Intersection of non-matching grids of different dimensions

Jan Březina[a,*], Pavel Exner[a]

[a] *Technical University of Liberec, Studentská 1402/2, 461 17 Liberec 1, Czech Republic*

Abstract

TODO: An abstract.

Keywords: non-matching gird, intersections, mixed-dimensional mesh, Plückercoordinates

1. Introduction

Structure of introduction:

- motivation for computing mesh intersections, other applications - basic overview of existing work, approaches - our contribution - new algorithms for element intersectins based on Plücker coordinates - improvement of the front tracking algorithm

[**?**] - Use *g*eometric predicates incircle and orientation. Use mesh of bounding boxes to search for intersection candidates. Only use point queries for all vertices of triangle/tetrahedron. Use compatible partitioning of elements by the crack elements.

[**?**] - Implementation of the Nitche method in Fenics. They need 2d-3d intersections, but only for distribution od quadrature points. So they do not store whole intersection data, but only the quad points. The boundary of one domain is partitioned into intersections with elements of the other domain. Use emph collision relations (pairs of faces of boundary and elmements of the background mesh) and *c*ollision maps (map face to all intersections and background element to all intersections, can be constructed from the collision relation), They cite books from computer graphics .. Mention geometric predicates. Present algorithms for: find intersection candidates, comput intersections, integrate over complex domains.

[**?**] Claim to provide robust algorithm covering all tiangle-trinagle degenerate cases. Basic idea is the same as our: use some tree (Binary SPace Tree) structure to find initial point on intersection curve and trace the curve in both

*Corresponding author.

Email addresses: jan.brezina@tul.cz (Jan Březina), pavel.exner@tul.cz (Pavel Exner)

directions. Quite clever data structure for the mesh. Adaptive precision geometric predicates. Store topological information about the intersection points (point on triangle, edge, vertex). Resolve various degenerate cases, dicuss mesh optimization after subdivision.

[**?**] Full algorithm for 2d=3D called PANG. Implemented in DUNE in 2010 (Bastian). Other software (Hecht, MpCCI). Triangle-Triangle and Tetra-Triangle intersections done similarly as in aur approach: set of points including vertices and then sort them countercolckwise. Local intersection algorithm do not provide neighbouring information so the traversal algorithm do not use it.

Review of algorithms: [**?**]

[**?**] Robust discretization of porous media. Complete error analysis with mortar non-compatible case. Seems they use normal fluxes accros fracture instead of pressure traces. This allows conforming discretization and error results.

[**?**] Application of mixed-dimensional approach in mechanics. Beam and shell elements ...

[**?**] Finite volume method, fully implicit two phase flow. Cite usage of FRAC3D generator.

[**?**] Space-time DG with adaptive mesh, need integration of product of functions on different meshes (in 2D). An algorithm for computing triangle-triangle intersections in 2D.

[**?**] DFN + MHFEM. Mortar method for intersections. Focused on formulation not on intersections.

[**?**] (Barbara) Mortad, non=matching, acustics. Theory, structured gird case. No complex intersections.

[**?**] Patent for curiosity.

[**?**] Immersed boundary. ??

[**?**] ??

[**?**]

[**?**] Original algorithm for line-thetrahedra intersections.

[**?**] Line-convex Polyhedra intersections using generalisation of line-clipping algorithm

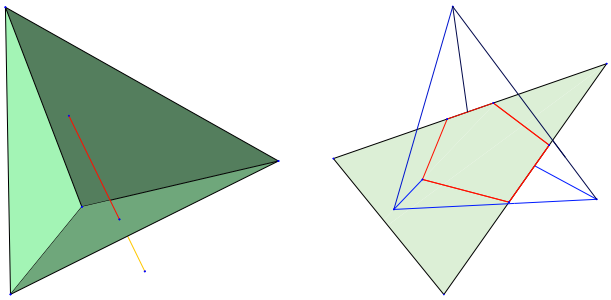
The initial problem that occurs when working with incompatible meshes of combined dimensions is the efficient computation of intersections of the mesh subdomains, in particular 1D-2D, 1D-3D, 2D-3D. There is a piece of code in Flow123d at the moment which uses means of analytic geometry. In particular, it computes an intersection of a line and a plane in 3D by solving Gaussian elimination. However it is not effiecient and it does not provide barycentric coordinates explicitly. In this section, we present our latest work on this problematics.

The topic is studied in the bachelor and master thesis of Viktor Friš [**?**], who was supervised by Jan Březina. They suggested a solution, based on [**?**] for the two aspects of the problem:

itemsep=-3pt an efficient identification of the intersecting pairs of elements,

iitemsep=-3pt an efficient computation of the actual intersection for the single element pair.

The algorithm is based on the methods used in the computer graphics, in particular for the task (b) Plückercoordinates are employed to efficiently compute 1D-2D intersection in three-dimensional space.



(a) 1D-3D (b) 2D-3D

For the 1D-3D and 2D-3D cases, the basic algorithm is applied on sides and edges of the simplices and all the topological information coming from the performed Plückerproducts is carefully collected to obtain final intersecting object. Further in 2D-3D case, an optimized tracing algorithm is suggested to directly obtain the correct order and orientation of the edges of the intersection polygon.

1.1. Element Intersections

In this section, we present algorithms for computing intersection of a pair of simpicial elements of different dimension in the 3D ambient space.

1.2. Plückercoordinates

Plückercoordinates represent a line in 3D space. Considering a line p, given by a point $\boldsymbol{A}$ and its directional vector $\boldsymbol{u}$, the Plückercoordinates of p are defined as

$$\pi_p = (\boldsymbol{u}_p, \boldsymbol{v}_p) = (\boldsymbol{u}_p, \boldsymbol{u}_p \times A).$$

Further we use a permuted inner product

$$\pi_p \odot \pi_q = \boldsymbol{u}_p \cdot \boldsymbol{v}_q + \boldsymbol{u}_q \cdot \boldsymbol{v}_p.$$

The sign of the inner product product gives us the relative position of the two lines, see Figure 1.

1.3. Intersection Line-Triangle (1D-2D)

Let us consider a line segment p with end points $\boldsymbol{A}$ and $\boldsymbol{A}+\boldsymbol{u}$ and a triangle $(\boldsymbol{V}_0, \boldsymbol{V}_1, \boldsymbol{V}_2)$ with oriented sides $s_i = (V_j, V_k)$, $j = (i+1) mod 3$, $k = (i+2) mod 3$. The inner products $\pi_p \odot \pi_{s_i}$, $i = 0, 1, 2$ have the same sign if and only if there is an intersection point inside the triangle. In such a case, the inner products

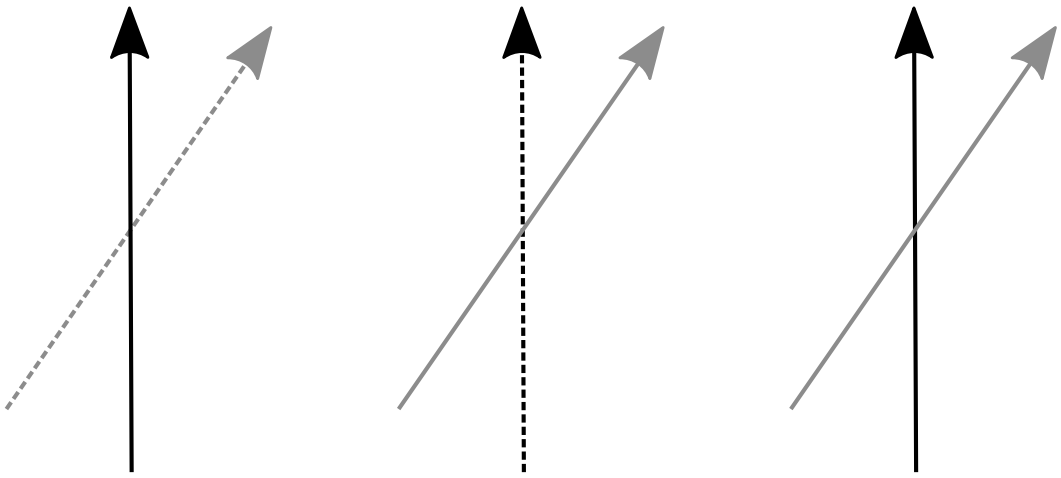


Figure 1: Sign of the Plückerproduct (from [**?**]). Dashed line symbolizes that the line is in the back. In the case (c), product is zero which means the lines intersect.

provides scaled barycentric coordinates of the intersection on the triangle. In particular for the barycentric coordinate w_i related to the vertex V_i we have

$$w_i = \frac{\pi_p \odot \pi_{s_i}}{\sum j = 0^2 \pi_p \odot \pi_{s_i}}$$

Indeed, using the barycentric coordinates the intersection point can be expressed as $\boldsymbol{X} = \boldsymbol{V}_0 + w_1\boldsymbol{s}_2 - w_2\boldsymbol{s}_1$. The line p have Plückercoordinates $(\boldsymbol{u}, \boldsymbol{u} \times \boldsymbol{X})$ (Plückercoordinates are invariant to change of initial point). Combining these two expressions we get

$$\pi_p \odot \pi_{s_1} = \boldsymbol{u} \cdot (\boldsymbol{s}_1 \times V_2) + \boldsymbol{s}_1 \cdot (\boldsymbol{u} \times [\boldsymbol{V}_0 + w_1\boldsymbol{s}_2 - w_2\boldsymbol{s}_1]) = -w_1\boldsymbol{u} \cdot (\boldsymbol{s}_1 \times \boldsymbol{s}_2).$$

Since $\boldsymbol{s}_0 + \boldsymbol{s}_1 + \boldsymbol{s}_2 = 0$ we have $\boldsymbol{s}_1 \times \boldsymbol{s}_2 = \boldsymbol{s}_2 \times s_0 = \boldsymbol{s}_0 \times \boldsymbol{s}_1$ and thus

$$\pi_p \odot \pi_{s_i} = -w_i\boldsymbol{u} \cdot (\boldsymbol{v}_1 \times \boldsymbol{v}_2).$$

Having the barycentric coordinates of the intersection on the triangle we can compute also its local coordinate on p from its parametric form:

$$X_i = A_i + tu_i, \text{ for } i = 1, 2, 3$$

In practical computation we consider i with maximal $|u_i|$.

Next, we shell discuss the case when some of the permuted inner products is zero. Various $p$athological cases are detected and further topological information is atached to the intersection data. In fact we test if a barycentric coordinate

is under given tolerance ($|w_i| \leq \epsilon$). In the case of one zero coordinate (non-coplanar intersection on the edge) or two zero coordinates, we add the local edge index or the local vertex index to the intersection, respectively. When all coordinates are (close to) zero, the line and the triangle are (nearly) coplanar. For every edge s_i we compute its intersection with the line segment p in terms of the local coordinates t_i and t_p. Only intersections with $t_i \in (-\epsilon, 1+\epsilon)$ are accepted, for t_i close to 0 and 1 we append the vertex index to the intersection. Edges parallel to p are skipped. Finally, we obtain zero, one, or two intersection points with additional topological information. Degenerated case of p laying on the triangle edge is also detected....

Starting with 1D-2D intersecting simplices, all the Plückercoordinates and products must be computed at first. In case that all Plückerproducts are nonzero and the intersection point exists, its barycentric coordinates can be obtained directly. Otherwise one of the pathologic cases occurs

itemsep=-3pt 1 zero product: line intersects triangle side,

itemsep=-3pt 2 zero products: line intersects triangle at its vertex,

itemsep=-3pt 3 zero products: line intersects triangle at its vertex and:

itemsep=-3pt intersects also the opposite side,
iitemsep=-3pt is parallel to the opposite side,
iiitemsep=-3pt intersects also another triangle vertex.

Anyway, in the pathologic case, the intersection of a line and triangle side is solved analyticaly which leads to a system of two equations with three unknowns. This is then solved using Crammer's rule.

So far, we considered the line and the triangle sides as bisectors. If the 1D-2D intersection is the final object, we check that the found intersection point lies on the abscissas. Otherwise all the computed data are used for further computation. Apart from the coordinates of the intersection points, we also save temporarily its topology information (vertex, side or edge index).

1D-3D (abscissa-tetrahedron). Tetrahedron has six edges, so six Plückercoordinates and products are computed at most. These data are passed to 1D-2D algorithm described above, which is then run for each pair line – tetrahedron side. There might be two intersection points maximally, which are finally checked that they belong o abscissas. If the line has its boundary point inside the tetrahedron, the intersection line must be cut and coordinates of intersection point are interpolated. Pathologic cases are again carefully processed so we obtain the most extensive topology information we can. See [**?**] for detailed description of all possible cases.

2D-3D (abscissa-tetrahedron). The intersection of a triangle and a tetrahedron is a polygon with 7 vertices at maximum. The vertices are found as intersection points of either triangle side and tetrahedron or tetrahedron edge and

triangle. Therefore we use both algorithms above for 1D-3D and 1D-2D, respectively. Data are again efficiently passed to lower dimensional problems, so 9 Plückercoordinates and 18 Plückerproducts are computed.

The array of intersection points is generally not sorted. We use two so called *tracing* algorithms and we intend to orient the edges of the polygon in the same direction as the triangle is oriented. If one of the intersection point is pathologic, a general convex hull method is applied using the Monotone chain[1] algorithm. The points are sorted using only their barycentric coordinates.

An optimized algorithm has been suggested for non-pathologic cases. At this moment all the collected topology data come into play. The algorithm takes advantage by using only the data already computed and also lowers the complexity to $O(N)$, compared with the Monotone chain complexity $O(N \log N)$ (N being number of intersection points).

1.4. Prolongation algorithm

Consider now a complex mesh of combined dimensions consisting of *components*, which are sets of connected elements of the same dimension (1D, 2D), in the space of 3D elements (tetrahedrons). Obtaining of all the 1D-3D and 2D-3D intersections is based on finding the first two elements intersecting each other. Then we can prolongate the intersection by investigating neighbouring elements of the component.

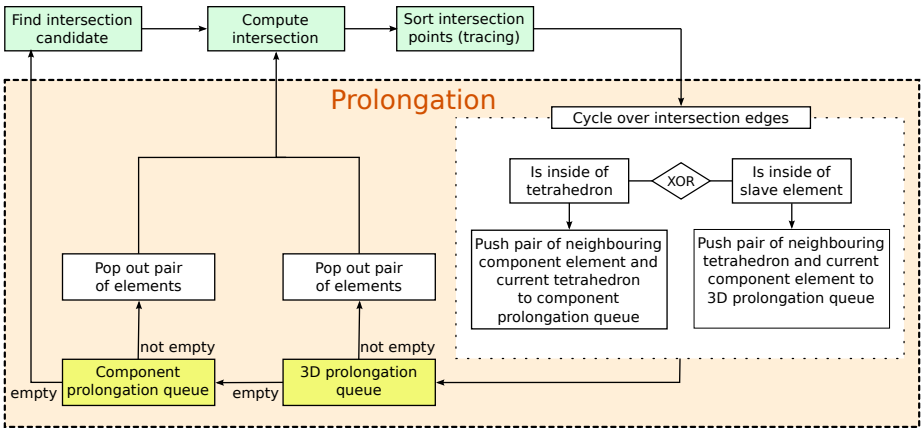


Figure 2: Prolongation algorithm for 1D-2D and 2D-3D intersections.

The prolongation algorithm is the same for both 1D and 2D components, see Figure 2. It can be seen as a *breadth-first search* [2] algorithm on the graph of 1D and 2D elements, respectively.

[1] Wikibooks, [online 2016-03-01], http://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain

[2] Wiki, [online 2016-03-01], https://en.wikipedia.org/wiki/Breadth-first_search

After computing the intersection of a pair of elements (line or triangle vs tetrahedron), we fill two queues with element pairs as candidates for further intersection. If the intersection edge (point of line in 1D, edge of polygon in 2D) is inside the tetrahedron, not on its surface, we get a neighbouring element of the component and push it back together with the current tetrahedron into *component prolongation queue*. If the intersection edge is inside the *slave* element (line or triangle), i.e. is on the surface of tetrahedron, we get a neighbouring element of the tetrahedron and push it back together with the current slave element into *3D prolongation queue*.

Then we empty the prolongation queues – the 3D prolongation queue at first, then the component prolongation queue. When both queues are empty, all intersections of a component have been found and we continue to look for another component.

The algorithm is now unified for 1D and 2D in contrast to [**?**], where the component prolongation queue is emptied at first.

1.4.1. 1D-3D prolongation

PE: *I suggest to denote IP, IPs intersection point and its plural respectively.*

A new candidate pair of elements is found during prologantion, based on the topological information of the intersection point. There are 3 possible cases:

- **IP lies at 1D element node and inside 3D element**
  We find the neighboring 1D element over the node and push a new candidate pair [1D neighbor – current 3D element] into component queue.
- **IP lies at 1D element node and on the surface of 3D element**
  We find all the faces of 3D element in which the IP lies (1 face, or 2 faces (IP on an edge), or 3 faces (IP at a node)). We find the corresponding neghboring 3D elements over the faces and push the following new candidates pairs into the bulk queue: [current 1D element – 3D neighbor], [1D neighbor – 3D neighbor]. If the candidates pair has been investigated already, we skip it.
- **IP lies inside 1D element node (therefore must be on the surface of 3D element)**
  The same as before, but we push only [current 1D element – 3D neighbor] candidates pairs, since there is no 1D neighbor.

2. Front tracking algorithm

3. Benchmarks

4. Conclusions

TODO: - line intersection tracking for accelerate 2D-2D intersections - better handling of special cases in paarticular in relation to prolongations - better calculation reuse (pass with prolongations) - optimisation of element intersection - skip unnecessary calculations

5. Acknowledgement

The paper was supported in part by the Project OP VaVpI Centre for Nanomaterials, Advanced Technologies and Innovations CZ.1.05/2.1.00/01.0005.