An Algorithm for Non-Matching Grid Projections with Linear Complexity

Martin J. Gander[1] and Caroline Japhet[2]

[1] Section of Mathematics, University of Geneva, 2-4 Rue du Lièvre, CP 64, 1211 Geneva 4
martin.gander@unige.ch
[2] LAGA, Université Paris 13, 99 Av. J-B Clément, 93430 Villetaneuse, France
japhet@math.univ-paris13.fr

1 Introduction

Non-matching grids are becoming more and more common in scientific computing. Examples are the Chimera methods proposed by [20] and analyzed in [2], the mortar methods in domain decomposition by [1], and the patch method for local refinement by [6], and [17], which is also known under the name 'numerical zoom', see [9]. In the patch method, one has a large scale solver for a particular partial differential equation, and wants to add more precision in certain areas, without having to change the large scale code. One thus introduces refined, possibly non-matching patches in these regions, and uses a residual correction iteration between solutions on the patches and solutions on the entire domain, in order to obtain a more refined solution in the patch regions. The mortar method is a domain decomposition method that permits an entirely parallel grid generation, and local adaptivity independently of neighboring subdomains, because grids do not need to match at interfaces. The Chimera method is also a domain decomposition method, specialized for problems with moving parts, which inevitably leads to non-matching grids, if one wants to avoid regridding at each step. Contact problems in general lead naturally to non-matching grids.

In all these cases, one needs to transfer approximate solutions from one grid to a non-matching second grid by projection. This operation is known in the literature under the name mesh intersection problem in [12], intergrid communication problem in [16], grid transfer problem in [18], and similar algorithms are also needed when one has to interpolate discrete approximations, see [13, Chap. 13].

2 Towards an Optimal Algorithm

There are two problems that need to be addressed in order to obtain an efficient projection algorithm, a combinatorial one and a numerical one: the combinatorial one stems from the fact that in principle, every element of one grid could be intersecting

with every element of the other grid, and hence the naive approach immediately leads to an $O(n^2)$ algorithm, where n is the number of elements. This is well known in the domain decomposition community, see for example [4]. The numerical difficulty is related to the calculation of the intersection of two finite elements, which is numerically difficult, because one needs to take numerical decisions whether two segments intersect or not, and whether one point is in an element or not. For the patch method, [17] state: "Some difficulties remain though since we must compute integrals involving shape functions that are defined on non-compatible meshes". They use as approximation a midpoint rule, computing only in which element the barycenter of the elements of the other grid lies. The influence of the error of a quadrature rule for this problem is studied in [15]. The authors in [4] mention the substantial complexity increase when going from one- to two-dimensional interfaces, and a sophisticated program with many special cases is used to compute the projection, as explained by [3].

If one needs to interpolate values only, the numerical intersection problem is avoided, and an elegant way to reduce the complexity to $O(n \log n)$ was introduced by [10], in form of an additional adaptively refined background Cartesian mesh, called quadtree in 2d and octree in 3d. This approach is currently widely used, for example in contact problems, see [11], where the overall complexity of the simulation process is still dominated by the nonlinear monotone multigrid method. A related approach is to use a binning (or bucket) technique, introduced by [14], see for example [18], and the MpCCI code from the Fraunhofer [8]. Faster algorithms can be obtained, if neighboring information for each element is available: in the case of interpolation, one can use an advancing front technique that starts, for each new point at which one needs to interpolate data, a local search in the neighborhood of the element where the previous point was interpolated. Only if this search is not successful in less than a constant number of steps a brute force search is launched, see [13]. This approach leads to an algorithm with close to linear complexity. A related technique uses a self-avoiding walk, see [12], with a vicinity search. This search can only fail after a boundary element had no intersection with any element of the other mesh, in which case a quad-tree search is employed. This leads to what the authors call approximately linear complexity. Further techniques for treating the boundary are given in [13].

Computing the intersection of elements numerically was first studied in the computer graphics community under the name "polygon clipping", see [21] and references therein. The basic algorithm works as follows: one marches along the edges of one polygon, and whenever an intersection is found, one switches the polygons and marches on the edges of the other one. As soon as one returns to a point already visited, the intersection polygon is obtained. This algorithm is extensively used in computer graphics, and a generalized version, which can also handle self-intersecting polygons can be found in [7], where we also find the quote: "So far we have tacitly assumed that there are no degeneracies, i.e. each vertex of one polygon does not lie on an edge of the other polygon. Degeneracies can be detected in the intersect procedure ... In this case we perturb the vertex slightly. If we take care that the perturbation is less than a pixel width, the output on the screen will be cor-

rect." While for computer graphics, a natural scale for the truncation is the pixel, it is more difficult to determine acceptable perturbations for numerical applications. Since we did not find an elegant and robust solution for degenerate cases in the context of mortar applications, we propose an entirely different algorithm below, which can also easily be generalized to three-dimensional interfaces. A numerically robust way to determine intersections is however presented in [19], who is using adaptive precision floating point arithmetic. The award winning mesh generator "triangle" by the same author computes intersections of two non-matching triangular grids using this approach.

For a problem in two dimensions, the mortar method has one-dimensional interfaces, and a simple algorithm based on the ideas of merge sort computes the projection in $O(n)$ steps, where n is the number of elements touching the interface, see [5]. We show in this paper a generalization of this algorithm to higher dimensions. We use an advancing front technique and neighboring information, which is often available in finite element meshes, in order to obtain an algorithm with linear complexity. Its implementation is surprisingly short, and we give the entire Matlab code. For computing the intersection, we use a new approach, which turns out to be numerically robust and can be generalized to higher dimensions. We show numerical experiments both in 2d and 3d, which illustrate the optimal complexity and negligible overhead of the algorithm.

3 The Algorithm for Computing the Intersection

We now present an algorithm that computes the intersection polygon of two arbitrary triangles. It first computes all edge intersections, and all corners of the triangles that are contained in the other one. Then the algorithm orders the set of points obtained counterclockwise in order to obtain the intersection polygon, see Figure 1. The graphic primitive `EdgeIntersections(X,Y)` computes all intersections of edges of triangle `X` (corner coordinates stored column-wise) with edges of triangle `Y`, including borderline cases by using greater or equal in the decisions. The routine `PointsOfXInY(X,Y)` computes corners of triangle `X` in triangle `Y`, again including borderline cases. The routine `SortAndRemoveDoubles(P)` sorts the points in `P` in counterclockwise order and removes duplicates, which turns out to make the algorithm numerically robust.

In addition to computing the intersection polygon, the algorithm also returns two more results needed later: in `n` which neighboring triangles of `X` will also intersect with `Y`, and in `M` the integrals on the intersection `P` of products of element shape functions of `X` with the ones of `Y`, or any related quantity obtained from the routine `MortarInt`.

This algorithm can be generalized to compute the intersection of tetrahedra in 3d (see also Section 5): one first calculates all points were an edge of one tetrahedron traverses the face of the other, and all corners of one tetrahedron contained in the other. Then one orders the points face by face counterclockwise. Note also that this intersection algorithm can easily be generalized to convex polygons and polyhedra.

```
function [P,n,M]=Intersect(X,Y);
% INTERSECT intersection of two triangles and mortar contribution
%   [P,n,M]=Intersect(X,Y); computes for two given triangles X and Y
%   the points P where they intersect, in n the indices of neighbors
%   of X that also intersect with Y, and the local mortar matrix M
%   of contributions of the element X on the element Y.

[P,n]=EdgeIntersections(X,Y);
Q=PointsOfXInY(X,Y);
if size(Q,2)>1                      % if there are two or more
  n=[1 1 1];                        % interior points, the triangle
end                                 % is candidate for all neighbors
P=[P Q];
P=[P PointsOfXInY(Y,X)];
P=SortAndRemoveDoubles(P);          % sort counterclockwise
M=zeros(3,3);
if size(P,2)>0
  for j=2:size(P,2)-1               % compute local Mortar matrix
    M=M+MortarInt(P(:,[1 j j+1]),X,Y);
  end;
end;
```

Fig. 1. Algorithm for computing the intersection polygon of two triangles.

4 The Projection Algorithm with Linear Complexity

We now show an algorithm that computes, for two non-matching triangular meshes representing the same planar geometry, the associated mortar projection matrix, see [1], or any other similar quantity on each intersection polygon defined by `MortarInt` in the `Intersect` procedure. The algorithm is using advancing fronts and the fact that each triangle knows which are its neighbors, see Fig. 2. The input of the algorithm are two triangular grids. The grid node coordinates are stored column wise in `N`. The triangles are stored row wise in `T`, the first three numbers referring to the nodal coordinates of the triangle in `N`, and the next three to the neighboring triangles in `T`, both ordered counterclockwise. The algorithm then works as follows: it starts with a pair of intersecting triangles (assumed to be the first ones in `Ta` and `Tb`), which are often trivially available at a corner, but otherwise could also be found by one direct search. We then compute first the intersection of these two triangles using the intersection routine from Section 3. We then add the neighbors of the triangle from mesh *a* as candidates in a list `al`, since they could intersect with our triangle from mesh *b*. Picking triangles from list `al` one by one, we compute their intersection with the current triangle from mesh *b* and add non treated neighbors to the list `al` until all triangles in `al` have been treated. This implies that the starting triangle from mesh *b* cannot intersect any triangles from mesh *a* any more. Now we put all the neighbors of the starting triangle of mesh *b* into a list `bl`, and perform the same steps as for the first triangle on each one in the list `bl`, until it becomes empty, and the algorithm terminates.

We now address the complexity of our algorithm: the key step is that we stored a starting candidate from list `al` for each of the triangles added to list `bl` in list `bil`. This information is obtained without extra calculation in the computation of the intersection. Thus there is never a search needed for a candidate triangle of mesh *a*

```
function M=InterfaceMatrix(Na,Ta,Nb,Tb);
% INTERFACEMATRIX projection matrix for non-matching triangular grids
%   M=InterfaceMatrix(Na,Ta,Nb,Tb); takes two triangular meshes Ta
%   and Tb with associated nodal coordinates in Na and Nb and
%   computes the associated mortar projection matrix M

bl=[1];                       % bl: list of triangles of Tb to treat
bil=[1];                      % bil: list of triangles Ta to start with
bd=zeros(size(Tb,1)+1,1);     % bd: flag for triangles in Tb treated
bd(end)=1;                    % guard, to treat boundaries
bd(1)=1;                      % mark first triangle in b list.
M=sparse(size(Nb,2),size(Na,2));
while length(bl)>0
  bc=bl(1); bl=bl(2:end);     % bc: current triangle of Tb
  al=bil(1); bil=bil(2:end);  % triangle of Ta to start with
  ad=zeros(size(Ta,1)+1,1);   % same as for bd
  ad(end)=1;
  ad(al)=1;
  n=[0 0 0];                  % triangles intersecting with neighbors
  while length(al)>0
    ac=al(1); al=al(2:end);   % take next candidate
    [P,nc,Mc]=Intersect(Nb(:,Tb(bc,1:3)),Na(:,Ta(ac,1:3)));
    if ~isempty(P)            % intersection found
      M(Tb(bc,1:3),Ta(ac,1:3))=M(Tb(bc,1:3),Ta(ac,1:3))+Mc;
      t=Ta(ac,3+find(ad(Ta(ac,4:6))==0));
      al=[al t];              % add neighbors
      ad(t)=1;
      n(find(nc>0))=ac;       % ac is starting candidate for neighbor
    end
  end
  tmp=find(bd(Tb(bc,4:6))==0); % find non-treated neighbors
  idx=find(n(tmp)>0);         % take those which intersect
  t=Tb(bc,3+tmp(idx));
  bl=[bl t];                  % and add them
  bil=[bil n(tmp(idx))];      % with starting candidates Ta
  bd(t)=1;
end
```

Fig. 2. Algorithm with linear complexity for computing the intersection of two non-matching triangular grids and the associated mortar projection matrix.

that could intersect the currently treated triangle from mesh *b*. The algorithm treats triangles of mesh *b* one by one, and checks for each triangle at most a constant number of triangles in mesh *a*, which shows that the average complexity is linear. The worst-case complexity however is quadratic, namely when the constant equals the total number of triangles in mesh *a*. This situation arises when every triangle of mesh *a* intersects every triangle of mesh *b*, and quadratic complexity is unavoidable in this case.

Note that our algorithm does not depend on the number of dimensions; it only uses the fact that each element has a given number of neighbors, which in the implementation shown is three. We however assumed that the two meshes are connected, and also that the intersection of one element with the elements of the other mesh are simply connected. Otherwise the algorithm would need extra starting points in order to find the complete intersections.

5 Numerical Experiments

We show in Fig. 3 a comparison of our algorithm with the brute force search, where

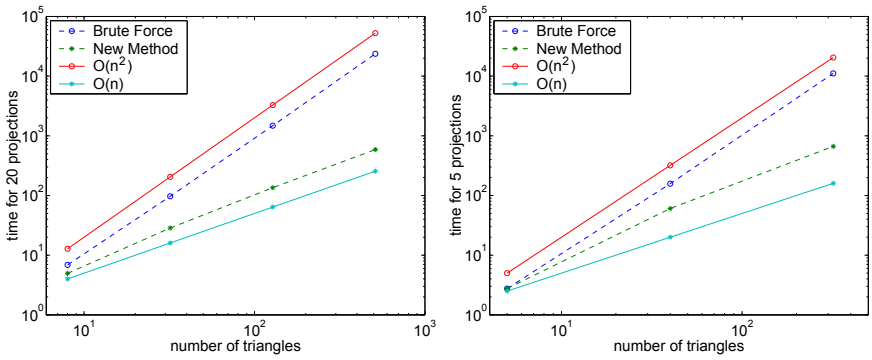

Fig. 3. Comparison in computing time for two-dimensional meshes on the left, and three-dimensional meshes on the right.

for every element in the first mesh the intersection with every element in the second mesh is computed. On the left, we show the average computing time for twenty projection calculations in two dimensions, each time with two random triangular meshes, and on the right a similar comparison for the three-dimensional case, where we show the average computing time for five projection calculations. In addition to the asymptotic superiority, we also see that the new algorithm is already competitive for small meshes, i.e. the algorithmic overhead is negligible.

6 Conclusions

The intersection algorithm we presented for two triangles can be made slightly faster by first using an inexact range test to quickly exclude non-intersecting triangles, before starting the actual computation of the intersection.

The projection algorithm itself has also been extended to contact problems, where the interfaces of the two neighboring domains do not quite lie in the same physical manifold, and an additional projection "normal" to the interface is necessary. All codes and a demo are available at `www.unige.ch/~gander`.

Acknowledgement. We thank a referee for detailed comments. This research was supported in part by the Swiss National Science Foundation Grant 200020-1 17577/1.

References

[1] Bernardi, C., Maday, Y., Patera, A.T.: Domain decomposition by the mortar element method. In H. G. Kaper and M. Garbey, eds., *Asymptotic and Numerical Methods for Partial Differential Equations with Critical Parameters*, vol. 384, 269–286. N.A.T.O. ASI, Kluwer Academic, 1993.

[2] Brezzi, F., Lions, J.-L., Pironneau, O.: Analysis of a Chimera method. *C. R. Acad. Sci. Paris Sér. I Math.*, 332(7):655–660, 2001.

[3] Flemisch, B., Hager, C.: Mortar projection algorithms. Private Communication, 2007.

[4] Flemisch, B., Kaltenbacher, M., Wohlmuth, B.I.: Elasto-acoustic and acoustic-acoustic coupling on nonmatching grids. *Internat. J. Numer. Methods Engrg.*, 67(13):1791–1810, 2006.

[5] Gander, M.J., Japhet, C., Maday, Y., Nataf, F.: A new cement to glue nonconforming grids with Robin interface conditions: The finite element case. In R. Kornhuber et al. eds., *Proceedings of the 15th international domain decomposition conference*, volume 40, 259–266. Springer LNCSE, 2005.

[6] Glowinski, R., He, J., Lozinski, A., Rappaz, J., Wagner, J.: Finite element approximation of multi-scale elliptic problems using patches of elements. *Numer. Math.*, 101(4):663–687, 2005.

[7] Greiner, G., Hormann, K.: Efficient clipping of arbitrary polygons. *ACM Trans. Graph.*, 17(2):71–83, 1998.

[8] Fraunhofer Institute: *MpCCI 3.0.6-12 Documentation*. Fraunhofer Institute for Algorithms and Scientific Computing, 2007.

[9] Apoung Kamga, J.-B., Pironneau, O.: Numerical zoom for multiscale problems with an application to nuclear waste disposal. *J. Comput. Phys*, 224:403–413, 2007.

[10] Knuth, D.N.: *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973.

[11] Krause, R., Sander, O.: Fast solving of contact problems on complicated geometries. In R. Kornhuber et al. eds., *Proceedings of the 15th international domain decomposition conference*, vol. 40, 495–502. Springer LNCSE, 2005.

[12] Lee, P., Yang, C.-H., Yang, J.-R.: Fast algorithms for computing self-avoiding walks and mesh intersections over unstructured meshes. *Adv. Engrg. Softw.*, 35: 61–73, 2004.

[13] Löhner, R.: *Applied CFD Techniques: An Introduction Based on Finite Element Methods*. Wiley, 2001.

[14] Löhner, R., Morgan, K.: An unstructured multigrid method for elliptic problems. *Internat. J. Numer. Methods Engrg.*, 24:101–115, 1987.

[15] Maday, Y., Rapetti, F., Wohlmuth, B.I.: The influence of quadrature formulas in 2d and 3d mortar element methods. In *Recent developments in domain decomposition methods (Zürich, 2001) Lect. Notes Comput. Sci.*, vol. 23, 203–221. Springer, 2002.

[16] Meakin, R.L.: A new method for establishing intergrid communication among systems of overset grids. *10th AIAA Comput. Fluid Dyn. Conf. (Honolulu, HI),*, 91(1586), 1991.
[17] Picasso, M., Rappaz, J., Rezzonico, V.: Multiscale algorithm with patches of finite elements. *Comm. Numer. Methods Engrg.*, 24(6):477–491, 2007.
[18] Plimpton, S., Hendrickson, B., Stewart, J.: A parallel rendezvous algorithm for interpolation between multiple grids. *J. Parallel Distr. Comput.*, 64(2), 2004.
[19] Shewchuk. J.R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Comput. Geom.*, 18(3):305–363, 1997.
[20] Steger, J.L., Dougherty, F.C., Benek, J.A.: A chimera grid scheme, advances in grid generation. In K.N. Ghia and U. Ghia, eds., *ASME FED*, vol. 5, 1983.
[21] Weiler, K., Atherton, P.: Hidden surface removal using polygon area sorting. *Siggraph* 11(2):214–222, 1977.