 CMES, vol.118, no.1, pp.111-137, 2019

LiToTac: An Interactive-Interface Software for Finite Element Analysis of Multiple Contact Dynamics

Lei Peng[1,2], Zhiqiang Feng[1,2,*], Pierre Joli[2] and Christine Renaud[2]

Abstract: In order to investigate the mechanical behavior of systems with complex architecture and a large number of contacting bodies, a finite element software, named LiToTac, has been developed by using the object-oriented programming technique. This software, with an interactive graphical user interface, is able to handle highly non-linear problems including multiple contacts and large deformation. More importantly, the contact detection based on a hybrid three-stages methodology can be performed automatically, which is more efficient than the common strategies of pre-defining contact zones in commercial FEM software like ANSYS, ABAQUS, etc. In addition, the contact solver in LiToTac is portable between dynamic and quasi-static codes and can accurately solve contact coupled with friction in a reduced system. Several numerical examples are carried out to illustrate the functionality and capacity of the software package.

Keywords: Finite element software, object-oriented programming, automatic contact detection, multiple contact dynamics.

1 Introduction

Contact/impact problem is a very difficult issue in computational mechanics and many engineering fields. The main difficulty comes from the high nonlinearities on the contact surface and the discontinuous nature of motion. Over the years, numerical solutions of such problems accounting for large deformation effects have been intensively studied [Mahmoud, El-Shafei, Abdelrahman et al. (2013); Mlika, Renard and Chouly (2017); Hartmann, Oliver, Weyler et al. (2009); Chen, Joli and Feng (2015)]. Consequently, systems composed of only a few interacting bodies may require significant computational efforts. More complex scenarios such as soft textile composite reinforcements [Misra, Dixit and Mali (2014)] and biomechanical systems [Bei and Fregly (2004); Lin, Walter, Banks et al. (2010)] are particularly challenging and prone to very long simulating time.

[1] School of Mechanics and Engineering, Southwest Jiaotong University, Chengdu, 610031, China.
[2] Laboratory of Mechanics and Energetics, Université Paris-Saclay, Evry, 91020, France.
[*] Corresponding Author: Zhi-Qiang Feng. Email: zhiqiang.feng@univ-evry.fr.

The finite element method has become one of the most powerful tools for solving complex contact problems during the past several decades. Integration of contact dynamics with the finite element method has been realized in different commercial CAE software packages like ANSYS, ABAQUS, NASTRAN, COMSOL, etc. These commercial software contain complete material libraries, computing solvers, advanced functionalities of pre- and post-processors, which enable them to solve different kinds of engineering or scientific problems. But to the viewpoints of the authors, these software have several drawbacks. Firstly, too many parameters are required to be set and a lot of complicated operations on the interface need to be performed during the modeling procedures. Whereas, for average users, they may have backgrounds or experiences only in one discipline. It is difficult for them to define appropriate parameters so as to obtain good results. Secondly, with regard to some specific problems, for example the layered composites [Liu, Cheng, Zeng et al. (2015)] which considers Monte Carlo simulation, incorporating new algorithms and models are not always possible since most commercial software packages are closed-source. This necessitates the development of special software for local problems.

Object-oriented programming (OOP) is currently a well-suited approach for designing new FEM applications, because the object-oriented concepts (abstraction, encapsulation, inheritance, polymorphism) present much similarities with the structure of FEM system. The entities such as nodes, elements, displacements, constraints can be viewed as objects with a hierarchical structure, which allows them to be accessed at various levels of abstraction. Besides, OOP also improves the code with reusability, modificability, and maintainability. Recently, the object-oriented philosophy has been applied by researchers to develop finite element programs in various domains of interest, such as nonlinear multi-physics [Yuan and Fish (2015)], pavement analysis [Fang, Hand, Haddock et al. (2007)], multi-body system [Feng, Joli and Seguy (2004)], etc.

In this work, our objective is to describe the development of an efficient object-oriented FEM software for solving a contact system with complex architecture and a large number of deformable bodies. We remark that the commercial software can also address this issue. However, pre-defining the slave-master contact sets in these software is particularly tedious. All the possibilities of contact pairs should be considered, nevertheless, the contact areas inside the scenario are difficult to be selected by simply picking operations on the interface. In addition, as the number of bodies becomes very large, complexity will increase quadratically, which makes predefining contact zones generally impossible. In LiToTac, we propose to overcome these shortcomings through the new function of automatic contact detection. This function offers improvements in the efficiency of contact inspections and simplicity of interface operations. The improvements are the result of a hybrid boxes-based identification strategy that uses a global Octree search and an optimal bounding volume hierarchy query. This algorithm also permits to directly get vertex-elementary contacting pairs, and can handle interactions between multiple solids, beams, and shells.

In general, the nonlinear contact system of equations in commercial software can either be solved by the penalty methods or the multiplier Lagrangian methods. As is known, the penalty methods are easy to be implemented, but the contact boundary conditions and friction laws can not be accurately satisfied. Besides, the value of penalty factor is problem-dependent, and depends particularly strongly on the stiffness ratio between the contact objects, which may result in spurious oscillations in contact forces. The multiplier Lagrangian methods are able to enforce the zero-penetration condition exactly by introducing a set of multipliers representing contact forces. Nevertheless, both the generalized coordinates and multipliers are unknown values, thus leading to an increase in the size of the equation systems. In LiToTac, the techniques of bi-potential [De Saxcé and Feng (1998)] are adopted, which shows several favorable properties as follows: (1) The reaction force appearing as the projection of a linear combination of the reaction with the relative velocity onto Coulomb's cones, can lead to a unique variational inequality rather than two minimum principles, as compared to classic augmented Lagrangian methods [Jean and Touzot (1988)]. (2) The unique mathematical operator of projection can can make a certain mathematic reduction, because it does not consider the Lagrangian multipliers as additional unknowns. (3) The nonlinearities of equations are separated to be solved, which can overcome the computational complexity resulting from the non-differentiable contact bipotential and the inequalities of frictional contact rules. (4) The whole solution process requires only one user-defined parameter: the friction coefficient. This enables users to obtain very good results even though they are not familiar with the contact mechanics theory.

The article is organized as follows: in Section 2, the object-oriented approach to design the software is presented. The structure of the developed software and the simulation methods are given in Section 3. Section 4 describes the main functionalities of the software. Section 5 presents several examples to highlight the graphical representation of the numerical solutions. In Section 6, a few concluding remarks are drawn.

2 Object-oriented programming (OOP)

LiToTac, as a prototype finite element software of the CAE family, employs the OOP approach for the designing process. Several crucial components have been incorporated into the software using an OOP architecture, these include an interactive graphical user interface, standard mechanical elements and material libraries to assist in the modeling, postprocessing for the manipulation and display of results. The code is implemented with careful software-engineering procedures, management of data, re-use of the programming strategy.

The basic features of object-oriented programming include data abstraction, encapsulation, data-hiding, modularity, hierarchy, inheritance, and polymorphism, etc, allowing the finite element code to be easily extended for implementing new ideas and new algorithms. The detailed explanation of these key concepts can be referred to the articles [Fenves (1990); Dubois-Pèlerin and Pegon (1998)]. In this section, we will only briefly present them as

they relate to our own program.

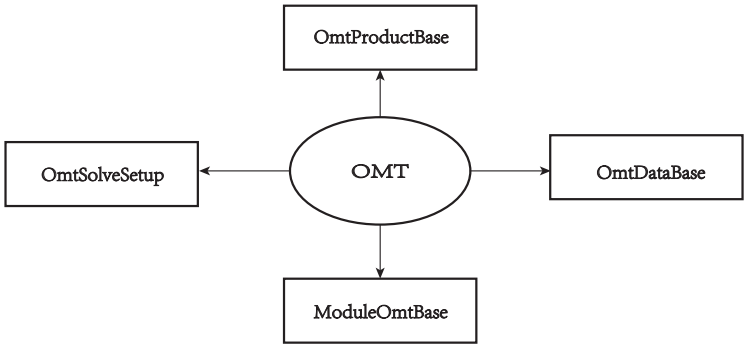


Figure 1: Base abstraction classes of OMT platform

One main characteristic of OOP is to define various classes. A class incorporates the set of objects that shares the same structure and behavior. An object is called an instance of the class, which can be treated as the realization of the data type defined by the class. In our code, the object-oriented architecture is organized around several important base classes of OMT (Objected and Methodological Technology), as illustrated in Fig. 1. The main functions of these classes are summarized as follows:

- **OmtProductBase:** this base class is in charge of managing the version, name, registered modules of a software developed on OMT platform.
- **ModuleOmtBase:** the user project is organized by splitting the problem into different modules, while each module derives from the base abstract class ModuleOmtBase.
- **OmtDataDef:** this base class is used to define specific data class in the user modules, such as the boundary condition, velocity, load, etc.
- **OmtSolveSetup:** this base class is generated to create different setups for different solver engines.

By using the inheritance concepts, new classes for contact system codes can be directly created from the existing base class. In other words, the derived classes inherit the data and member functions of the base class. And at the same time, some additional methods and attributes relating to it can be implemented. As is shown in Fig. 2, we register the new developed software as "ProductLiToTac". Then some version information about the current developed software would be provided to the OMT platform according to the virtual methods in OmtProductBase class. ModuleContactDetection, ModuleInitVelocity,

ModuleLimits, ModuleLoad, ModuleMesh, ModuleMaterial, ModuleSolution are the professional concrete modules derived from the base abstract class ModuleOmtBase. Each module, as its name explains, corresponds to a necessary function for contact dynamic simulation. These modules are created individually, and inherit the basic function members like InitalModule(), ReadModule(), GetModuleActions(), etc from the OmtProductBase class. As a result, they can independently manage their own data and behaviors.

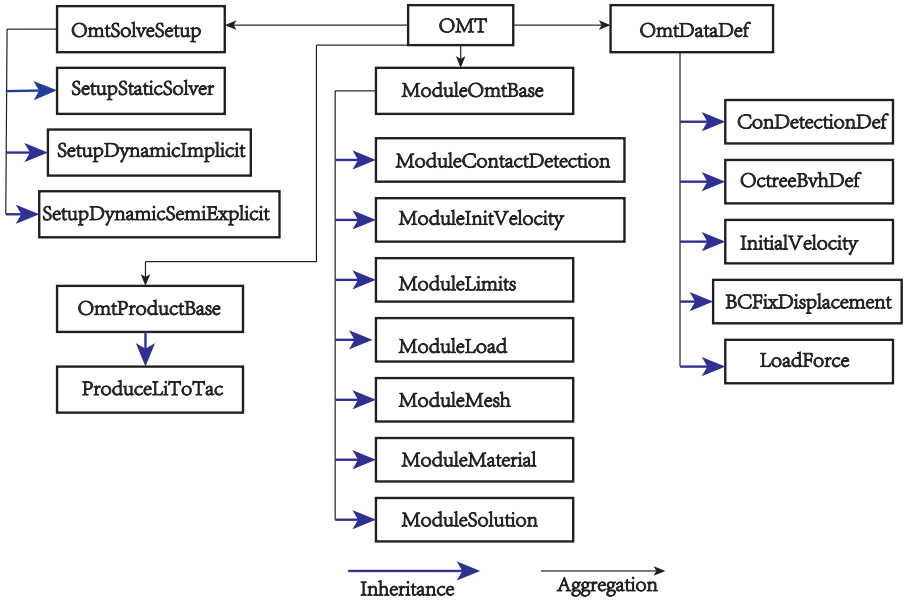


Figure 2: Derived class from OMT base class

It is clear that the modularization by splitting one project into several small modules can be a big advantage of OOP in large scale software design, because each programmer just needs to take charge of one part of them, then the total design time can be significantly reduced. Also, one can easily extend the software to multiple suites by plugging in other functional modules. For instance, ModuleInitialVelocity is in charge of adding initial velocities to nodes before performing time loops. It manages the related date from the geometry and from the interactive interface. If someone wants to use acceleration as the initial condition, ModuleInitalVelocity can be deleted, and then be replaced with a ModuleInitialAcceleration. These kinds of operations do not affect the running of other modules and also have no influence on the final simulation results.

The encapsulation of OOP is a way of defining private data members and function members that are hidden from view outside of the object's definition. That means only the object's own method can directly have access to manipulate its field. In our code, the ModuleMesh class has two private data: NumNodes and NumElements. The value of these two items

can be obtained by inputting data operation defined by the ModuleMesh, but they are not allowed to be recalled or modified by the methods of other modules, even though these modules have to gather information from mesh geometry. By allowing the developers to limit the inter-dependencies between software components, the encapsulation mechanism can enormously help reduce the system complexity and increase robustness.

Polymorphism, which means a variety of forms", is the fundamental technique of OOP. Polymorphism is generally implemented by virtual member functions, which allow redefinition in the subclasses. This characteristic enables objects to respond to messages differently for the same function. Here, we take the operation of saving data function as an example to illustrate this concept in our code. It is very common to click the "save project" button in the menu items when one wants to reuse the current data or simulation operations for the next time. By this operation, the OMT manager will firstly create a project file, and then send messages to each module to call the function "SaveProductData". SaveProductData is a virtual function in the base class of ModuleOmtBase, while its subclasses override this method too. As a result, although all the modules are told to save ProductData, the concrete module only saves its own data inside the project file. Also, through polymorphism, the ModuleMesh can draw any geometry components (selected nodes, elements, surface sets, etc by the operation on the interface) in run-time without pretreatment of geometrical types before compiling. This is so-called dynamic polymorphism, which ensures multiple methods can be implemented by one connector.

3 General structure of the software and computational methods

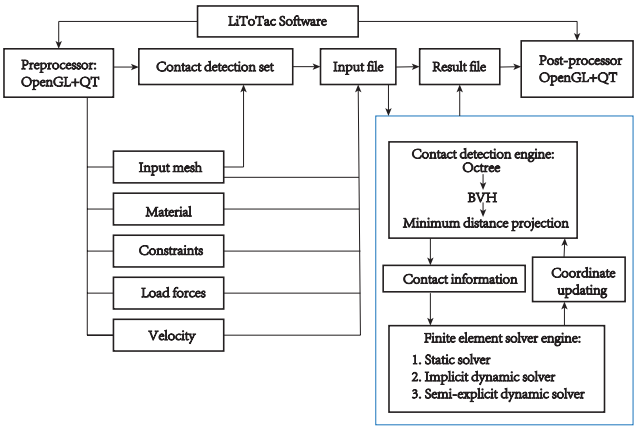


Figure 3: Flow diagram of the developed software

Fig. 3 gives the flow diagram of the software, in which the contact detection module and the finite element module are the core components. Once the preprocessing and the contact detection are finished, an input file will be produced for the contact detection engine and

solver engine. Within a time step, the contact detection engine based on a hybrid methodology consisting of Octree and bounding volume hierarchy (BVH), can calculate the contact information (actual contact area, nodes, penetration, etc) and transform these data to the finite element solver engine. The Octree and BVH need to be updated based on the current coordinates, thus they have to consider the solved displacements for each time step. The solver engine includes three analysis types: static solver, implicit dynamic solver, and semi-explicit dynamic solver. The final computed results are written in an output file and displayed by the post-processor.

3.1 Automatic contact detection

Contact detection is usually considered as a prerequisite for solving contact forces. It can be found in applications of virtual reality, game development, and robotics, etc. In the literature, the range of contact detection algorithms proposed for the finite element modeling is various. These include methods based on the slave-master surface [Hallquist, Goudreau and Benson (1985)], the nested bucket [Benson and Hallquist (1990)], the sweep and prune [Cohen, Lin, Manocha et al. (1995)], the closest features [Gilbert, Johnson and Keerthi (1988); Cameron (1997); Lin and Canny (1991)], the LC-grid [Chen, Lei and Zang (2014)], etc. For survey articles on contact detection, the reader is referred to [Jiménez, Thomas and Torras (2001); Lin and Gottschalk (1998)].

In the multiple contact systems with a large number of bodies, contact detection becomes quite complicated. For one thing, bodies within the scene can move and deform unpredictably from the previous loading steps. For another thing, highly frequent occurrences of contact events can significantly increase the computational cost. The most widely used methods in CAE software such as ANSYS and ABAQUS, are based on slave-master approaches, in which the user should pre-define contact zones. This may result in a heavy burden for users to find potential contact nodes and surfaces in order to establish a list of contact zones. Moreover, for systems with very complex geometry architecture, contact zones may be inside the body, which is generally impossible to be predefined by simple operations like picking nodes or elements from the interface. In our code, these problems can be solved by performing automatic contact detection based on a three-stages methodology. The main idea behind each stage will be briefly introduced in the following subsections.

3.1.1 Top stage for broad inspections

The top-stage inspections implemented in ModuleContactDetection is based on a pointer-based Octree data structure [Wilhelms and Van Gelder (1992)], which can quickly enumerate pairs of objects that may potentially collide. The Octree (Fig. 4) represents the 3D volume as a hierarchy of discrete octants, in which each parent octant is recursively subdivided into eight child octants. The octant can be added or removed as required, thus is well suited to dynamic problems that update frequently.

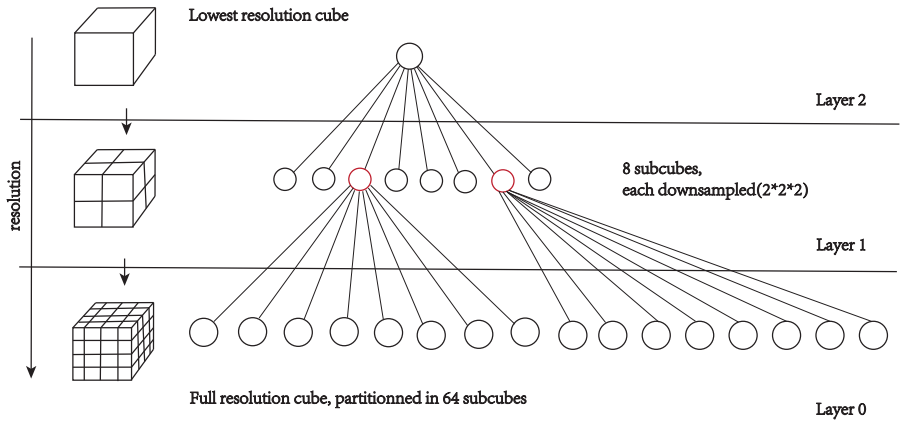


Figure 4: Octree partition [Ericson (2004)]

```
Struct OctreeNode
{
 public:
     Vector3f   MinCor;       //minimum coordinate of bounding box
     Vector3f   MaxCor;       //maximum coordinate of bounding box
     Vector3f   Center;       //center coordinate
     OctreeNode *Children[2][2][2]:          //eight children pointers
     OctreeNode *Parent;
     bool HasChild ;           //if the node needs to be partitioned
     int NodeHeight;           //node height
 }
```

Figure 5: Octree node struct

The definition of a typical Octree node is shown in Fig. 5. Each octant stores pointers to its eight children. But only these non-empty octants are considered to take memory, while others are defined *Null*. This allows all the search paths to be tested with a lower memory

storage. The recursive location of objects terminates when the branch reaches the tree depth, or the number of objects per octant below the predefined value. The Octree structure is searched from the root down to determine collisions. Only objects that share the same octants are taken to have potential contacts.

Note: each object in an Octree hierarchy should be firstly culled by an approximated bounding box for a rough test. To locate the real contact nodes or elements in the following stages, each object is also decomposed of a finite element mesh.

3.1.2 Middle stage for interactions between two FEM meshes

After the filtering of the first stage, the middle stage lies in how to precisely detect the contact elementary pairs (e.g., node-segment(2D), node-triangle(3D), node-quad(3D)) be-tween two meshes. Accordingly, the technique of bounding volume hierarchies (BVH) is adopted for primitive queries. Types of BVs include spheres, axis aligned bounding boxes (AABBs), discretely oriented polytopes (K-DOPs), or a hybrid of them. In this work, we employ the popular hierarchy of AABBs as the fundamental BVHs, because they provide a good trade-off between tightness of fit and computation cost.

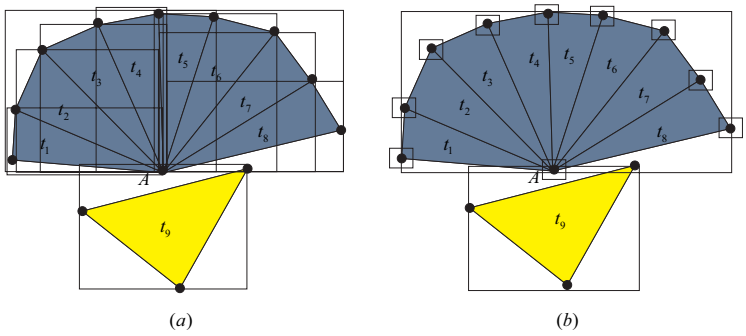
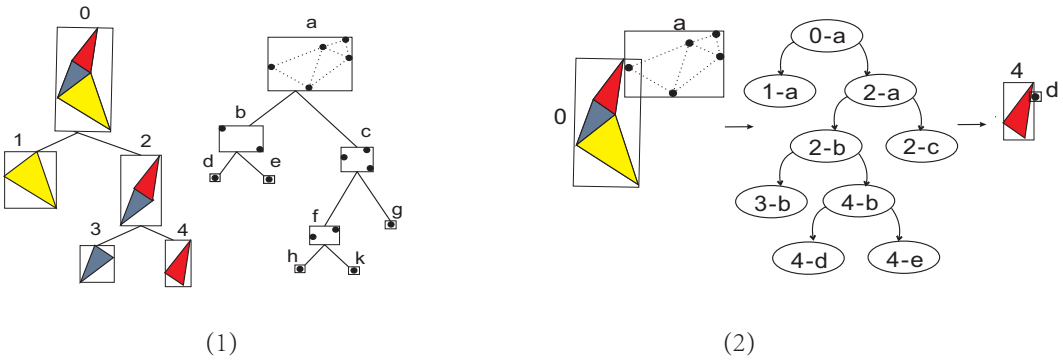


Figure 6: Elementary test between vertex A and face t_9

We create bounding volumes based on specific geometric components, i.e., vertices, edges and faces, see Fig. 6(b). This strategy can largely reduce duplications of elementary queries compared to traditional methods by merely using triangle elements (Fig. 6(a)). For example, the vertex A (Fig. 6(a)) is incident to eight triangles (t_1,,,t_8), and A comes into contact with triangle t_9, which will produce eight times triangle-triangle tests, and 24 times vertex-triangle tests. But in Fig. 6(b), each bounding volume enclosing a vertex is represented only once in the hierarchy, thus the overlapping between vertex A and triangle t_9 would be unique. Furthermore, the separate hierarchies can improve the culling efficiency of elementary testing, because the volume of a vertex can be much smaller than any other type of geometric component.



(1) (2)

Figure 7: (1) Triangle-based BVH and Vertex-based BVH, (2) Hierarchy traversal between two meshes

A triangle-based hierarchy or a vertex-based hierarchy can be built in a typical top-down manner [Bergen (1997)], see Fig. 7(1). It starts out by enclosing the hull of all finite element primitives with a tight Axis-aligned bounding box (AABB). Then these primitives are successfully sorted into two smaller subsets from one level to a deeper lever, thus producing a binary tree hierarchy. The recursion stops when the leaf nodes of the tree consist of only a single primitive. For the constructing process, the most important part is to find a suitable splitting plane to partition the primitives. To ensure a balanced tree, we position the splitting plane orthogonal to the longest AABB axis, which should pass through the median of the centroid coordinates. Any primitive is classified depending on its location with respect to the splitting plane.

In Fig. 7(2), all the contacting elementary pairs between object 0 and a can be searched by a recursive traversal algorithm. The two trees are traversed down from the root, and recursively descend to deeper levels. The primitive contact tests are finally performed between the leaf BV 4 and BV d. This tree search complexity is in logarithmic order, which is particularly efficient for cases with very few contact elements, but a very large data system. Note that, for finite element contact calculation, the simplest linear triangle element can be replaced by other more complicated types in the context of finite element analysis, e.g., 6-node triangle, 4-node quad, 8-node quad, etc.

3.1.3 Bottom stage for local calculation

Finally, the bottom stage functions to discern whether these pairs detected by middle stage are really in penetrated condition. Consider a surface φ, any point on the surface can be presented by a parametric form: $\mathbf{S}(\varepsilon,\eta) = \sum N_i(\varepsilon,\eta)\mathbf{x}_i$, where N_i denotes the Lagrangian shape function associated with node position $\mathbf{x_i}$. Let point $\mathbf{y}$ be the potential counterpart of the contact surface φ as illustrated in Fig. 8.

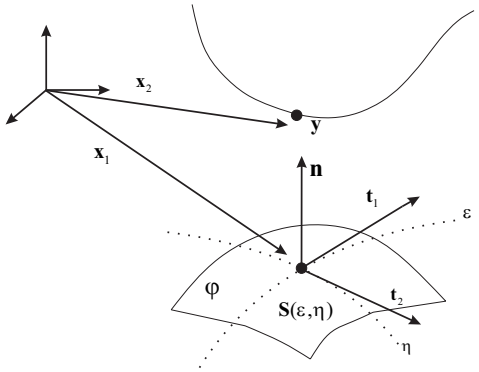


Figure 8: Contact node location

Therefore, the closest point from surface φ to an arbitrary point $\mathbf{y}$ can be obtained after solving the following system of equations:

$$f(\mathbf{l}) = \begin{bmatrix} f_1(\mathbf{l}) = (\mathbf{S}(\varepsilon,\eta) - \mathbf{y}) \cdot \mathbf{v}_\varepsilon \\ f_2(\mathbf{l}) = (\mathbf{S}(\varepsilon,\eta) - \mathbf{y}) \cdot \mathbf{v}_\eta \end{bmatrix} = 0, \ \mathbf{l} = \begin{bmatrix} \varepsilon \\ \eta \end{bmatrix} \tag{1}$$

where $\mathbf{v}_\varepsilon = \frac{\partial \mathbf{S}(\varepsilon,\eta)}{\partial \varepsilon}$, $\mathbf{v}_\eta = \frac{\partial \mathbf{S}(\varepsilon,\eta)}{\partial \eta}$. The problem stated by Eq. (1) may be nonlinear when the surface φ is curved, for example, the quadrilateral face. Therefore, the solution can be obtained by the Newton-Raphson iterative process:

$$\begin{cases} J(\mathbf{l}^{(k)})(\Delta\mathbf{l}^{(k)}) = f(\mathbf{l}^{(k)}) \\ \mathbf{l}^{(k+1)} = \mathbf{l}^{(k)} - \Delta\mathbf{l}^{(k)} \end{cases} \tag{2}$$

where $J(\mathbf{l}^{(k)}) = \begin{pmatrix} \frac{\partial f_1(\mathbf{l}^{(k)})}{\partial \varepsilon} & \frac{\partial f_1(\mathbf{l}^{(k)})}{\partial \eta} \\ \frac{\partial f_2(\mathbf{l}^{(k)})}{\partial \varepsilon} & \frac{\partial f_2(\mathbf{l}^{(k)})}{\partial \eta} \end{pmatrix}$. The iterations usually converge in four or less iterations. The final solution (ε^*, η^*) should satisfy the conditions: $-1 \leq \varepsilon^* \leq 1, -1 \leq \eta^* \leq 1$.

Herein, the outward normal vector of point (ε^*, η^*) is determined by the cross product of $\mathbf{r}_\varepsilon$ and $\mathbf{r}_\eta$

$$\mathbf{n} = \frac{\partial \mathbf{S}(\varepsilon,\eta)}{\partial \varepsilon} \times \frac{\partial \mathbf{S}(\varepsilon,\eta)}{\partial \eta}|_{\varepsilon=\varepsilon^*,\eta=\eta^*} \tag{3}$$

The gap vector is given by

$$\mathbf{g} = \sum \mathbf{N_i}(\varepsilon^*, \eta^*)\mathbf{x_i} - \mathbf{y} \tag{4}$$

3.2 Finite element computation

The solver in ModuleSolution is based on the finite element method for multiple contact dynamic analysis. Typically, the finite element formulation of this problem in the discrete form can be written as:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{A}\dot{\mathbf{u}} = \mathbf{F}_{ext} + \mathbf{F}_{int} + \mathbf{R} \tag{5}$$

where $\mathbf{F}_{ext}$ the vector of external loads, $\mathbf{F}_{int}$ the internal forces, $\mathbf{R}$ the contact reaction vector, $\mathbf{M}$ the mass matrix, $\mathbf{A}$ the damping matrix, $\dot{\mathbf{u}}$ the velocity vector and $\ddot{\mathbf{u}}$ the acceleration vector. It is worth noting that the internal forces $\mathbf{F}_{int}$ are derived from the physics-based deformable model related to the stiffness effect.

The variables such as velocity and stress inside the continuum field are not known. The shape interpolating function $\mathbf{N}$ is needed to approximate them by the nodal values of the field variable. The shape functions $\mathbf{N}$ are generally written in the form of polynomials depending on the element's shape. If the displacement vector $\mathbf{u}$ in an element Ω^e is approximated by $\mathbf{u} = \mathbf{N} \cdot \mathbf{u}^e$, where $\mathbf{u}^e$ represents the displacements at nodes. Then the mass matrix $\mathbf{M}^e$, the damping matrix $\mathbf{A}^e$ can be defined as:

$$\mathbf{M}^e = \int_{V^e} \rho^e \mathbf{N}^T \mathbf{N} dV, \quad \mathbf{A}^e = \int_{V^e} \kappa^e \mathbf{N}^T \mathbf{N} dV, \tag{6}$$

where ρ^e is the mass density, κ^e the damping parameter, V^e the volume of Ω^e.

The solution to Eq. (5) differs between explicit and implicit approaches. The explicit method appears to be efficient, but unstable, and without checking convergence. The implicit method is supposed to be accurate. Nevertheless, the computation is much more expen-sive for large deformations. Currently, our code offers three solvers for different types of analysis: (1) the implicit solver [Feng, Joli, Cros et al. (2005); Feng, Magnain and Cros (2006)] to address low-velocity or quasi static cases; (2) the semi-explicit solver [Peng, Feng and Joli (2018)] to handle high-velocity impact problems; (3) the static solver [Feng, Hjiaj, De Saxcè et al. (2006)] to deal with static frictional cases.

3.2.1 Computation of contact forces

Our contact solver is based on the bi-potential method [De Saxcé and Feng (1998)], in which a formulation extended by the augmented Lagrangian method is provided. Considering a local reference frame (normal vector $\mathbf{n}$ and tangential vector $\boldsymbol{\tau}$), there exists an implicit relationship between the constraint displacement $\mathbf{x}$ and the contact reaction forces $\mathbf{r}$:

$$\mathbf{x} = \mathbf{W}\mathbf{r} + \tilde{\mathbf{x}} \tag{7}$$

where $\mathbf{W}$ and $\tilde{\mathbf{x}}$ represent respectively the global compliance matrix and the free displacement. For each contact point α, among N_c instantaneous contact points, the relationship can be written as:

$$\mathbf{x}_\alpha - \mathbf{W}_{\alpha\alpha}\mathbf{r}_\alpha = \sum_{\beta=1}^{\alpha-1} \mathbf{W}_{\alpha\beta}\mathbf{r}_\beta + \sum_{\beta=\alpha+1}^{N_c} \mathbf{W}_{\alpha\beta}\mathbf{r}_\beta + \tilde{\mathbf{x}}_\alpha \tag{8}$$

where $\mathbf{W}_{\alpha\beta}$ is an influence matrix that takes account of the coupling between contact points α and β . With regard to point α, Eq. (8) can be treated by considering other contact points ($\alpha \neq \beta$) as "frozen".

In the work of [De Saxcé and Feng (1998)], the contact bi-potential is formulated as follows:

$$b_c(-\mathbf{x},\mathbf{r}) = \underset{\Re_-}{\cup}(-\mathrm{x}_n) + \underset{K_\mu}{\cup}(\mathbf{r}) + \mu \mathrm{r}_n ||\mathbf{x}_\tau|| \tag{9}$$

where $\Re_-$ is the set of negative and null real numbers, K_μ the so-called Coulomb cone. $\underset{K_\mu}{\cup}(\mathbf{r})$ denotes the indicator function of the closed convex set K_μ.

In order to avoid non-differentiable potentials, it is convenient to use the Augmented La-grangian method [De Saxcé and Feng (1991)], Eq. (9) can be equal to

$$\mathbf{r} = \mathrm{Proj}_{K_\mu}(\mathbf{r}^*) \tag{10}$$

and

$$\mathbf{r}^* = \mathbf{r} - \rho\mathbf{x}^* \quad \text{with} \quad \mathbf{x}^* = \mathbf{x} + \mu \|\mathbf{x}_\tau\| \mathbf{n} \tag{11}$$

where $\mathbf{r}^*$ is the so-called augmented contact forces, and ρ is a positive real parameter, μ the friction coefficient. $\mathrm{Proj}_{\mathrm{K}_\mu}(\mathbf{r}^*)$ means that $\mathbf{r}$ is the projection of $\mathbf{r}^*$ onto the closed convex Coulomb cone.

The Uzawa technique can be used to solve the implicit Eq. (8), which leads to a prediction-correction procedure as:

$$\begin{array}{l} \text{Prediction} : \mathbf{r}^{*(i+1)} = \mathbf{r}^{(i)} - \rho^{(i)}(\mathbf{x}^{(i)} + \mu ||\mathbf{x}_\tau^{(i)}|| \mathbf{n}) \\ \text{Correction} : \mathbf{r}^{(i)} = \mathrm{Proj}_{\mathrm{K}_\mu}(\mathbf{r}^{*(i+1)}) \end{array} \tag{12}$$

where the correction is explicitly carried out with respect to three possible contact statues as:

$$\begin{array}{ll} \text{if} \quad ||(\mathbf{r}_\tau^*)^{i+1}|| \leq \mu (r_n^*)^{i+1} \quad \text{then} \quad \mathbf{r}^{i+1} = (\mathbf{r}^*)^{i+1} & \text{(sticking)} \\ \text{else if } \mu ||(\mathbf{r}_\tau^*)^{i+1}|| < -(r_n^*)^{i+1} \quad \text{then} \quad \mathbf{r}^{i+1} = \mathbf{0} & \text{(no contact)} \\ \text{else} \quad \mathbf{r}^{i+1} = (\mathbf{r}^*)^{i+1} - (\frac{||(\mathbf{r}_\tau^*)^{i+1}|| - \mu (r_n^*)^{i+1}}{1+\mu^2})(\frac{(\mathbf{r}_\tau^*)^{i+1}}{||(\mathbf{r}_\tau^*)^{i+1}||} + \mu \mathbf{n}) & \text{(sliding)} \end{array} \tag{13}$$

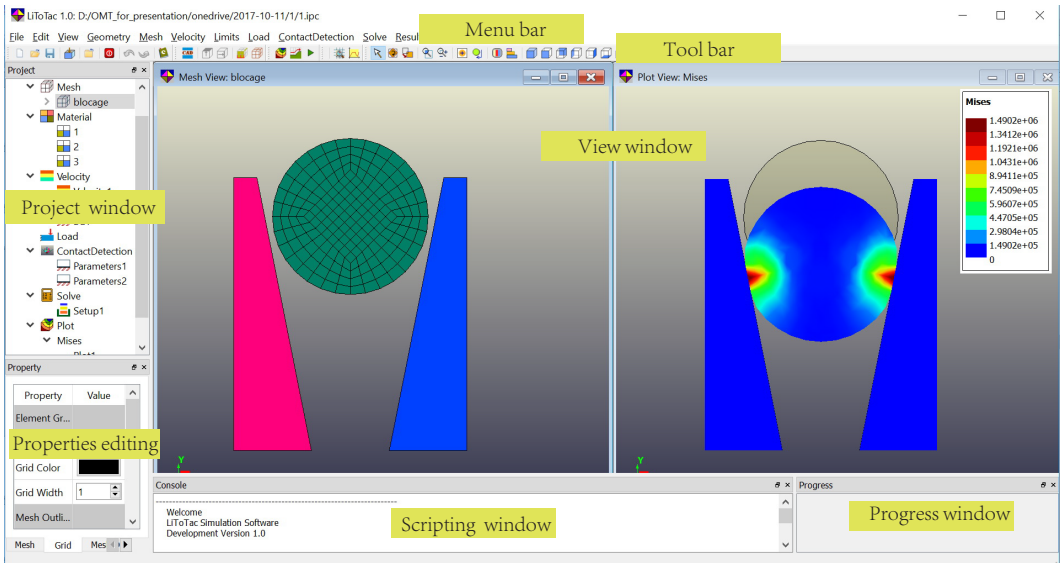


Figure 9: GUI of LiToTac

After the prediction-correction procedure converges, the global contact forces $\mathbf{R}$ can be obtained by

$$\mathbf{R} = \mathbf{H}(\mathbf{u})\mathbf{r} \tag{14}$$

where $\mathbf{H}(\mathbf{u})$ is the mapping matrix from the local frame to the global frame.

4 Functionalities of the software

Qt is an efficient application framework for developing multi-platform applications, by which the code can be run on various systems and hardware platforms with little or even no change. In the current project, Qt has been largely used to design the graphical user interface (GUI) objects such as Dialog, Menu, Toolbar, Label, Icon, etc. Also, to render the scene with high fidelity, the OpenGL library is applied to draw graphic primitives like points, lines, surfaces, volumes, etc.

As mentioned, the software LiToTac is an integrated environment which consists of four main parts: the preprocessor, the contact detector, the solver, and the post-processor. The GUI of the program is shown in Fig. 9. Some main characteristics of LiToTac are presented as follows.

4.1 Preprocessor

- **Mesh:** to allow the user to make certain operations on the model geometry such as creating, deleting, modifying the sets of primitives (nodes, elements, surfaces, etc). In addition, it helps to render the view environment based on user's habits. The user can achieve some special effects by defining light, texture, transparency, color, and so on.

- **Material:** to set the parameters for different material models. Several material models including linear elasticity, and non-linear hyperelasticity have been implemented in our code. Particularly, users are allowed to quickly add their own material models into the library in very few steps, which provides better convenience over general commercial softwares.
- **Create input files:** after finishing the procedures as described above, an input file with only necessary information is produced. This file can also be generated for supporting ANSYS or other commercial software.

4.2 Contact detection

- The contact detection is performed automatically to gather the basic data for calculating contact forces. The construction of Octrees and bounding volume hierarchies should rely on the current coordinates provided by the initial mesh as well as the motion displacements.
- For some particular materials, different areas within a body may produce several different frictional coefficients when interacting with another kind of material. This may add complexity to the computation of frictional force and the management of data. In our software, the user can well handle this issue by defining frictional coefficients between two kinds of groups, wherein, one is the node groups, the other is the element edge groups. The value of coefficients will be recorded in a 1D array. Consequently, once a contact detection query is solved, each of the resulting contact pairs can have a individual frictional coefficient based on the node ID and its counterpart element ID. This methodology allows users to make further researches on inhomogeneous friction and anisotropic friction problems.

4.3 Solver

The solver consists of the following tasks:

- to read the input file from the preprocessor.
- to exchange data with the contact detection engine.
- to solve contact problems via implicit, semi-explicit, and static methods.
- to perform hourglass control for semi-explicit simulation, when one point Gauss integration is applied to accelerate the computing speed.
- to create output files of the solutions for the purpose of post-processing.

4.4 Post-processor

The roles of the post-processor can be summarized as follows:

- to read the result file produced by the solver engine.
- to import the stress analysis solutions.
- to display the structure of a solid body by scanning.

- to perform animation of the deformation evolution and the stress distribution along time.
- to show the evolution of actual contact areas.
- to create reports about the energy including total energy, kinematic energy, and elastic energy with change of time.

5 Example

In order to test and validate the functions and the capability of LiToTac, several specific examples have been carried out. It is noted that the following analysis are performed on a PC (i7-8550U, 1.80 GHz).

5.1 Example 1: 2D quasi-static contact simulation

The first model consists of 16 hyperelastic bodies (spheres) and 4 rigid bodies (rectangles), as is shown in the left side of Fig. 10. The highlighted yellow points on the rigid bodies are fixed, while the left rectangle body is imposed by constant force loading. The plots of Von-Mises stress at the final step are presented in the right side of Fig. 10. This example is a very special quasi-static case, which can not be modeled by a traditional static solution. Because several spheres inside the system are enforced without displacement limits, but only with contact constraints, which leads to a singularity of the global stiffness matrix. It is also difficult to apply a general dynamic solver to perform this simulation. Since quasi-static simulations requires extremely accurate and robust computation of dynamic contact forces. Whereas, most dynamic solvers in commercial softwares may produce numerical oscillations when taking account of impact effects. By using the implicit method [Feng, Joli, Cros et al. (2005)], LiToTac can well address this case. As is shown in the left side of Fig. 11, the hyperelastic bodies can deform stably in very low velocity, since it generally exists no obvious oscillations in the evolution of Von-Mises versus time.

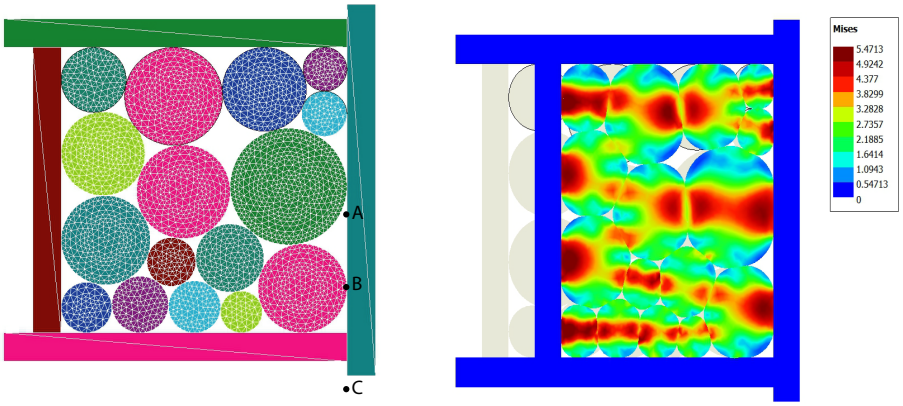


Figure 10: A quasi static model with 20 interacting bodies. Left side: initial mesh. Right side: the final distribution of Von-Mises stress

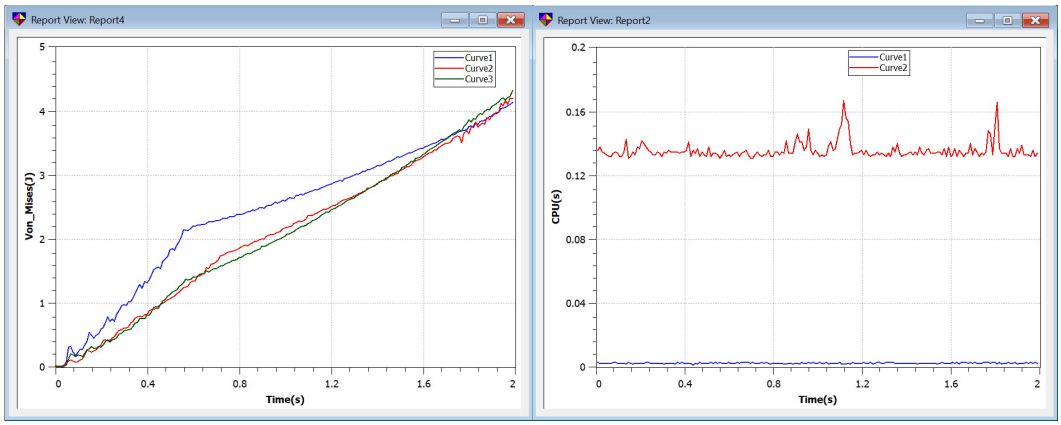


Figure 11: Left side: evolution of Von-Mises stress versus time of node A, B, C. Right side: the comparison of CPU time between our automatic method and the brute force method, where Curve1 is the automatic contact detection, and Curve2 the brute force detection

By means of the powerful post-processing capabilities of LiToTac, users can import the results of different situations and plot the reports together to view the influences by applying different computing algorithms. For example, an analysis report of contact searching is given in the right side of Fig. 11. The result shows the computational time of the automatic contact detection (Curve1) is significantly lower than the brute force (Curve2). Therefore, the efficiency of automatic contact detection is confirmed. Additionally, it is worthy noting a multi-window function of LiToTac. In Fig. 9 or Fig. 11, the mesh window and plots window are able to be viewed at the same time. This function can provide great convenience for users when they need to gather information from different views.

5.2 Example 2: 3D soft rope system under pure torsion

This 3D example is a typical "1+6" structure [Ghoreishi, Davies, Cartraud et al. (2007)] composed of seven hyperelastic fibers. Large deformation and sliding are analyzed by using the static solver. The total torsion angle is $\theta = 95^{\circ}$, the bottom surface is fixed, and the displacements of Z direction on the top surface are set to zero. 25 load steps are performed for this problem, so a pure torsion of $3^{\circ}48'$ is applied to each step.

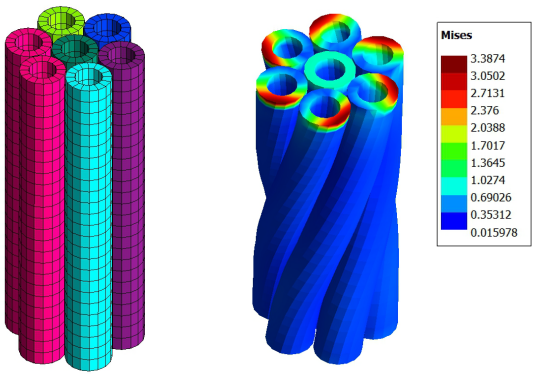


Figure 12: Left side: Initial mesh. Right side: Final state of Von-Mises stress distribution

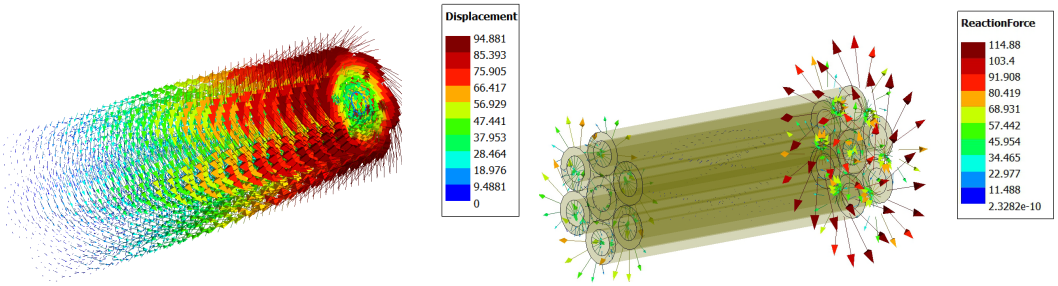


Figure 13: Left side: Vector quantity of displacements at Step 25. Right side: vector quantity of reaction force at Step 25

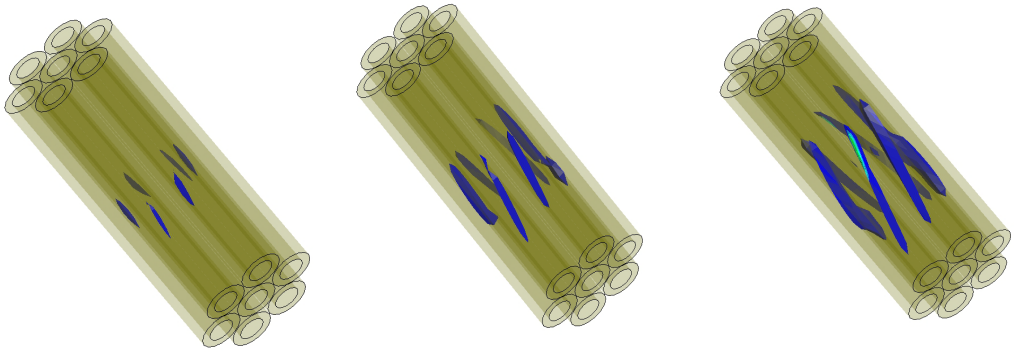


Figure 14: Contour plots of contact force at real interacting nodes. Left-Step 13, middle-Step 18, right-Step 25

The initial mesh and the final state of distribution of Von-Mises stress are shown in Fig. 12. From the obtained results, it can be seen that the stress distribution is symmetrical to the

center point. The vector quantity of displacements and reaction forces at Step 20 are given in Fig. 13. Fig. 14 depicts the contour plots of contact forces at real contacting nodes at different loading steps, which allows us to observe the inter fiber-fiber interactions.

Numerical modeling of fiber systems still remains a very difficult issue in FEM field. The inter contacts between fibers may become unpredicted because of its complicated architecture and large number of bodies. Example 2 presented here aims at illustrating the capacity of LiToTac, thus no detailed analysis of mechanical behavior is performed. The functionalities mentioned such as multi-windows, cut plane of inter structure, animation of real contact areas, view plotting of contact force contour, etc are quite convenient, which permits LiToTac to handle complicated fiber-fiber interacting simulations with much less operations than commercial software.

5.3 Example 3: 2D comparison between LiToTac and commercial software

To illustrate the computational efficiency of LiToTac, we consider here one example to make a comparison with the commercial software ANSYS and ABAQUS. This example is inspired by the work of Wriggers et al. [Wriggers, Van and Stein (1990)], which involves a hyperelastic cylinder impacting upon two oblique rigid symmetric surfaces, see Fig. 15. During the pro-cess, there exists no damping except for Coulomb friction between contact surfaces. The Yeoh material model is considered to describe the hyperelasticity behavior. The character-istics of this example are: $c_{10} = 3794000\ Pa$, $c_{20} = 232000\ Pa$, $c_{10} = -3000\ Pa$, $d_1 = 1e^{-7}$, $d_2 = 1e^{-7}$, $d_3 = 1e^{-7}$, initial mass density $\rho = 1207\ kg/m^3$, initial velocity $v_y = -30\ m/s$. The total simulation time is $3e^{-3}\ s$, and the time step is $\Delta t = 10e^{-5}\ s$. The cylinder consists of 209 nodes and 192 linear quadrilateral plane strain elements. The cylinder is positioned at center point O (0.0, 0.03) and its radius is $R = 0.01\ m$. The right side of the rigid block is located by four points: A(0.005, 0.0), B(0.015, 0.0), C(0.015, 0.035), D(0.012, 0.035).

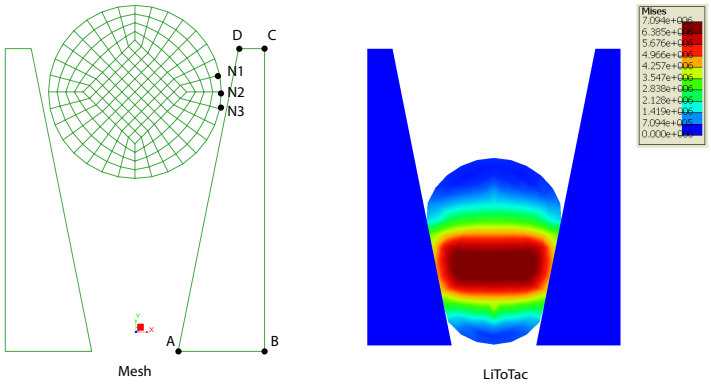


Figure 15: Deformable-rigid impact

Firstly, the distribution of Von-Mises stress contour at time $t = 0.002\ s$ is displayed in Fig. 16. From the results, we can observe that the maximum value of Von-Mises stress in LiToTac is slightly lower than ANSYS and ABAQUS. In Tab. 1, N1, N2, N3 are three selected nodes that interpenetrate with surface AD at time $t = 0.002\ s$. As is shown, the contact penetrations δ of these nodes in LiToTac can be controlled in the order of 10^{-16}, while the value just approaches to 10^{-5} in ANSYS, and 10^{-6} in ABAQUS. This difference indicates that LiToTac can be more accurate than ANSYS and ABAQUS to satisfy the contact impenetrability condition. The performance of computational time is reported in Tab. 2. Through the result, we can find that LiToTac possesses a small performance difference with ABAQUS, but is much faster than ANSYS.

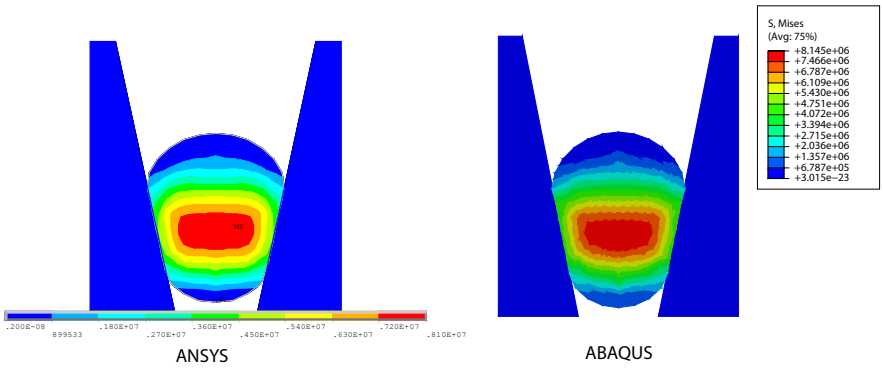


Figure 16: Distribution of Von-Mises stress

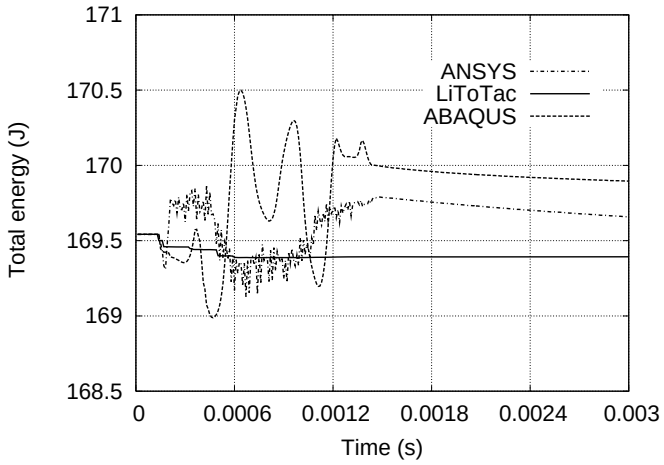


Figure 17: Comparison of total energy

It is also interesting to compare the energy evolution of the two softwares. Fig. 17 plots the total energy as a function of time in the case of frictionless contact. We can see that the total energy for these three softwares is conserved within an acceptable range of error (less than 0.5%). However, the change of energy curves confirms a better robustness of LiToTac over ANSYS and ABAQUS.

Table 1: Comparison of contact penetration

Code	Method	δ_{N_1} (m)	δ_{N_2} (m)	δ_{N_3} (m)
ANSYS	Penalty	$-1.114e^{-5}$	$-1.643e^{-5}$	$-1.824e^{-5}$
ABAQUS	Lagrange-multiplier	$-3.567e^{-6}$	$-4.040e^{-6}$	$-4.653e^{-6}$
LiToTac	Bi-potential	$-7.758e^{-18}$	$-6.939e^{-18}$	$-2.194e^{-17}$

Table 2: Comparison of CPU performance

Code	Formulation	Contact algorithm	Total CPU (s)
ANSYS	Second-order implicit	Penalty	71.3
ABAQUS	Second-order implicit	Lagrange-multiplier	18.5
LiToTac	First-order implicit	Bi-potential	16.3

5.4 Example 4: 3D comparison between LiToTac and commercial software

As a comparison, a 3D example consisting of two deformable blocks is also analyzed here, see Fig. 18. For convenience, the normalized units are used. The smaller block is created by its diagonal corner coordinates (0.5, 0.0, 1.05) and (1.5, 1.0, 2.05), and the larger one is defined by (0.0, 0.0, 0.0) and (2.0, 2.0, 1.0). The smaller block impacts onto the large block with a initial rigid-body velocity (0.0, 2.0, -1.0), and the base of the larger block is fixed. The whole structure is set with the same density 0.01. We consider again the Yeoh model: $c_{10} = 0.3794$, $c_{20} = 0.0232$, $c_{10} = -0.0003$, $d_1 = 0.01$, $d_2 = 0.01$, $d_3 = 0.01$. The total simulation time is 0.5 scaled time unit, and the time step is $\Delta t = 0.005$. The frictional coefficients are set as: $\mu = 0.0$. During the process, no damping effects are considered. The finite discretization includes 208 nodes and 102 eight-node hexahedrons.

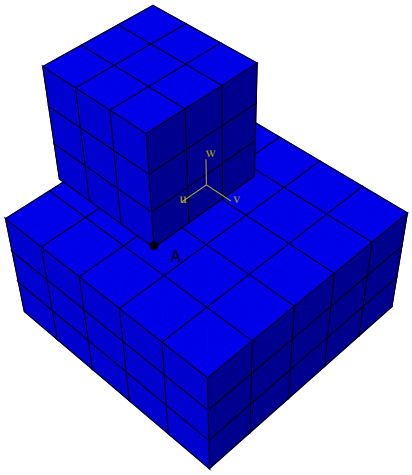


Figure 18: Initial configuration

Fig. 19 shows the plots of the Von-Mises stress of Node *A* versus time. We can observe that the curves produced by LiToTac and ABAQUS are close to each other, and are more robust than the case of ANSYS. Fig. 20 indicates the evolution of total energy as a function of time. The results confirms that LiToTac allows a better conservation of energy than ABAQUS for 3D frictionless contact problem. Whereas, for ANSYS, an increase of total energy can be produced because of numerical instability.

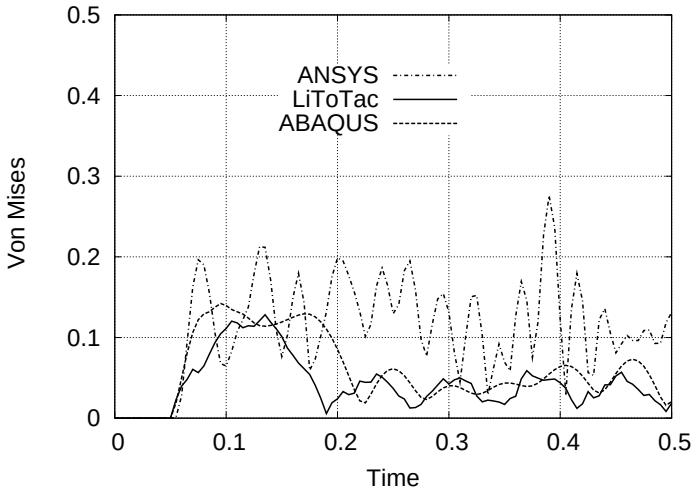


Figure 19: Von-Mises stress at point *A*

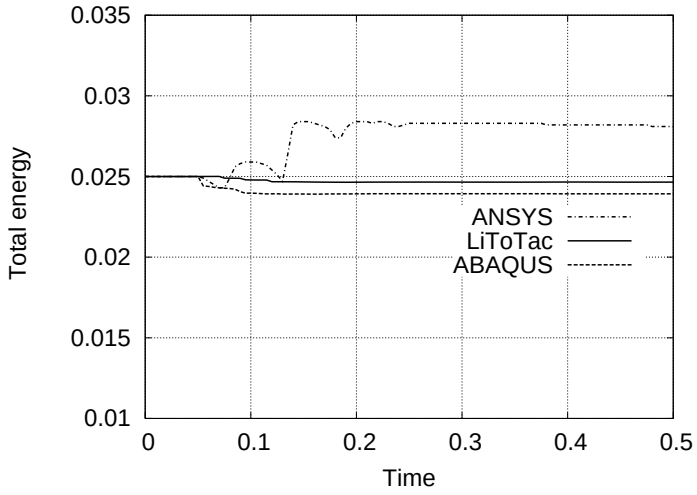


Figure 20: Evolution of total energy for $\mu = 0.0$

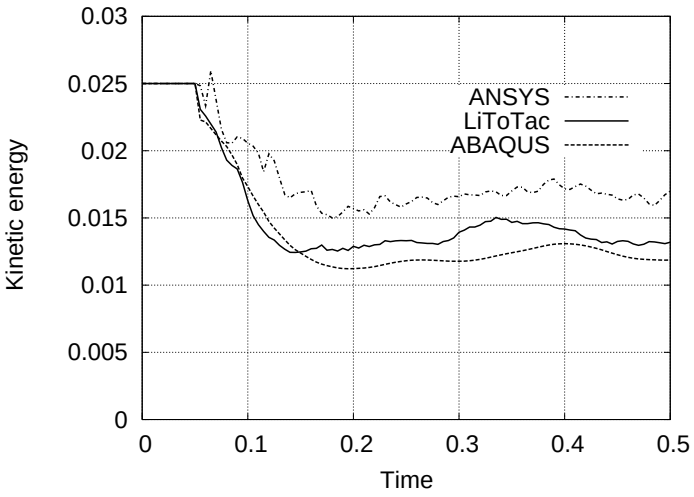


Figure 21: Evolution of total energy for μ = 0.5

To investigate the frictional effects on the algorithm performance, we change the frictional coefficient to $\mu = 0.5$. Fig. 21 gives the plots of whole kinetic energy versus time. In LiToTac and ABAQUS, the kinetic energy decreases as expected by frictional contact effects after time $t = 0.05$ when the impact occurs. However, in ANSYS, a jump takes places at time $t = 0.065$. The performance of the applied algorithms in each software is summarized in Tab. 3. The Lagrange-multiplier method in ABAQUS is replaced by penalty method because of its non convergence in dealing with this frictional case. Though the results obtained by computing a 3D example, we can see that LiToTac is faster than ANSYS, but a little bit slower than ABAQUS.

Table 3: Comparison of CPU performance

Code	Algorithm	Total CPU (s) $\mu = 0.0$	Total CPU (s) $\mu = 0.5$
ANSYS	Second-order implicit+Penalty	21.8	24.7
ABAQUS	Second-order implicit+Penalty	7.1	8.1
LiToTac	First-order implicit+Bi-potential	9.3	9.6

6 Conclusion

In this paper, we have presented a new finite element software for the modeling and analysis of multiple contact dynamics. This software is able to handle strong non-linearities including multiple contacts and large deformations, and it can also perform automatic contact detections without predefining contact zones. The program has been designed by using the object-oriented principles and the OMT technology. These techniques allow us to simplify the architecture of the program and to incorporate quite easily new features to other specific suites. Through several complicated numerical examples, the new function of automatic contact detection confirms a good convenience over slave-master approaches. Also, by comparing with the commercial software ANSYS and ABAQUS, the bi-potential techniques in LiToTac are demonstrated to possess higher accuracy and better robustness than penalty and Lagrange multiplier algorithms.

The software presented can be further extended to study local phenomenon occurring in polymer chains, and DNA, or in the domain of computer graphics to improve the rendering of hairs. It does not need to completely redefine the software architecture. Some optimization methods can also be added to perform parallel computation of large-scale problems.

Acknowledgement: We gratefully acknowledge the financial support of the National Key R&D Program of China (Grant No. 2017YFB0703200) and the National Natural Science Foundation of China (Grant No. 11772274).

References

Bei, Y.; Fregly, B. J. (2004): Multibody dynamic simulation of knee contact mechanics. *Medical Engineering & Physics*, vol. 26, no. 9, pp. 777-789.

Benson, D. J.; Hallquist, J. O. (1990): A single surface contact algorithm for the post-buckling analysis of shell structures. *Computer Methods in Applied Mechanics and Engineering*, vol. 78, no. 2, pp. 141-163.

Bergen, G. V. D. (1997): Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, vol. 2, no. 4, pp. 1-13.

Cameron, S. (1997): Enhancing GJK: computing minimum and penetration distances between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, vol. 4, pp. 3112-3117.

Chen, H.; Lei, Z.; Zang, M. (2014): LC-Grid: a linear global contact search algorithm for finite element analysis. *Computational Mechanics*, vol. 54, no. 5, pp. 1285-1301.

Chen, Z. W.; Joli, P.; Feng, Z. Q. (2015): Anisotropic hyperelastic behavior of soft biological tissues. *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 18, no. 13, pp. 1436-1444.

Cohen, J. D.; Lin, M. C.; Manocha, D.; Ponamgi, M. (1995): I-COLLIDE: an interactive and exact collision detection system for large-scale environments. *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pp. 189-196.

De Saxcé, G.; Feng, Z. Q. (1991): New inequality and functional for contact with friction: the implicit standard material approach. *Journal of Structural Mechanics*, vol. 19, no. 3, pp. 301-325.

De Saxcé, G.; Feng, Z. Q. (1998): The bipotential method: a constructive approach to design the complete contact law with friction and improved numerical algorithms. *Mathematical and Computer Modelling*, vol. 28, no. 4-8, pp. 225-245.

Dubois-Pèlerin, Y.; Pegon, P. (1998): Object-oriented programming in nonlinear finite element analysis. *Computers & Structures*, vol. 67, no. 4, pp. 225-241.

Ericson, C. (2004): *Real-Time Collision Detection*. CRC Press, USA.

Fang, H.; Hand, A. J.; Haddock, J. E.; White, T. D. (2007): An object-oriented frame-work for finite element pavement analysis. *Advances in Engineering Software*, vol. 38, no. 11-12, pp. 763-771.

Feng, Z. Q.; Hjiaj, M.; De Saxcè, G.; Mróz, Z. (2006): Influence of frictional anisotropy on contacting surfaces during loading/unloading cycles. *International Journal of Non-Linear Mechanics*, vol. 41, no. 8, pp. 936-948.

Feng, Z. Q.; Joli, P.; Cros, J. M.; Magnain, B. (2005): The bi-potential method applied to the modeling of dynamic problems with friction. *Computational Mechanics*, vol. 36, no. 5, pp. 375-383.

Feng, Z. Q.; Joli, P.; Seguy, N. (2004): FER/Mech-a software with interactive graphics for dynamic analysis of multibody system. *Advances in Engineering Software*, vol. 35, no. 1, pp. 1-8.

Feng, Z. Q.; Magnain, B.; Cros, J. M. (2006): Solution of large deformation impact problems with friction between blatzko hyperelastic bodies. *International Journal of Engineering Science*, vol. 44, no. 1-2, pp. 113-126.

Fenves, G. L. (1990): Object-oriented programming for engineering software development. *Engineering with Computers*, vol. 6, no. 1, pp. 1-15.

Ghoreishi, S. R.; Davies, P.; Cartraud, P.; Messager, T. (2007): Analytical modeling of synthetic fiber ropes. Part II: a linear elastic model for 1+6 fibrous structures. *International Journal of Solids and Structures*, vol. 44, no. 9, pp. 2943-2960.

Gilbert, E. G.; Johnson, D. W.; Keerthi, S. S. (1988): A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193-203.

Hallquist, J. O.; Goudreau, G. L.; Benson, D. J. (1985): Sliding interfaces with contact-impact in large-scale lagrangian computations. *Computer Methods in Applied Mechanics and Engineering*, vol. 51, no. 1-3, pp. 107-137.

Hartmann, S.; Oliver, J.; Weyler, R.; Cante, J. C.; Hernández, J. A. (2009): A contact domain method for large deformation frictional contact problems. Part 2: numerical aspects. *Computer Methods in Applied Mechanics and Engineering*, vol. 198, no. 33-36, pp. 2607-2631.

Jean, M.; Touzot, G. (1988): Implementation of unilateral contact and dry friction in computer codes dealing with large deformations problems. *Journal de Mécanique Theorique et Appliquée*, pp. 145-160.

Jiménez, P.; Thomas, F.; Torras, C. (2001): 3D collision detection: a survey. *Computers & Graphics*, vol. 25, no. 2, pp. 269-285.

Lin, M.; Gottschalk, S. (1998): Collision detection between geometric models: a survey. *Proceedings of IMA Conference on Mathematics of Surfaces*, vol. 1, pp. 602-608.

Lin, M. C.; Canny, J. F. (1991): A fast algorithm for incremental distance calculation. *IEEE International Conference on Robotics and Automation*, pp. 1008-1014.

Lin, Y. C.; Walter, J. P.; Banks, S. A.; Pandy, M. G.; Fregly, B. J. (2010): Simultaneous prediction of muscle and contact forces in the knee during gait. *Journal of Biomechanics*, vol. 43, no. 5, pp. 945-952.

Liu, Y. F.; Cheng, L. F.; Zeng, Q. F.; Feng, Z. Q.; Zhang, L. T. (2015): PCLab-a software with interactive graphical user interface for monte carlo and finite element analysis of microstructure-based layered composites. *Advances in Engineering Software*, vol. 90, pp. 53-62.

Mahmoud, F. F.; El-Shafei, A. G.; Abdelrahman, A. A.; Attia, M. A. (2013): Modeling of nonlinear viscoelastic contact problems with large deformations. *Applied Mathematical Modelling*, vol. 37, no. 10-11, pp. 6730-6745.

Misra, R. K.; Dixit, A.; Mali, H. S. (2014): Finite element (FE) shear modeling of woven fabric textile composite. *Procedia Materials Science*, vol. 6, pp. 1344-1350.

Mlika, R.; Renard, Y.; Chouly, F. (2017): An unbiased Nitsche's formulation of large deformation frictional contact and self-contact. *Computer Methods in Applied Mechanics and Engineering*, vol. 325, pp. 265-288.

Peng, L.; Feng, Z. Q.; Joli, P. (2018): A semi-explicit algorithm for solving multibody contact dynamics with large deformation. *International Journal of Non-Linear Mechanics*, vol. 103, pp. 82-92.

Wilhelms, J.; Van Gelder, A. (1992): Octrees for faster isosurface generation. *ACM Transactions on Graphics*, vol. 11, no. 3, pp. 201-227.

Wriggers, P.; Van, T. V.; Stein, E. (1990): Finite element formulation of large deformation impact-contact problems with friction. *Computers & Structures*, vol. 37, no. 3, pp. 319-331.

Yuan, Z.; Fish, J. (2015): Nonlinear multiphysics finite element code architecture in object oriented fortran environment. *Finite Elements in Analysis and Design*, vol. 99, pp. 1-15.