

# Generic Modelica Framework for MultiBody Contacts and Discrete Element Method

Hilding Elmqvist<sup>1</sup>, Axel Goteman<sup>1,2</sup>, Vilhelm Roxling<sup>1,2</sup>, Toheed Ghandriz<sup>1</sup>

<sup>1</sup>Dassault Systèmes, Lund, Sweden, {Hilding.Elmqvist, Toheed.Ghandriz}@3ds.com

<sup>2</sup>Lund Institute of Technology, Lund, Sweden, {axel.goteman, vilhelm.roxling}@gmail.com

## Abstract

A generic framework for mechanical modeling of objects that collide and have contact is presented. It can be used in combination with the Modelica MultiBody library and to model granular objects using DEM (Discrete Element Method). The shapes of the objects are given by general triangular meshes.

*Keywords:* MultiBody models, Discrete Element Method, Collision detection, Contact handling

## 1 Introduction

Many real-world system behaviors depend on contact between mechanical bodies. Examples are walking, vehicle on a road, mechanisms in mechanical watches and many types of manufacturing machines.

### 1.1 Earlier Modelica Based Solutions

There have already been several developments to support collision handling in Modelica. In (Otter, et al 2005) is described a solution based on a collision handling software called Solid. The contact force calculations take into account the contact patch, i.e. also rotational friction torque is handled. The paper (Oestersötebier et al, 2014) introduces non-central contact blocks in which the contact surfaces are defined. (Hofmann, 2014) discusses the use of the Bullet Physics Library and deepest point penetration for force calculations. Unfortunately this point might not be unique which then results in unrealistic simulation results.

### 1.2 Discrete Element Method

The Discrete Element Method (DEM) has a focus on handling many interacting objects. Typically, both positional and rotational degrees of freedom are handled, i.e. 6 DOFs per object. The geometries can be complex, e.g. polyhedral. Many different force models can be used depending on what matter is studied.

It has been noted that simply using penetration depth, which is often calculated by collision packages, are not suited for continuous time simulation because of the discontinuity in the forces and where the forces act. DEM typically handles this by calculating penetration volume. See, for example, (Chen, 2012).

(Hippman, 2003, 2013) also considers the penetration volume and makes force calculations based on the surface patches.

### 1.3 Contribution

A generic framework for mechanical modeling of objects that collide and have contact is presented. It can be used in combination with the Modelica MultiBody library and to model granular objects using DEM (Discrete Element Method). The shapes of the objects are given by general triangular meshes. The special case of sphere is also supported in order to handle tens thousands of objects for DEM. The contact handling is organized using ExternalObjects, i.e. C and C++ code. Each body in the scene registers its current position which is given as the solution of the Modelica motion equations. After that a centralized routine of the scene calculates and adds all forces between pairs of bodies in contact. The force calculation is done using the intersection volume found by the CSG (Constructive Solid Geometry) intersect operator. We have used a generalization of the Hertz contact model developed by (Nassauer, 2013), where the force is proportional to  $\sqrt{Vd}$ , with  $V$ =penetration volume and  $d$ =penetration depth. The force is acting in the centroid of the penetration volume. The details are given in section 6.1.

### 1.4 Example: Simple Objects

As an example of how the presented library can be used, consider the billiard like set-up in Figure 1 of 100 layers of balls, i.e. 5050 balls on a table plus one ball hitting from below.

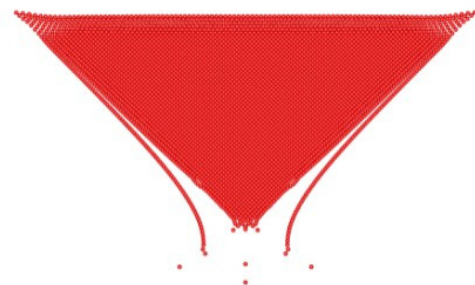
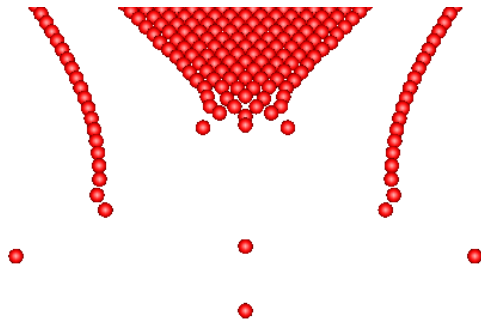


Figure 1. 5050 balls hit from below

The area around where the first collision happens is enlarged in Figure 2.



**Figure 2.** Zoomed in on the first colliding balls  
Modeling such scenario is quite simple:

```
model Billiard
parameter Integer layers=100;
parameter Integer n=div(layers*(layers+1),2);
inner CollidingWorld collidingWorld;
Sphere sphere[n]
(x0={{layer(i)*sqrt(3)/2,
column(i)-(layer(i)-1)*0.5, 0}
for i in 1:n}, each radius=0.5) ;
Sphere sphere1(x0={-5,0,0}, v0={1,0,0}, radius=0.5);
end Billiard;
```

The trivial functions layer and column calculates the position of the  $i^{\text{th}}$  ball in the triangle.

## 2 Modeling Solids

### 2.1 Triangle Mesh

One popular method of representing solids is to define its closed surface by a set of triangles with its counter clockwise normal pointing outwards. The following Modelica record is used:

```
record TriangleMesh "Defines solid by triangle mesh of its surface"
parameter Real vertices[: ,3] "3D coordinates of points";
parameter Integer triangles[: ,3](min=1)
"Triangle structure based on vertices (index array)";
end TriangleMesh;
```

i.e. all vertices are defined in a separate vector and the triangles are defined using indices into this vector.

### 2.2 Solids Defined by Functions

A cube can be defined as a function:

```
function cube
output TriangleMesh mesh=TriangleMesh(
vertices={{0,0,0}, {0,1,0}, {0,0,1}, {0,1,1},
{1,0,0}, {1,1,0}, {1,0,1}, {1,1,1}},
triangles={{1,4,2}, {1,3,4}, {1,5,7}, {1,7,3}, {5,8,7}, {5,6,8},
{2,4,8}, {2,8,6}, {4,3,8}, {3,7,8}, {1,6,5}, {1,2,6}});
end cube;
```

The library contains a set of functions for defining the triangular meshes of basic primitive shapes such as box, cylinder and sphere. In addition, functions for translating, rotating and scaling triangle meshes are available.

### 2.3 Polygon Extrusion

A convenient method of defining solids, especially for early concept studies, is to extrude polygons.

#### 2.3.1 Editing Polygon as Icon

Polygon editing is available in Modelica tools since one of the icon primitives is Polygon. Dymola has an API function to retrieve annotations from Modelica classes. This function is used to define a parameter vector of 2D coordinates. We propose that such a function is included in the Modelica standard.

An example of such use is shown in Figure 3 for a pallet level of a watch. A picture was imported into the icon editor and a polygon was drawn over the picture.

#### 2.3.2 Concave Polygon Triangulation

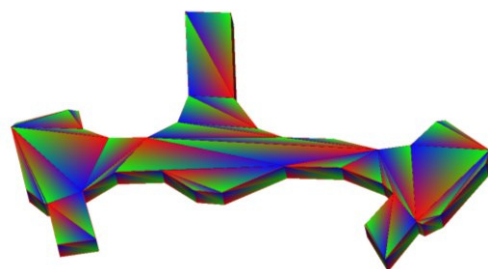
The polygon, which can be concave, needs to be triangulated in order to fit into the triangular mesh representation.

It is assumed that the polygon is defined counter clockwise. The vertices of the polygon need to be traversed and inspected for “ears”, i.e. when the edges make a “right turn”. A simple algorithm is to remove vertices at “left turns” and generate the corresponding triangles first. The polygon is therefore traversed as long as there are at least 3 remaining vertices.

#### 2.3.3 Extrusion

The extrude function takes a potentially concave polygon as input and an extrusion height parameter. It triangulates the polygon and converts to 3D coordinates for front and back. The extrusion sides are trivially triangulated by splitting the rectangles defined by the polygon vertices from the front and back polygon respectively.

The resulting triangular mesh for extruding the pallet lever polygon is shown in Figure 3. The triangle vertices are colored red, green and blue and interpolation has been performed over the triangle surface.



**Figure 3.** Triangle mesh of extruded part

### 2.3.4 Regular Forms

For regular forms such as a gear wheel, it's convenient to just define the polygon for one tooth and use replication to define the entire form. In order to replicate around a circle, the polygon needs to be rotated. Functions to translate, rotate and scale polygons are defined.

Figure 4 shows how the above operations can be used to define an escape wheel of a watch.

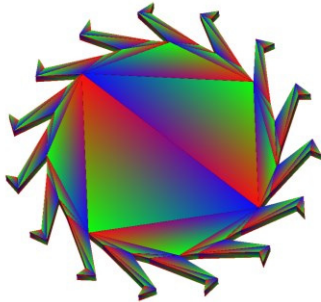


Figure 4. Triangle mesh of extruded regular form

### 2.4 Constructive Solid Geometry Operations

More complex geometries can be defined using constructive solid geometry (CSG). It enables combining triangular meshes by the operations: union, difference and intersection.

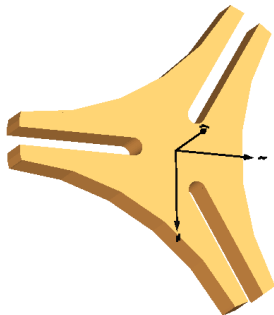


Figure 5. Triangle mesh of part created by CSG operations

The solid in Figure 5 has been created using the following Modelica code (the parameters are defined in section 8.2):

```

mesh := rotate(Cylinder(radius, height=height), {pi/2,0,0});
for i in 1:nSlots loop
  mesh := difference(mesh, rotate(translate(
    Box({slotLength,height,slotWidth}),
    {radius-slotLength,-height,-slotWidth/2}),
    {0,2*pi/nSlots*(0.5+i),0}));
  mesh := difference(mesh, rotate(translate(rotate(
    Cylinder(slotWidth/2, height=height), {pi/2,0,0}),
    {radius-slotLength,0,0}), {0,2*pi/nSlots*(0.5+i),0}));
  mesh := difference(mesh, rotate(translate(rotate(
    Cylinder(stopArcRadius, height=height), {pi/2,0,0}),
    {centerDistance,0,0}), {0,2*pi/nSlots*i,0}));
end for;

```

A Cylinder is first constructed. A for loop is then used to remove boxes and cylinders from the solid by using the CSG difference operator.

This method of defining solids, i.e. a textual definition of object creation, extrusion, CSG operations, etc, can also be found in certain CAD packages such as OpenSCAD and OpenJSCAD.org. The latter uses JavaScript to define the solid objects and the operations.

### 2.5 Use of CAD data

The presented Modelica library also contains functions for reading triangle meshes from CAD files. The current implementation allows reading DXF files. By using converters other CAD formats can also be used such as VRML.

## 3 Modeling Idiom for Pairwise Coupling of Objects

It is important that the end user does not have to care about setting-up individual possible contact pairs. It should be possible to just drag an object into the scene, which is possible in Playmola (Elmqvist, et al., 2015), and automatically get collision behavior.

A unique Integer id needs to be defined by the user both for the solutions of (Otter, et al., 2005) and (Oestersötebier et al., 2014).

Our solution solves this problem as follows. The contact handling is organized using ExternalObjects, i.e. C and C++ code. Each body in the scene registers its position which is given as the solution of the Modelica motion equations. After that a centralized routine of the scene calculates and adds all forces between pairs of bodies in contact. These forces are then retrieved for each body and used in their motion equations.

The difficulty is to properly synchronize the external calculations of the forces. All current positions and orientations must be known then.

We use inner outer constructs together with flow variable declarations in the following way. When a body calls `setBodyTransformation`, it also assigns `bodyMoved` which is defined as `sync.synchronize.done`. When the body calls `getBodyForces` an extra term is added: `collidingWorld.forceCalculated`.

```

model Body
  extends Contacts.Synchronize.CollidingObject;
  Real bodyMoved = sync.synchronize.done;
  Modelica.SIunits.Force f[3];
equation
  bodyMoved = setBodyTransformation(...);
  f = getBodyForces(b,time) +
    collidingWorld.forceCalculated;
  ...
end Body;

```

The term `forceCalculated` will not have a value until all bodies have defined `bodyMoved` and the `calculateForces` function of the external scene object has been called. This is accomplished in an algorithm in the inner instance of `CollidingWorld`:

```

model CollidingWorld
  ExternalScene scene=ExternalScene(id="s");
  Synchronize.SynchronizeConnector synchronize;
  Real bodyMoved = synchronize.done;
  Real forceCalculated[3];
  Real dummy;
equation
  synchronize.do = 0;
algorithm
  dummy := bodyMoved;
  calculateForces(scene);
  forceCalculated :=fill(0, 3);
end CollidingWorld;

```

The trick is then that `bodyMoved` of `CollidingWorld` should depend on the `bodyMoved` of all the bodies. `bodyMoved` is defined using a `Real` flow variable `done`.

```

connector SynchronizeConnector
  Real do;
  flow Real done;
end SynchronizeConnector;

```

This means that all the outer `done` variables of `CollidingWorld.synchronize` are summed together with the inner `done` of `CollidingWorld.synchronize`. However, to achieve this zero sum semantics, there needs to be some connections to `collidingWorld.synchronize`. Each body extends `CollidingObject` and gets the outer declaration, an instance of `SynchronizeModel` and the needed connect statement.

```

model CollidingObject
  outer CollidingWorld collidingWorld;
  SynchronizeModel sync;
equation
  connect(collidingWorld.synchronize, sync.synchronize);
end CollidingObject;

model SynchronizeModel
  SynchronizeConnector synchronize;
end SynchronizeModel;

```

This is indeed complex. However, a useful consequence of Modelica semantics of inner-outer combined with connectors having a flow variable. Fortunately, the end user does not need to care about this.

This idiom is documented here since it might be useful in other circumstances when external objects are involved and careful synchronization of calls to external functions are necessary.

## 4 Extension to MSL MultiBody library

A general body with collision behavior and general triangle mesh shape has been developed as a wrapper to `Modelica.Mechanics.MultiBody.Parts.Body`. The mass and inertia are automatically calculated. It has the usual frame connector, i.e. it can be used together with joints and force elements in the usual way.

## 5 Contact detection

Often, the most time consuming part of a DEM simulation is the contact detection. It is therefore crucial that it is as efficient as possible, for speed considerations. The contact detection can be divided into two phases, the broad phase and the narrow phase. The broad phase consists of finding potential collision pairs among all the bodies in the scene, while in the narrow phase, those pairs are checked in detail for collision. For more detailed descriptions than present in this section, see (Goteman, et al., 2015).

### 5.1 Without a broad phase

To begin with, a broad phase is not necessary in cases where the number of bodies in the scene,  $n$ , is low. In those cases the body shapes may also be more complex, which means that the narrow phase will take the majority of the time, and a broad phase would not make an impact anyway.

Without a broad phase, every body has to be checked against all other bodies, resulting in a time complexity of  $O(n^2)$ . This does not mean that a detailed investigation is needed between every pair of bodies. Normally, a *bounding volume* of every body is determined. If the bounding volumes of two bodies intersect, that pair can be checked in more detail. The most common bounding volume type is the *axis aligned bounding box (AABB)*, but one can also use e.g. the *bounding sphere* (centered in the centroid of the objects).

### 5.2 Broad phase

As  $n$  increases, it gets expensive to even check bounding volumes for all bodies. The point of having a broad phase is to remove the quadratic time dependency. The most common approaches uses some kind of tree structure into which the objects are placed, resulting in an  $O(n \log(n))$  time complexity.

As an example, imagine 1000 balls placed in a straight line, every pair being precisely at contact. Without a broad phase, approximately half a million bounding volume checks would be carried out, assuming that every pair of bodies is only checked once. With a good broad phase however, only 999 checks would be needed.



### 5.2.1 Morton codes

In this solution, the so called *Z-order* curve has been used in the broad phase. This space filling curve arises when doing the following:

Discretize 3d-space into *base cells* so that the coordinates of each base cell can be described by an integer. Translation of coordinates should also be done, so that the integers are unsigned. Now the idea is to assign a so called *Morton code* to each base cell. The Morton code is determined by interlacing the binary representation of the three coordinates, according to the example in Figure 6.

$$\begin{array}{r}
 x = 10010 \\
 y = 11100 \\
 z = 01001 \\
 \Downarrow \\
 \begin{array}{r}
 x = \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \\
 y = \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \\
 z = 0 \quad 1 \quad 1 \quad 0 \quad 0 \\
 c = 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0
 \end{array}
 \end{array}$$

Figure 6. Morton code construction

$c$  is the Morton code, and as shown above, starting from the least significant bit, the lowest x-bit comes first, then y's lowest, then z's lowest. Then the second lowest x-bit, and so on.

If the base cells are sorted according to their Morton codes, the order will follow the curve shown in Figure 7 (in 2D).

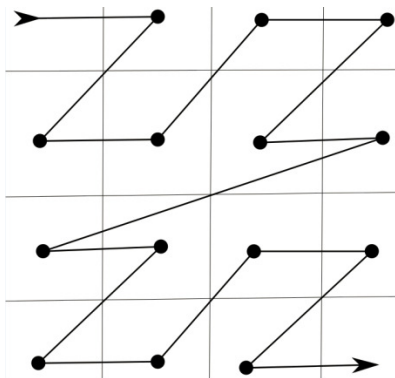


Figure 7. Z-curve ordering

For more details on the Z-order curve and Morton encoding, see e.g. (Karras, 2012).

### 5.2.2 The algorithm

The algorithm used in this solution is based on the algorithm presented by (Lavrov, 2014). The idea is that every body is occupying one to eight *cells*, where the sizes of the cells are represented by the level of the corresponding octree-node. A cell is not necessarily a base cell, but a cubic cell within which the Z-order curve covers every base cell before leaving. Given the

size and lowest Morton code of a cell, the maximal Morton code can also be calculated, and the cell can be represented by a one dimensional interval. A short description of the algorithm:

- 1.) For each body, generate its Morton code-based intervals (one to eight). This should be done according to the body's AABB. The information should be inserted in some data structure, where each element contains the interval, and a body ID.
- 2.) Sort the data structure according to the start code of the intervals.
- 3.) For each interval:
  - a. Iterate forward in the data structure until a start code is found that is greater than the end code of this interval.
  - b. For every start code passed within this interval, push a pair containing this intervals ID and the others, to another data structure.
- 4.) Remove duplicates from the data structure with pairs.
- 5.) For each pair:
  - a. Check the bodies AABBs against each other.
  - b. If they intersect, a more detailed analysis is needed (narrow phase).

### 5.2.3 Parallelization using GPU

In this section, a short explanation of possible parallelizations of step 1-4 is presented. The NVIDIA CUDA programming model was used for this parallelization, and we refer to (Elmqvist, et al., 2015) for a brief introduction to general GPU architectures and CUDA.

The most computationally heavy part of the algorithm is sorting. There are in fact two sorts in the algorithm, as step 4 also requires a sorted vector to operate on. In the first sort, the generated intervals are sorted, and the number of elements to be sorted is therefore of the same magnitude as the number of bodies in the scene. In the second sort, possible collision pairs are sorted, and the numbers of pairs are therefore directly linked to how crowded the scene is. Sorting in the broad phase can be done in parallel, by using the CUDA library Thrust's parallel radix sort.

Also the Morton encoding can be done in parallel, the only problem being that the number of generated intervals per object is unknown. This is a problem since pushing data to the end of a storage container results in the need of atomic operations, which generally has a negative effect on GPU performance. However, using the fact that an object generates at most eight intervals, each thread on the GPU can work on its own separate piece of memory, with gaps removed after encoding. Because of how the Morton encoding is done and the choice of grid size, most

objects generate eight intervals, minimizing the gap removal.

Step three could not be effectively parallelized, since no effective method was found for determining the number of generated collision pairs per interval. Without a guaranteed upper limit on collision pairs, the Morton encoding method could not be used, and using atomic operations proved to be too slow, due to the high number of access conflicts.

### 5.3 Narrow phase

When the AABBs of two bodies are colliding, a method is needed to detect if the bodies are actually colliding. If a collision is detected, the overlapping region also has to be determined in order to calculate the resulting forces.

For this, the CSG (Constructive Solid Geometry) intersect operator, using BSP (Binary Space Partitioning) trees, is used (Segura, 2013). It is a suitable method since it works well and fast on arbitrarily shaped bodies that are described by their polygon surfaces.

In a BSP tree, every node has two children. A node has a plane in 3d space, and its left child represents one of the half spaces created by this plane, and the right child represents the other. All operations on the tree are naturally defined recursively. BSP trees are widely used in e.g. computer graphics, where the planes might be chosen as the walls of a room.

When using BSP trees for CSG operations, the planes are chosen as coplanar with the surface polygons of the bodies. It results in the left child being inside the polygon, and the right child being outside (or the other way around, depending on how you define your tree), see Figure 8.

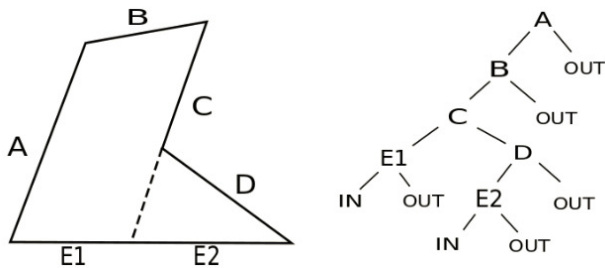


Figure 8. BSP tree

Note that polygon (line in 2D) E has to be split by the plane coplanar to polygon C in the tree construction, but not A by D. This is because the tree structure depends on which polygon is chosen as root.

It is now possible to efficiently determine e.g. which part of a body surface is inside another. Polygon by polygon, the surface can be traced down the tree, until it comes to an empty node, which is either in or out. The polygons are split if they span both sides of a plane.

The implementation is made in C++, building on principles from a ported version of the JavaScript library csg.js, by (Wallace, 2012).

### 5.3.1 Multiple Contacts

The intersection from a collision between concave bodies may consist of multiple contact regions. The result of the CSG intersection contains no information about this, so an algorithm is needed, that splits up the provided set of polygons (if needed). A short description of a solution:

The algorithm assumes that the original intersection consists of one or more closed volumes, which are to be represented as a vector of intersections. The algorithm goes through all polygons in the original intersection and checks the polygons vertices against the vertices of each intersection. If no match is found, a new intersection is created. If one match is found, the polygon is added to that intersection. If multiple matches are found, the corresponding intersections are merged and the polygon added.

### 5.4 Iterative reformulation

Motivated by the fact that the narrow phase generally was the computationally heaviest part of a simulation, except for simulations of spheres, attempts to accelerate this phase were made. The critical part of the narrow phase is when polygons of one body are traversing the other body's tree, so that became the focus.

Every polygon of a body traverses the tree independently, which motivated to do the traversal in parallel. Then, by traversing all polygons in all collision pairs simultaneously, high parallelism is possible both with few complex objects, and many simple ones, see figure 9.

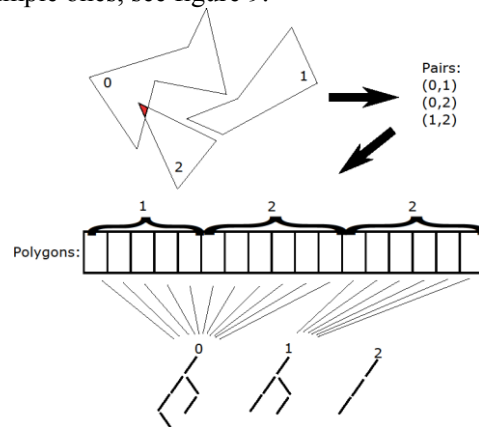


Figure 9. Illustration of the new traversal algorithm.

To achieve this kind of traversal, a major restructuring of the CSG algorithm were carried out. Incidentally, this restructuring resulted in a large performance boost for CPU execution as well. In fact, it performed as good as or better than the GPU in most cases.

The three major reasons for the lack of GPU acceleration are:

- High instruction divergence. Nearby threads may take different paths in the tree.
- High data divergence. Nearby threads might access non-coalesced memory locations.
- Each thread requires a large amount of registers, decreasing the occupancy.

## 6 Force Calculations

### 6.1 Normal force

When the intersection between two bodies is determined, the resulting forces need to be calculated. Because the colliding bodies are arbitrarily shaped, the overlapping region (the intersection) is given as a polyhedron with arbitrary shape, which implies that a classification of type of collision is hard to make. Thus a collision type independent model is needed. One such model is proposed in (Nassauer, 2013), where the contact force for elastic response is volume dependent. The derivations are based on Hertz model for contacts between spheres. With the assumption that the contact region is small with respect to the bodies in contact, it leads to:

$$\mathbf{F} = E k \sqrt{V d}$$

Here  $E$  is Young's modulus of the objects,  $V$  is the volume of the overlapping region,  $d$  is the penetration depth and

$$k = \frac{4}{3\sqrt{\pi}}$$

It is shown that this is actually a generalization of Hertz model.

The force is applied at the centroid of the overlapping region. To determine the force direction, constant pressure is assumed within the intersection. The direction can then be determined by weighing each polygon's normal with its area, sum over the polygons from one of the bodies, and normalize:

$$\mathbf{n}_f = \frac{\sum_j A_j \mathbf{n}_j}{\sum_j A_j}$$

$A_i$  and  $\mathbf{n}_i$  are the area and normal direction of polygon  $i$  respectively. This gives a behavior similar to that of a body floating in a liquid, the only difference being the assumption of constant pressure. An illustration of the polygonal contributions for the force direction is shown in Figure 9, taken from the debug window developed together with this package.

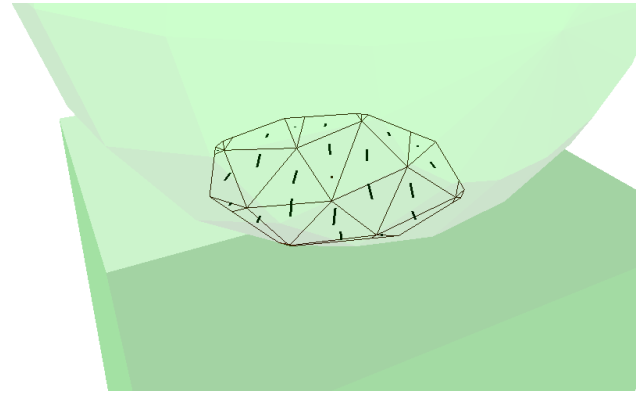


Figure 9. Intersection volume and forces

The penetration depth is defined as the extension of the overlapping region in the direction of  $\mathbf{n}_f$ , see Figure 10, and can be found as

$$d = \max_j (\mathbf{n}_f \cdot \mathbf{p}_j) - \min_j (\mathbf{n}_f \cdot \mathbf{p}_j)$$

$\mathbf{p}_i$  is the position of vertex  $i$ , and  $j$  ranges over all vertices in the intersection.

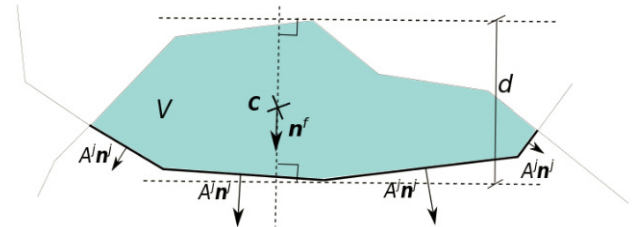


Figure 10. Penetration depth

### 6.2 Additional forces

In this context, the normal force is not sufficient to describe the contact interaction between bodies. (Nassauer, 2013) also proposes models for damping and friction, described below.

#### 6.2.1 Damping

Energy dissipation can be introduced by modifying the normal force equation to

$$\mathbf{F} = E k \sqrt{V d} (\mathbf{1} + c \mathbf{v}_d),$$

where  $c$  is the damping constant and  $\mathbf{v}_d$  is the relative velocity between the bodies in the orientation of  $\mathbf{n}_f$ . The points for which the velocities are calculated are chosen as the centroid of the intersection for both bodies.

#### 6.2.2 Friction

The classical Coulomb friction model turns out to be very hard to implement in DEM simulations. There are many reasons for this, the most important being that, in the case of static friction, all other forces acting on the body have to be known.

So instead, the following, velocity dependent model is used:

$$F_f = \left( (2\mu_{s*} - \mu_k) \frac{x^2}{x^4 + 1} + \mu_k - \frac{\mu_k}{x^2 + 1} \right) F$$

$F$  is the normal force,  $x=v_t/v_s$ , where  $v_t$  is the magnitude of the relative tangential velocity between the bodies, and  $v_s$  is the velocity for transition from static to kinetic friction.  $v_t$  can be found by projecting the relative velocity between the bodies, again taken at the centroid of the intersection, onto the plane of which  $\mathbf{n}_f$  is normal. Also

$$\mu_{s*} = \mu_s (1 - 0.09 \left( \frac{\mu_k}{\mu_s} \right)^4),$$

where  $\mu_s$  and  $\mu_k$  are the coefficients of friction for static and kinetic friction respectively. This generates the following friction behavior.

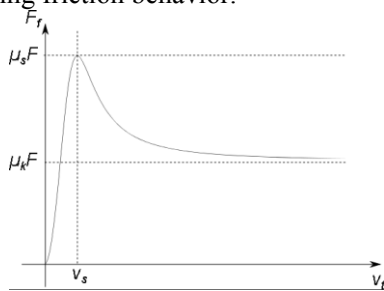


Figure 11. The friction model

### 6.2.3 Rotational damping

To simulate the friction caused by rotation at the contact point, a model with a damping effect is proposed. It applies a torque in the opposite direction of the relative angular velocity at the contact point (in the direction of  $\mathbf{n}_f$ ). Its magnitude is proportional to the angular velocity, the area of the contact region, and the normal force, according to

$$\mathbf{M}_r = -AFc_r(\boldsymbol{\omega}_1 - \boldsymbol{\omega}_2)\mathbf{n}_f$$

$A$  is the area of the contact region,  $c_r$  is a constant for calibration of the model, and  $\boldsymbol{\omega}_i$  is the angular velocity of body  $i$ , which is the same in any point of a rigid body. The area can be approximated by projecting the triangles from one body onto the plane to which  $\mathbf{n}_f$  is normal, and summarize:

$$A = \sum_j A_j \mathbf{n}_j \mathbf{n}_f$$

This model does not take into account the fact that the friction force most probably is not uniformly distributed across the contact area. A better solution (not yet implemented) would be to integrate over the area, and apply the friction model from section 6.2.2 on the triangles.  $F$  should then be replaced by the corresponding local contribution to the normal force, and  $v_t$  by the tangential component of  $\boldsymbol{\omega} \times \mathbf{r}_i$ , where  $\mathbf{r}_i$  is the position of the triangle relative to some fix point, e.g. the centroid of the intersection.

## 6.3 Additional algorithms for polyhedrons

### 6.3.1 Volume and centroid

To complete the force calculations, algorithms to compute the volume and the centroid of the intersection are needed. Those algorithms are also needed to compute center of mass and mass of a triangular meshed body. This is done by (Nürnberg, 2013), for closed polyhedrons of arbitrary shape. Conveniently, the algorithms only depends on the surface of the polyhedron, which is what is produced from the CSG intersection operation.

### 6.3.2 Moment of inertia

An algorithm to compute the inertia tensor of the bodies is also needed. This can be done for an arbitrary polyhedron (with triangle faces) in the following way:

Pick a point, e.g. the point around which the inertia tensor is needed. If another point is chosen, a simple translation to the desired point has to be carried out afterwards.

The integration over the body's volume can be transformed to a sum of integrals over tetrahedrons. Those tetrahedrons are the ones created by connecting the chosen point with the triangles of the body surface. Now, for one such tetrahedron, if the normal of the triangle from the object's surface points into the tetrahedron, the contribution from this tetrahedron is to be subtracted, otherwise added, to the total inertia tensor.

What remains is to actually calculate the moment of inertia of arbitrary shaped tetrahedrons. (Tonon, 2004) presents expressions for this in terms of the vertex coordinates.

## 7 Animation

### 7.1 Additional debug window

In order to allow for debugging and analysis of contact behavior, OpenGL was used to create a new animation window for Dymola. This window has expanded capabilities in order to simplify debugging of contact mechanics.

It allows viewing of the individual polygons of a body, which gives the option of viewing the intersections in the event of a collision. The window also supports drawing of the forces, both the contribution of every individual intersection polygon and the combined collision force. The resulting torque of the collision force is also viewable.

### 7.2 New ModelicaServices Model for Triangle Mesh

The ModelicaServices package which might have different implementations for different tools, has some models for animation of certain predefined shapes such



as box, cylinder and sphere, DXF-files and parametric surfaces. Parametric surfaces are defined by a function of two independent parameters and return the corresponding 3D position.

The above capabilities are not sufficient for animation of triangular meshes. We propose that ModelicaServices is extended with such a model. It will be similar to ModelicaServices.Animation.Surface, defining the surface by:

```
input Real vertices[:,3]
  "3D coordinates of points";
parameter Integer triangles[:,3]
  "Triangle structure based on vertices"
```

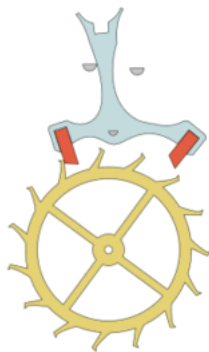
Note that vertices can be time dependent, i.e. it is possible to animate flexible bodies.

Three different coloring schemes are proposed: one color for entire shape, separate color for each triangle and separate color for each vertex with interpolation over triangle. The last most powerful alternative supports animating stresses in flexible bodies.

## 8 Examples

### 8.1 Lever Escapement

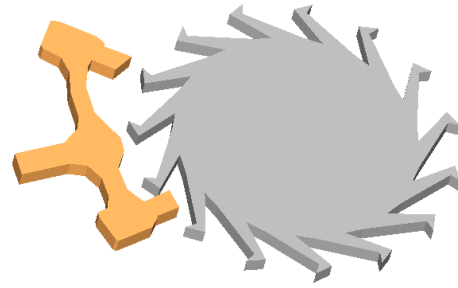
The lever escapement was invented around 1755 and is used in watches to convert an oscillation to a rotational motion, see Wikipedia ("Lever escapement"), Figure 12:



**Figure 12.** Lever escapement of mechanical watch

There are two contact problems: One for making sure the rotation ticks at the desired frequency; the other to insert more energy from the spring of the watch to the oscillator.

The shape of the pallet lever was extracted from (British Horological Institute, 2011) and a polygon was created and extruded. The escape wheel was defined as a regular form as described in section 2.3, see Figure 13.



**Figure 13.** Pallet lever and escape wheel

### 8.2 Geneva Mechanism

The Geneva mechanism has been used since long ago to achieve intermittent motion (Bickford, 1972). It is used, for example, in mechanical watches and in film projectors to move the film.

The drive wheel (grey in the figures below) has a pin which rotates the output wheel (yellow) when in contact in the slots. When the pin is not in contact, the output wheel locks its locking surface by sliding against the locking ring of the drive wheel.

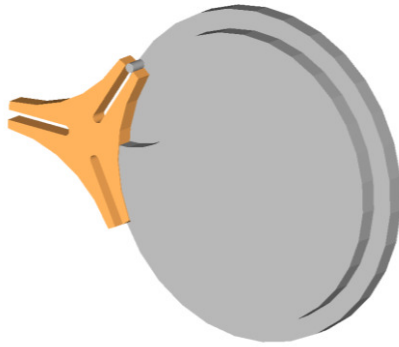
The dynamics is thus quite complex and there is a fast change in the acceleration when the drive pin enters and leaves the slot.

#### 8.2.1 Parametric mechanism

Depending on how many output slots there are, the geometry of the parts is given. The Modelica function generating the output wheel by CSG operations in section 2.4 has the following variable declarations showing the dependencies.

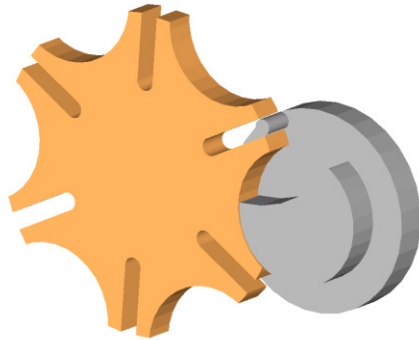
```
input Real radius=1 "Geneva wheel radius";
input Integer nSlots(min=3)=3 "Number of driven slots";
input Real pinDiameter=radius/10 "Drive pin diameter";
input Real clearance=radius/100;
input Real height=radius/5;
output TriangleMesh mesh;
protected
Real pi=Modelica.Constants.pi;
Real centerDistance = radius/cos(pi/nSlots);
Real driveCrankRadius = sqrt(centerDistance^2-radius^2);
Real slotLength = radius + driveCrankRadius - centerDistance;
Real slotWidth = pinDiameter + clearance;
Real stopArcRadius = driveCrankRadius - pinDiameter*1.5;
Real stopDiscRadius = stopArcRadius-clearance;
Real clearanceArc = radius*stopDiscRadius/driveCrankRadius;
```

It is therefore possible to define a parametric mechanism, i.e. that the shapes of all parts of the mechanism are parametric. The Geneva mechanism with  $nSlots=3$  is shown in Figure 14.



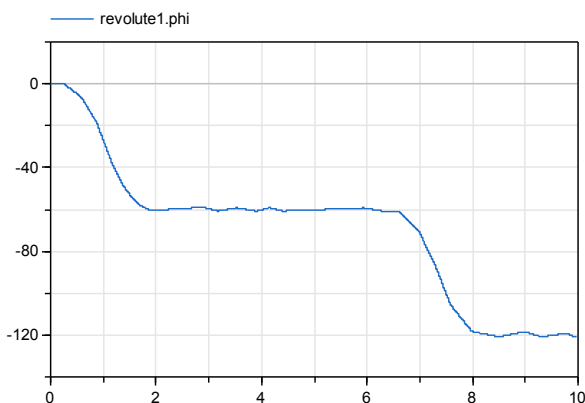
**Figure 14.** Geneva mechanism with 3 slots

The Geneva mechanism with  $nSlots=6$  is shown in Figure 15. Notice that the drive wheel is then relatively smaller.



**Figure 15.** Geneva mechanism with 6 slots

The angle of the output wheel is plotted in Figure 16 in the case of  $nSlots=6$ :



**Figure 16.** Output angle when 6 slots

It should be noted that the contact handling is very complex. In particular, the locking surface is concave. It is sliding against the convex locking ring which has smaller radius. So theoretically there would be one contact surface. However, due to the discretization of

triangulated mesh, there might be several simultaneous contact surfaces which our approach handles.

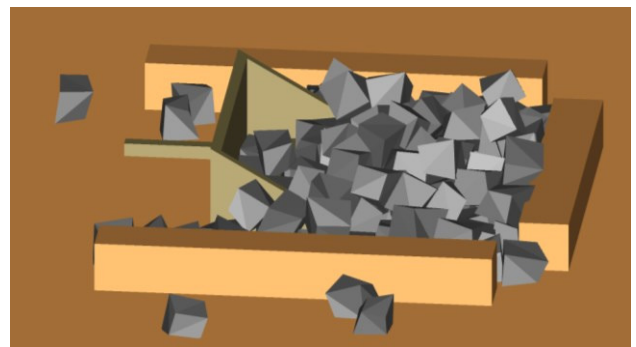
### 8.3 Bucket Digging in Pile of Belgian Blocks

Some of the roads in the old central parts of the city of Lund, Sweden are built by Belgian Block stones, see Figure 17.



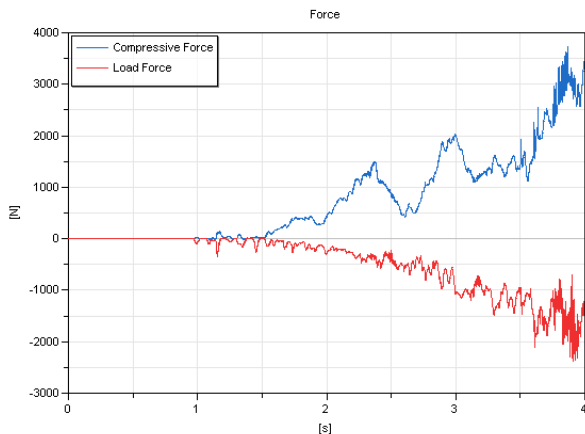
**Figure 17.** Belgian Block stones

To demonstrate that our solution can handle many objects (DEM) defined by triangular meshes, we want to calculate the force needed on the bucket to grab the stones by an excavator or loader tractor, see Figure 18. The time to simulate 4 seconds of real time took 27 minutes. During that time more than 7 million collisions took place.



**Figure 18.** Bucket digging pile of Belgian Block stones

The plot of the force is shown in Figure 19 when the bucket has constant velocity.



**Figure 19.** Plot of horizontal and vertical forces during digging

### 8.4 Tippe Top

The Tippe Top is a special toy that has fascinated a lot of physicists for a long time. It consists of a hollow sphere with a stem, as can be seen in Figure 20.



**Figure 20:** Snapshot of Tippe Top inversion, from left to right

As a result of its special geometry, its center of mass is located below the center of the sphere. When spun, the Tippe Top will invert itself, and spin on the stem, Figure 20. Since the behavior of the Tippe Top is quite complex, the fact that we can simulate it serves as a good indicator of the potential of this package.

Simulating the behavior until inverting after 3.3 seconds took 12 minutes.

## 9 Extensions to Flexible Bodies

The presented framework could naturally be extended to flexible bodies as well. This section introduces related work not yet integrated in the framework which shows the possibilities.

A prototype library has been developed in Modelica to model and simulate *planar* flexible multibody systems which also has the ability of simulating contact problems. The simulation of the flexible multibody system is based on the floating frame of reference approach and a model reducing technique, Kraig-Bampton method (Ghandriz, 2014, Shabana, 2013 and Simeon, 2013). The geometry of a body is a set of polygons defining the outer and inner boundaries of the body. A standalone code was developed for generating the finite element mesh by implementing Delaunay triangulation (Shewchuk, 2012). The reduced model is generated as Modelica code. Once a flexible or a rigid

body is generated inside Dymola it can be used together with joints, drivers, constraints, etc. to build a multibody system using the PlanarMultiBody library (Zimmer, 2012).

Having solved the equation of motion of the model, the nodal elastic deformations of the flexible components are retained from the modal coordinates which is used to calculate the time history of the planar stresses.

A few examples will be given below to show the capabilities.

### 9.1 Flexible Bodies – FEM

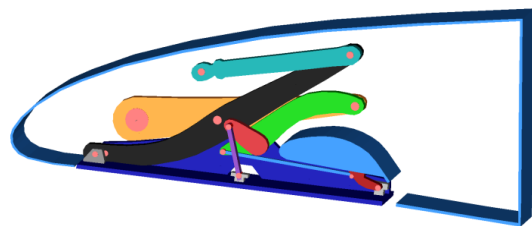
The first example is a simplified version of a mechanism so called *wing variable camber leading edge flap* used in Boeing aircraft (Cole, 1967). A similar mechanism is shown in Figure 21.



**Figure 21.** Wing variable camber leading edge flap

The behavior of the system can be analyzed using the planar flexible library. The simplified mechanism in Figure 22 consists of 13 rigid and two flexible bodies.

The bending of the flap and the resulting stress distribution at an instance of time can be seen in Figure 23.



**Figure 22.** Folded wing flap

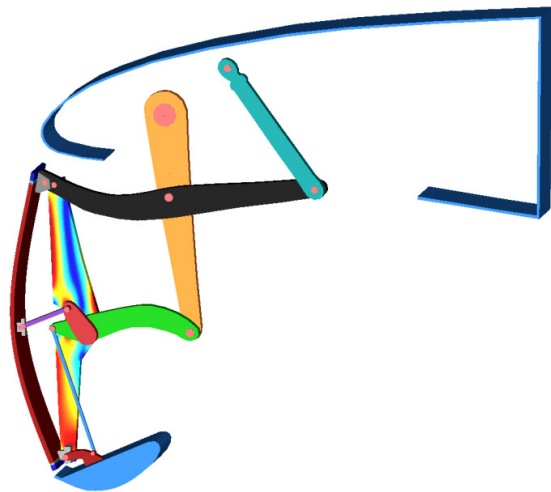


Figure 23. Unfolding wing flap

### 9.2 Flexibility and Topology Optimization

One of the excellent applications of flexible multibody dynamics can be realized as its combination with the theory of structural optimization. In particular, structural topology optimization is a part of conceptual design of a mechanical product where the material distribution, i.e. topology, of the body is iteratively updated to reach a constraint optimal state (Bendsoe, 2003). It means that, for example, with the optimal topology, the body can be stiffer or stronger but lighter. It is the purpose of many mechanical design engineers to build a mechanical part which shows the highest strength on the operation with the minimum amount of material used. In (Ghandriz, 2015) a method for applying structural topology optimization on multibody systems (TOMBS) can be found; where, large rotational and transitional motion, transient inertia and reaction forces of flexible bodies are accounted for in the optimization process.

For applying TOMBS on a flexible multibody system it is required to solve equation of motion in every optimization iteration; thus, the modal reduction and retaining all nodal elastic deformations must also be repeated in accordance with the new topology.

We apply TOMBS on one of the flexible bodies in the above example. The optimization problems are defined as to minimize the sum of the strain energy stored in the body over the operation time, while the maximum allowed volume is 60% of the initial volume shown in Figure 22.

If the thickness of the non-optimized body is changed such that it has the same weight as the optimized body, the generated stresses during operation in the body with optimal topology is smaller than the stresses of the non-optimized one. Figure 24 shows the change of the maximum stress of the two designs (optimized and non-optimized with reduced thickness) over time. The stress distribution of the non-optimized body with reduced thickness is shown in Figure 25 (upper part). The optimal result is shown in

Figure 25 (lower part). The colors are set for illustrative purpose. The highest stress is well below the yield stress. In later design stages, the high local stresses can be cured using shape optimization.

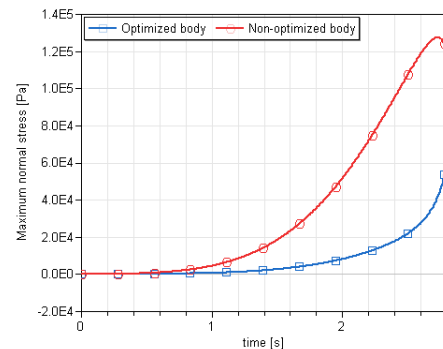


Figure 24. Maximum stresses for optimized and non-optimized body

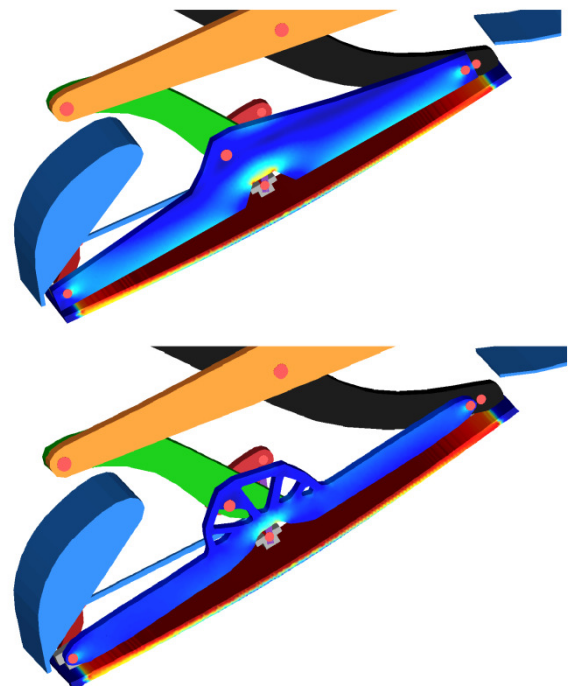


Figure 25. Stress distribution of the non-optimized body with reduced thickness (upper part); resulting shape of topology optimization (lower figure)

### 9.3 Flexibility and Contact

The last example is the lever escapement where both bodies are flexible. In flexible bodies, the forces generated due to the contacts must be distributed to the nodes of the finite elements involved in contact. The portion of the total contact force which each node receives is proportional to the node's penetration depth and its distance from the center of the contact region.

To obtain a more exact local deformation of the finite elements involved in contact, corresponding



nodes must be excluded from the modal reduction, i.e. they should be in the set of Master nodes in Kriag-Bampton method. Figure 26 shows the finite element mesh. In Figure 27, three snapshots of the model and the resulting stresses at the moment of contact are illustrated.

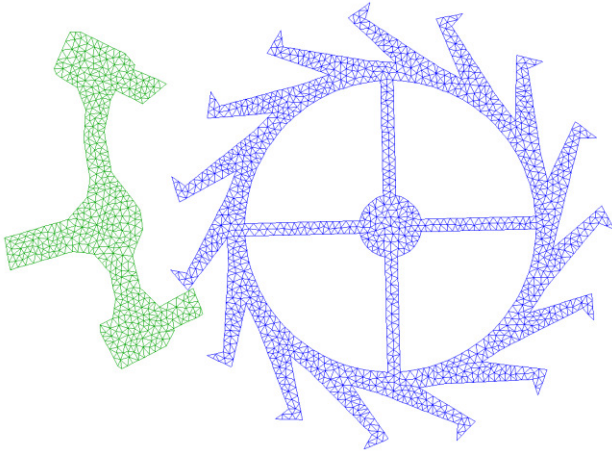


Figure 26. FEM mesh

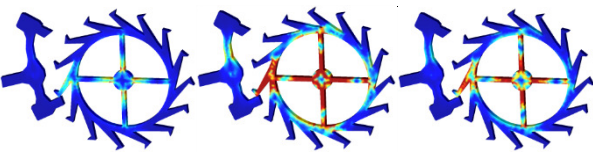


Figure 27. Stresses during the contact

It is interesting to see that the highest stresses in the Pallet body, i.e. the body on the left, are caused by the inertia forces. These stresses cannot be captured if the problem is analyzed statically.

## 10 Conclusions

A new Modelica framework for collision handling and DEM has been presented. It allows construction of 3D parts by use of Modelica functions for CSG in the early concept phase. CSG is also used to find contact region. Special considerations are taken in order to be able to handle DEM.

A discussion about the possibility to extend this framework to FEM with contact and for topology optimization is given.

We propose that this framework serves as a starting point for a working group within Modelica Association to define a standard Modelica library for collision, contact and DEM.

## Acknowledgements

This work has partly been performed as a master thesis project at Lund Institute of Technology. The first author served as an industrial advisor and Michael Doggett as the formal supervisor.

The authors want to thank Hans Olsson for extending Dymola with the capability to animate triangular meshes.

## References

- Bendsoe M. P., Sigmund O. (2003): Topology optimization: theory, methods and applications. Springer Science & Business Media
- Bickford J.H. (1972). Geneva Mechanisms. Mechanisms for intermittent motion. New York: Industrial Press inc. 128. ISBN 0-8311-1091-0, [http://ebooks.library.cornell.edu/k/kmoddl/pdf/002\\_010.pdf](http://ebooks.library.cornell.edu/k/kmoddl/pdf/002_010.pdf)
- British Horological Institute (2011): Drawing Clock and Watch Escapements - Distance Learning Course. pp 17-30, <http://www.bhi.co.uk/sites/default/files/Drawing%20Escapements%20Version%201.4%20DS.pdf>
- Chen J. (2012): Discrete Element Method for 3D Simulations of Mechanical Systems of Non-Spherical Granular Materials. The University of Electro-Communications, Japan, [http://ir.lib.uec.ac.jp/infolib/user\\_contents/9000000625/900000625.pdf](http://ir.lib.uec.ac.jp/infolib/user_contents/9000000625/900000625.pdf)
- Cole J. B., Island M., Weiland R. H. (1967): AIRCRAFT WING VARIABLE CAMBER LEADING EDGE FLAP. United States Patent Office, <https://docs.google.com/viewer?url=patentimages.storage.googleapis.com/pdfs/US3504870.pdf>
- Elmqvist H., Baldwin A.D., Dahlberg S. (2015): 3D Schematics of Modelica Models and Gamification. Proceedings 11th International Modelica Conference, Versailles, September 21-23, 2015.
- Elmqvist H., Olsson H., Goteman A., Roxling V., Zimmer D., Pollok A. (2015) Automatic GPU Code Generation of Modelica Functions. Proceedings 11th International Modelica Conference, Versailles, September 21-23, 2015.
- Ghandriz T. (2014): An algorithm for structural topology optimization of multibody systems, Master's thesis, Lund University. <https://sharepoint.srv.lu.se/sites/mimer/kursplanering/gu/S/PBstipendium/Nomineringar%202015/MATEMATIK%20Tohee%20Ghandriz.pdf>
- Ghandriz T., Führer C., Elmqvist H. (2015): Structural Topology Optimization of Multibody Systems. ECCOMAS Thematic Conference on Multibody Dynamics, Barcelona, Catalonia, Spain.
- Goteman A., Roxling V. (2015): GPU Usage for Parallel Functions and Contacts in Modelica, Master's thesis, Lund Institute of Technology, Lund, Sweden. (To be published)
- Gottland N. (2012): Make Geneva wheels of any size, <http://newgottland.com/2012/01/08/make-geneva-wheels-of-any-size/>
- Hippman G. (2003): An Algorithm for Compliant Contact Between Complexly Shaped Surfaces in MultiBody Dynamics. MultiBody Dynamics 2003, Lisbon, Portugal, [http://www.pcm.hippmann.org/doc/eccomas03\\_hippmann.pdf](http://www.pcm.hippmann.org/doc/eccomas03_hippmann.pdf)
- Hippman G. (2013): Polygonal Contact Model. <http://www.pcm.hippmann.org/>

- Hofmann A., Mikelsons L., Gubsch I., Schubert C. (2014): Simulating Collisions within the Modelica MultiBody Library. Proceedings 10th International Modelica Conference, Lund, March 10-12, 2014, <http://www.ep.liu.se/ecp/096/099/ecp14096099.pdf>
- Karras T. (2012): Thinking Parallel, Part III: Tree Construction on the GPU. <http://devblogs.nvidia.com/parallelforall/thinking-parallel-part-iii-tree-construction-gpu/>
- Lavrov D. (2014), Collision Detection Using Z Order Curve Aka Morton Order. [http://dmytry.com/texts/collision\\_detection\\_using\\_z\\_order\\_curve\\_aka\\_Morton\\_order.html](http://dmytry.com/texts/collision_detection_using_z_order_curve_aka_Morton_order.html)
- Mueller R.K. (2015): OpenJSCAD.org User & Programming Guide. <http://openjscad.org/>
- Nassauer B., Kuna M. (2013): Contact forces of polyhedral particles in discrete element method. DOI 10.1007/s10035-013-0417-9, Springer Verlag.
- Nürnberg R. (2013): Calculating the volume and centroid of a polyhedron in 3d. <http://wwwf.imperial.ac.uk/~rn/centroid.pdf>
- Oestersötebier F., Wang P., Trächtler A. (2014): A Modelica Contact Library for Idealized Simulation of Independently Defined Contact Surfaces. Proceedings 10th International Modelica Conference, Lund, March 10-12, 2014, [http://www.ep.liu.se/ecp\\_article/index.en.aspx?issue=96;article=97](http://www.ep.liu.se/ecp_article/index.en.aspx?issue=96;article=97)
- Otter M., Elmqvist H., Diaz Lopez J. (2005): Collision Handling for the Modelica MultiBody Library. Proceedings 4th International Modelica Conference, Hamburg, March 7-8, 2005, pp. 45-53, <http://elib.dlr.de/12299/1/otter2005-modelica-collision.pdf>
- Segura C., Stine T., Yang J. (2013): Constructive Solid Geometry Using BSP Tree. [https://www.andrew.cmu.edu/user/jackiey/resources/CSG/CSG\\_report.pdf](https://www.andrew.cmu.edu/user/jackiey/resources/CSG/CSG_report.pdf)
- Shabana. A. A. (2013): Dynamics of multibody systems. Cambridge university press.
- Shewchuk J. R. (2012): Lecture Notes on Delaunay Mesh Generation, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley.
- Simeon B. (2013): Computational flexible multibody dynamics: a differential-algebraic approach. Springer Science & Business Media.
- Tonon F. (2014): Explicit Exact Formulas for the 3-D Tetrahedron Inertia Tensor in Terms of its Vertex Coordinates. <http://docsdrive.com/pdfs/sciencepublications/jmssp/2005/8-11.pdf>
- Wallace E. (2012) csg.js. <http://evanw.github.io/csg.js/>
- Zimmer D. (2012): A Planar Mechanical Library for Teaching Modelica. Proceedings of the 9th International Modelica Conference, September 3-5, 2012, Munich, Germany