Mesh Association: Formulation and Algorithms*

Xiangmin Jiao† Herbert Edelsbrunner‡ Michael T. Heath†

Abstract

In computational simulation of coupled, multicomponent systems, it is frequently necessary to transfer data between meshes that may differ in resolution, structure, and discretization methodology. Typically, nodes from one mesh must be associated with elements of another mesh. In this paper, we formulate *mesh association* as a geometric problem and introduce two efficient mesh association algorithms. One of these algorithms requires linear time in the worst case if the meshes are well shaped and geometrically well aligned. Our formulation of the problem and our algorithms are more general than previous work and can be applied to surface meshes with curved elements.

Keywords. Mesh generation, computational geometry, point association, data transfer, search, interpolation.

1 Introduction

Mesh association is a problem that arises frequently in the numerical simulation of complex, multicomponent systems. In such systems, data must be transferred across interfaces between adjacent domains whose respective meshes may differ in resolution, structure, and discretization methodology. We call the 2-dimensional portion of a discretized 3-dimensional domain that is adjacent to another such domain its *interface mesh*. Given two domains, we refer to their interface meshes as G and H, alluding to a *guest* and a *host* mesh. Although the neighboring physical domains abut each other, the two corresponding interface meshes may not precisely coincide because of discretization or rounding errors.

There are two distinct phases in data transfer between domains, one geometric and the other numerical. The first phase is *mesh association*, in which each node in G is associated with a face or *element* in H. The second phase computes approximate local coordinates in H for the nodes in G, and then interpolates field data using these computed coordinates. This paper provides a systematic formulation and efficient algorithms for the first phase. For a discussion of the second phase, readers are referred to [2, 3, 6].

When the interface meshes of two domains are identical, with coincident nodes and elements, mesh association is trivial. Domains often have nonconforming interface meshes, however, due to different discretization methodologies or resolution requirements. For example, in *fluid-solid interaction*, which is a typical multidisciplinary problem, a finite difference or finite volume method with a (block) structured mesh is often used for the fluid, whereas a finite element method with an unstructured mesh is typically used for the solid. Generally, the fluid also has finer resolution requirements than the solid. Such discrepancies in interface meshes make mesh association decidedly nontrivial.

Mesh association has attracted considerable attention in recent years, but formulations of the problem and algorithms for solving it have not been completely satisfactory. Löhner [8] suggests an algorithm for mesh association, called the *advancing-front vicinity algorithm*, but with a restrictive assumption that all nodes of G lie on the underlying space of mesh H. Maman and Farhat [9] propose a scheme for associating nonconforming meshes that relaxes this assumption. They define the *associate* of a point x to be the normal projection of x onto H, and the corresponding element would then be the host of that associate. With this definition, however, there is a nonnegligi-

*Research supported by the Center for Simulation of Advanced Rockets funded by the U.S. Department of Energy under Subcontract B341494.

†Department of Computer Science, University of Illinois, Urbana, IL 61801. {jiao,heath}@cs.uiuc.edu.

‡Department of Computer Science, Duke University, Durham, NC 27708. edels@cs.duke.edu.

ble possibility for a point to have more than one associate or no associate, as illustrated in Figure 1. In their study, they assume that these problematic cases do not occur. Their algorithm uses exhaustive search to find the associate of each node of G, which is not very efficient, though easy to parallelize.

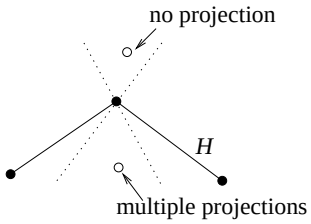

Figure 1: 1-dimensional example where associate may be ill-defined using Maman and Farhat's definition. Points in upper region between dotted lines do not project onto H and hence have no associate. Points in lower region between dotted lines project onto two elements of H and thus have multiple associates.

In this paper, we consider the mesh association problem with nonconforming meshes. In particular, we focus on cases arising from 3-dimensional fluid-solid interaction, in which the 2-dimensional interface meshes must be associated. Section 2 formulates mesh association as a geometric problem. Section 3 introduces efficient generic algorithms for mesh association based on the new formulation. Finally, Section 4 concludes the paper with a discussion of related problems.

2 Problem Definition

We first formulate mesh association as a geometric problem with physical meaning in mesh applications. In this paper an *interface mesh* or *mesh* refers to a collection of cells of dimension 2, 1, and 0. We call the 2-dimensional cells *elements*, the 1-dimensional cells *edges*, and the 0-dimensional cells *nodes*. Elements are required to be closed topological disks, edges are closed topological intervals, and nodes are points, all embedded in $\mathbb{R}^3$. We also require the mesh to be a pure complex. In particular, the boundary of each element is a finite union of edges, each edge or node belongs to at least one element, and any two elements either are disjoint, intersect in a single node, or intersect along a single edge. The meshes in our application are portions of surfaces of 3-dimensional domains, so we may further assume that each mesh is a 2-manifold with boundary. This means that each edge belongs to either one or two elements, and each node belongs to either a linear or a cyclic sequence of elements. We also assume that the number of elements is at most a constant times the number of nodes. Occasionally, we will talk about the set of points contained in the elements of a mesh. This is traditionally called the *underlying space* of the mesh, but we find it easier to ignore the difference between a mesh and its underlying space.

2.1 Point Association

We start by defining point association, which is the key component of mesh association.

DEFINITION. *An **associate** of a point $x \in \mathbb{R}^3$ is a point $x' = x'_H$ in a mesh H with minimum distance to x. An **associated element** of x is an element $A(x) = A_H(x)$ in H containing an associate of x. The **distance** from x to H is the Euclidean distance between x and x', denoted by $d(x) = d_H(x) = \|x - x'\|$. Given x and H, **point association** is the problem of computing the associated elements of x in H.*

This definition does not make any assumption about the type of mesh, so it is applicable to meshes of arbitrary structure and with curved elements, and can also be generalized to higher dimensions. Figure 2 illustrates the definitions. In the figure, the mesh H lies in a plane, which generally is not the case.

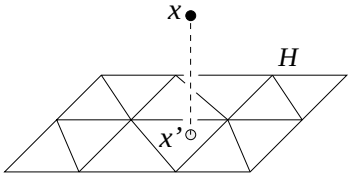

Figure 2: Orthogonal projection of x onto plane of mesh H is x'. It is contained in interior of associated element $A(x)$. Length of line segment xx' is $d(x)$.

This formulation of point association makes sense both geometrically and numerically. Geometrically, the associate of a point x is *unique* in the sense that for any x, an arbitrarily small perturbation to x suffices to make x' unique. The perturbed point might still have more than one orthogonal projection onto H, but only one will minimize the distance to x. Consequently, we say a point having more than one associate is a *degeneracy*. Without loss of generality,

we can then assume *general position*, which is the absence of such degeneracy. Degenerate cases can be handled using techniques such as symbolic perturbation [5, 12].

A point having multiple associated elements, however, is not a degeneracy. This is because $A(x)$ is not unique when x' is a node of H or lies in the interior of an edge, and the probability of this happening is nonzero when H is flat. Nevertheless, assuming general position of x, any reasonable tie-breaking scheme for choosing $A(x)$ suffices to make it unique. This causes no harm to the overall objective of mesh association since no matter which associated element is chosen, the interpolation will always give the same result. Henceforth, we use the notation $\boldsymbol{A}(x)$ to denote the set of all associated elements of x, and $A(x)$ to denote the unique associated element determined by some tie-breaking strategy, if necessary. We have thus shown that point association is a well-posed problem.

Numerically, the associate will be used to compute local coordinates γ in H for a node x for subsequent interpolation. We assume that for each element h there is a diffeomorphic map, $\eta_h : [0,1]^2 \to h$, which maps local to physical coordinates. Given the physical coordinates of the point x, if $x \in h$, the local coordinates γ have an exact solution. If x does not lie on the host mesh H, however, then the local coordinates can be solved for only approximately. The best solution in the least squares sense minimizes the distance from the point x to the point $\eta_h(\gamma)$, where $h = A(x)$, and $\eta_h(\gamma) = x'$, which agrees with our definitions of associate and associated element. In summary, a closest element is optimal for computing the local coordinates in H of a point $x \in \mathbb{R}^3$.

Before we move on to mesh association, we note that the distance function $d : \mathbb{R}^3 \to \mathbb{R}$ defined by $d(x) = \|x - x'\|$ is Lipschitz continuous.

LEMMA 1. $|d(x) - d(y)| \leq \|x - y\|$.

To see this, observe that the associate y' of y lies on or outside the sphere with center x and radius $d(x)$. The triangle inequality implies the claimed inequality.

2.2 Mesh Association

DEFINITION. ***Mesh association*** *is the problem of identifying the associated element in a host mesh H for each node in a guest mesh G, i.e., mesh association is point association for all nodes of G.*

As in point association, we assume that the nodes are in general position, so that every node in G has a unique associate. This is a reasonable assumption, since the number of nodes is finite. By contrast, assuming general position of *all* points of the surface represented by G would be unrealistic, as this surface contains uncountably many points. Without loss of generality, we also assume that mesh G is connected. If G has more than one component, associating G and H can be treated as several independent mesh association problems, one per component of G.

The definition of mesh association applies to arbitrary meshes G and H, which is nice but overly general. In practice, G and H are typically similar in shape and close in space, because in principle they discretize the same surface. In this section, we propose a criterion for capturing this similarity and closeness relationship to restrict mesh association to a manageable scope. Our criterion is based on the notions of medial axis and local feature size (see also Ruppert [11]).

DEFINITION. *Given a surface mesh H, the* ***medial axis*** *is the set $M = M_H$ of points x with at least two disjoint associated elements. The* ***local feature size*** *is the map $f : H \to \mathbb{R}$ defined such that $f(y)$ is the minimum distance from y to any point of M.*

The medial axis is a surface with measure 0 in $\mathbb{R}^3$, which is consistent with our earlier observation that for every point with more than one associate, we can find an arbitrarily close point with only one associate. Intuitively, the medial axis M relates to mesh resolution and an appropriately discretized notion of curvature of H. In particular, the finer the resolution and the greater the curvature, the closer M is to H. Note in particular that the medial axis does *not* extend all the way to the edges and vertices of the mesh because we require multiple associates on *disjoint* elements. A point x whose distance to the associate x' is less than the local feature size at x' cannot have disjoint associated elements. These observations motivate us to call a mesh G *close* to H if $d(x) < f(x')$ for every point x of G. In any attempt to solve the mesh association problem, it seems reasonable to assume that G be close to H. Essentially, this links how far G can be locally from H to the local curvature and resolution of H. This agrees with our intuitive expectation on the geometric relationship between G and H. Closeness implies a connectedness property for associated elements that is more general than just for nodes of G. This property turns out to be the key to efficient

algorithms for mesh association. Let X be a possibly uncountably infinite point set, and let $\boldsymbol{A}(X)$ be the union of sets $\boldsymbol{A}(x)$ over all points $x \in X$. Similarly, define $A(X)$ as the set of (pointwise unique) associated elements of points of X.

LEMMA 2. *If a point set $X \subseteq \mathbb{R}^3$ is connected and close to H, then $A(X)$ is connected.*

PROOF. We prove the claim by contradiction, using Lemma 1 and the intermediate value theorem of differential calculus. Let $C \subseteq X$ be a connected curve with endpoints x_1 and x_2. Assume $A(x_1)$ and $A(x_2)$ belong to two distinct connected components A_1 and A_2 of $A(X)$. Let $g(x)$ be the difference in Euclidean distances from x to A_1 and to A_2, $g(x) = d_{A_1}(x) - d_{A_2}(x)$. Since d is continuous, so is g. By definition of associated element, $d_{A_1}(x_1) < d_{A_2}(x_1)$ and $d_{A_2}(x_2) < d_{A_1}(x_2)$. Therefore, $g(x_1) < 0$ and $g(x_2) > 0$. According to the intermediate value theorem, there is a point $x \in C$ such that $g(x) = 0$, which implies $x \in M$. This contradicts the closeness assumption. ■

3 Mesh Association Algorithms

In this section, we consider algorithms for mesh association. Except for the brute-force one, all algorithms depend on the guest mesh G being close to the host mesh H. However, we equip the algorithms with safeguards so that a violation of that assumption influences only the running-time and not their correctness.

3.1 Brute-Force Algorithm

A simple but robust algorithm for mesh association performs point association independently for each node in G.

BRUTEFORCE
```
forall nodes v ∈ G do
   d(v) ← ∞;
   forall elements h ∈ H do
      if d(v) > d_h(v) then
         d(v) ← d_h(v); A(v) ← h
      endif
   endfor
endfor.
```

The inner `for`-loop is a brute-force method for point association. At the end of each iteration, $d(v)$ stores the distance from v to H, and $A(v)$ stores the associated element of v. When the algorithm terminates, the arrays d and A contain the distances and associated elements for all guest nodes.

This algorithm is not very efficient. If m is the number of nodes in G and n the number of nodes in H, then the algorithm requires $\Theta(mn)$ time, assuming the distance from a point to an element is computed in constant time. The algorithm is slow but robust, and neither its correctness nor its running-time depends on G being close to H. In the following, we will improve the performance of this algorithm in various respects, but we shall use brute-force as a fallback if all else fails.

One obvious source of inefficiency in the brute-force algorithm is that it ignores neighborhood information. In practice, the associated elements of nearby nodes tend to be nearby as well. We take advantage of this observation by making two changes to the algorithm. In the outer loop, we traverse the nodes in G from neighbor to neighbor. In the inner loop, we perform a local search in H instead of a blind global search. Implementing local search is the main issue here, and it will be discussed shortly.

Another improvement takes advantage of the numerical nature of mesh association. Recall that the purpose is to compute approximate local coordinates in H of nodes in G. These coordinates must be within some numerical tolerance so that the interpolated result is reasonably accurate. We will use the concept of tolerance to improve performance and maintain robustness of our algorithms.

3.2 Steepest Descent

Steepest descent is a greedy method for finding local minima [7]. The basic idea is to move in the direction of locally steepest slope. We use this idea in the inner loop of the mesh association algorithm. After finding $A(u)$ for a node $u \in G$, we search for the associated element of a neighboring node $v \in G$ by starting from $A(u)$ and walking in H until we reach a local minimum. The outer loop is implemented as a depth-first search of the graph of nodes and edges in G, see e.g. [4]. To simplify the description of the algorithm we just write "`forall` nodes $v \in G$ in df-order `do`" to indicate depth-order traversal of the nodes. This order provides information about the predecessor of

v, which is a neighbor u for which $d(u)$ and $A(u)$ are already known. To avoid the case where v has no predecessor, we compute $d(u)$ and $A(u)$ for some arbitrary node u using BRUTEFORCE and start the search at a neighbor v of that initial node u.

```
STEEPESTDESCENT
  forall nodes v ∈ G in df-order do
    h ← A(u);
    do
      mark h with v;
      d(v) ← d_h(v); A(v) ← h;
      h ← CLOSEST(h, v)
    while d(v) ≥ d_h(v)
  endfor.
```

Function CLOSEST returns the element adjacent to h in H that is not yet marked with v and that minimizes the distance to v. For the marking mechanism we can either use a bit array or an integer array that identifies the marking node by index. In the former case we need to unmark elements before repeating the inner `do`-loop in search for the associated element of the next node.

Reaching a local minimum, however, is not the same as finding $A(v)$. Figure 3 shows a 1-dimensional example where STEEPESTDESCENT fails, even though G is close to H. If we start from $A(u)$, the greedy search will never reach $A(v)$. To overcome this difficulty, we introduce a small positive parameter ε, called a *safeguard*, which is a tolerance for the distance between a node and its associated element. Specifically, if the distance between v and the returned $h = A(v)$ exceeds the tolerance times a measure of the length of h such as its diameter, $d_h(v) > \varepsilon \cdot \operatorname{diam} h$, then the element is rejected and BRUTEFORCE is used to recompute the associated element.

The choice of safeguard value ε may be delicate. If it is too small, then the safeguard test may frequently call upon BRUTEFORCE and slow down STEEPESTDESCENT. On the other hand, if the safeguard value is too large, then the computed coordinates may be inaccurate.

3.3 Vine Search

We expect the steepest descent algorithm to be efficient in practice, but it is somewhat problematic

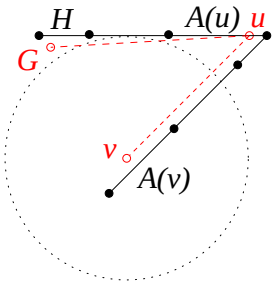

Figure 3: Edges of host mesh are solid, and those of guest mesh are dashed. Steepest descent algorithm fails because gradient of distance function from v points in wrong direction.

because it depends on the choice of a safeguard parameter ε. This subsection proposes a new algorithm that is efficient and safe without depending on any parameter. It still relies on the assumption that G is close to H.

The key to the new algorithm is the connectedness of $A(uv)$, for every edge uv in G, as claimed in Lemma 2. In effect, $A(uv)$ provides a path that leads us from the associated element of u to that of v. Based on the Chinese proverb, "to find the melon, follow the vine," we call this the *vine search algorithm*. It is convenient to follow a path in a slightly larger set $\boldsymbol{B}(uv)$ of elements. We need some definitions to say exactly what the larger set is. For an element, edge, or node $t \in H$ we denote by $H(t)$ the subset of elements that contain t. In the case of an element we have $H(t) = \{t\}$, but for edges and nodes the set $H(t)$ usually, but not necessarily, contains more than one element.

DEFINITION. *Let $x \in \mathbb{R}^3$, $h \in H$ an element, and $x' = x'_h$ the associate of x in h. Let t be h, an edge of h, or a node of h, so that x' lies in the interior of t. We say x **locally associates** with h if $x' = x'_{H(t)}$.*

Observe that local association is weaker than association. In other words, x locally associates with every element $h \in \boldsymbol{A}(x)$, but there can be other elements with which it also locally associates. The set $\boldsymbol{B}(uv)$ is by definition the collection of elements h such that there is at least one point $x \in uv$ that locally associates with h. We trivially have $\boldsymbol{A}(uv) \subseteq \boldsymbol{B}(uv)$. Since $\boldsymbol{A}(uv)$ is connected by Lemma 2, the associated elements $A(u)$ and $A(v)$ belong to the same connected component of $\boldsymbol{B}(uv)$. Hence there is a path of elements we can follow. This is exactly what the vine search algorithm does. For an element $h \in H$

it checks local association, and if that holds it marks h and recursively visits unmarked neighboring elements. Here by neighboring elements we mean the elements that share an edge or a node with h.

```
VineSearch (h)
  if isLocallyAssociated(h, uv) then
     mark h with v;
     if d(v) > d_h(v) then
        d(v) ← d_h(v); A(v) ← h
     endif
     forall unmarked neighbors k of h do
        VineSearch(k)
     endfor
  endif.
```

Function isLocallyAssociated decides whether or not there is a point $x \in uv$ that locally associates with h. This is a local test and we can reasonably assume that the decision can be made in time proportional to the number of elements that share an edge or a node with h. The complete algorithm combines the outer depth-first loop of SteepestDescent with VineSearch as the inner loop replacing the steepest descent search.

Testing local association usually takes constant time, but it is somewhat more involved than computing distance. One may therefore consider modifying vine search so it first follows the strategy of steepest descent and resorts to the more elaborate vine search only if the identified element is not as close to the node as expected. This will speed up the algorithm at the cost of reintroducing a heuristic parameter. Also, if we are not sure whether the guest mesh G is indeed close to the host mesh H, in the technical sense introduced above, we can safeguard vine search against disaster by using a parameter $\varepsilon > 0$ as we did for SteepestDescent.

3.4 Analysis of Vine Search

The running time of vine search depends on the type of the meshes and how they relate to each other geometrically. We analyze the algorithm under admittedly favorable assumptions. We believe that in typical applications of mesh association, the meshes violate the assumptions only mildly and thus incur only slightly higher running-time than analyzed here. Note that the following assumptions are needed only for the analysis and not for the correctness of the algorithm.

(1) G is a convex region contained in a plane in $\mathbb{R}^3$. The same holds for H. The planes of G and H are parallel and the orthogonal projection of G onto the plane of H is contained in H.

(2) All elements of G and H are triangles whose angles are bounded from below by some constant $\delta > 0$.

While typical meshes are not flat, as required by (1), they represent differentiable surfaces and are locally approximately flat. The assumptions of parallel planes and convexity are not very important and serve mainly to simplify the argument below. Assumption (2) is essential because the size of angles has a direct influence on the running time.

Let G' be the mesh obtained by projecting all elements of G orthogonally onto the plane of H. The crucial combinatorial quantity in the analysis of vine search is the number of intersections between edges in H and in G'. We prove below that the number of intersections is at most some constant times the number of edges in H and in G'. The argument reduces the problem to counting the number of holes in the union of a finite set of triangles, again assuming all angles are bounded from below by the same constant δ. Matoušek et al. prove that the number of holes is at most some constant times the number of triangles, where the constant depends of course on δ [10]. Let n be the total number of edges in H and in G combined.

Lemma 3. *The number of pairs of edges, one in H and the other in G', that have non-empty intersection is at most $c \cdot n$, where c is some constant depending on δ.*

Proof. The outside region of H is the complement of the geometric support within the plane of H. Add this region to H, thicken each edge by a tiny amount, and remove the resulting narrow strips from all triangles and from the outside region. We get a collection of slightly smaller triangles, which are separated from each other and from the outside region by narrow channels. Do the same for G'. Consider the union of all the resulting triangles. If we make the channels sufficiently narrow, we get a hole in the union for each intersection point involving the edge in the original H. To apply the result by Matoušek et al., we replace the two outside regions by a collection of triangles whose union is exactly the union of the outside regions, at least in the vicinity of G and H.

That result now implies that the number of intersection points is at most some constant times n. Each intersection point x corresponds to a single pair of edges that intersect at x, unless x is a node of H or G', in which case it corresponds to at most $(2\pi/\delta)^2$ pairs, which is again a constant. ■

We now prove that under assumptions (1) and (2), vine search takes only time proportional to the number of edges in H and in G. Note first that because H lies in a plane, we have $A(uv) = \boldsymbol{B}(uv)$ for every edge $uv \in G$. The number of elements in $A(uv)$ is at most one larger than the number of edges in H that intersect the projection of uv. Lemma 3 thus implies that the total size of all marked sets of elements is at most

$$\begin{aligned} \sum_{uv \in G} |\boldsymbol{B}(uv)| &= \sum_{uv \in G} |A(uv)| \\ &\leq (c+1) \cdot n. \end{aligned}$$

For each marked triangle h, vine search visits h together with all elements that share an edge or a node with h. The angle bound implies that the number of such triangles is at most some constant, namely less than $6\pi/\delta$. This completes the proof that vine search takes time only linear in the size of the guest and host meshes.

4 Discussion

In this paper we considered the mesh association problem, which arises in numerical simulations with multiple components represented by disparate meshes. We provided a precise formulation of the problem and introduced algorithms for solving it. We analyzed vine search, which is the most advanced of the three algorithms, and showed that it takes time only linear in the size of the meshes, under some assumptions. The first author is in the process of implementing the algorithms of this paper, and applying them in coupled, multicomponent simulation of solid propellant rockets, with the intention to report experimental findings later.

A different, but closely related problem, which we call *mesh tracking*, arises in simulations with moving boundaries [2]. In such applications we must update the associated elements as the mesh moves. The relationship between mesh association and mesh tracking is analogous to that between mesh generation and adaptive mesh refinement. Mesh association associates the meshes starting from scratch, without any hint from a previous computation. Mesh tracking, on the other hand, has a previous computed result as a starting point, and each iteration is typically cheaper than a complete mesh association from scratch. The techniques discussed in this paper, such as steepest descent and vine search, are also applicable to mesh tracking.

Point association is related to some well-known problems in computational geometry. One of these is point location [1, Chapter 5], which can be considered as a special case of point association in which the associate of a point is itself. Our approach to mesh association is similar to algorithms described for overlaying two meshes in the plane, see e.g. [1, Chapter 2]. The strongest hint for this similarity is the cost per edge intersection made explicit in the analysis of the vine search algorithm. The same cost occurs in overlaying two meshes, where each intersection becomes a node of the new mesh.

References

[1] M. Berg, M. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[2] J. R. Cebral and R. Löhner. Fluid-structure coupling: extensions and improvements. In *35th AIAA Aerospace Sciences Meeting*, Reno, NV, 1997. AIAA-97-0858.

[3] S. Chippada, C. N. Dawson, M. L. Martinez, and M. F. Wheeler. A projection method for constructing a mass conservative velocity field. *Computer Methods in Applied Mechanics and Engineering*, 157:1–10, 1998.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1989.

[5] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990.

[6] C. Farhat, M. Lesoinne, and P. LeTallec. Load and motion transfer algorithms for fluid/structure interaction problems with nonmatching discrete interfaces: momentum and en-

ergy conservation, optimal discretization and application to aeroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 157:95–114, 1998.

[7] M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw–Hill, New York, 1997.

[8] R. Löhner. Robust, vectorized search algorithms for interpolation on unstructured grids. *Journal of Computational Physics*, 118:380–387, 1995.

[9] N. Maman and C. Farhat. Matching fluid and structure meshes for aeroelastic computations: a parallel approach. *Computers & Structures*, 54:779–785, 1995.

[10] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM Journal on Computing*, 23:154–169, 1994.

[11] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18:548–585, 1995.

[12] R. Seidel. The nature and meaning of perturbations in geometric computing. *Discrete & Computational Geometry*, 19:1–17, 1998.