BrickFEM: An Automated Finite Element Model for Static and Dynamic Simulations of Simple Lego® Sets

Martin Pletz and Matthias Drvoderic

Chair of Designing Plastics and Composite Materials, Montanuniversitaet Leoben, Otto Gloeckl Strasse 2, Leoben, Austria

Abstract

BrickFEM is a tool that can automatically generate, mesh, run and evaluate Finite Element Method (FEM) models of simple sets of bricks in the commercial FE software Abaqus. It uses Python scripts to generate Lego® sets, which can contain regular bricks, plates, tiles and base plates of any size. The model calculates the full stress and strain fields of the bricks and can show how Lego bricks disassemble under applied loads. The clamping connection between the Lego bricks is realized by widening the lower cavities of the bricks before defining the contact between them. Loads can then be applied in a static or dynamic step on regions of the bricks or using rigid bodies that contact the Lego set.

Finite element modeling; toy bricks; clamping connection; Simulia Abaqus

Program title	BrickFEM
Developer's repository link	`https://github.com/mpletz/BrickFEM`
Code license	MIT
Programming languages and tools	Simulia Abaqus, Python, imagemagick
Support email address	martin.pletz@unileoben.ac.at

1 Introduction

The key feature of Lego bricks is that they can be clamped together. The studs of one brick can be stuck into the bottom cavity of another brick, see Figure 1. The friction between the studs and the cavities, combined with the normal force, results in a clamping force. The design of the lower cavity is thus a key feature for the clamping connection and was the main development in the original Lego brick patent [1].

Building Lego sets with a desired shape, color, and stability is a classical mathematical optimization problem [2]. Some of these works consider the stability of the Lego sculpture [3, 4, 5, 6] using simplified systems of forces with contacting bricks and applying the dead load and external forces on the Lego sets. In this way, large Lego sets can be analyzed

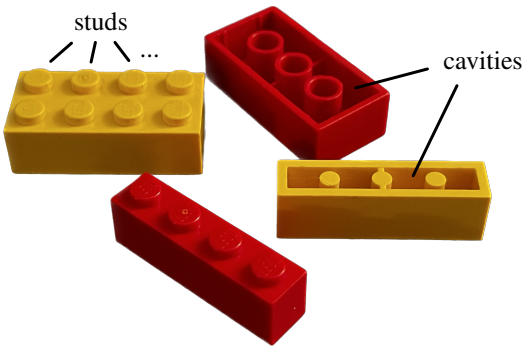


Figure 1: Some 1×4 and 2×4 Lego bricks with their studs and cavities.

for static loads. However, the clamping between bricks is simplified to a great extent. For example, Luo [3] uses a critical frictional load between two assembled bricks.

A dynamic Lego model with a massive number of bricks modeled the crash of two Lego cars in an explicit finite element model [7, 8]. The regular bricks in the model of Gerlinger et al. [8] are connected by tied contact with parameters calibrated by experiments. Such simplifications are necessary for models that contain thousands of bricks and therefore have a rather coarse mesh.

This work presents a package for the automated generation of Lego set FEM models to study this clamping connection in more detail. The full FEM model can compute realistic stress- and strain fields in the model in a static or dynamic analysis using Lego bricks, plates, tiles, and base-plates of any size. It is therefore able to accurately predict the forces required for disassembly depending on the brick geometry, the material behavior of the bricks, and the friction coefficient between the bricks.

2 Software description

The main challenge in modeling a Lego set is to establish the clamping of the bricks in an accurate and efficient way. In a finite element model, the contact faces cannot initially overlap. Therefore, the model cannot start with the bricks in their positions in the set but must stick them together in some way.

This could be done by simulating how all the bricks are stuck together, one brick at a time. Since the time complexity for adding bricks is exponential, this becomes infeasible even for small models. Therefore, we developed a method for BrickFEM that uses only three static steps (the *clamping steps*) to realize the clamping of all bricks at the same time, see Figure 2:

- *widen*: The initial overlap of surfaces is resolved by displacements of the lower cavities. This means that nodes on the face that would lie inside the stud of another brick are moved so that there is no overlap. The top faces of the studs are fixed in this step to prevent relative movement between the bricks,

- *contact*: In the second step, the contact between the studs and the lower cavities of the bricks starts and the applied displacements of the *widen* step are discontinued. This results in a first clamping between the bricks. Since the top faces of the studs are still fixed, this clamping is unrealistic.

- *free*: The *free* step frees the top faces of the studs, resulting in a realistic clamping connection between the bricks is obtained. This is the starting point for applying the loads in another step.

BrickFEM automatically generates these clamping steps.

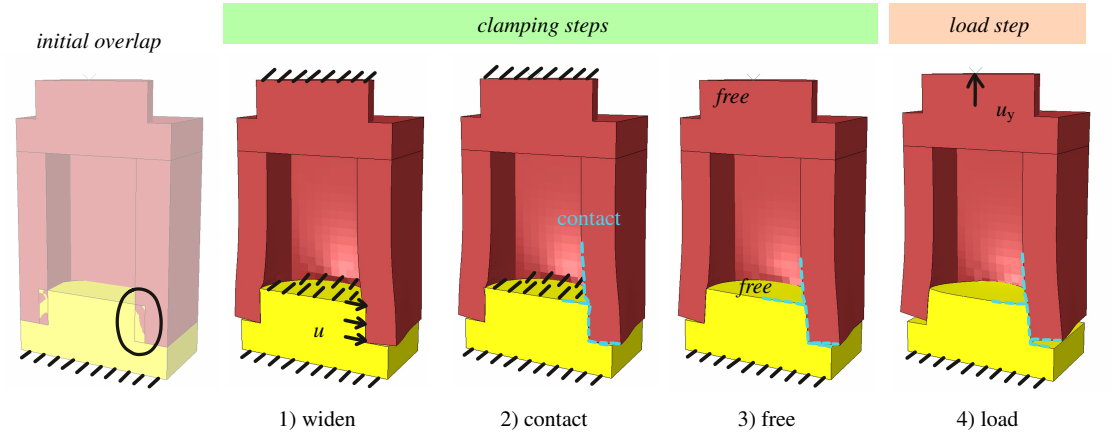


Figure 2: Steps in the Lego model: The initial penetration of the contact surfaces is resolved in the *widen* step. Then, the contact is defined. In the next step, the initially fixed faces are released. Then, the clamped Lego bricks can be loaded in the *load* step, which can be either static (implicit) or dynamic (explicit).

2.1 Software architecture

The function *make_model* generates a Lego model based on the input dictionaries that define the brick arrangement (*assembly*), the loads of a dynamic, explicit step (*explicit_par*), and the basic Lego dimensions (*lego_geom*), see Figure 3. *make_model* is the main function of BrickFEM that calls the internal functions that create the geometry, mesh the geometry, apply loads, set boundary conditions, run the model execution, and evaluate the results. If no *explicit_par* is passed to the function, the Lego set is loaded in a static, implicit step. Note that the static analysis will terminate as soon as the bricks disassemble because the FEM does not converge. Therefore, implicit loading is only relevant for Lego sets that deform slightly and do not disassemble.

BrickFEM uses the geometry of the 1961 Lego patent [1] including ribs between the inner tubes at every second inner tube or cylinder, see Figure 1. The dictionary *lego_geom*, which is an optional input for the *make_model* function, contains all the general geometry parameters as well as the elastic properties of the brick material and the friction coefficient

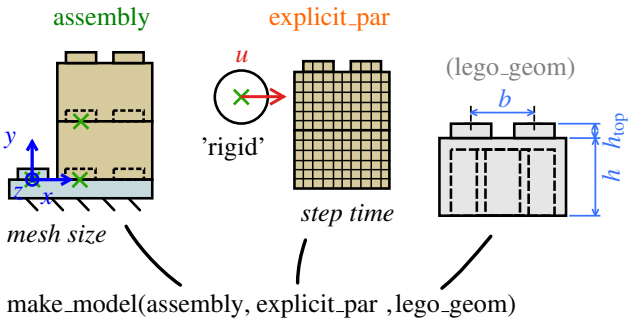


Figure 3: Structure of the model function *make_model* that creates, runs, and evaluates the Lego model from the input dictionaries *assembly*, *explicit_par*, and *lego_geom*. Note that both *explicit_par* and *lego_geom* are optional. By default, *lego_geom* contains basic Lego dimensions and Lego material properties.

between the bricks. For details on the *lego_geom* dictionary, see the BrickFEM documentation. If not passed to the *make_model* function, BrickFEM uses the geometry of the Lego patent [1] with typical dimensions.

2.2 Software features

In the following, the possible brick types, types of boundary conditions, and types of loads on the Lego sets are defined and the necessary form of the input dictionaries *assembly* and *explicit_par* is described.

2.2.1 Definition of brick assembly and loads

The assembly of a Lego set in BrickFEM is defined in the dictionary *assembly*. The subdictionary *bricks* lists the bricks used in the model. The subdictionary *parts* places these bricks, referenced by their *brick_id*, at the locations *loc*, see Figure 4a. Note that each of the bricks defined in the *bricks* subdictionary can be used multiple times in the model by referring to it multiple times in the *parts* subdictionary. Optionally, the color of each part for the output video can be defined as *c*, specifying either one of the solid Lego brick colors from [9] or a 6-digit hex code as a string.

The assembly dictionary further defines the boundary conditions *bc*, the loads on the brick sets *loads_rp*, the mesh size *mesh_size*, and the friction coefficient *mu*:

```
assembly = {
 'name':'case1-pull-1x1',
 'bricks':{
   1:{'type':'base-plate','nx':4,'nz':2},
   2:{'type':'regular','nx':2,'nz':2}},
 'parts':{
   1:{'brick_id':1,'loc':(0,0,0),'c':'Yellow'},
   2:{'brick_id':2,'loc':(8,0,0)},'c':'Red'},
   3:{'brick_id':2,'loc':(8,0,0)},'c':'Yellow'},
   4:{'brick_id':2,'loc':(8,0,0)},'c':'Red'},
```

```
    5:{'brick_id':2,'loc':(8,0,0)},'c':'Yellow'}},
'bc':{1:{'part_id':1,'set_name':'BOTTOM'}},
'loads_rp':{},
'mesh_size':0.75,'mu':0.2
 }
```

The assembly dictionary is sufficient to define the Lego set, including all loads for the static implicit analysis. For a dynamic, explicit analysis, additional parameters must be defined and loads by rigid bodies can be applied, see the next section.

The bricks in the *assembly* are defined by their brick type *type* which can be *regular*, *plate*, *tile*, or *base-plate*. The size of the brick is defined in terms of the number of studs in the x-direction and in the z-direction by the parameters *nx* and *nz*, respectively. So a 2×4 brick with the longer axis in x-direction has $n_x = 4$ and $n_z = 2$. Note that there is no brick rotation implemented in the model; so a 2×4 brick must be defined twice if its longer axis points in the x-direction for some parts and in the z-direction for others.

Figure 4b shows the shape and coordinate system origin (green x) of the four possible brick types *regular*, *plate*, *tile*, and *base-plate*. The height without studs h for the plate and tile is only one third of a *regular* brick. The y-origin of the bricks lies in the bottom plane for all brick types except for the *base-plate*, where it lies at the bottom of the stud. The x- and z-coordinates of the origin of the brick lie in the center of the upper left stud. When referring to the stud of a brick, this is done in terms of their indices i_x,i_z, with i_x and i_z starting at 1.

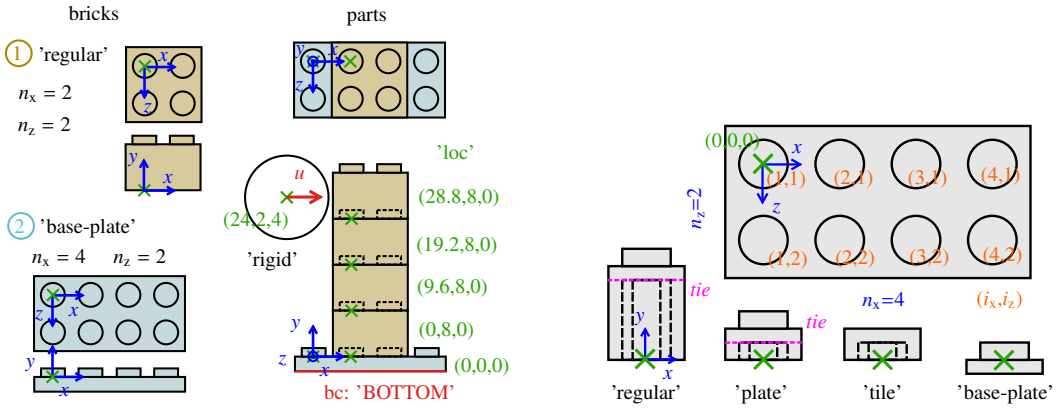


Figure 4: a) Definition of the bricks used in the model and their positions, as defined in the *assembly* dictionary. The load in this model is applied by a rigid cylinder with a displacement u in the x-direction, as defined in the *explicit_par* dictionary. The green coordinates indicate the locations of the instances *loc* of the bricks and b) defines the origins of the bricks for all brick types and the numbering of the studs for applying loads.

Boundary conditions are defined in the *bc* subdictionary of *assembly*. It needs the part id and the name of the set to fix. Possible sets are *'BOTTOM'*, *'TOP-FACES'*, and *'STUD-ij'* for the bottom face, all top faces, or the top face of the stud with index i=i_x and j=i_z, respectively.

Loads applied to sets via reference points work like boundary conditions: The part id and the *set_name* must be specified. Then a reference point is automatically created in the center of the set and all nodes of the set are rigidly coupled to this reference point. The displacements (in mm) and rotation angles (in radians) of the RP can be specified with the parameters *ux*, *uy*, *uz*, *rotx*, *roty*, or *rotz*. If one or more of those displacements is not specified, that displacement is free in the model.

The *mesh_size* defines the global mesh size in all Lego bricks. Linear hexahedral elements with reduced integration are used for implicit load steps and linear tetrahedral elements for explicit load steps. For hexahedral elements, the *regular* bricks and *plates* are split into two parts at $y = h - h_{\text{top}}$ and then connected using surface-based tie constraints of Abaqus, see Figure 4b.

BrickFEM models the contact using the penalty contact algorithm of Abaqus with the friction coefficient *mu*. For the implicit steps (including the clamping steps), the Surface to Surface contact option of Abaqus is used. For the explicit load step, BrickFEM uses General Contact.

2.2.2 Definition of explicit load parameters

Parameters of the explicit load step and loads by rigid bodies are defined in the *explicit_par* dictionary. If *make_model* is parsed an empty dictionary for *explicit_par*, the model is run implicitly using only the parameters from the *assembly* dictionary. By default, *explicit_par* is an empty dictionary.

For the Lego set defined in Figure 4a, the *explicit_par* dictionary looks like this:

```
explicit_par = {
 't_step':0.001,'is_acc':0,'mass_scale_t':0,
 'load_str':'',
 'loads_rigid':{
     1:{'shape':'sphere','u':(20,0,0),
        'radius':4.,'loc':(-8.1,9.6*4.5,4)}}
}
```

The following parameters must be defined in *explicit_par* to run an explicit load step in BrickFEM:

- *t_step*: The time of the explicit step (float) or the times of the explicit steps (list) that are independently computed.
- *is_acc*: Whether the load should be applied with constant acceleration (1) or constant velocity (0) to reach the total displacement at the end of the step.
- *mass_scale_t*: If 0, no mass scaling is used. If unequal 0, this is the target time step used for mass scaling.
- *load_str*: String to add to the model name. If the same Lego set is loaded in different ways defined in separate *explicit_par* dictionaries, it may be convenient to identify these load cases in the model name.

The subdictionary *loads_rigid* defines rigid parts for loading, which can be either spheres or cylinders. Both need a location of their center and a radius, which are specified in the dictionary as *loc* and *radius*. For the cylinder, the user also needs to specify the *direction* of the cylinder axis and the cylinder length *len*. Note that the reference point of the cylinder is located at half of its length for the cylinder and at the center of the sphere.

The displacement of the rigid part is given in u as a list (u_x, u_y, u_z). The rotations of the rigid part are always fixed. As an alternative to applying a displacement u, the rigid part can have an initial velocity and then move freely in the load step. This can be done by specifying m and *v0* in *loads_rigid* to specify the mass and the initial velocity of the rigid part, respectively. The moments of inertia I_{jj} are calculated from the mass m and the radius r as $I_{xx} = I_{yy} = I_{zz} = 2/5\ m\ r^2$ for the sphere. For the cylinder with a length l, the moments of inertia are set to $I_{xx} = 1/2\ m\ r^2$ and $I_{yy} = I_{zz} = 1/12\ l\ m^2$ with the x-axis as the cylinder axis.

3 Example problem

To illustrate the use of BrickFEM, the example of a Lego tower made of six 2×2 regular bricks on a 2×2 baseplate (which is fixed at the bottom) is used. The tower is loaded by a rigid sphere hitting it from the left. This is the *assembly* dictionary for the tower example:

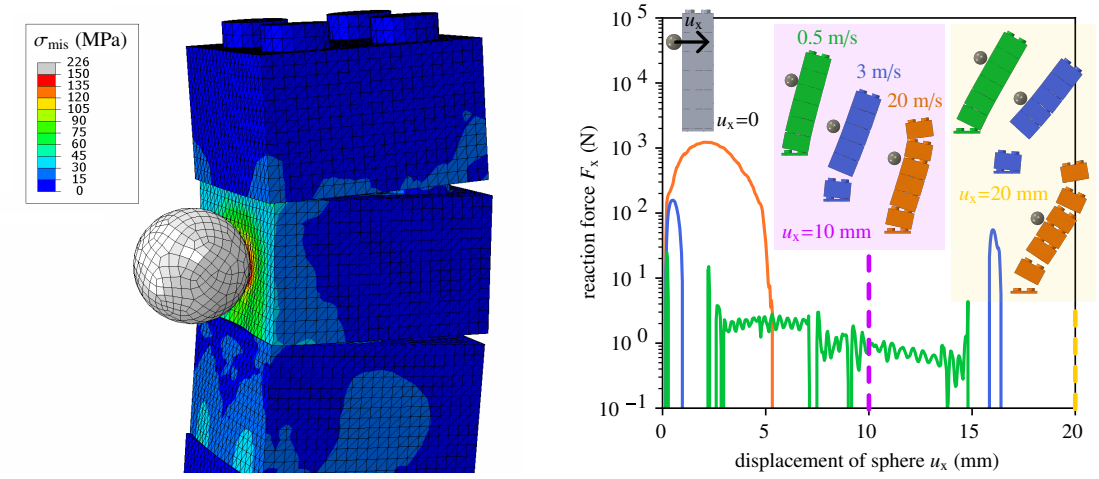
```
assembly_tower = {
  'name':'tower6_2x2',
  'bricks':{
     1:{'type':'base-plate','nx':2,'nz':2},
     2:{'type':'regular','nx':2,'nz':2}},
  'parts':{
     1:{'brick_id':1, 'loc':(0,0,0)},
     2:{'brick_id':2, 'loc':(0,0,0)},
     3:{'brick_id':2, 'loc':(0,9.6,0)},
     4:{'brick_id':2, 'loc':(0,2*9.6,0)},
     5:{'brick_id':2, 'loc':(0,3*9.6,0)},
     6:{'brick_id':2, 'loc':(0,4*9.6,0)},
     7:{'brick_id':2, 'loc':(0,5*9.6,0)}},
  'bc':{1:{'part_id':1,'set_name':'BOTTOM'}},
  'loads_rp':{},
  'mesh_size':0.75,'mu':0.2}
```

The sphere is moved by 20 mm at a speed of 20 m/s, 3 m/s, or 0.5 m/s. The velocities can be set by adjusting the time of the step *t_step* to apply the displacement. For example, the step time can be 1 ms, so that the speed is 20 mm/1 ms = 20 m/s:

```
explicit_par_tower = {
 't_step':0.001,'if_acc':0,'mass_scale_t':0,
 'load_str':'',
 'loads_rigid':{
     1:{'shape':'sphere','u':(20,0,0),
```

```
6           'radius':4.,'loc':(-8.1,9.6*4.5,4)}}
7 }
```

Figure 5 shows the resulting von Mises stress field for a sphere velocity of 20 m/s and the force-displacement curves and deformation of the Lego bricks for all three velocities. Depending on the velocity of the sphere, the tower tilts or dynamically disassembles. The highest reaction forces in the sphere occur at the highest velocity.



(a) (b)

Figure 5: Results of the Lego tower example in terms of a) the von Mises stress in the bricks when the sphere hits the tower (with the linear-elastic material model used, local stresses reach about 230 MPa) and b) the force-displacement curves of the sphere for three sphere velocities.

4 Conclusions

Creating FEM models of Lego sets is not straightforward because the initial clamping connection needs to be established first. Contact surfaces in a FEM model must not penetrate each other at the beginning of an analysis. There is no simple way of establishing this clamping connection in the beginning. This work uses initial displacements to widen bottom cavities of the bricks and then defines the contact to clamp the bricks. The automated model can then be statically or dynamically loaded. Due to the full mesh of all bricks, the number of bricks in the models is limited to a few. For these, however, a detailed analysis of the clamping with the full stress- and strain fields can be studied.

By providing a versatile tool to compute the detailed clamping behavior of Lego bricks in simple setups, BrickFEM can help to better understand the connection of Lego bricks better and improve the analytical models for the stability of Lego designs. In addition, it can predict dynamic effects that are not captured in existing analytical models or considerably simplified in existing dynamic models [8]. It can also be adapted to help understand and optimize interlocking mechanisms of materials with high toughness [10, 11]. Furthermore, it can show how mechanical simulation and the FEM method can be applied to

everyday objects and that FEM and engineering mechanics does not need to be boring.

References

[1] G. K. Christiansen, Toy Building Brick (1961).

[2] B. Stephenson, A multi-phase search approach to the LEGO construction problem 7 (1) 89–97. doi:10.1609/socs.v7i1.18385.

[3] S.-J. Luo, Y. Yue, C.-K. Huang, Y.-H. Chung, S. Imai, T. Nishita, B.-Y. Chen, Legolization: optimizing LEGO designs. 34 (6) 222:1–222:12. doi:10.1145/2816795.2818091.

[4] R. Testuz, Y. Schwartzburg, M. Pauly, Automatic Generation of Constructable Brick Sculptures, in: M.-A. Otaduy, O. Sorkine (Eds.), Eurographics 2013 - Short Papers, The Eurographics Association, 2013. doi:10.2312/conf/EG2013/short/081-084.

[5] T. Kollsker, Mathematical Models and Algorithms for Optimisation of the LEGO Construction Problem, Ph.D. thesis, Technical University of Denmark, Lyngby (2020).

[6] T. Kollsker, E. Malaguti, Models and algorithms for optimising two-dimensional LEGO constructions, European Journal of Operational Research 289 (1) (2021) 270–284. doi:10.1016/j.ejor.2020.07.004.

[7] T. Gerlinger, D. Koch, A. Haufe, N. Karajan, T. Weckesser, P. Glay, A. Saharnean, M. Thiele, On the Setup and Simulation of Large Scale LEGO® Models built with LS-DYNA® and LoCo, Koblenz, Germany, 2019, p. 15.

[8] T. Gerlinger, D. Koch, A. Haufe, N. Karajan, T. Weckesser, P. Glay, A. Saharnean, M. Thiele, Simulation Data Management from CAD to Results with LoCo and CAViT for Large Scale LS-DYNA® LEGO® Crash Models, 2020.

[9] Bricklink, Bricklink color guide (Dec. 2022).
URL https://www.bricklink.com/catalogColors.asp

[10] P. Fratzl, O. Kolednik, F. D. Fischer, M. N. Dean, The mechanics of tessellations – bioinspired strategies for fracture resistance, Chemical Society Reviews 45 (2) (2016) 252–267. doi:10.1039/C5CS00598A.

[11] L. Djumas, G. P. Simon, Y. Estrin, A. Molotnikov, Deformation mechanics of non-planar topologically interlocked assemblies with structural hierarchy and varying geometry 7 (1) 11844. doi:10.1038/s41598-017-12147-3.