for soft contacts where the contact volume is large enough. To summarize, collision handling with variable-step integrators and penetration depth computation can most likely only work reliably for *point contacts* that are smoothly changing.

To cope with this severe difficulty in a simple way, the Modia3D supported primitives are currently being regularized to enforce point contacts. For example, if a contact material is defined for a box, a small sphere (default radius $= 1\ mm$) is attached automatically at every vertex of a box. An example is shown in Fig. 10, where the left side of the figure shows the original size of these spheres and the right figure shows the spheres enlarged by a factor of 30. When such a box bounces on a table or is in rest on a table, point contacts between four spheres and the table occur and the integrator has no problem to compute a solution.

4. Example: Billiard Table with 16 Billiard Balls

The billiard table, in Fig. 11, has 16 billiard balls and is used as one benchmark for Modia3D. The material constants are shown in Table 1. The left most ball has an initial velocity to the right and hits the 15 other balls after a short time interval exactly at the top of the ball triangle. This results in a symmetric evolution of the balls, as one would expect. All previously described effects (sliding, rolling, colliding) act together. The hybrid DAE system has $\dim(\boldsymbol{x}) = 13 \cdot 16 = 208$ and there are about 200 possible collision pairs. Simulation on a standard PC needs about 20 minutes for 5 seconds simulation time. At the moment the code is implemented for functionality and not tuned for efficiency, so a speed-up is expected in the future.

Table 1. Material constants for billiard table.

			$E\ [N/m^2]$	ν
ball			5.4e9	0.34
table			1.1e10	0.4
		cor	μ_k	μ_r
ball	table	0.0	0.6	0.02
ball	ball	1.0	0.0	0.0
ball	cushion	0.8	0.0	0.0

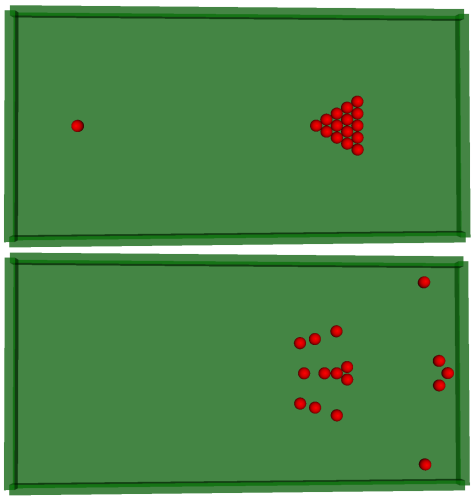

Fig. 11. Initial setting of 16 billiard balls (top). Billiard balls after 5 s (bottom).

5. Conclusion

In this article, a short overview about the experimental 3D modeling environment Modia3D is given. In particular, collision handling with a variable-step integrator has been sketched and a novel formulation for elastic response calculation is proposed. The Modia3D collision and contact handling is demonstrated with several examples.

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 in general to compute results that are close to physics.

Modia3D is still a prototype implementation and several important parts are under development, especially the integration with Modia is missing. 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 been performed yet.

6. References

[1] T. Bellmann. Interactive Simulations and advanced Visualization with Modelica. In Francesco Casella, editor, *Proc. of the 7th International Modelica Conference*. LiU Electronic Press, Sept. 2009.

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

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

[4] H. Elmqvist, A. Goteman, V. Roxling, and T. Ghandriz. Generic Modelica Framework for MultiBody Contacts and Discrete Element Method. In Peter Fritzson and Hilding Elmqvist, editors, *Proc. of the 11th International Modelica Conference*. LiU Electronic Press, Sept. 2015.

[5] H. Elmqvist, T. Henningsson, and M. Otter. Systems Modeling and Programming in a Unified Environment based on Julia. In *Proc. of ISoLA Conference*. Springer, Oct. 2016.

[6] H. Elmqvist, T. Henningsson, and M. Otter. Innovations for Future Modelica. In J. Kofranek and F. Casella, editors, *Proc. of the 12th International Modelica Conference*. LiU Electronic Press, May 2017.

[7] T. Erez, Y. Tassa, and E. Todorov. Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

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

[9] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[10] M. Hellerer, T. Bellmann, and F. Schlegel. The DLR Visualization Library - Recent development and applications. In Hubertus Tummescheit and Karl-Erik Arzen, editors, *Proc. of the 10th International Modelica Conference*. LiU Electronic Press, March 2014.

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

[12] B. Kenwright. Generic Convex Collision Detection using Support Mapping. Technical report, 2015.

[13] A. Neumayr and M. Otter. Collision Handling with Variable-step Integrators. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT'17, pages 9–18. ACM, 2017.

[14] A. Neumayr and M. Otter. Component-Based 3D Modeling of Dynamic Systems. In M. Tiller, H. Tummescheit, and L. Vanfretti, editors, *Proceedings of the American Modelica Conference*, Oct. 2018.

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

[16] R. Nystrom. *Game Programming Patterns*. Genever Benning, 2014.

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

[18] C. Rackauckas and Q. Nie. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 2017.

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

[20] G. Snethen. Xenocollide: Complex collision made simple. In Scott Jacobs, editor, *Game Programming Gems 7*, pages 165–178. Charles River Media, 2008.