3B.4

Overlaying Surface Meshes: Extension and Parallelization

Ankita Jain[⋆] and Xiangmin Jiao[⋆⋆]

College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332

Summary. Many computational applications involve multiple physical components and require exchanging data across the *interface* between them, often on parallel computers. The interface is typically represented by surface meshes that are non-matching, with differing connectivities and geometry. To transfer data accurately and conservatively, it is important to construct a *common refinement* (or *common tessellation*) of these surface meshes. Previously, Jiao and Heath developed an algorithm for constructing a common refinement by overlaying the surface meshes. The original algorithm was efficient and robust but unfortunately was complex and difficult to implement and parallelize. In this paper, we present a modified algorithm for overlaying surface meshes. Our algorithm employs a higher-level primitive, namely *face-face intersection*, to facilitate easy parallelization of mesh overlay while retaining the robustness of the original algorithm. We also introduce a safeguarded projection primitives to improve the robustness against non-matching features and potential topological inconsistencies. We present numerical examples to demonstrate the robustness and effectiveness of the new method on parallel computers.

Key words: Computational geometry, surface mesh overlay, common refinement, robustness, parallel algorithms

1 Introduction

Many emerging scientific and engineering applications involve interactions of multiple surface meshes. An example is multiphysics simulations, such as simulations of foil flutter of airplane wings in aerodynamics or of blood flows in biomedical applications. These multiphysics systems nowadays constitute one of the greatest challenges in computational science, and they are also very computational demanding and often can be solved only on parallel computers. In these applications, the different

[⋆] Current affiliation: Cisco Systems, Inc.

[⋆⋆] Corresponding author. Current affiliation: Department of Applied Mathematics and Statistics, Stony Brook University. Email: jiao@ams.sunysb.edu sub-systems have their own computational domains, which are discretized into volume meshes. A sub-system communicates at the boundaries with other sub-systems, and an integrated simulation of the entire system requires accurate and conservative data exchange at these common boundaries or interfaces. An interface between these components has more than one realization, one for each subdomain abutting the interface. These surface meshes are in general non-matching with differing connectivities and geometry. Many computer graphics applications, such as texture mapping and shape morphing, also involve correlating multiple surface meshes. Correlating these non-machines meshes is an important and challenging task.

There are at least two different types of techniques for correlating surface meshes. The first and the simpler type is *mesh association* [6], which maps the vertices of one mesh onto a discrete surface defined by another mesh. The second and the more sophisticated type is *mesh overlay* [8, 9, 10], which computes a finer mesh, called the *common refinement* or *common tessellation*, whose vertices are the "intersections" of the edges of the input meshes and whose faces are the "intersections" of the faces of the input meshes. While mesh association is useful sometimes, such as for pointwise interpolation, the mesh overlay provides a more comprehensive correspondence of the meshes and can enable more accurate data transfers across different meshes in scientific applications [7].

The subject of this paper is to develop a robust algorithm for mesh overlay that can be used in large-scale scientific simulations, especially on parallel computers. In [8], Jiao and Heath proposed an algorithm for overlaying two surface meshes. Albeit the original algorithm was efficient and robust, it was difficult to implement and parallelize. The goal of this paper is to overcome the complexity and the inherently sequential nature of the original algorithm for mesh overlay.

The first contribution of this paper is the simplification of the serial algorithm while maintaining the accuracy and robustness. The new algorithm introduces a high-level primitive, *face-face intersection*, which not only simplifies the implementation but also streamlines the logic for analyzing potential inconsistencies. Secondly, we introduce a new safeguarded primitive for projecting a vertex onto a discrete surface, and also make mesh association a preprocessing step of mesh overlay for improved robustness and flexibility. Thirdly, our new algorithm can be applied to partially overlapping meshes. Even for meshes that have disparate topologies, the new algorithm can build correspondence whenever possible, unlike the original algorithm, which required the input meshes to have the same topology and matching features.

The remainder of the paper is organized as follows. Section 2 defines some basic terminology and reviews the original algorithm for surface mesh overlay and some other related work. Section 3 describes the modified primitives and the modified serial algorithm. Section 4 describes the parallel algorithm. Section 5 contains some preliminary results with our algorithms. Section 6 concludes the paper with a brief discussion on future research directions.

2 Background

2.1 Terminology

We first define some terms that are used throughout this paper. A *surface mesh* is a discretization or tessellation of the geometry of a surface. We refer to the topological objects of a mesh as *cells*. The 0-D cells are called *vertices*, the 1-D cells *edges*, and the 2-D cells *faces*. Each d-dimensional cell σ has a *geometric realization* in $\mathbb{R}^3$, denoted by $|\sigma|$. The realization of an edge is the line segment between its two vertices, and that of a triangle or quadrilateral is a linear or a bilinear patch bounded by their edges. The *realization* of a mesh M, denoted by $|M|$, is the union of the realizations of its faces. We refer the $|M|$ as a *discrete surface*.

A mesh R is a *refinement* of a mesh M if their geometric realizations $|R|$ and $|M|$ are homeomorphic (i.e., there is a mapping between them that is bijective and continuous and whose inverse is also continuous) and every cell of M is partitioned into one or more cells of R. Given two meshes whose realizations are homeomorphic, a *common refinement* (or *common tessellation*) of them is a mesh that is a refinement of both given meshes. For meshes that are not homeomorphic globally, a *common refinement* may be defined for the subsets of the input surfaces that are homeomorphic. We refer to the 0-D, 1-D, and 2-D cells of R as *subvertices*, *subedges*, and *subfaces,* respectively, and refer to them collectively as *subcells*.

An *overlay* of two meshes is special type of common refinement. A subvertex of an overlay is either a vertex of the input meshes or an "intersection" of a blue and a green edge. The subedges of the overlay are the intervals in blue and green edges between the subvertices. A subface corresponds to a 2-D "intersection" of a blue and a green face. Figure 1 shows a blue and a green mesh and their overlay. The subfaces in the overlay are polygons with up to eight edges, which are inconvenient for many applications. A common-refinement mesh may further subdivide these subfaces into triangles. A cell in the blue or green mesh *hosts* one or more subcells of a common refinement. The lowest dimensional blue or green cell that contains a subcell is called the blue or green *parent* of the subcell, respectively. Each subcell has two realizations, one in each of its parents. Through the subcells, the common refinement defines a piecewise one-to-one correspondence between the input discrete surfaces, which can be used in applications such as exchanging data between the surface meshes accurately and conservatively.

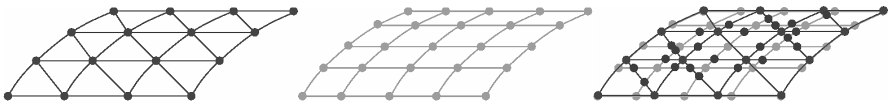
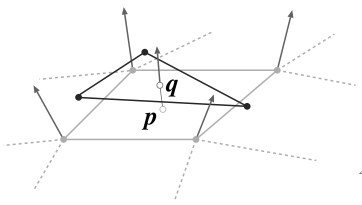
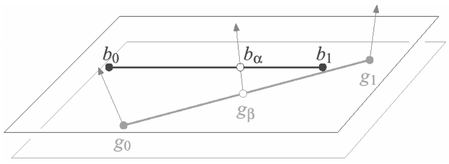

(a) Input blue mesh. (b) Input green mesh. (c) Overlay mesh.

Fig. 1. Sample blue (a) and green (b) meshes and their overlay mesh (c).



(a) Point projection.



(b) Edge-edge intersection.

Fig. 2. Illustrations of projection and intersection primitives.

2.2 Basic Primitives

The surface mesh overlay has two basic primitives. The first is *point projection*, which maps a point on one discrete surface onto a nearby point on another, and vice versa. Different types of mapping can be used, such as nearest-point projection. In [8], a nearly orthogonal projection was defined through interpolating the normals at green vertices using the shape functions of the green mesh, where the vertex normals can be given as an input or be estimated numerically. Let us first define the projection from $|G|$ to $|B|$ and then use its inverse to define the projection from $|B|$ to $|G|$. For any point $\mathbf{p} \in |G|$, let its projection $\mathbf{q} \in |B|$ be $\mathbf{p} + \gamma\mathbf{d}(\mathbf{p})$, where $\mathbf{d}$ is the unit projection vector at $\mathbf{p}$ obtained from interpolating the green vertex normals (c.f. Fig. 2(a)). Let $\mathbf{g}_i$ denote the ith green vertex and $\mathbf{d}_i$ its associated projection direction. Let ϕ_i denote the shape function corresponding to the ith vertex of the green mesh. For any point $\mathbf{p} = \sum_i \phi_i\mathbf{g}_i$, its projection direction $\mathbf{d}$ is $\sum_i \phi_i\mathbf{d}_i/\|\sum_i \phi_i\mathbf{d}_i|$. Let $\mathbf{b}_j$ denote the jth vertex of a blue face $b \in B$, and ψ_j the corresponding basis function associated with the vertex. The projection of $\mathbf{p}$ onto b is then the solution of the system of equations

$$\sum_j \psi_j\mathbf{b}_j - \gamma\mathbf{d}(\mathbf{p}) = \mathbf{p}. \tag{1}$$

Because ψ_j is a function of the two local coordinates of b, this system has three equations (one for each component of the physical coordinates) and three unknowns (γ and the two local coordinates in b). This system of equations is linear for triangular meshes and bilinear for quadrilateral meshes. Its solution gives the local coordinates of the projection, which determines whether $\mathbf{q}$ is in the interior, on an edge, at a vertex, or in the exterior, of b. The projection from a point $\mathbf{q} \in |B|$ to a green face $g \in G$ is computed by solving the system of equations

$$\sum_i \phi_i\mathbf{g}_i - \gamma\sum_i \phi_i\mathbf{d}_i = \mathbf{q}, \tag{2}$$

which also has three equations and three unknowns. This system is in general nonlinear and can be solved using Newton's method.

The second primitive is *edge-edge intersection*. Given a blue edge b and a green edge g, the primitive computes the "intersection" of them. Again, the intersection has two realizations, one in $|b|$ and one in $|g|$. Let the blue edge be $b = \mathbf{b}_0\mathbf{b}_1$ and the green edge $g = \mathbf{g}_0\mathbf{g}_1$. We parameterize them as $\mathbf{b}_0+\alpha(\mathbf{b}_1-\mathbf{b}_0)$ and $\mathbf{g}_0+\beta(\mathbf{g}_1-\mathbf{g}_0)$, respectively (c.f. Fig. 2(b)). Let $\mathbf{d}_0$ and $\mathbf{d}_1$ denote the unit normal vectors at $\mathbf{g}_0$ and $\mathbf{g}_1$, respectively. The green realization of the intersection is contained in the plane containing b with normal direction $\mathbf{n} = (\mathbf{b}_1 - \mathbf{b}_0) \times (\mathbf{d}_0 + \beta(\mathbf{d}_1 - \mathbf{d}_0))$. Therefore, the parameter β of the intersection in g can be obtained by solving

$$\mathbf{n}^T(\mathbf{g}_0 + \beta(\mathbf{g}_1 - \mathbf{g}_0) - \mathbf{b}_0) = 0, \tag{3}$$

which is a quadratic equation with the unknown variable β, and we in general choose the positive solution with the smaller magnitude. After solving for β, the parameter α of the intersection in b is the solution to the linear equation

$$\mathbf{m}^T(\mathbf{b}_0 + \alpha(\mathbf{b}_1 - \mathbf{b}_0) - \mathbf{g}_0) = 0, \tag{4}$$

where $\mathbf{m} = (\mathbf{g}_1 - \mathbf{g}_0) \times (\mathbf{d}_0 + \beta(\mathbf{d}_1 - \mathbf{d}_0))$. A solution corresponds to an actual edge intersection if $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ simultaneously. The intersection is in the interior of b if $0 < \alpha < 1$ and at a vertex if $\alpha = 0$ or 1; similarly for g. Because of the nonlinearity of some of the primitives, extra care must be taken due to the inevitable numerical errors. We refer readers to [8, 9] for more detail.

2.3 Original Edge-Based Algorithm

The original mesh-overlay algorithm in [8] proceeds in two phases. The first phase determines the subvertices of the overlay and their parents using the aforementioned primitives and sorts the subvertices on their parent edges. Specifically, it traverses the edges of B in a breadth-first order. For each edge $b \in B$ it locates the subvertices within b using edge-edge intersection and determines the parents of the end-vertices of b using point projection. This operation is repeated until all the blue edges have been processed. To start the algorithm, it is necessary to know the green parent of a *seed* blue vertex within each connected component of B, and the parent of the seed can be obtained by brute-force. Thereafter, the subvertices are sorted in their green parent edges, and the green vertices are projected onto blue faces. During the first phase, robustness is achieved through thresholding and a set of consistency rules [9].

The second phase of the algorithm uses the parent information of the subvertices to create the subfaces of the overlay using divide-and-conquer. In particular, it first creates a list of the subvertices in the edges of a blue face in counterclockwise order. It then identifies a sequence of green subedges to subdivide the face into two parts. The subdivision procedure is then applied recursively to each part until no part is further divisible.

2.4 Other Related Work

There are various algorithms to construct mesh association and mesh overlay. For mesh association, a few methods were reviewed in [6], including a locality-based search called *vine search*. The key primitive in mesh association is point projection, which commonly uses a nearest-point projection or orthogonal projection. Recently, van Brummen proposed a smoothed normal field for the projection to improve accuracy [17]. A related work in computer graphics is the construction of homeomorphisms. In particular, Chazal et al. [2] defined a mapping (Orthomap) between two (n-1)-dimensional manifolds S and S' in $\mathbb{R}^n$, which associates a point $\mathbf{x} \in S$ with the closest point on S' lying on the line passing through the normal to S at $\mathbf{x}$. This mapping is homeomorphic if the minimum feature sizes of S and S' both exceed $h/(2-\sqrt{2})$, where h is the Hausdorff distance between S and S'. More recently, Chazal et al. [3] proposed a ball mapping that relaxed the requirements of the minimum feature sizes. However, these mappings are presently limited to smooth surfaces and are not directly applicable to discrete surfaces.

For mesh overlay, some algorithms have also been developed in the computer graphics community. Some of them use a virtual reference surface, such as a plane or sphere, onto which the input meshes are first projected. Such methods have sparked extensive research on spherical and planar parameterizations of surfaces. An algorithm to compute the overlay of planar convex subdivisions was proposed by Guibas and Seidel in [5]. Alexa [1] proposed a method to compute overlay on a unit sphere, which is limited to overlaying genus-0 polyhedral meshes. More recently, Schreiner et al. proposed a method to construct a common refinement (which they referred to as a *meta-mesh*) of two triangle meshes [15]. The method partitions the two surfaces into a corresponding set of triangular patches by tracing a set of corresponding paths and then creates progressive mesh representations of the input meshes. A number of researchers have also developed other types of inter-surface correspondences. For example, the zippering algorithm of Turk and Levoy [16] establishes a correspondence between nearby overlapping surface patches to "zipper" them into one mesh. In [11], Kraevoy and Sheffer developed an inter-mesh mapping by constructing a common base mesh, which in general is not a common refinement. These methods are advantageous for their specific application domains but are not well suited for scientific applications that require accurate correspondence for physically meaningful and numerically accurate data transfers.

In the scientific community, parallelization of mesh correlation have drawn some attention for large-scale simulations, but they have thus far mostly focused on mesh association instead of mesh overlay. In particular, Farhat et al. proposed a parallel algorithm for mesh association in [12]. Plimpton et al. proposed a so-called rendezvous algorithm for mesh association [14], which constructs a separate partitioning of the input meshes for better load balancing. The MpCCI (www.mpcci.com) is a commercial software package that supports parallel mesh association and data interpolation. Different from these previous efforts, the focus of this paper is on mesh overlay for accurate and conservative data transfer and their effective parallelization.

3 Simplified Serial Algorithms

The original algorithm in [8] was efficient and robust, but it is difficult to implement and parallelize. Part of the complexity is due to its edge-based traversal of the meshes. In this section, we propose a modified algorithm to traverse the blue mesh face-by-face, which simplifies the implementation, streamlines the logic, and will also ease its parallelization. We also propose some enhancement to the primitives to improve robustness.

3.1 Modified Primitives

Safeguarded Point Projection

Our first modification is to introduce a safeguard for point projection to improve robustness. The point-projection primitive constructs a nearly orthogonal projection by interpolating the green vertex normals. If the normals change rapidly, such as near features or large curvatures, this primitive may project a vertex onto more than one face and lead to ambiguity.

To address this issue, we define a *volume of influence* for each green face g, denoted by $V(g)$, so that the blue vertex b projects onto g only if b falls within $V(g)$. We define $V(g)$ in a way consistent with the point projection primitive. Specifically, the interpolated projection directions along an edge form a bilinear surface, which subdivide $\mathbb{R}^3$ into two half-spaces, and $V(g)$ contains all the points that lie in the same half-space as the face with respect to the bilinear surfaces of all the edges of g. If the edges of the face are in counter-clockwise order, then a point in $V(g)$ must lie to the left of bilinear surfaces for all the edges of g.

Whether a particular point is to the left of the bilinear surface can be determined by the sign of the vector from the point to its orthogonal projection onto the bilinear surface. Given a point $\mathbf{p}$, let its projection be $\mathbf{q} = \mathbf{p} + \gamma\mathbf{n}$, where $\mathbf{n}$ is normal to the bilinear surface (c.f. Fig. 3(a)). We parameter $\mathbf{q}$ by α and β, i.e.,

$$\mathbf{q} = \alpha\mathbf{g}_1 + (1-\alpha)\mathbf{g}_0 + \beta(\alpha\mathbf{d}_1 + (1-\alpha)\mathbf{d}_0), \tag{5}$$

where α corresponds to its parameterization in the green edge $\mathbf{g}_0\mathbf{g}_1$ and β is the parameterization along the green normal directions. Therefore, $\mathbf{t}_1 = (\mathbf{g}_1 + \beta\mathbf{d}_1) - (\mathbf{g}_0 + \beta\mathbf{d}_0)$ and $\mathbf{t}_2 = \alpha\mathbf{d}_1 + (1-\alpha)\mathbf{d}_0$ correspond to two tangent vectors of the bilinear surface at $\mathbf{q}$, and the normal to the bilinear surface at $\mathbf{q}$ is

$$\mathbf{n} = \mathbf{t}_1 \times \mathbf{t}_2 = (\mathbf{g}_1 - \mathbf{g}_0 + \beta(\mathbf{d}_1 - \mathbf{d}_0)) \times (\alpha\mathbf{d}_1 + (1-\alpha)\mathbf{d}_0). \tag{6}$$

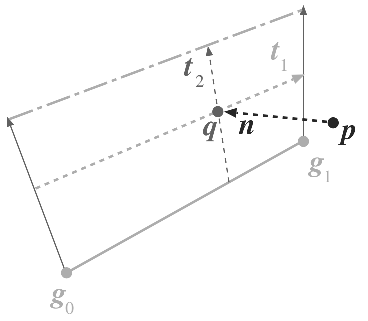
Plugging (5) and (6) into $\mathbf{q} = \mathbf{p} + \gamma\mathbf{n}$, we obtain a system of three equations for three unknowns. A negative γ would indicate that $\mathbf{q}$ is to the left of the green edge. To avoid degeneracy, we scale $\mathbf{d}_0$ and $\mathbf{d}_1$ by a factor c so that

$$\left(\mathbf{g}_1 - \mathbf{g}_0 + c(\mathbf{d}_1 - \mathbf{d}_0)\right)^T (\mathbf{g}_1 - \mathbf{g}_0) > 0. \tag{7}$$

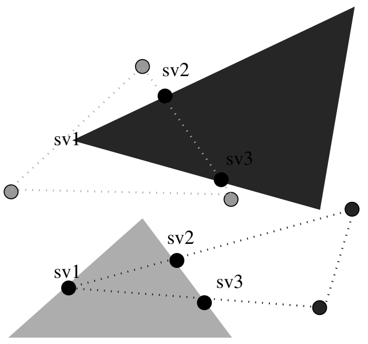
This scaling does not change the sign of γ. In addition, the resulting β must also satisfy

$$\left(\mathbf{g}_1 - \mathbf{g}_0 + \beta c(\mathbf{d}_1 - \mathbf{d}_0)\right)^T (\mathbf{g}_1 - \mathbf{g}_0) > 0. \tag{8}$$

If this condition is violated, then the point lies outside the volume of influence of g. If a blue vertex falls into the volumes of influence of multiple green faces, then



(a) Safeguard for point projection.



(b) Face-face intersection.

Fig. 3. Illustrations of safeguarded point projection and face-face intersection.

we project it onto the closer face. If a blue vertex falls outside of the volumes of influence of all the green faces, then the surfaces mismatch locally and we will not project the vertex onto the green mesh.

Face-Face Intersection

To facilitate face-based traversal, we define a new primitive *face-face intersection*, which identifies the subvertices hosted by a pair of green and blue faces. Specifically, this primitive computes the intersections of the blue and green edges of the faces and determines the projections of the blue vertices onto the green face and vice versa. Face-face intersection therefore is higher level than the point-projection and edge-edge-intersection primitives. Naively, given a blue face b and a green face g, one can independently compute the pairwise intersections of the edges of b and g, compute the projections of the vertices of b onto g, and vice versa. However, such an approach may potentially not only introduce redundant computations but also lead to inconsistencies due to numerical errors.

To avoid these potential redundant computations and inconsistencies, we compute the intersection in two passes (c.f. Fig. 3(b)). First, we determine the projections of the vertices of b onto $|g|$ and the projections of the vertices of g onto $|b|$. In general, this is done using the safeguarded point-projection primitive outlined earlier. If the projection has already been determined previously, then this computation is skipped for efficiency. Thereafter, we compute the intersection of a pair of blue and green edge by checking the barycentric coordinates of the vertices and compute the intersections only if the two blue vertices fall onto different sides of the green edge and the two green vertices fall onto different sides of the blue edge. In addition, we skip the computation of the intersection of a pair of edges if it has already been computed previously. For robustness, we also compare the normals of the blue and green faces and reject the intersection if their face normals point opposite to each other (i.e., their dot product is negative).

3.2 Face-Based Serial Algorithm

We now present our new face-based algorithm for overlaying surface meshes using the modified primitives. The algorithm is composed of three steps, as we explain in the following subsections. During the process, we pay special attention to the robustness in the presence of numerical errors.

Projecting blue vertices onto green surface

The projection and intersection primitives of mesh overlay in general assume the surface meshes are close to each other. This assumption is valid for most physical applications, but it may be violated sometimes near large curvatures. For best generality and robustness, we first project the blue vertices onto the green discrete surface. We use the safeguarded point projection to project the blue vertices onto $|G|$. For each blue vertex b, we identify a unique point b' on $|G|$ and store the green face that b projects onto and the local coordinates of b' in that green face. The projected points along with the connectivity of B compose a mesh B' and a discrete surface $|B'|$. If a blue vertex projects outside $|G|$, we omit the blue vertex in later steps so that partially overlapping meshes can be overlaid.

Essentially, this projection step is *mesh association*, which we can solve using a number of different techniques, such as *vine search* [6]. For ease of implementation, our present implementation uses a kd-tree [4]. The kd-tree is a multidimensional search tree for points in k-dimensional space, where $k = 3$ in our case. We insert the centroids of the green faces into a kd-tree. We then iterate through the blue vertices. For each blue vertex b we extract from the kd-tree the green faces that are nearby and project b onto them. For better efficiency, an optimized implementation can take advantage of locality similar to vine search. In particular, starting from a blue face with a known intersection with a green face, one can visit its incident blue face c and attempt to project the blue vertices of c onto nearby green faces. This procedure constitutes a face-based version of the vine search.

Separating out the projection step from the remainder of the overlay algorithm has a number of advantages. First, since this step is mesh association, which may suffice for some simpler applications such as point-wise interpolation, the later steps of the algorithm may be skipped for those applications. More importantly, this modification allows enhancing the robustness of later steps in a number of ways. During the process, we can perform additional pre-processing steps, such as aligning the features, smoothing the normal directions of the green vertices, or optimizing the projections of the blue vertices. Furthermore, for applications with moving meshes, the projections of the blue vertices can be updated efficiently from the previous projections. Finally, after the projection step, the discrete surface $|B'|$ will in general be closer to $|G|$. We therefore will use the mesh B' instead of B in the later steps, which will substantially reduces the chances of inconsistencies.

Locating subvertices

This step identifies the subvertices and their blue and green parents. A subvertex is either a vertex in one of the meshes or is the intersection of a blue edge and a green edge. We identify the subvertices by traversing the blue faces and for each blue face apply the face-face intersection primitive to the nearby green faces. To identify the nearby green faces of a blue face b (contained in $|B'|$), we first make a list of the green faces that are projected onto by the vertices of b. For each green face g that intersects with b, we add a neighbor face of g, say h, into the list if $h \cap g$ intersects with b. When applying the face-face intersection primitive, we minimize redundant computations as much as possible as discussed previously. In particular, we never compute the intersection of a blue and a green edge more than once.

The non-linear primitives for mesh overlay can be solved only approximately. This may lead to potential inconsistencies of the subvertices. Such inconsistencies must be identified and resolved properly for a valid overlay of the input meshes. As noted in [8], there are three common types of inconsistencies that may arise. The inconsistencies and their remedies are shown in Fig. 4. In the first two cases, numerical computations return two intersections between the blue edge b and green edges, with the difference that in Case (a) one of the intersection is close to (or at) the origin of b, say $\mathbf{b}_0$, whereas in Case (b) both the intersections are far away from $\mathbf{b}_0$. Case (a) occurs only when $\mathbf{b}_0$ is too close to one of the green edges. We perturb $\mathbf{b}_0$ onto its nearby green edge (by reassigning the green parent of $\mathbf{b}_0$). Case (b) occurs only when b is too close to the common vertex of the green edges that intersect b. We resolve this inconsistency by perturbing the intersection onto the green vertex. In Case (c), numerical errors cause b to fall into an artificial gap at a given green vertex between the green edges so that no edge intersection is found. Case (c) happens only when b is too close to the green vertex, and we also perturb the intersection to the green vertex in this case.

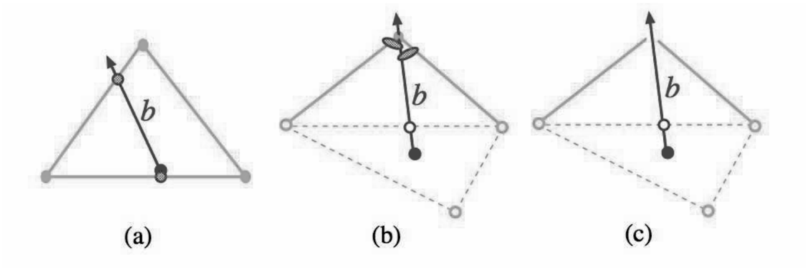


Fig. 4. Different types of inconsistencies and their remedies.

Creating and triangulating subfaces

We enumerate the subfaces for the blue mesh and green mesh simultaneously. This step is broken down into three sub-steps. First, we sort the subvertices on their host edges in both input meshes. Sorting is done based on the distance of a subvertex from the starting vertex of the face in counter-clockwise direction, with the subvertices lying farther away from the starting vertex listed earlier in the list. During the second sub-step, we create polygonal subfaces. Unlike the original algorithm, we directly construct the subface for each pair of intersecting faces from their shared subvertices and subedges. Starting from a known subvertex, we traverse the subvertices hosted by the blue and green faces in counter-clockwise order, as illustrated in Fig. 5. The sorted lists of the subvertices within their parent edges are used to identify the ordering. The subfaces obtained from face-face intersections are polygons with up to eight edges, and they are inconvenient to use for most applications. In the third sub-step, we further triangulate each subface. We use the ear-removal algorithm [13] to triangulate the subfaces formed above.

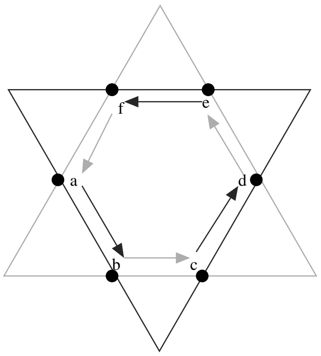


Fig. 5. Illustration of construction of subface.

3.3 Analysis and Comparison

Let N_b and N_g be the numbers of faces in the blue and green mesh, respectively. The first step traverses the nodal graph of B to locate the subvertices in its incident edges. With the simple implementation using kd-trees, the cost of building the kd-tree is $O(N_g log N_g)$ and the cost of each query is $O(N_g^{2/3} + s)$, where s is the maximum number of returned points. The later steps intersect each blue face b only with the green faces that are projected onto by b, so they take linear time in input plus output size. Therefore, the total cost of using the kd-tree is $O(N_b(N_g^{2/3} + s))$. However, when an optimized locality-based search is used during point projection, the total computational cost can be reduced to linear complexity in input plus output size, which is expected to be linear in the input size for well-shaped input meshes.

A main difference of the original and modified algorithms is that they use edge-based and face-based traversals, respectively. Face traversal allows replacing the lower-level primitive by the higher-level primitive, face-face intersection, which simplified the data structures and also simplified the resolution of inconsistencies. When using this new primitive, the inconsistencies could be resolved using the knowledge of the location of the subvertices relative to their blue and green parents. Introducing a higher-level primitive, as explained before, may potentially introduce some redundant computations, which we minimized through additional bookkeeping. The modified algorithm can also be more easily applied to partially overlapping meshes, which allows parallelization of the algorithm.

4 Parallel Algorithm

The main motivation behind modifying the original algorithm was to make mesh overlay easily parallelizable. The key challenges in the parallelization come from the fact that the input meshes are often partitioned independently of each other, so inter-processor communication is in general unavoidable. In particular, the resolution of inconsistencies along the partition boundaries requires coordinations across different processors. In addition, each processor may have a different numbering system for the elements and vertices that it owns, and the different numbering systems on different processors introduce additional implementation complexity. We present a relatively simple parallel algorithm, which follows the same control flow as the serial algorithm, with one additional communication step at the beginning and one at the end.

4.1 Communicating Overlapping Green Patches and Blue Ghost Layers

During this communication step, we collect all the green faces that may potentially intersect with the blue faces that are local to a processor. A naive approach is for each processor to broadcast its green partition to all the processors. Such an approach leads to too much communication overhead as well as memory requirement. Instead, we let each process compute the bounding box of each connected component of its blue partition and send the bounding boxes to all the other processors. Thereafter, each processor p_i sends to processor p_j the green faces that intersect with the bounding boxes of the blue partition on p_j. Note that each vertex along the partition boundary of the green mesh may be shared by multiple processors. Each instance of a shared vertex may be identified by the two-tuple of its partition ID and its local vertex ID within the partition. We assign a unique global ID to each shared vertex, taken to be the two-tuple ID of the instance with the smallest partition ID. When sending the green faces, we also send the global IDs of the shared vertices along the green partition boundaries to facilitate the identification of shared vertices.

Besides sending the green faces, each processor also collects the blue faces from the other processors that are adjacent to the faces of the blue mesh. This builds up a blue mesh with a layer of “ghost elements,” which will substantially simplify the resolution of inconsistencies as explained shortly. We implement the above communications using non-blocking sends and receives.

4.2 Applying Face-Based Serial Algorithm

After receiving the green faces from all the processors, we then apply the serial algorithm to compute the intersections. One potential strategy to compute the subvertices is to use the serial algorithm for every set of green faces received from the other processors, one at a time. The advantage of this method is that a processor can then overlap computation with communication. However, such an approach makes it very difficult to identify and resolve inconsistencies along partition boundaries and hence defeats the goal of the new algorithm, namely, to simplify the implementation.

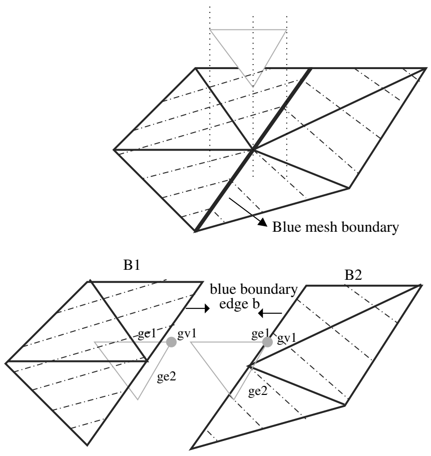


Fig. 6. Inconsistencies along blue partition boundary.

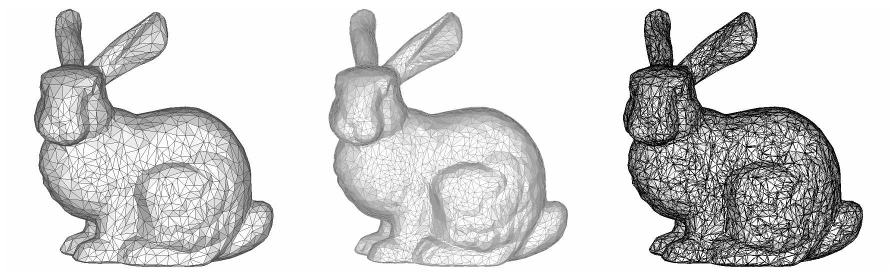
To overcome this disadvantage, we gather all the green faces from different processors first. We number the received vertices and faces using the local numbering scheme of the sender process, and compute the subvertices and the subfaces using the serial algorithm. For the green faces from the same sender, we resolve the inconsistencies in the same way as in the serial algorithm. The resolution of the inconsistencies along partition boundaries deserve special attention. If a blue face produces different intersections with the green faces on two different processors, as shown in Fig. 6, then some significant amount of communication would be required to exchange data back and forth to resolve them. In our algorithm, if two processes share a blue vertex or edge, then they would contain all the incident blue faces and the intersecting green faces. In the implementation, we further ensure that the primitives are invoked consistently on different processors to avoid inconsistent numerical results due to round-off errors. Therefore, the algorithm would produce identical results along partition boundaries on different processors, eliminating the needs of additional communication for resolving inconsistencies across different processors.

4.3 Redistributing Subfaces

After all the subvertices have been computed and the subfaces are created, a processor sends all the subvertices and subfaces to the senders of their green parent faces during the pre-communication step. Therefore, at the end of the algorithm every processor has a list of the subvertices and subfaces lying on its blue and green mesh partitions.

5 Experimental Results

We present some preliminary experimental results and demonstrate the performance of the new serial and parallel algorithms. Figure 7 shows the two meshes of the Stanford bunny and their overlay mesh computed using the new serial algorithm. The blue mesh contains 1,844 vertices and 3,471 triangles, and the green mesh contains 3,546 vertices and 6,944 triangles. The common refinement contains 20,853 vertices and 40,945 triangles. The original algorithm would have failed for this test due to non-matching features caused by the roughness of the meshes.



(a) Blue mesh. (b) Green mesh. (c) Common refinement.

Fig. 7. Blue and green meshes of Stanford bunny and their common refinement

To demonstrate our parallel algorithm, we used a relative simple geometry of two cylinders connected by conic patches. The blue mesh contains 5,903 vertices and 11,806 triangles, and the green mesh contains 22,730 vertices and 45,460 triangles. The two meshes are shown in Fig. 8. Figures 9 and 10 show the different partitions of the common refinement, where the colored edges indicate the input meshes and black edges correspond to the common refinement. Figure 11 shows a zoomed-in view of a blue partition. We conducted the experiments on the IA-64 Linux cluster at San Deigo Super Computing Center (SDSC), which has 262 nodes with dual 1.5 GHz Intel Itanium 2 processors. Figure 12 shows the speed-ups of our algorithm up to 16 processors. The achieved speed-ups were nearly linear. On a single processor, the current unoptimized implementation using kd-trees took about 24 seconds, which

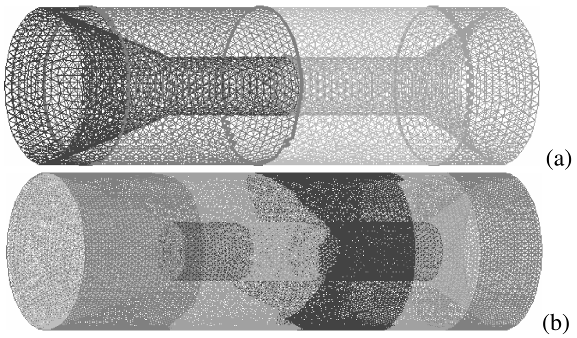


Fig. 8. Blue (a) and green (b) meshes for testing of parallel mesh overlay. Each mesh is decomposed into four partitions, indicated by the different colors.

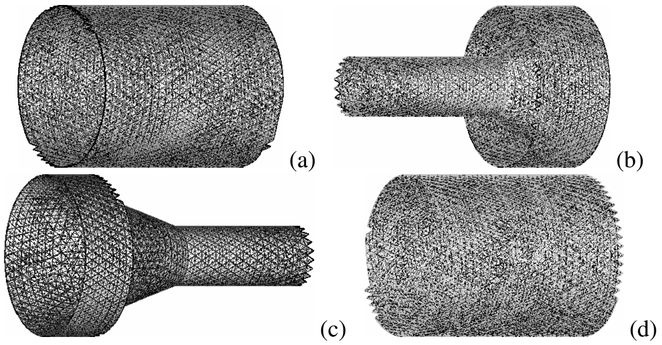


Fig. 9. Partitions of blue meshes and the common refinement.

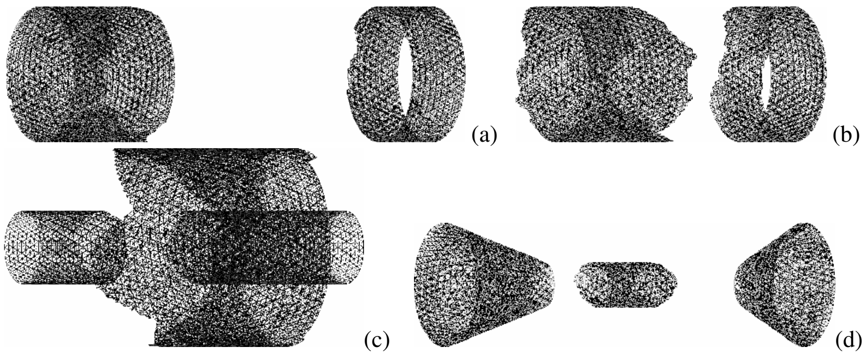


Fig. 10. Partitions of green meshes and their common refinement.

is relatively slow compared to the original algorithm. A primary cause of this slowdown of the serial algorithm is due to the use of kd-trees during the point-projection step. We are presently working on optimizing this step for better performance.

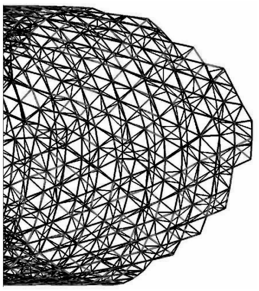


Fig. 11. Zoomed in view of common refinement.

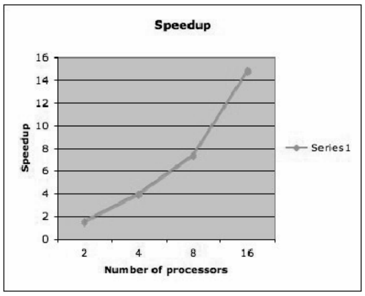


Fig. 12. Speed up of parallel algorithm.

6 Conclusion

In this paper, we presented a simplified serial algorithm and a parallel algorithm for overlaying two surface meshes, which has applications in many areas of computational science, such multi-physics or multi-component simulations. Different from the previous algorithm in [8], which used edge-based traversals, the new algorithms are face-based, which simplified the logic of the algorithm and its implementation. We also introduced a new safeguarded point-projection primitive for improved robustness of mesh overlay, and also defined a higher-level primitive, namely, face-face intersection. Our parallel algorithm uses the serial algorithm for the kernel, requiring only a pre-communication step to distribute the blue and green meshes and a post-communication step to redistribute the resulting subfaces, while retaining the robustness of the serial algorithm. We presented some preliminary experimental results with the new algorithms. Our present implementation is not yet optimized in terms of performance, but the experimental results are promising in terms of robustness and speed-ups. Our future research directions will focus on further improving the projection step of the algorithm, i.e., to project the blue vertices on the green discrete surface, in terms of more accurate treatment of singularities, smoothing of the projected blue vertices and the green projection directions, as well as improving its efficiency.

Acknowledgement. This work was supported by a subcontract from the Center for Simulation of Advanced Rockets of the University of Illinois at Urbana-Champaign funded by the U.S. Department of Energy through the University of California under subcontract B523819 and in part by the National Science Foundation grants ATM-0326431 and 0325046.

References

1. M. Alexa. Merging polyhedral shapes with scattered features. *Vis. Comput.*, 16(1):26–37, 2000.

2. F. Chazal, A. Lieutier, and J. Rossignac. Orthomap: Homeomorphism-guaranteeing normal-projection map between surfaces. In *ACM Symposium on Solid and Physical Modeling (SPM)*, pages 9–14, 2005.
3. F. Chazal, A. Lieutier, J. Rossignac, and B. Whited. Ball-map: Homeomorphism between compatible surfaces. Technical Report GIT-GVU-06-05, Georgia Institute of Technology, 2006.
4. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications*. Springer, second edition, 2000.
5. L. Guibas and R. Seidel. Computing convolutions by reciprocal search. In *SCG '86: Proceedings of the Second Annual Symposium on Computational geometry*, pages 90–99. ACM Press, 1986.
6. X. Jiao, H. Edelsbrunner, and M. T. Heath. Mesh association: formulation and algorithms. In *Proceedings of 8th International Meshing Roundtable*, pages 75–82, 1999.
7. X. Jiao and M. Heath. Common-refinement-based data transfer between nonmatching meshes in multiphysics simulations. *Int. J. Numer. Methods Engr.*, 61(14):2402–2427, 2004.
8. X. Jiao and M. Heath. Overlaying surface meshes, part i: algorithms. *Int. J. Comput. Geom. Appl.*, 14(6):379–402, 2004.
9. X. Jiao and M. Heath. Overlaying surface meshes, part ii: topology preservation and feature matching. *Int. J. Comput. Geom. Appl.*, 14(6):379–402, 2004.
10. X. Jiao and M. T. Heath. Efficient and robust algorithms for overlaying surface meshes. In *Proceedings of 10th International Meshing Roundtable*, 2001.
11. V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, 23(3):861–869, 2004.
12. N. Maman and C. Farhat. Matching fluid and structure meshes for aeroelactic computations: a parallel approach. *Comput. Struct.*, 54(4):779–785, 1995.
13. J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.
14. S. J. Plimpton, B. Hendrickson, and J. R. Stewart. A parallel rendezvous algorithm for interpolation between multiple grids. *J. Parallel Distrib. Comput.*, 64(2):266–276, 2004.
15. J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe. Inter-surface mapping. In *SIGGRAPH '04*, pages 870–877. ACM Press, 2004.
16. G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH '94*, pages 311–318. ACM Press, 1994.
17. E. van Brummelen. Mesh association by projection along smoothed-normal-vector fields: association of closed manifolds. Technical Report 1574-6992, Delft Aerospace Computational Science(DACS), Kluyverweg 1, 2629HS Delft, The Netherlands, 2006.