Acta Polytechnica CTU Proceedings **34**:15–19, 2022
https://doi.org/10.14311/APP.2022.34.0015

Published by the Czech Technical University in Prague

A MODULAR EXTENSION OF A FINITE ELEMENT CONTACT IMPLEMENTATION

Ondřej Faltus*, Martin Horák

Czech Technical University in Prague, Faculty of Civil Engineering, Department of Mechanics, Thákurova 7, 166 29 Prague 6, Czech Republic

* corresponding author: ondrej.faltus@cvut.cz

Abstract. The structural mechanics module of the OOFEM finite element software has been developed to include various algorithms for solving contact mechanics problems. After developing small strain contact algorithms in the 2D domain and their extension to large strain applications [1, 2], the present contribution extends the existing framework to the 3D domain. After reviewing the current code and comparing existing solutions found in the literature, we identify the common ground between 2D and 3D applications, then propose and implement the necessary changes and additions to smoothly integrate the 3D support into existing code. Tests and example problems are presented to confirm the functionality of the resulting implementation.

Keywords: Contact, large strains, penalty method, 3D contact, node-to-segment, OOFEM.

1. Introduction

Contact mechanics, as a specific subdiscipline of structural mechanics, lend themselves very handily to computational implementations. Analytical solutions to contact mechanics problems are scarcely found. Since the end of the 19$^{\text{th}}$ century, when Heinrich Hertz studied the contact of convex elastic bodies without friction [3], progress in this originally fringe discipline had been very slow until the advent of computational methods, mainly the finite element method (FEM) in the latter half of the 20$^{\text{th}}$ century [4, 5]. Today, more and more complicated problems can be studied with the help of more and more involved algorithms, owing to the technological advances in computational hardware.

In our past work [1, 2], we have developed the basis of contact algorithms in the OOFEM software. OOFEM [6] is a finite element computational software package developed at the Czech Technical University in Prague. From a programming perspective, OOFEM is built as a strictly object-oriented C++ codebase. In our past implementations, we have paid close attention to this fact, adapting some of the commonly in literature presented programmatic solutions to suit the object-oriented philosophy better. So far, contact algorithms in OOFEM can handle contact problems in the 2D domain, using the node-to-node and node-to-segment discretizations and alternatively the penalty method or Lagrangian multiplier method of contact condition enforcement. Of these options, the penalty method node-to-segment approach is adapted for use with geometrically nonlinear problems as well.

This work aims to extend our previous implementation to the 3D domain. Keeping in with the idea of object-oriented code, we try to achieve this by extending the existing objects and algorithms they represent instead of coding everything anew. Chiefly, we would like to avoid the "contact element" approach often presented in literature [4, 7, 8] and create instead a universal contact condition object able to interact with every possible contact segment type.

2. Theoretical Basis

At the core of our modular code extension lies the desire to adapt the existing geometrically nonlinear penalty contact formulation for 2D domains to fully work in the 3D domain as well. For this purpose, we would like to derive a universal formulation for the forces and stiffness applicable to all cases equally.

2.1. Formulation of Penalty Contact

The penalty formulation counts as one of the most straightforward possible implementations of contact conditions into the FEM framework. Its governing idea is the introduction of an arbitrary penalty parameter to the force vector, aimed at penalizing unwanted states of penetration among the contacting bodies. In other words, we express the force on the contact boundary as $H(-g_c)pg_c$, where p is the chosen penalty parameter and g_c the contact gap (negative in penetration) [1, 4, 5, 7]. H is the Heaviside function.

To this end, the energy contribution of the contact process can be expressed as

$$W^c(\boldsymbol{u}) = \int_{\Gamma_c} \frac{1}{2} H(-g_c(\boldsymbol{u}))pg_c^2(\boldsymbol{u}) \, \mathrm{d}\Gamma \qquad (1)$$

where $\boldsymbol{u}$ is the global vector of displacements and Γ_c is the contact boundary. Note that for a node-to-segment contact discretization, the integral morphs into a sum over a finite set of discrete contact points.

With the first variation of the contact energy, we obtain the internal force vector as

$$\boldsymbol{f}_c = \int_{\Gamma_c} p g_c \left(\boldsymbol{N}^{*T} \frac{\boldsymbol{n}}{||\boldsymbol{n}||} \right) \mathrm{d}\Gamma \tag{2}$$

where $\boldsymbol{n}$ is the normal vector to the contact boundary in the deformed configuration and $\boldsymbol{N}^*$ is the extended N-matrix: a matrix serving to distribute the forces to the according degrees of freedom of both the external node and the nodes of the element forming the contact surface (see [1] for a more detailed discussion of the concept).

Under the assumption of small strains, the normal vector and the extended N-matrix would remain independent of $\boldsymbol{u}$, giving with the second variation of contact energy a simple formula for the algorithmic stiffness contribution [1, 2]

$$\boldsymbol{K}_c = \int_{\Gamma_c} p \left(\boldsymbol{N}^{*T} \frac{\boldsymbol{n}}{||\boldsymbol{n}||} \right) \left(\boldsymbol{N}^{*T} \frac{\boldsymbol{n}}{||\boldsymbol{n}||} \right)^T \mathrm{d}\Gamma \tag{3}$$

However, those assumptions no longer hold with nonlinear geometry, and a more complex stiffness formula has to be derived. Keeping in mind the desired universality of the solution for both 3D and 2D domains, we shall do that in the following paragraph.

2.2. A Universal Stiffness Formula

In expression (2) for the internal forces of contact, the gap function g_c depends on the current deformation state of the domain, with the variation

$$\delta g_c = \boldsymbol{N}_v^T \delta \boldsymbol{d} \tag{4}$$

with $\boldsymbol{N}_v = \boldsymbol{N}^{*T}\boldsymbol{n}/||\boldsymbol{n}||$ and $\delta\boldsymbol{d}$ being the variational displacements of the related degrees of freedom.

Moreover, the extended N-matrix $\boldsymbol{N}^*$ and the unit normal vector $\boldsymbol{\nu} = \boldsymbol{n}/||\boldsymbol{n}||$ depend on the convective natural coordinates $\boldsymbol{\xi}$ denoting the contact point on the element surface. The β-th natural coordinate ξ^β has the variation

$$\delta\xi^\beta = A^{-1}_{\alpha\beta} \left(\boldsymbol{T}_{v,\alpha} - g_c \boldsymbol{B}_{v,\alpha} \right) \delta\boldsymbol{d} = \boldsymbol{D}_{v,\beta} \delta\boldsymbol{d} \tag{5}$$

where:

$$\boldsymbol{B}_{v,\alpha} = \boldsymbol{B}^{*T}_{\alpha} \boldsymbol{\nu} \qquad \boldsymbol{T}_{v,\alpha} = \boldsymbol{N}^{*T} \boldsymbol{t}_\alpha \tag{6}$$

the matrix $\boldsymbol{B}^*_\alpha$ being the derivative of $\boldsymbol{N}^*$ with respect to ξ^α, the vector $\boldsymbol{t}_\alpha$ being the tangent vector to the surface in the α-th direction and $\boldsymbol{A}$ being an extended metric tensor of the surface, composed of the covariant metric tensor $\boldsymbol{m}$ and the curvature tensor $\boldsymbol{\kappa}$ [4]

$$A_{\alpha\beta} = m_{\alpha\beta} - g_N \kappa_{\alpha\beta} \tag{7}$$

Summation convention over α and β (parametric surface coordinate indicies) is implied wherever applicable.

After substitution of these variations into the known expressions and exploitation of the orthogonality of the normal and tangent vectors, we can express the desired tangential stiffness as

$$\begin{aligned} \boldsymbol{K}_c = \int_{\Gamma_c} & \quad p \boldsymbol{N}_v^T \boldsymbol{N}_v \\ & + \; p g_c \left(\boldsymbol{B}_{v,\alpha} \boldsymbol{D}^T_{v,\alpha} + \boldsymbol{D}_{v,\alpha} \boldsymbol{B}^T_{v,\alpha} \right. \\ & + \left. \kappa_{\alpha\beta} \boldsymbol{D}_{v,\beta} \boldsymbol{D}^T_{v,\alpha} + g_c m^{\alpha\beta} \bar{\boldsymbol{B}}_{v,\alpha} \bar{\boldsymbol{B}}^T_{v,\beta} \right) \mathrm{d}\Gamma \end{aligned} \tag{8}$$

where

$$\bar{\boldsymbol{B}}_{v,\alpha} = \boldsymbol{B}_{v,\alpha} + \kappa_{\alpha\beta} \boldsymbol{D}_{v,\beta} \tag{9}$$

and $m^{\alpha\beta}$ is the contravariant metric tensor (i.e. $\boldsymbol{m}^{-1}$). Note that the first term in equation (8) corresponds perfectly to the geometrically linear formula (3).

This formula has been derived with the express aim to make no assumptions about the dimension of the domain space or the number of nodes on the element boundary. It is, therefore, a universal formula for all cases of nonfrictional contact. Letting α and β only assume the value of 1, setting $\boldsymbol{m}$ to element length squared and $\boldsymbol{\kappa}$ to 0, an expression for a linear element in 2D space is obtained, which reduces perfectly to the stiffness formula found for this case in [4] and used in [2].

3. Implementation

This section shall discuss the precise intricacies of the chosen approach to implementing the contact algorithms.

3.1. The Contact Condition and Contact Segment Objects

Contact usually occurs among two bodies within the domain space, of which at least one is deformable and described in FEM by a mesh of finite elements. By definition, when the contact gap is open, no solid matter occupies the area between the two bodies, and therefore no mesh is present.

From a practical programmatic perspective, an essential part of any FEM algorithm is the assembly process. In this process, all the elements within the domain space contribute to the global stiffness matrix and the global vector of internal forces, assembling their contributions to their proper places.

In the case of contact, it is unclear how to introduce contact forces and resulting algorithmic stiffness into this process. In literature [4, 5, 7], it is customary to introduce so-called contact elements. Those are finite elements occupying the physically empty space in between the contacting objects. Those elements perform the aforementioned force and stiffness assembly function. Therefore, the gap area in the reference configuration has to be meshed with those special elements describing the contact algorithm.

This approach has several noticeable drawbacks. Firstly, in the node-to-segment approach, for example, a separate contact element has to be formulated for

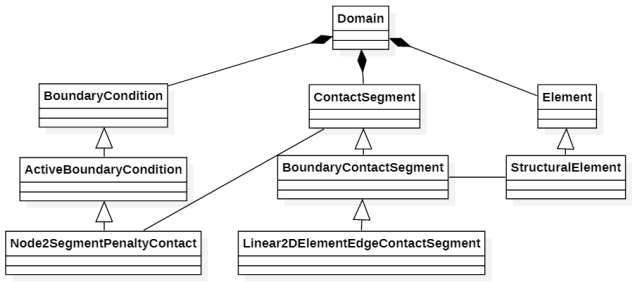

FIGURE 1. A UML diagram describing the concept of object relationships within OOFEM.

each possible case of element edge or surface forming the segment (in the segment-to-segment approach, matters would be even much worse). For a codebase with a reasonable claim to universality, this could shortly result in a significant increase of necessary code.

Secondly, the formulation of those elements may be restrictive in terms of the possible contact cases. A contact element usually only pairs one node with one boundary segment. Each node deemed possible to come in contact with a given segment, therefore, requires a separate element defined, and this has to be decided already at the time of meshing.

Already in the existing implementation in OOFEM, we have diverged from this standard practice. In line with object-oriented programming principles, the aforementioned assembly process in OOFEM is handled with the use of objects. Apart from object classes representing elements, another class capable of contributing to the global stiffness matrix and force vectors is the `ActiveBoundaryCondition` class. We have chosen to handle contact algorithms through subclasses descended from that.

The principle is illustrated diagrammatically in Figure 1. All OOFEM objects are collectively stored in the overarching `Domain` class. A `Node2SegmentPenaltyContact` object is a subtype of an `ActiveBoundaryCondition`, which itself is a subtype of a `BoundaryCondition`. This contact condition class maintains a generalized reference to a `ContactSegment` object, which, in the case it is of the `BoundaryContactSegment` subtype, can reference element objects relevant for its task of describing an element boundary.

This way of structuring the code offers us a great potential of generality in large parts of the code and presents a more intuitive way of modeling the real physical idea of the contact phenomenon.

3.2. Universality of the Contact Condition Object

In our previous work, four boundary condition objects have been created, namely `Node2NodePenaltyContact`, `Node2NodeLagrangianMultiplierContact`, `Node2SegmentPenaltyContact`, and `Node2SegmentLagrangianMultiplierContact` [1, 2]. In the case of the node-to-node approach, the contact condition object itself handles the references to the nodes concerned, requiring no interaction with any other objects. Its universality and extension to the 3D domain is therefore trivial and not a subject of this paper.

In the node-to-segment case, however, the operations within the code rely heavily on an interaction between the condition and an object representing the contact segment. As described by Figure 1, we aimed to maintain the connection of the boundary condition class only to the generalized `ContactSegment` parent class, rather than differentiating among its subtypes, which include formulations as diverse as an analytical polynomial function in 2D space and a segment representing multiple 3D element surfaces.

The formulas necessary to this universal approach have already been discussed in section 2.2. The only necessary part of the implementation part is to ensure the ability of the code to handle all possible matrix sizes of the various contributions. Extensive use has been made of the `std::list` array object in the C++ language, which allows for an unspecified number of objects to be passed between C++ functions. To this end, the functions in the 2D contact segment have been altered to return an array of size one (of, i.e., tangent vectors) as opposed to a single object. On the level of the `Node2SegmentPenaltyContact` class, the code is written to always loop through those arrays regardless of their size.

Key to the computations in the `Node2SegmentPenaltyContact` class are the `computeTangentFromContact()` and `computeExternalForcesFromContact()` methods, which compute the stiffness and force contributions, respectively, for a given node-segment pair. In addition to those, auxiliary methods exist to compute the various expressions seen in formula 8, e.g. the `computeNvMatrixAt()` method for the $\boldsymbol{N}_v$ matrix or the `computeModifiedBvMatricesAt()` method for the (one or multiple) $\bar{\boldsymbol{B}}_v$ matrices. Those then poll the contact segment for the relevant information, such as gap value, metric and curvature tensor, normal and tangent vector(s), the $\boldsymbol{N}$ matrix etc.

3.3. The universality of Contact Search Algorithm

Another issue with code universality arises specifically in the 3D domain. Contrary to the 2D case, in 3D, even the limitation to finite elements with linear formulation does not necessarily ensure the linearity of their boundary [9]. More specifically, only surfaces of triangular shape can be described linearly. We have come to the conclusion that limiting our implementation to those would severely hamper the usability of the resulting code; it was, therefore, necessary to take the possible nonlinearity of the contact surface into account when conducting a contact point search. The solution

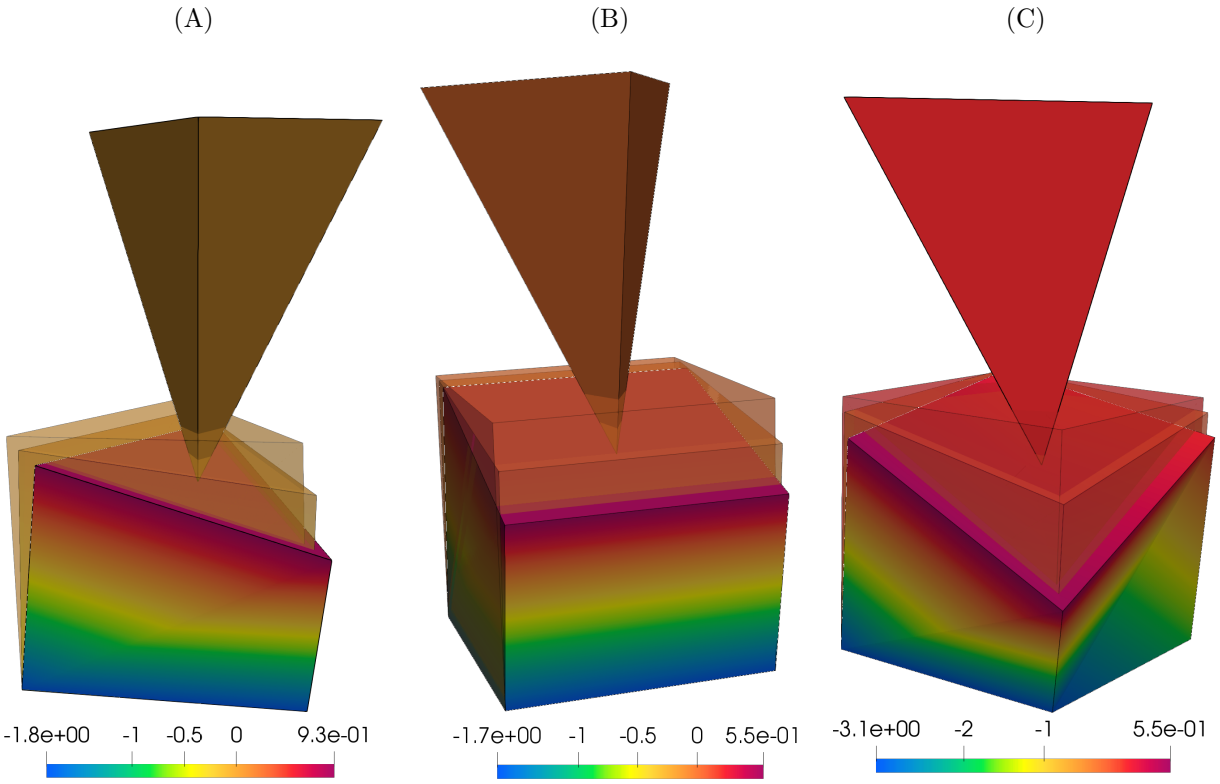

Figure 2. Node-to-segment contact with penalty formulation in the 3D domain: tests on a linear wedge and a linear brick element. Displayed geometries correspond to the 0th (undeformed configuration), 40th, and 60th steps of the computation. Coloring represents the vertical normal stress σ_{yy}. A) A test on a linear wedge element. B) A test on a linear brick element, with the initial contact point at an axis of symmetry of the top surface. The deformed state remains symmetric along this axis. C) A test on a linear brick element with an arbitrary initial contact point. The apparent gap is only a consequence of the imprecise rendering of the resulting bilinear surface.

is an implementation of a closest-point-projection algorithm in the `computeContactPoint()` method of the `Linear3DElementSurfaceContactSegment` class.

The parametric coordinates of the desired contact point (i.e., the point on the surface closest to the contacting node, regardless of whether contact occurs) can be expressed as a vector $\boldsymbol{\xi}$ minimizing the distance function

$$F(\boldsymbol{\xi}) = \frac{1}{2}(\boldsymbol{r} - \boldsymbol{\rho}(\boldsymbol{\xi})) \cdot (\boldsymbol{r} - \boldsymbol{\rho}(\boldsymbol{\xi})) \tag{10}$$

where $\boldsymbol{r}$ are the global coordinates of an external point (the node), and $\boldsymbol{\rho}$ are the global coordinates of the contact point (on the surface).

Solving this minimization problem with the Newton-Raphson scheme leads to a convenient expression for the iterative increment $\delta\boldsymbol{\xi}$ in the form

$$\delta\boldsymbol{\xi} = -\boldsymbol{A}^{-1}\frac{dF}{d\boldsymbol{\xi}} \tag{11}$$

where $\boldsymbol{A}$ is the already well-established extended metric tensor of the contact surface (see (7)).

4. Computational Examples

In light of the original aim of this paper, the extension of OOFEM's contact formulations to the 3D domain, we shall demonstrate the functionality of the described implementation on several 3D test examples.

The tests concern the node-to-segment formulations for geometrically nonlinear cases. In one case, the contact segment is made of a triangular surface of a linearly formulated wedge element, while in the other two cases, the used element is a linear brick. Therefore, the first element represents an example of a fully linear surface without curvature, while the latter concerns a bi-linear surface, where the curvature tensor is generally nonzero. On the brick element, two different tests are performed; in the first one, the initial contact point is positioned on one of the axes of symmetry of the surface, in the other one, an arbitrary point is chosen.

The elements in question are the OOFEM elements `LWedge` and `LSpace`, respectively. Above them, a single `LTRSpace` element is positioned. This is a linear tetrahedral element, which serves, however, only to provide the contacting node. All its nodes are fixed, and the lower node is loaded by a prescribed vertical displacement. This node is designed to come into contact with the top surface of the lower element and push it downwards. All nodes of the contact surface are free; the lower element is fixed by Dirichlet boundary conditions on its lower base.

All elements in these test simulations have a linear

elastic material model assigned, with Young's modulus and Poisson's ratio $E = 500\,\mathrm{MPa}$ and $\nu = 0$. In OOFEM, such a material model translates to the Saint Venant-Kirchhoff law in the case the geometrical nonlinearity of the element is turned on [6].

Loading is performed in 60 steps in each case. The total displacement prescribed for the external node is 0.3 m, i.e. 0.005 m per loading step.

The results of the test are pictured in Figure 2. For the visualization of the results, ParaView, an open-source software package, has been used [10].

We can observe that the deformation of the surface follows the positioning of the contact point. For the brick element, in the first case, the contact point remains on the axis of symmetry, and the rotation of the contact surface only follows the other axis, while in the second case, the entire surface twists and assumes a bi-linear deformed shape. Note that the somewhat apparent gap between the element surface and the contacting node is only a result of the rendering software's inability to portray this bi-linear surface properly.

5. Conclusions

The paper describes an object-oriented approach to handling contact mechanics in FEM. As a result of this object-oriented approach, a universal contact algorithm is necessary. This is derived on a theoretical basis, and the implementation in an open-source finite element code OOFEM is then discussed in detail. To verify the practical usability of the chosen approach, tests are performed on contact in the 3D domain, which is a feature previously unavailable in the existing OOFEM contact implementation.

Acknowledgements

The authors would like to acknowledge the support received for work on this project from a CTU grant no. SGS21/037/OHK1/1T/11.

References

[1] O. Faltus. *Object-Oriented Design and Implementation of Contact Mechanics into Finite Element Code OOFEM*. Master's thesis, Czech Technical University in Prague, 2020.

[2] O. Faltus, M. Horák. Finite element implementation of geometrically nonlinear contact. *Acta Polytechnica CTU Proceedings* **30**:18–23, 2021.

[3] H. Hertz. On the contact of elastic solids. *Z Reine Angew Mathematik* **92**:156–171, 1881.

[4] P. Wriggers. *Computational contact mechanics*. Springer, New York, 2nd edn., c2006.

[5] V. A. Yastrebov. *Numerical methods in contact mechanics*. Wiley, Hoboken, NJ, 2013.

[6] B. Patzák. OOFEM home page, 2000. Http://www.oofem.org.

[7] A. Konyukhov, R. Izi. *Introduction to computational contact mechanics*. Wiley, Chichester, West Sussex, 2015.

[8] T. A. Laursen. *Computational contact and impact mechanics: fundamentals of modeling interfacial phenomena in nonlinear finite element analysis*. Springer Science & Business Media, 2003.

[9] O. C. Zienkiewicz, R. L. Taylor. *The finite element method*. Butterworth-Heinemann, Boston, 5th edn., 2000.

[10] U. Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware, 1st edn., 2015.