11-1-2015

An Immersed Boundary Geometric Preprocessor for Arbitrarily Complex Terrain and Geometry

Inanc Senocak
Boise State University

Micah Sandusky
Boise State University

Rey DeLeon
Boise State University

Derek Wade
Boise State University

Kyle Felzien
Boise State University

See next page for additional authors

Authors

Inanc Senocak, Micah Sandusky, Rey DeLeon, Derek Wade, Kyle Felzien, and Marianna Budnikova

An Immersed Boundary Geometric Preprocessor for Arbitrarily Complex Terrain and Geometry

INANC SENOCAC, MICAH SANDUSKY, REY DELEON, AND DEREK WADE

Department of Mechanical and Biomedical Engineering, Boise State University, Boise, Idaho

KYLE FELZIEN AND MARIANNA BUDNIKOVA

Department of Computer Science, Boise State University, Boise, Idaho

(Manuscript received 27 January 2014, in final form 18 November 2014)

ABSTRACT

There is a growing interest to apply the immersed boundary method to compute wind fields over arbitrarily complex terrain. The computer implementation of an immersed boundary module into an existing flow solver can be accomplished with minor modifications to the rest of the computer program. However, a versatile preprocessor is needed at the first place to extract the essential geometric information pertinent to the immersion of an arbitrarily complex terrain inside a 3D Cartesian mesh. Errors in the geometric information can negatively impact the correct implementation of the immersed boundary method as part of the solution algorithm. Additionally, the distance field from the terrain is needed to implement various subgrid-scale turbulence models and to initialize wind fields over complex terrain. Despite the popularity of the immersed boundary method, procedures used in the geometric preprocessing stage have received less attention. The present study found that concave and convex regions of complex terrain are particularly challenging to process with existing procedures discussed in the literature. To address this issue, a geometric preprocessor with a distance field solver was presented, and the solver demonstrated its versatility for arbitrarily complex geometry, terrain, and urban environments. The distance field solver uses the initial distance field at the immersed boundaries and propagates it to the rest of the domain by solving the Eikonal equation with the fast sweeping method.

1. Introduction

The immersed boundary (IB) method was first proposed by [Peskin \(1972\)](#) for use in biomedical engineering. The intended purpose was to simulate flow through a human heart valve as a fluid–structure interaction problem on a fixed structured mesh. In the original IB method, the effect of the moving or stationary boundaries was explicitly represented as a body-force term in the momentum equations. However, researchers have reported numerical stability issues associated with having an explicit body force in the equations. The direct forcing approach introduced by [Mohd-Yusof \(1997\)](#) and [Fadlun](#)

[et al. \(2000\)](#) provided an alternative formulation for the body force and eliminated the stability issues arising from the explicit body-force formulation. The popularity of the immersed boundary approach has increased markedly after the introduction of the direct forcing approach ([Mittal and Iaccarino 2005](#)). In the direct forcing approach, the body force to represent an immersed boundary is implicitly taken into account by a reconstruction step on the primitive variables at the cells cut by the solid boundaries.

There is a growing interest to adopt Cartesian immersed boundary methods to simulate environmental flows under realistic conditions ([Senocak et al. 2004](#); [Lundquist et al. 2010, 2012](#); [Kang et al. 2011](#); [DeLeon et al. 2012](#)). Simulations of wind over arbitrarily complex terrain are an active area of research because of the rapidly expanding wind energy field. However, IB methods have been primarily proposed for engineering fluid flow applications at modest Reynolds numbers and, therefore, they do not readily extend to atmospheric flows over arbitrarily complex terrain. Successful simulations of atmospheric flows over arbitrarily complex terrain necessitate reconstruction

Denotes Open Access content.

Corresponding author address: Inanc Senocak, Department of Mechanical and Biomedical Engineering, Boise State University, 1910 University Drive, Boise, ID 83725-2085.
E-mail: senocak@boisestate.edu

DOI: 10.1175/JTECH-D-14-00023.1

FIG. 1. A triangulated surface mesh fitted to the terrain of Buckman Springs.

methods and subgrid-scale models that are suitable for the underlying rough terrain resolved by a relatively coarse spatial resolution used in the computations. To make progress on this front for arbitrarily complex terrain, we need a versatile IB preprocessor that can calculate the geometric information needed for a variety of reconstruction methods. In addition, the preprocessor needs to calculate the distance field to an arbitrarily complex terrain because distance information is needed in several turbulence models, in initializing three-dimensional wind fields over complex terrain and extracting flow variables over a surface contour at a specified height above ground level for data analysis.

Computer implementation of the IB method inside an existing flow solver can be done with minor modifications to the rest of the solver. The nonintrusive nature of the implementation is one of the reasons why IB methods have become so popular for various applications. Researchers have adopted different interpolation methods (e.g., linear, quadratic, logarithmic) as part of the reconstruction of the primitive variables at the cells cut by an immersed boundary (Fadlun et al. 2000; Gilmanov and Sotiropoulos 2005; Choi et al. 2007; Mittal et al. 2008). In most validation studies, researchers have relied on analytical shapes (sphere, Witch of Agnesi Curve, sinusoidal ridges, etc.) to validate the underlying numerical method for fluid flow simulation. However, for arbitrarily complex geometries, such as a mountainous terrain or a three-dimensional complex geometry, a robust preprocessing program is needed to correctly tag the cells as fluid, solid, and immersed boundary nodes; to identify neighboring cells for interpolation; to bind an IB node to a surface element; and to calculate the distance to the surface for use in the interpolation part of the reconstruction step. The preprocessing stage for arbitrarily complex geometry is composed of algorithms from the computational geometry field. The

majority of published work on the IB method has focused on the numerical accuracy of the flow solution over analytical geometries. Description and computer implementation of the preprocessor for arbitrarily complex geometry has received scant coverage. An exception is the work of Gilmanov and Sotiropoulos (2005), who applied the immersed boundary method to complex geometries, such as an undulating fishlike body and a planktonic copepod, and described the algorithms used in treating the complex flexible immersed boundaries. Although Gilmanov and Sotiropoulos claimed their preprocessing procedure to be general, we find that their procedure does not readily apply to arbitrarily complex mountainous terrain. Choi et al. (2007) and Edwards et al. (2010) also point to problems with existing procedures and describe an alternative approach.

In what follows, we present a new procedure to handle arbitrarily complex geometry, terrain, and urban environments. Additionally, we apply the fast sweeping method (Zhao 2004) to solve the Eikonal equation to efficiently calculate the distance field from any complex geometry for the full flow domain. We test these two components of the preprocessor for challenging test cases to demonstrate the versatility of the geometric preprocessor. We do not perform any flow simulations in this study because the preprocessing stage for an arbitrarily complex terrain is imperative for correct implementation of any IB method and subgrid-scale models. Therefore, we focus entirely on detailing the steps in the geometric preprocessor and conduct a thorough investigation over different kinds of geometry.

2. Review of existing procedures

Consider a small portion of the terrain geometry as shown in Fig. 1, intersecting a background Cartesian mesh as illustrated in Fig. 2a. Note that the Cartesian mesh

spacing does not have to be uniform (i.e., $\Delta x \neq \Delta y \neq \Delta z$) First, we must identify all the Cartesian points that are in close proximity of the terrain. These are called *near boundary nodes*. This can be accomplished by sweeping through all triangular elements of the surface mesh and identifying all the Cartesian points that are within a search radius r_s from the centroid of the triangular element of interest. An optional bounding box can be created to eliminate Cartesian points irrelevant to the search problem, and the search can be distributed to multiple parallel processes. [Gilmanov and Sotiropoulos \(2005\)](#) suggest r_s to be approximately equal to the smallest Cartesian mesh dimension, which we find to be insufficient in practice.

Next, we sweep through all the near boundary nodes to distinguish them as being interior or exterior relative to the terrain. We find the centroid of all triangular elements within a search radius of r_s from each node. We create a position vector $\mathbf{d}$ from the centroid of the triangular element to the near boundary node at hand. We then perform a dot product between the position vector and the surface normal vector of the triangular element. The dot product is evaluated for every triangular element that is found to be within the search radius of a near boundary node. [Gilmanov and Sotiropoulos \(2005\)](#) suggest that if the dot product is greater than zero for *at least one* boundary node within the search radius, the node is exterior to the geometry. If the dot product is less than zero for *all* triangular elements, the node is interior to the geometry.

[Gilmanov and Sotiropoulos \(2005\)](#) state that the above procedure is general and applicable to arbitrarily complex geometry. However, we find that the above procedure is not entirely general for all concave configurations of a terrain. Depending on the resolutions of the surface terrain, and the Cartesian mesh at hand, we are not able to separate a node as exterior or interior to a terrain for some challenging concave configurations following their procedure. [Choi et al. \(2007\)](#) also report that the above procedure does not work for some complex geometries.

A problematic configuration, which can be common on a complex terrain surface mesh, is illustrated in 2D in [Fig. 2a](#). In this schematic we tag the nodes manually to illustrate how some of the existing procedures can fail. The filled circular markers show the near boundary nodes solid points, and the filled square markers show the immersed boundary nodes upon which a reconstruction scheme is to be imposed.

Application of the interior/exterior identification stage of the above procedure to the grid point of interest identified in [Fig. 2a](#) will result in two iterations of the dot product check, as two triangular elements fall within the

search radius. Triangular element 1 gives $\mathbf{d} \cdot \mathbf{n} > 0$, terminating the identification state, incorrectly labeling the grid point as exterior.

To remedy the above logic failure for some complex geometries, [Choi et al. \(2007\)](#) and [Edwards et al. \(2010\)](#) proposed an angle-weighted pseudonormal vector, which essentially averages the variations in the local surface normal vector to ensure smoothness of the underlying geometry. However, as also mentioned by Choi et al., there is no guarantee that the angle-weighted normal vector fixes the issue for all cases. [Mittal et al. \(2008\)](#) use a slightly different method for distinguishing between solid, fluid, and IB nodes, which involves finding the closest triangular element to a node and then taking the dot product of the position vector from the centroid of the triangular element to the node with the surface normal vector of the same triangular element. We find that this procedure works as long as the surface mesh has nearly uniform triangular elements. However, when the surface mesh is composed of large and small triangular elements, the procedure also fails to correctly tag all the points.

[Figure 2b](#) displays an area in which a procedure described in [Mittal et al. \(2008\)](#) could fail. In this convex scenario, test node 1 would wrongly pick the triangular element to its left because its centroid is the closest, despite being positioned above a different triangular element. Consequently, the dot product with the surface normal of the triangular element would cause test node 1 to be wrongly tagged as solid. Therefore, identifying triangular elements based on the closest centroid is not a fail proof logic.

[Figure 2b](#) illustrates another potential problem that arises when binding a node to a surface element. In the reconstruction stage of the IB method, boundary conditions need to be assumed on a surface element. In analyzing test node 2, we see that distance to the surface should be the Euclidean distance calculated between the test node and the vertex. However, because of binding the test node with a surface triangular element, the distance can be wrongly calculated as the shortest distance between the test node and the plane that includes the triangular element as illustrated in [Fig. 2b](#). This issue does not cause the IB method to fail, but it introduces geometry errors by altering the underlying geometry, which would affect both the reconstruction step and the calculation of the distance field using the Eikonal equation. In situations like this, an additional logic is needed to decide whether the appropriate distance is to the triangular element, the edge, or the vertex.

We emphasize that these logic failures only occur for some portion of the complex terrain. [Figure 3](#) demonstrates the result of these logic failures as four dips in the

FIG. 2. The 2D schematics illustrating inherent discrepancies in published work. The distance vector $\vec{d}$ is from the centroid of the triangular element to the near boundary node, and the surface normal $\vec{n}$ is from the triangular element. (a) Concave region and (b) convex region.

FIG. 3. Isosurface visualization of the solid node flags for a section of complex terrain. Cylindrical dips from the surface indicate unflagged nodes where the interior/exterior logic fails.

isosurface visualization of solid/interior node flags for only a portion of Buckman Springs, California, terrain. For a reasonably large complex terrain area (e.g., $10\text{ km} \times 10\text{ km}$), these dips result from errors in the solid/fluid node tagging process and can be numerous. These errors may go unnoticed, which will result in missing or incorrect geometric information for the IB method, and will also pollute the calculation of the distance field through the solution of the Eikonal equation.

We should emphasize that the deficiencies we have identified in the above procedures may not manifest in all geometries. The resolutions of the Cartesian and the triangular surface meshes relative to each other can cause these inherent discrepancies to go undetected. We have experienced this issue in our own practice. In the following section, we introduce a new procedure that addresses the shortcomings described above.

3. Description of the geometric preprocessor

Our new procedure to obtain the essential geometric information (i.e., node tagging, distance to surface from nodes used in the reconstruction, binding an IB node with a surface element) to implement the immersed boundary method benefits from various components discussed in different studies (Gilmanov and Sotiropoulos 2005; Mittal et al. 2008; Choi et al. 2007; Sunday 2001a; Eberly 2008; Holcombe 2012).

In our practice, we obtain terrain maps from the U.S. Geological Survey website and convert it to a digital elevation model (.dem) file. The publicly available MICRODEM software (Guth 2013) can be used to crop a smaller area of interest from a DEM map. We use DEM maps with 10-m resolution in the horizontal. We

then fit a triangulated surface mesh to the terrain using a function from MATLAB Central File Exchange (McDonald 2004) and save the surface mesh in stereolithography (STL) file format. An STL file contains the vertices, centroid, and surface normal of every triangular element in the mesh. Figure 1 shows the triangulated surface mesh for an area in Buckman Springs.

The first component of our procedure is to classify Cartesian mesh points as interior or exterior relative to the immersed geometry. Concave and convex zones resulting from the surface mesh of the complex geometry require a robust method. We developed our own procedure by using some of methods from Holcombe (2012). We follow a 2D approach for the interior/exterior classification step. We first define a 2D search radius that is a multiple of the norm of the maximum x and y spans of an STL triangle. We use a conservative search radius that is 4 times the norm. Note that the size of the search radius only affects the execution time. Next, a cylindrical slab of this radius is projected down in the z direction from each Cartesian point. Triangular elements of the surface mesh that fall within this slab are tagged, and then those elements are examined to identify the triangular elements that intersect a ray projection from the Cartesian point in the z direction.

There are a few methods for determining whether a ray projection intersects a triangle. Figure 4 helps illustrate the method presented by Sunday (2001a). This method involves, first, finding the intersection of the ray, projected in the z direction with the plane of the triangular element. After the intersection point is identified, an inclusion check is applied to find out whether this point is within the triangular element. Following Sunday (2001a), the intersection point can be parameterized in terms of s and t coordinates:

FIG. 4. Illustration of the methods used to determine whether a point is within a triangle.

$$s = \frac{(\mathbf{u} \cdot \mathbf{v})(\mathbf{w} \cdot \mathbf{v}) - (\mathbf{v} \cdot \mathbf{v})(\mathbf{w} \cdot \mathbf{u})}{(\mathbf{u} \cdot \mathbf{v})^2 - (\mathbf{v} \cdot \mathbf{v})(\mathbf{u} \cdot \mathbf{u})}, \quad (1)$$

$$t = \frac{(\mathbf{u} \cdot \mathbf{v})(\mathbf{w} \cdot \mathbf{u}) - (\mathbf{u} \cdot \mathbf{u})(\mathbf{w} \cdot \mathbf{v})}{(\mathbf{u} \cdot \mathbf{v})^2 - (\mathbf{v} \cdot \mathbf{v})(\mathbf{u} \cdot \mathbf{u})}. \quad (2)$$

As long as these parameters follow the inequalities $s \geq 0$, $t \geq 0$, and $(s + t) \leq 1$, the point is within the bounds of the triangle. For the interior/exterior classification step of our procedure, we rely on the winding algorithm presented in O'Rourke (1994) with an implementation similar to the one presented by Holcombe (2012). In this 2D approach, the triangle in question and the Cartesian point are projected onto the x - y plane by simply neglecting their Z coordinates. The same $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ are constructed, as in Fig. 4, for each of three vertices. The angles between $\mathbf{w}$ and $\mathbf{u}$, $\mathbf{w}$ and $\mathbf{v}$, and $\mathbf{u}$ and $\mathbf{v}$ are found from the dot product of the unit vectors as follows:

$$\begin{aligned} \theta_1 &= \arccos(\hat{\mathbf{w}} \cdot \hat{\mathbf{u}}), \\ \theta_2 &= \arccos(\hat{\mathbf{w}} \cdot \hat{\mathbf{v}}), \\ \theta_3 &= \arccos(\hat{\mathbf{v}} \cdot \hat{\mathbf{u}}). \end{aligned} \quad (3)$$

Note that quantities with a hat symbol ($\hat{\cdot}$) are unit vectors. If the conditions $\theta_1 \leq \theta_3$ and $\theta_2 \leq \theta_3$ hold true when applied to each of the vertices (i.e., V_0 , V_1 , V_2), then the point lies within the triangle in the x - y plane. If no

triangle intersects the 2D projection of the point, then the point is exterior and marked as fluid.

Note that a ray projected in the z direction may intersect multiple triangular elements for a watertight geometry. Therefore, we identify the triangular element closest to the Cartesian point in the z direction by comparing the z coordinate of the Cartesian node against the z coordinate of the centroid of the triangular elements that have been intersected. Once the closest triangle is identified, we perform a three-dimensional dot product between the surface normal $\mathbf{n}$ of the closest triangular element and the position vector $\mathbf{r}$ drawn from the centroid to the Cartesian node. If the dot product is greater than zero, then the Cartesian node is exterior to the geometry and tagged as a fluid node. Else, the node is interior to the geometry and is tagged as a solid node. Once all the Cartesian nodes are tagged as exterior (fluid) or interior (solid), we then sweep through the fluid nodes and tag any fluid node that has, at least in one direction, a neighboring solid node as an IB node.

This procedure does not produce any false outputs from the dot product, as the Cartesian point is always positioned directly above or below the triangular element, avoiding the problematic situations described in Fig. 2.

a. Preprocessing for the IB reconstruction step

In our approach the immersed boundary reconstruction method relies on interpolation along the vector pointing

from the closest point on the STL surface to the IB node. In this step, a separate logic is needed to bind the IB node to the closest surface element, which can be a triangle, an edge, or a vertex. Flow reconstruction can then be done on a line along this unit vector $\hat{\mathbf{n}}_{BV}$ that binds the IB node to the closest surface element. Depending on the flow regime, the reconstruction could assume a linear or a logarithmic profile.

After all of the IB nodes are determined, the shortest distance to the solid must be found for each IB node. This initial distance is then propagated to the rest of the fluid domain through the solution of the Eikonal equation with the fast sweeping algorithm of Zhao (2004). First, the closest triangle vertex is established for each IB node. Next, the triangles that share this vertex are identified and their indices are stored. A line that passes through the IB node and is parallel to the surface normal of the triangular element at hand is intersected with the plane that includes the triangular element. Equations (1) and (2) are used to determine whether the intersection point falls inside the triangular element. If the intersection point is within the triangular element, then the IB node is bound to that triangular element. The distance from the intersection point to the IB node is stored as the closest distance. If none of the triangles are intersected by the line that passes through the IB node, we move on to check the edges sharing the common vertex using an algorithm described in Eberly (2008) and Sunday (2001b). We calculate a vector $\mathbf{w}$ pointing from the vertex to the Cartesian IB node and separate unit vectors $\hat{\mathbf{u}}_i$ along each triangle edge with a common origin at the vertex. The shortest distance d to each edge is calculated as follows:

$$d = \min(\|\hat{\mathbf{u}}_i \times \mathbf{w}\|), \quad (4)$$

where the subscript i is an identifier index that refers to the set of edges that share the closest vertex to the IB node. Note that if the angle between $\hat{\mathbf{u}}$ and $\hat{\mathbf{w}}$ is greater than 90° for an edge, then the intersection point is not contained within the edge element, and that edge is ignored.

If we do not find an intersection point lying within any of the edges that share the closest vertex to the IB node, we then simply calculate the Euclidean distance from the IB node to the closest vertex. Whether it is a triangle, edge, or a vertex, we refer to the unit vector along the line that connects the intersection point on the closest element to the IB node as the binding vector $\hat{\mathbf{n}}_{BV}$.

When we applied these binding methods to the Stanford Bunny geometry, 68.4% of the IB nodes were bound to triangular elements, 31.5% were bound to edges, and 0.1% were bound to vertices. The respective

FIG. 5. A 3D example of the surface normal of a triangle intersecting the nearest Cartesian cell face after passing through the immersed boundary node.

proportions for the Buckman Springs terrain were 74.4%, 25%, and 0.6%. These percentages suggest that the IB node binding step is an important part of the preprocessor.

Once the IB node is bound to its closest element on the surface mesh, a line that passes through the IB node in the direction of binding unit vector $\hat{\mathbf{n}}_{BV}$ is extended toward the fluid domain. Figure 5 illustrates the situation for an IB node bound to a triangular element. The geometric information needed to interpolate values onto the projected line is extracted as follows.

- 1) Identify the closest Cartesian cell face intersected by the projected line. To accomplish this task, we solve the plane–line intersection problem for each face of the Cartesian cell and compare the Euclidean distances. The parametric equation of a 3D line is substituted into the standard vector equation of a plane. The Cartesian cell faces in the x , y , z directions will have the following unit normal vectors: $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$, respectively. Therefore, the parameter t of the parametric equation of the line can be found by

$$t = \frac{r_{P,i} - r_{BV,i}}{\hat{\mathbf{n}}_{BV,i} \cdot \hat{\mathbf{n}}_{BV,i}}, \quad (5)$$

where the subscript i indicates the direction, and no summation over the repeated indices is implied. Term $\hat{\mathbf{n}}_{BV,i}$ is the i component of the unit vector in the direction of the line. Term $r_{P,i}$ is a coordinate of a point on the plane, and $r_{BV,i}$ is the coordinate of reference point for the 3D line. Note that we calculate the parameter t for each face of the Cartesian cell. Regardless of their values, the quantity $r_{P,i} - r_{BV,i}$ is equal to a Cartesian mesh spacing δ_i in the direction of a unit normal vector belonging to a

FIG. 6. Contour plot of the solid nodes for the Stanford Dragon showing good detail. A 2D plane of the Cartesian mesh with one of every eight lines is also displayed.

cell face. We then make use of the parameter t of the parametric equation of the line. At this stage, we do not know which of three faces have been intersected first by the line. The closest Cartesian cell face is determined by the smallest Euclidean distance.

- 2) Find the coordinates of the intersection point on the cell face using the parametric equation of the line that passes through the IB node as follows:

$$\begin{aligned} x_{\text{int}} &= x_{\text{IB}} + \hat{\mathbf{n}}_{\text{BV},x}t, \\ y_{\text{int}} &= y_{\text{IB}} + \hat{\mathbf{n}}_{\text{BV},y}t, \\ z_{\text{int}} &= z_{\text{IB}} + \hat{\mathbf{n}}_{\text{BV},z}t, \end{aligned} \quad (6)$$

where subscripts int and IB represent the intersection point and the IB node, respectively, for each of the coordinate directions. The subscript BV indicates the unit binding vector as discussed earlier.

- 3) Calculate the weights to apply a bilinear interpolation on the flow variables on the cell face.

Knowing the flow variables at the intersection point and the immersed surface enables the reconstruction of flow variables (e.g., velocity components and scalar quantities) at the IB node using linear, logarithmic, quadratic, or any other interpolation methods (Fadlun et al. 2000; Gilmanov and Sotiropoulos 2005; Choi et al. 2007; Mittal et al. 2008).

b. Parallel implementation

The above procedure is implemented in parallel to accelerate the preprocessing stage. In our computer implementation, we partition the surface mesh in STL format in the same manner as the computational domain

(i.e., in the x direction), but with a buffer zone on both sides to make sure all the triangular element information pertinent to the process is available to each message passing interface (MPI) process. We then execute each section independently because communication between processors is not necessary. We successfully tested our parallel implementation up to 32 processors. We use bounding boxes to find where the terrain sits in the Cartesian domain, meaning the processors do not have to waste computer memory on empty space.

4. Distance field calculation

Signed distance fields and level set methods have been used in a number of applications, such as computer-generated water surfaces for visualization (Osher and Fedkiw 2003) and improved imaging for medical applications (J. Pu 2008). The distance field is especially important for our application because we wish to implement a variety of turbulence models to simulate winds over complex terrain. The distance from a surface is an essential piece of information to apply near-surface parameterizations in subgrid-scale (SGS) turbulence models. Additionally, the distance field information helps to impose specific wind profiles (e.g., power law) at lateral boundary conditions and also for solution initialization.

The level set method is a fast algorithm to calculate the signed distance from a surface. Two popular approaches to implement the level set method are the fast marching method (FMM) (Sethian 1999) and the fast sweeping method (FSM) (Zhao 2004). The FMM loops over the grid points closest to the surface and moves outward. FMM has the advantage of allowing for calculations of narrow bands of data near the surface when

FIG. 7. Composite view of the distance field around the Stanford Dragon and Bunny geometries. (a) A 2D plane of the Cartesian mesh with one of every eight lines shown can be seen below the dragon. (b) A 2D plane of the Cartesian mesh with one of every four lines shown is visible below the bunny. A portion of the surface mesh is also displayed over the bunny head.

only those data are desired. This approach can be advantageous for dynamic surfaces. The runtime complexity of FMM is $O[N \log(N)]$, where N is the number of mesh points. On the other hand, computer implementation of FSM is relatively easier as compared to FMM and has a runtime complexity of $O(N)$. Therefore, we use FSM as part of our geometric preprocessor.

a. Signed distance calculation and discretization

After the application of the immersed boundary preprocessor, we are left with a computational domain that is split into three parts: the solid, the fluid, and a layer separating the solid from the fluid. This separation layer contains the immersed boundary points that hold the

FIG. 8. Placement of IB nodes (blue squares) on Buckman Springs. IB nodes conform well to the STL surface. The fluid (green) points are all above the surface, and the solid (red) points are all below the surface.

shortest distance to the underlying geometry. However, we also need the distance field in the fluid points away from the surface for use in certain turbulence models and wind field initialization. Therefore, we propagate the distance field at the immersed boundary points to other mesh points using the FSM.

The signed distance function implicitly defines the terrain on which $\phi(\mathbf{x}) = 0$. The distance field is positive in the fluid domain and negative in the solid domain.

Following the derivation in Osher and Fedkiw (2003), the Eikonal equation is written as follows.

$$\|\nabla\phi\| = 1. \quad (7)$$

We follow the work of Zhao (2004) to numerically solve the Eikonal equation but allow a directionally non-uniform mesh (i.e., $dx \neq dy \neq dz$) in our formulation. The discretized form of the Eikonal equation in three dimensions can be written as follows.

$$\left(\frac{\phi_{i,j,k} - \phi_{x\min}}{dx}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_{y\min}}{dy}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_{z\min}}{dz}\right)^2 = 1$$

$$i=2, \dots, NX-1, \quad j=2, \dots, NY-1, \quad k=2, \dots, NZ-1, \quad (8)$$

where $\phi_{x\min} = \min(\phi_{i-1,j,k}, \phi_{i+1,j,k})$, $\phi_{y\min} = \min(\phi_{i,j-1,k}, \phi_{i,j+1,k})$, and $\phi_{z\min} = \min(\phi_{i,j,k-1}, \phi_{i,j,k+1})$. After solving

the above equation for $\phi_{i,j,k}$, we are able to calculate the signed distance at any mesh point. The distance field propagation requires initialization that is provided by the immersed boundary geometric processor, as stated earlier.

b. Fast sweeping algorithm

Now that we have a method to solve Eq. (7) at any point on our computational domain, we must select a method of iteration that will ensure convergence. Since the calculation at any given point depends on the distance values surrounding that point, it is not sufficient to arbitrarily cycle through the grid points in the domain. Initially, there are only distance values on one layer near the implicit surface that were calculated during the immersed boundary process. As in Zhao (2004), all other points outside the surface (not including our initialization layer) are set to a large positive number that will be updated later during the iteration process. We then sweep the domain and update the distance values, $\phi_{i,j,k}$, in the following way: $\phi_{i,j,k}^{\text{new}} = \min(\phi_{i,j,k}^{\text{old}}, \phi_{i,j,k}^{\text{calc}})$, where $\phi_{i,j,k}^{\text{calc}}$ is the value calculated in Eq. (8) for the current sweep iteration. This step ensures that the distance value at each grid point remains nonincreasing and that the value will only update when a smaller distance value has been calculated. The order for a single complete sweep of the domain is critical, and it is given as follows:

- (1) $i = 1: NX, j = 1: NY, k = 1: NZ$; (2) $i = NX: 1, j = 1: NY, k = NZ: 1$;
- (3) $i = NX: 1, j = 1: NY, k = 1: NZ$; (4) $i = 1: NX, j = 1: NY, k = NZ: 1$;
- (5) $i = NX: 1, j = NY: 1, k = NZ: 1$; (6) $i = 1: NX, j = NY: 1, k = 1: NZ$;
- (7) $i = 1: NX, j = NY: 1, k = NZ: 1$; (8) $i = NX: 1, j = NY: 1, k = 1: NZ$.

FIG. 9. (a) Distance field propagation for the complex terrain of Buckman Springs. (b) Distance field propagation for a portion of Oklahoma City urban environment. Cartesian mesh shows one of every eight lines.

The above loop order covers all possible sweep directions for a three-dimensional domain. As suggested in Bridson (2008), the sweep process may be repeated several times for increased accuracy.

5. Results

Our geometric preprocessor can handle mountainous terrain, urban environments, and complex geometry. We use 3D test models that are popular within the computer graphics community to demonstrate versatility of our preprocessor. To illustrate the success of the interior/exterior node tagging algorithm, a contour plot of the interior (solid) nodes for the so-called Stanford Dragon geometry can be viewed in Fig. 6. This image shows that the highly complex geometric detail of the dragon can be handled with our procedure.

Figures 7a,b presents computer-aided design (CAD) geometries of the Stanford Dragon and Bunny stored as

an STL file. These geometries are immersed inside a 3D Cartesian domain to calculate the geometric information in the vicinity of the surface. The distance field at the immersed boundary cells are then propagated using the fast sweeping method. From these images, one can see that the distance field emanates smoothly away from the surface without any kinks or disturbances. Kinks or disturbances in the distance field typically hint problems in the initial distance field that the geometric preprocessor calculates. The underlying Cartesian meshes used in the calculations are also displayed in these images. Once the interior/exterior classification is done, we place an IB node between a solid and a fluid node. A check that we perform on the preprocessor is to make sure there is always an immersed boundary node between a fluid and a solid node. To perform this check, we color-code each of the Cartesian mesh points in the vicinity of the geometry. Figure 8 shows that this condition is successfully satisfied for the

complex terrain of Buckman Springs. Figure 9a shows the terrain for a portion of the Buckman Spring area in California. The terrain was immersed into a Cartesian domain of approximately $8.8\text{ km} \times 6.8\text{ km}$ with horizontal and vertical mesh spacings of 35 and 10 m, respectively. The initial distance field is then propagated to the rest of the domain using the FSM approach. We can observe from contour plots that the shortest distance to the terrain is computed as expected.

Finally, we tested our procedure with different Cartesian mesh resolutions successfully. The bunny and the dragon geometries were both tested on meshes with 128^3 and 256^3 points. The size of the Cartesian domain was 100^3 m^3 for the bunny and $100 \times 60 \times 70\text{ m}^3$ for the dragon. The surface meshes had 88 753 triangular elements with an average edge length of 0.53 m, and they had 66 991 triangular elements with an average edge length of 0.27 for the bunny and dragon geometries, respectively. The Cartesian domain for the Buckman terrain covers a volume of $8800 \times 6800 \times 1270\text{ m}^3$. The Cartesian domain was discretized on meshes with $256 \times 192 \times 128$, 192^3 , and $512 \times 256 \times 192$ points. The surface mesh for the terrain had 9962 triangular elements with an average edge length of 27.04 m. For the urban environment shown in Fig. 9b, we used a single 256^3 mesh for a Cartesian domain of $80 \times 90 \times 50\text{ m}^3$ with a surface mesh of 987 523 triangular elements with an average edge length of 0.12 m.

6. Conclusions

We presented a geometric preprocessor for arbitrarily complex geometry, terrain, and urban environments that calculates the necessary geometric information needed to implement the Cartesian immersed boundary method within a flow solver. We identified potential discrepancies in published work and addressed those issues with a new procedure that combines elements from various studies. The preprocessor classifies Cartesian mesh points as fluid, solid, and immersed boundary nodes. It then proceeds to calculate the necessary information for reconstruction along a surface normal direction. Additionally, the preprocessor calculates the shortest distance to the surface from the Cartesian mesh nodes through the solution of the Eikonal equation with the fast sweeping method of Zhao (2004). We successfully demonstrated the versatility of the preprocessor for complex geometries, terrain, and urban environment. We expect our geometric procedure to help broaden the adoption of the immersed boundary method for atmospheric flows over arbitrarily complex terrain.

Acknowledgments. This material is based upon work supported by the National Science Foundation under

Grants 1056110, 1043107, and 1229709. The authors thank Donna Calhoun for the helpful discussions on the solution of the Eikonal equation, and Ralph Preffer for his help in revising the original manuscript.

REFERENCES

- Bridson, R., 2008: *Fluid Simulation for Computer Graphics*. A K Peters, 228 pp.
- Choi, J.-I., R. C. Oberoi, J. R. Edwards, and J. A. Rosati, 2007: An immersed boundary method for complex incompressible flows. *J. Comput. Phys.*, **224**, 757–784, doi:10.1016/j.jcp.2006.10.032.
- DeLeon, R., K. Felzien, and I. Senocak, 2012: Toward a GPU-accelerated immersed boundary method for wind forecasting over complex terrain. *Proc. ASME Fluids Engineering Division Summer Meeting*, Rio Grande, Puerto Rico, ASME, FEDSM2012-72145.
- Eberly, D., 2008: Distance between point and line, ray, or line segment. Geometric Tools, 2 pp. [Available online at <http://www.geometrictools.com/Documentation/DistancePointLine.pdf>.]
- Edwards, J. R., J.-I. Choi, S. Ghosh, D. A. Giesekeing, and J. D. Eischen, 2010: An immersed boundary method for general flow applications. *Proc. Third Joint US-European Fluids Engineering Summer Meeting*, Montreal, QC, Canada, FEDSM-ICNMM2010-31097.
- Fadlun, E. A., R. Verzicco, P. Orlandi, and J. Mohd-Yusof, 2000: Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *J. Comput. Phys.*, **161**, 35–60, doi:10.1006/jcph.2000.6484.
- Gilmanov, A., and F. Sotiropoulos, 2005: A hybrid Cartesian/immersed boundary method for simulating flows with 3D, geometrically complex, moving bodies. *J. Comput. Phys.*, **207**, 457–492, doi:10.1016/j.jcp.2005.01.020.
- Guth, P. L., 2013: MICRODEM. [Available online at <http://www.usna.edu/Users/oceano/pguth/website/microdem/microdem.htm>.]
- Holcombe, S., 2012: Inpolyhedron—Are points inside a triangulated volume? MATLAB Central File Exchange. [Available online at <http://www.mathworks.com/matlabcentral/fileexchange/37856-inpolyhedron-are-points-inside-a-triangulated-volume->.]
- Kang, S., A. Lightbody, C. Hill, and F. Sotiropoulos, 2011: High-resolution numerical simulation of turbulence in natural waterways. *Adv. Water Resour.*, **34**, 98–113, doi:10.1016/j.advwatres.2010.09.018.
- Lundquist, K. A., F. K. Chow, and J. K. Lundquist, 2010: An immersed boundary method for the Weather Research and Forecasting Model. *Mon. Wea. Rev.*, **138**, 796–817, doi:10.1175/2009MWR2990.1.
- , —, and —, 2012: An immersed boundary method enabling large-eddy simulations of flow over complex terrain in the WRF Model. *Mon. Wea. Rev.*, **140**, 3936–3955, doi:10.1175/MWR-D-11-00311.1.
- McDonald, B., 2004: SURF2STL. MATLAB Central File Exchange. [Available online at <http://www.mathworks.com/matlabcentral/fileexchange/4512-surf2stl>.]
- Mittal, R., and G. Iaccarino, 2005: Immersed boundary methods. *Annu. Rev. Fluid Mech.*, **37**, 239–261, doi:10.1146/annurev.fluid.37.061903.175743.
- , H. Dong, M. Bozkurtas, F. Najjar, A. Vargas, and A. von Loebbeck, 2008: A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *J. Comput. Phys.*, **227**, 4825–4852, doi:10.1016/j.jcp.2008.01.028.

- Mohd-Yusof, J., 1997: Combined immersed-boundary/B-spline methods for simulations of flow in complex geometries. *Annual research briefs 1997*, Center for Turbulence Research Annual Research Briefs, 317–327.
- O'Rourke, J., 1994: *Computational Geometry in C*. Cambridge University Press, 346 pp.
- Osher, S., and R. Fedkiw, 2003: *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 269 pp.
- Peskin, C., 1972: Flow patterns around heart valves: A numerical method. *J. Comput. Phys.*, **10**, 252–271, doi:10.1016/0021-9991(72)90065-4.
- Pu, J., B. Zheng, J. K. Leader, X.-H. Wang, and D. Gur, 2008: An automated CT based lung nodule detection scheme using geometric analysis of signed distance field. *Med. Phys.*, **35**, 3453–3461, doi:10.1118/1.2948349.
- Senocak, I., A. S. Ackerman, D. E. Stevens, and N. N. Mansour, 2004: Topography modeling in atmospheric flows using the immersed boundary method. *Annual research briefs 2004*, Center for Turbulence Research, 331–341.
- Sethian, J., 1999: *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. 2nd ed. Cambridge University Press, 378 pp.
- Sunday, D., 2001a: Intersection of a ray/segment with a triangle. [Available online at http://geomalgorithms.com/a06-_intersect-2.html#Segment-Triangle.]
- , 2001b: Lines and distance of a point to a line. [Available online at http://geomalgorithms.com/a02-_lines.html.]
- Zhao, H., 2004: A fast sweeping method for Eikonal equations. *Math. Comput.*, **74**, 603–627, doi:10.1090/S0025-5718-04-01678-3.