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, Plücker coordinates

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

[1] - 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.

[2] - 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.

[3] 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.

[4] 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: [5]

[6] 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.

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

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

[9] 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.

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

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

[12] Patent for curiosity.

[13] Immersed boundary. ??

[14] ??

[15]

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

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

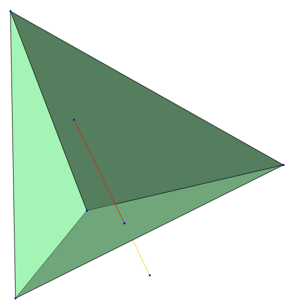
The initial problem that occurs when working with incompatible meshes of combined dimensions is the efficient computation of intersections of the mesh subdomains, in particular 1D-2D, 1D-3D, 2D-3D. There is a piece of code in Flow123d at the moment which uses means of analytic geometry. In particular, 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.

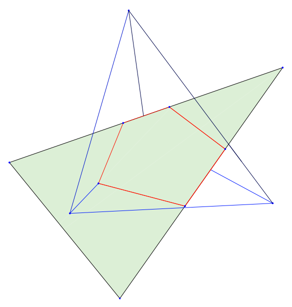
The topic is studied in the bachelor and master thesis of Viktor Friš [**?**], who was supervised by Jan Březina. They suggested a solution, based on [**?**] for the two aspects of the problem:

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) Plücker coordinates 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 Plücker products 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.

2. Element Intersections

In this section, we present algorithms for computing intersection of a pair of simplicial elements of a different dimension in the 3D ambient space. In particular we are interested in the intersections in 1D-2D, 1D-3D, 2D-2D, 2D-3D cases. It will be suitable to denote S_i a simplicial element with i vertices (of dimension $i-1$). In general, an intersection can be a point, a line segment or a polygon. All these three objects are represented as a vector of vertices called *intersection points* (later denoted IP, or IPs in plural).

The fundamental idea is to compute intersection of 1D-2D simplices using the Plücker coordinates and reduce all other cases to this one.

2.1. Plücker Coordinates

Plücker coordinates represent a line in 3D space. Considering a line p, given by a point $\boldsymbol{A}$ and its directional vector $\boldsymbol{u}$, the Plücker coordinates 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 permuted inner product gives us the relative position of the two lines, see Figure 1.

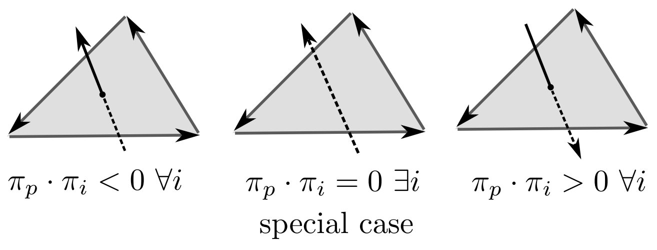

$\pi_p \cdot \pi_i < 0 \; \forall i$
$\pi_p \cdot \pi_i = 0 \; \exists i$
special case
$\pi_p \cdot \pi_i > 0 \; \forall i$

Figure 1: Sign of the permuted inner product is related to the relative position of the two oriented lines. Dashed line symbolizes that the line is in the back, the lines intersect in the middle case. **PE:** *circle dot permuted inner product*

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

PE: *In this paragraph we investigate real lines. Trimming comes later when tolerance is denoted, see next PE note.* Let us consider a line p with parametric equation $\boldsymbol{X} = \boldsymbol{A} + t\boldsymbol{u}$, $t \in \mathbf{R}$ and a triangle given by vertices $(\boldsymbol{V}_0, \boldsymbol{V}_1, \boldsymbol{V}_2)$ with oriented sides $s_i = (\boldsymbol{V}_j, \boldsymbol{V}_k)$, $j = (i+1) \bmod 3$, $k = (i+2) \bmod 3$. The permuted 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 with the line p inside the triangle. In such a case, the inner products provides scaled barycentric coordinates of the intersection on the triangle. In particular for the barycentric coordinate w_i related to the vertex $\boldsymbol{V}_i$ 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 Plücker coordinates $(\boldsymbol{u}, \boldsymbol{u} \times \boldsymbol{X})$ (Plücker coordinates are invariant to change of initial point). Combining these two expressions we get

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

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

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

We use i with maximal $|u_i|$ for practical computation.

Next, we shall discuss the case when some of the permuted inner products are zero. In fact, we test if any barycentric coordinate is under given tolerance ($|w_i| \leq \epsilon$). Various *pathological* cases are detected and further topological information is atached to the intersection data. In the case of one zero coordinate (non-coplanar intersection on the edge) or two zero coordinates (intersection in the vertex), 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 or lying on the p are skipped.

Finally, we obtain zero, one, or two intersection points with additional topological information. The intersection point data structure keeps for each of the two intersecting elements:

- Barycentric coordinate of the intersection.
- Dimension of the most specific subelement (face, edge, vertex) where the point lies.
- Local index of the most specific subelement.

Further, we store the sign of the permuted inner product in order to detect various special and degenerate cases higher dimensional cases.

TODO: We should distinguish "special" case and "degenerate" case. The later one is the case when line lies in the plane of the triangle. The first case should be treated as usual only set aditional topology info about vertex or edge. The degenerate case is not necessary to investigate in the case of auxiliary intersection only in the case of final intersection. Need to verify it since it is not the case in current code.

We distinguish two variants of the desired result: the auxiliary intersection used as subprocedure of the other intersections and the final intersection. In the first case we just return the array of the intersection points with the line. **PE:** *In the first case, we do not trim the line (We need both IPs (if there are two), for 1D-3D). In the final case, we need to check that also $t_p \in (-\epsilon, 1+\epsilon)$. That is the current state.* Degenerated case of p laying on the triangle edge is also detected.... **JB:** *No, in 1D-3D we doesn't need result of the degenerate case (all products zero), which is the only case we can obtain two IPs. I'm going to do not deal with the two variants and focus more on correctness then on optimality of the described algorithm.*

Intersection with line segment ... (final 1D-2D intersection)

2.3. Intersection Line-Teraherdon (1D-3D)

JB: *Has to be completed yet, however we consider that we obtain 0,1, or 2 distinguish IPs for all cases including various touch cases. Anyway every IP results from some non-degenerate 1d-2d intersection. We also assume that the IPs are in order according to the orientation of the side which is counterclockwise aroung the triangle.* We consider a line p and a tetraherdon... Tetrahedron has six edges, so six Plücker coordinates and inner products are computed at most.

These data are passed to the 1D-2D algorithm described above, which is applied to the every face of the tetrahedron face.

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.

2.4. Intersection Triangle-Tetrahedron (2D-3D)

The intersection of a triangle S_3 and a tetrahedron S_4 is an n-side polygon, $n \leq 7$. The sides of the polygon lie either on sides of S_3 or on faces of S_4. Thus each vertex (IP) of the polygon can arise either from side-face intersection, or from edge-triangle intersection, or be a vertex of S_3. So we have to compute at most 12 side-face intersections and at most 6 edge-triangle intersections. However, to this end we only need to compute 9 Plücker coordinates (3 sides, 6 edges) and 18 permuted inner produts, one for every side-edge pair. Computation of the intersection polygon consists of two parts: calculation of side-tetrahedron IPs (Algorithm 1) provides all IPs on the boundary of S_3, calculation of edge-triangle IPs (Algorithm 2). **PE:** *The following sentence is somehow weird.. I would say that we store IPs I; to sort I during the computation, we use the two tables; at the end, I is ordered correctly.* Correct order of the IPs, stored in the list I, is defined by the *connection tables* $F_g(:)$ and $F_p(:)$. **PE:** *The following sentece is a copy of the second sentence is this subsection.* Every side of the intersection polygon lies either on a side of S_3, or on a face, or on an edge of S_4. Let us denote M_3 the set of sides of S_3 and M_4 the set of volume, faces, edges, and vertices of S_4. Every side of polygon that lies on $x \in M_3 \cup M_4$ is followed by an IP given by $F_g[x]$ and every IP p is followed by the side that lies on $F_p[p] \in M_3 \cup M_4$.

Algorithm 1 passes through every side s of the triangle S_3 and computes the line-tetrahedron intersection L. In the regular case we process each of the two IPs in L. The IP p is added to the list I unless the last point is the same. This condition is effective just for the first IP in L and merges IPs at the same vertex of S_3. Then we identify an object $x \in M_4$ the point p lies on (line 7). **PE:** *I am not sure, if the term 'object' is clear..*

PE: *Why calling it vertex v instead of IP p here? What is y?* If the vertex v is between sides s_1, s_2, we effectively set connections $F_g[s_1] = v$, $F_g[y] = v$, $F_p[v] = s_2$. The backward temporary link on the line 13 is used in Algorithm 2 to fix $F_p[p]$ for the second p lying on an edge or a vertex of S_4. The condition at the end deals with the case in which two sides intersect the same object $x \in M_4$.

For the regular intersection L, we pass through its IPs, as described above. **PE:** *For special cases*, we firstly compute IPs on the boundary of S_3 using the line-tetrahedron intersection algorithm for every side. We store single point intersections into separate list J and skip filling the connection tables. This happens when the side touches S_4 at its edge or vertex. These IPs will be rediscovered again in Algorithm 2 with better topological information, however

Algorithm 1: 2d-3d intersection, points on triangle boundary

```
Input: input data
Output: List of IPs on sorted output data
$F_g(:) = -1,\ F_p(:) = -1$                                // Unset links.
for side $s$ of $S_3$ do
    $L$ = intersection$(s, S_4)$
    if $|L| = 0$ then continue
    if $|L| = 1$ then append $p$ to $J$ continue
    for $p$ in $L$ do
        if $p \neq I[-1]$ then append $p$ to $I$
        $p$ lies on $x \in M_4$
        if $p$ is first in $L$ then
            $F_g[x] = p,\ F_p[p] = s$
        else $p$ is the last in $L$
            $F_g[s] = p,\ F_p[p] = x$
            if $F_g[x] = -1$ then $F_g[x] = p$
if $I[-1] = I[0]$ then
    $x = F_p[I[-1]]$
    if $F_g[x]] = I[-1]$ then
        $F_g[x] = I[0]$
    $F_g[s_2] = I[0]$
    remove $I[-1]$
```

this is not the case if the touched edge e of S_4 is coplanar with S_3 and the IP is inside of e. Therefore we keep a separate list J of IPs to deal with this case.

```
Algorithm 2: 2d-3d intersection, points in triangle interior
   Input: $I$ with IPs on $S_3$ boundary, partially filled $F$
   Output: all IPs in $I$, complete $F$
 1 for edge $e$ of $S_4$ do $L[e]$ = intersection($e, S_3$)
 2 for unmarked edge $e$ of $S_4$ do
 3     $p = L[e]$
 4     if $p$ is inside of $e$ then
 5         ($f_0$, $f_1$) = edge faces($e$)
 6     else $p$ on the vertex $v$ of $S_4$
 7         ($f_0$, $f_1$) = vertex faces($v$)              // Algorithm 3
 8     if $p$ is on boundary of $S_3$ then
 9         $p$ lies on edge or vertex $x \in M_4$
10         $q = F_g[x]$                        // $q$ is already computed $p$
11         if $F_p[q] = x$ then $F_p[q] = f_1$
12         else $F_g[f_0] = q$
13         $F_g[x] = -1$                          // remove the backlink
14     else
15         append $p$ to $I$
16         $F_g[f_0] = p$                      // overwrites the backlink
17         $F_p[p] = f_1$
18         mark all edges coincident with $p$
19 if $|I| < 3$ then return $J$
20 else return $I$ sorted according to connectivity in $F_g$ and $F_p$
```

Algorithm 2 use the line-triangle intersection algorithm for the edges of S_4 (line 2). The loop produce IPs from interior of S_3 and possibly those already computed IPs with special position on vertex or edge of S_4. Every edge e of S_4 is oriented so that the pair of adjacent faces f_0, f_1 apear in the same order on the intersection polygon when the IP of e have negative sign (see Figure 1). The function "edge faces" used on line 4 and later on use the sign of the intersection to return the pair of faces in the correct order. Similarly the function "vertex faces" (Algorithm 3) described latter on returns pair of generalized faces (face or edge) possibly adjacent to the IP $L[e]$ at the vertex v of S_4.

2.4.1. Vertex Faces Algorithm

This function get an IP p at vertex v of S_4 as a parameter. The IP is special vertex case of non-degenerate edge-triangle intersection. The functin returns pair of generalized faces of S_4 preceeding and succeeding p on the polygons boundary in the case that p is at interior of S_3. Basic idea is to use the signs of IPs of the three edges coincident with v. Possible cases are:

Algorithm 3: 2d-3d intersection, vertex faces

```
Input: vertex $v$ of $S_4$, $L[:]$ intersection results for edges of $S_4$
Output: $(x_1, x_2)$, $x_1, x_2 \in M_4$, coincident with $v$ and intersected by the
        plane of $S_3$
1  $e_0$, $e_1$, $e_2$ edges coincident with $v$ oriented out of $v$ $s[i] = L[e_i]$, for
     $i = 0, 1, 2$,
2  if $s[:]$ have 1 non-degenerate edge $e$ then
3      return pair of degenerate edges sorted according to edge faces $(e)$
4  else if  $s$ have 1 degenerate edge $e$ then
5      $f$ is face oposite to $e$ if other two edges $e_a$, $e_b$ have different sign then
6          $z =$ edge faces$(e_a)$
7          replace $g \in z$, $g \neq f$ with $e$ return $z$
8      else  append IP of $v$ to $J$ return anything
9
10 else if  $s$ have edge $e$ with sign oposite to other two then
11     return edge faces$(e)$
12 else $s$ have all signs same
13     append IP of $v$ to $J$ return anything
```

- **All IPs have same sign.** (line 11) We return any pair of faces. S_3 is touching S_4 in the vertex v, the polygon ddegenerates to the single IP p, no connectionn information from table F necessary.

- **Single IP have sign oposite to the other two.** (line 9) Let e be the edge of the single IP with different sign. The plane of S_3 separates e from the other two edges so it goes through the faces adjacent to e. The order is detemined by the function edge faces.

- **Single degenerated IP.** (line 3) Let us denote e the edge with degenerated IP and f the face between the other two edges. The other two (non-degenerate) edges may have either oposite sign (the plane cut S_4) or the same sign (the plane touching S_4 at the edge e). In the first case, the call of edge faces for e returns (f_x, f) or (f, f_x), then thr vertex faces function returns (e, f) or (f, e), respectively.

  The edge e lies in the the singel edge

- **Two degenerated IPs.** (line 1) A face of S_4 lies in the plane of S_3, single edge e have non-degenerate IP. We treat the two degenerate edges as special case of faces adjacent to e and return them sorted like the faces given by edge faces fro edge e.

Either all signs are equal (S_3 touching S_4) wee , or a single edge have the sign oposite to other twotwo edges have a sign oposite then the thi. of According to the and need to insert them into I to correct place. To this end we exploit additional information about coincidence of points with faces of S_4. Every point p from the first part of the algorithm **PE:** *I would use proper references instead, I got confused, what 'part of algorithm' is..* that is not a vertex of S_3 belongs to a face f of S_4; we know from the order of the points whether the face comes before or after the point in the intersection polygon; in the first case we set $F[0, f] = p$, in the second case we set $F[1, f] = p$. Every point p from the second part of the algorithm lies on an edge e which is between two faces f_0, f_1. The correct order of the faces on the intersection polygon is determined by the sign of the edge-triangle intersection. So, having the faces in the correct order we set $F[0, f_0] = p$ and $F[1, f_1] = p$.

Finally in the third part (Algorithm 4), the table F than allows us to modify array of successors P and get I in correct order as the list K.

Algorithm 4: 2d-3d intersection, finish sort of points

```
Input: all points in $I$
Output: polygon in $K$ in correct order
1 $P = (1, \dots, n-1, 0)$// array of sucessors
2 for $f$ is face of $S_4$ do $P[F[1, f]] = F[0, f]$
3 $i = 0$
4 for $n = 0$ to $|I| - 1$ do $K[n] = I[i]$ $i = P[i]$
```

List $in[f]$ contains index of the intersection point that follows after f on the boundary of traced polygon, similarly $out[f]$ stores index of the intersection point that preeceeds the face f.

Possible cases for processing L:

1. Regular case, L consists of two intersections p, q sorted by orientation of s, laying inside of s.
   If p is on the edge e of S_4 compute sign of intersection(e, S_3), sort the faces f_0, f_1 coincident with e and set $in[f_0]$ to index of p in L. Similarly if q is on the edge, set $out[f_1]$ to index of q in L.
   If p is in vertex v of S_4, for every face f coincident with v set in[f] to index of p unless there is some index already set. So, we do not over ride entries comming from the edge intersections. Similarly set out[f] if q is in vertex of S_4.
   If p is on face f_0 of S_4, set $in[f_0]$ to index of p. Similarly, if q is on face f_1 of S_4, set $out[f_1]$ to index of q. laying on faces f_p, f_q of S_4.
2. L consists of a single intersection point p (touching S_4)
   If p is on edge, compute sign of intersection(e, S_3), sort the faces f_0, f_1, set $in[f_0]$ and $out[f_1]$ to index of p.
   If p

How tracing works.

- If there are no intersections in vertex of S_4.

intersection polygon 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

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).

2.5. Tracking boundary of the intersection polygon

PE: *It seems to me, that we can really describe the prolongation in general, for 1D and 2D, refering to them as components. I tried to do so..*

PE: *We need proper definitions of terms we use:*
candidates pair *is a pair of a component element and a bulk element, that might intersect each other (due to intersection of their bounding boxes or prolongation result)* pathologic, special, degenerate case ??

PE: *Do you want to use American 'neighbor' or all other English 'neighbour'? (I prefer non-american.. to be unified at the end..)*

2.6. Prolongation algorithm

Consider now a complex mesh of combined dimensions consisting of *components*, which are sets of connected elements of the same lower dimension (1D, 2D), in the space of 3D elements, which we shall call a *bulk*. Obtaining all of 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.

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 component elements.

breadth first search algorithm:

1. Get next unprocessed component element k.
2. Find intersection candidates $\mathcal{K}$ in bulk mesh (3D elements).
3. For $K \in \mathcal{K}$ compute intersection (k, K).
4. Push the intersection neigbours into queues: $(k, L) \to Q_b$, $(l, K) \to Q_c$.

[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

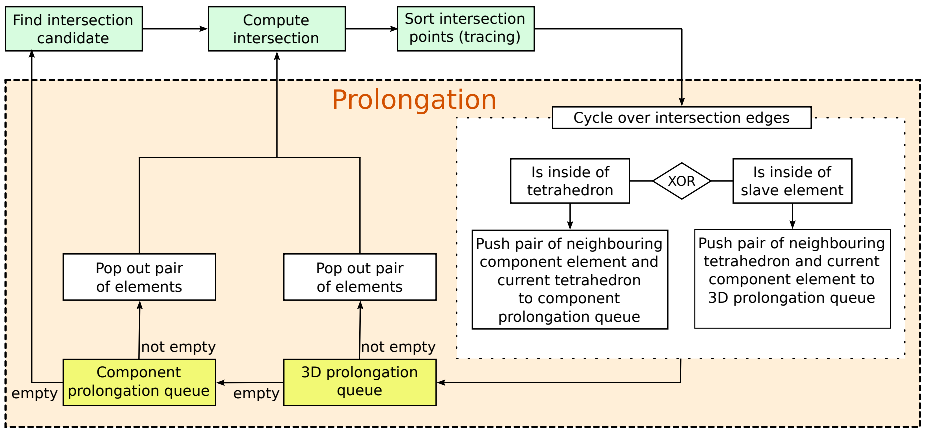

Find intersection candidate
Compute intersection
Sort intersection points (tracing)
Prolongation
Cycle over intersection edges
Is inside of tetrahedron
XOR
Is inside of slave element
Push pair of neighbouring component element and current tetrahedron to component prolongation queue
Push pair of neighbouring tetrahedron and current component element to 3D prolongation queue
Pop out pair of elements
Pop out pair of elements
not empty
not empty
empty
Component prolongation queue
empty
3D prolongation queue

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

5. While $(l, L) \in Q_b$ check intersection (l, L). Append queues.
6. Pop pair from Q_c. (move to the next component element: go to 3).
7. If Q_c is empty, go to 1.

Prerequisites:

- need information about element connectivity (mesh preprocessing)
- find the starting intersection – AABB (Axes Aligned Bounding Boxes)
- possibly create Bounding Interval Hierarchy (BIH)

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.

A new candidate pair of elements is found during prologantion, based on the topological information of the intersection point. There are 3 possible cases (applies both for 1D-3D and 2D-3D):

- **IP lies on component element side and inside bulk element**
  We find all the sides of component element in which the IP lies (IP can be at node and connect more sides). We find the component neighboring elements over the sides and push a new candidate pair [component neighbor – current bulk element] into component queue.

- **IP lies on component element side and on the surface of bulk element**
  We find all the faces of bulk 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 neighboring bulk elements over the faces and push the following new candidates pairs into the bulk queue: [current component element – bulk neighbor], [component neighbor – bulk neighbor]. If the candidates pair has been investigated already, we skip it.

- **IP lies inside component element (therefore must be on the surface of bulk element)**
  The same as before, but we push only [current component element – bulk neighbor] candidates pairs, since there is no component neighbor.

3. Front tracking algorithm

4. Benchmarks

5. 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

6. 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.

[1] N. Sukumar, N. Mos, B. Moran, T. Belytschko, Extended finite element method for three-dimensional crack modelling 48 (11) 1549–1570. doi:10.1002/1097-0207(20000820)48:11<1549::AID-NME955>3.0.CO;2-A.
URL http://doi.wiley.com/10.1002/1097-0207%2820000820%2948%3A11%3C1549%3A%3AAID-NME955%3E3.0.CO%3B2-A

[2] A. Massing, M. G. Larson, A. Logg, Efficient implementation of finite element methods on nonmatching and overlapping meshes in three dimensions 35 (1) C23–C47. doi:10.1137/11085949X.
URL http://epubs.siam.org/doi/abs/10.1137/11085949X

[3] A. H. Elsheikh, M. Elsheikh, A reliable triangular mesh intersection algorithm and its application in geological modelling 30 (1) 143–157. doi:10.1007/s00366-012-0297-3.
URL http://link.springer.com/article/10.1007/s00366-012-0297-3

[4] M. J. Gander, C. Japhet, Algorithm 932: PANG: Software for nonmatching grid projections in 2d and 3d with linear complexity 40 (1) 1–25. doi:10.1145/2513109.2513115.
URL http://dl.acm.org/citation.cfm?doid=2513109.2513115

[5] M. J. Gander, C. Japhet, An algorithm for non-matching grid projections with linear complexity, in: M. Bercovier, M. J. Gander, R. Kornhuber, O. Widlund (Eds.), Domain Decomposition Methods in Science and Engineering XVIII, no. 70 in Lecture Notes in Computational Science and Engineering, Springer Berlin Heidelberg, pp. 185–192, DOI: 10.1007/978-3-642-02677-5_19.
URL http://link.springer.com/chapter/10.1007/978-3-642-02677-5_19

[6] W. M. Boon, J. M. Nordbotten, Robust discretization of flow in fractured porous media.
URL http://arxiv.org/abs/1601.06977

[7] S. Bournival, J.-C. Cuillire, V. Franois, A mesh-geometry based approach for mixed-dimensional analysis, in: Proceedings of the 17th International Meshing Roundtable, Springer, pp. 299–313.
URL http://link.springer.com/chapter/10.1007/978-3-540-87921-3_18

[8] V. Reichenberger, H. Jakobs, P. Bastian, R. Helmig, A mixed-dimensional finite volume method for two-phase flow in fractured porous media 29 (7) 1020–1036.
URL http://www.sciencedirect.com/science/article/pii/S0309170805002150

[9] K. Sldkov, V. Dolej, Bakalsk prce: Integration of piecewise polynomial functions on non-matching grids.

[10] G. Pichot, J. Erhel, J.-R. de Dreuzy, A generalized mixed hybrid mortar method for solving flow in stochastic discrete fracture networks 34 (1) B86–B105.
URL http://epubs.siam.org/doi/abs/10.1137/100804383

[11] B. Flemisch, M. Kaltenbacher, S. Triebenbacher, B. Wohlmuth, Non-matching grids for a flexible discretization in computational acoustics 11 (2) 472–488. doi:10.4208/cicp.141209.280810s.
URL http://www.journals.cambridge.org/abstract_S1815240600002292

[12] Hybrid local nonmatching method for multiphase flow simulations in heterogeneous fractured media, u.S. Classification 703/2, 703/10; International Classification G06F17/50; Cooperative Classification G01V2210/646, E21B43/26, G01V1/30, E21B41/00, G01V1/301, G06F17/5018.
URL http://www.google.com/patents/US20140046636

[13] R. Mittal, G. Iaccarino, IMMERSED BOUNDARY METHODS 37 (1) 239–261. doi:10.1146/annurev.fluid.37.061903.175743.
URL http://www.annualreviews.org/doi/abs/10.1146/annurev.fluid.37.061903.175743

[14] S. J. Owen, A survey of unstructured mesh generation technology., in: IMR, pp. 239–267.
URL http://ima.udg.edu/~sellares/ComGeo/OwenSurv.pdf

[15] Y. A. Kuznetsov, Overlapping domain decomposition with non-matching grids 6 299–308.
URL http://emis.ams.org/proceedings/DDM/DD9/Kuznetsov.ps.gz