Proceedings of the ASME 2012 Fluids Engineering Summer Meeting
FEDSM2012
July 8-12, 2012, Rio Grande, Puerto Rico

FEDSM2012-20002

ROBUST AND EFFICIENT SETUP PROCEDURE FOR COMPLEX TRIANGULATIONS IN IMMERSED BOUNDARY SIMULATIONS

Jianming Yang *
IIHR – Hydroscience and Engineering
University of Iowa
Iowa City, IA 52242
Email: jianming-yang@uiowa.edu

Frederick Stern
IIHR – Hydroscience and Engineering
University of Iowa
Iowa City, IA 52242
Email: frederick-stern@uiowa.edu

ABSTRACT

Immersed boundary methods have been widely used for simulating flows with complex geometries, as quality boundary-conforming grids are usually difficult to generate for complex geometries, especially, when motion and/or deformation is involved. Complex geometries can be conveniently represented using triangulated surfaces in a Lagrangian manner. A major task in immersed boundary simulations is to inject the immersed boundary information into the background Cartesian grid, such as the inside/outside status of a grid point with regard to the immersed boundary and the accurate sub-cell position of the immersed boundary for a grid point next to it. For high resolution simulations with moving/deforming boundaries, this step can be very expensive. In this paper, a simple, fast, and robust procedure is developed for setting up complex triangulations. Several cases with complex geometries are performed to demonstrate the efficiency and robustness.

INTRODUCTION

An important ingredient of non-boundary conforming methods is the description of an immersed boundary (IB) on the underlying (usually, Cartesian) grid. For simple geometries such as cylinder, sphere, and other frequently used primitives in computer aided design, implicit surfaces by analytical functions can be used to efficiently define inside/outside status of a grid point and easily locate the intersections of the grid lines with immersed surface. Sometimes parametrized curves/surfaces also can be simple and efficient choices. However, for general complex geometries, surface meshes (usually with triangle elements) are the most popular choices due to the well-balanced simplicity and efficiency. Surface triangulations can be used to represent arbitrary geometries, either open or closed, solid or deformable, standalone or interconnected. In IB methods, triangulations have been frequently used to represent immersed objects, but the efficiency and robustness of setup procedure has not been paid enough attention to and adequately addressed in most immersed boundary studies.

Gilmanov et al. [1] considered single convex body represented by a triangulated surface in their IB method. The grid points inside the solid could be easily determined using the surface normal vectors of all surface elements. For a convex body, the projection of the IB point could be calculated readily using the normal vector of the surface element. Then a probe point was located in the fluid phase at the intercept with the grid plane by the ray starting from the surface projection and passing the IB point. In [2], this simplified algorithm was generalized to complex geometries as follows. First, all grid points within a certain distance (of magnitude about the near-body grid spacing) to the centroid of any surface element were identified as near-boundary points. Then for each near-boundary point, the surrounding surface elements with distances from their centroids to this point less than the prescribed threshhod given above were located. The surface normal vectors of these elements were then used to distinguish internal (inside the body) and external (IB) points. Similar to [1], a grid point could be determined as an internal point if all nearby surface elements gave inside status to this point. And

*Address all correspondence to this author.

external IB points were determined otherwise. After the near-boundary points were separated, other internal points could be simply determined by searching along grid lines and all points within two internal points would be internal too. The projection on the surface for an IB point and the probe point in the fluid phase were determined the same as discussed in [1].

Choi et al. [3] presented a detailed discussion of their IB setup procedure. The immersed objects were represented using triangulations in STL (stereolithography) format. ADMesh [4] was used to provide a consistent normal vector for each surface element. For each element, the centroid and three vertices together with the the corresponding surface normal vector and three angle-weighted pseudo-normal vectors at vertices were stored/calculated. Then for a grid point within the bounding box surrounding a surface element, the unsigned distance to the surface was approximated by the distance to the nearest surface point among the four candidates (one centroid and three vertices). The surface normal vector and three pseudo-normal vectors were used to assign the shortest distance a sign, which gave the inside/outside status of the grid point. They also used a consensus algorithm to improve the robustness of the above process. In general, for each surface element, if the number of surface points that gave inside status to a grid point was larger than the number of surface points that gave outside status, that grid point would be inside with regard to the given surface element; and vice versa. The global signed distance function could be then simply defined as the minimum one among all local signed distance functions from nearby surface elements. After the global signed distance function was obtained, the grid points could be easily classified into three different categories similar to Fadlun et al. [5]. For a grid point requiring solution reconstruction, a probe point was positioned in the direction of the outward normal vector from the surface passing that point. Then interpolation stencils for velocity components and pressure were established using surrounding fluid points for solution reconstruction at the probe point. Since the distance function was approximated only using the centroid and vertices instead of the accurate geometric distance, a major limitation of this approach was that the surface resolution had to be similar to that of the local grid, e.g., the average triangle size in [3] was at most twice of the local grid cell size, and larger triangles could result in wrong inside/outside classification.

In [6], the inside/outside status of a grid point with regard to a triangulated immersed body was determined using the surface normal vector of the surface element closest to that grid point. Compared with the additional consensus algorithm in [3], this approach was much simpler with the sacrifice of overall algorithm robustness. The determination of closest surface point for a ghost-cell grid point (at which the boundary conditions of the immersed surface was applied) was similar to [3] in the sense that the closest vertex of all surface elements to the ghost-cell point was located. Then all surface elements sharing this vertex were used to calculate the projected point of the ghost-cell point on the surface along the surface normal directions. For degenerate cases with more than one projected points or without any projected points on the surface, complicated search involving more surrounding surface elements would be triggered. After the closest surface point was identified, an image point for the ghost cell could be defined in the fluid phase and tri-linear interpolation was used to determine the field value at the image point.

The approaches discussed above made use of surface normal vectors to determine inside/outside status of a grid point. Actually this is a classical problem in computational geometry usually termed as point in polyhedron probelm [7]. For example, recently in [8] an algorithm based on ray-casting method was developed for IB method on general structured grids. Basically the bounding box enclosing the immersed body was covered by a coarse uniform Cartesian grid. Standard ray-triangle test as given in [7] was performed for this Cartesian grid to obtain three categories of control cells similar to those in the Cartesian grid methods. For a control cell intercepting triangles, a list of these triangles was created for a further ray-triangle interaction test of all curvilinear grid points contained with this control cells.

Using ray tracing technique for grid point classification in IB methods was first reviewed by Iaccarino and Verzicco [9]. They reported the employment of STL format for description of immersed objects and grid point tagging using the geometric algorithm in O'Rourke [7], i.e., 3D segment-triangle intersection test. A healing process using many additional random rays besides three perpendicular rays (i.e., along three grid line directions for a Cartesian grid) was conducted for problematic STL files. Note for the one-dimensional interpolation scheme reported in [5], the intersections from the three perpendicular rays provided all necessary interpolation information. However, for interpolation along the surface normal direction, additional information such as projections of IB points on the surface had to be obtained too. Compared with the inside/outside status determination using surface normals and surface centroids/vertices, a very different property of the ray tracing technique is that it does not rely on the resolution of surface triangulations as STL files directly from CAD software packages may contain very skewed triangles in surface areas with low curvature. Of course, to obtain a good representation of flow information distribution on the immersed surface such as pressure and shear stresses, a better surface triangulation that with a resolution comparable with the local grid spacing is still required.

Inside/outside classification and segment-triangle intersection using ray tracing algorithms are also frequently employed in Cartesian-based grid generation [10]. Although there are some similarities and many techniques developed for Cartesian grid generation could be directly used for IB methods, the information required for IB method is usually much less since irregular cells from geometric cutting usually are not considered in most IB methods.

IMMERSED BOUNDARY SETUP PROCEDURE

In this part a robust and efficient setup procedure for complex triangulated surfaces frequently used in IB simulation will be discussed in detail. The major components are the data structure for surface triangulations, inside/outside status determination of grid points, and closest surface point computation for an IB point. These techniques are applicable to different variants of the IB method.

Immersed Boundary Description

There are many different file formats (or data structures) for representing triangulations. For example, STL format can be used to describe a triangulated surface by only storing the unit normal and three vertices of each triangle in Cartesian coordinates. Usually the the geometry is a watertight solid and the normal vectors can define the inside/outside of the solid. It is a very simple format and available from major CAD and software packages. Therefore, many IB methods for complex geometries chose to use STL format. One of the major issues of STL format for IB applications is the STL files from various sources may contain errors such as holes, cracks, and triangles with wrong normal vectors, and a preprocessing step using tools like ADMesh is usually a necessary process to fix possible problems in the files. In addition, a STL file gives many redundant vertices as each element is stored as three independent vertices without any connectivity or topology information.

Another simple format gives the numbers of unique vertices and triangles and lists of vertex coordinates and triangles (indices of the three vertices), which is frequently used for representing finite element surface triangulations, e.g., a Tecplot finite element triangular surface data format [11]. This format is the most memory-effective one, but it does not give topology information either. Due to the lack of adjacency table, consistent treatment of neighboring triangles during the geometric operations required for the IB setup procedure is difficult to achieve.

In this study, we adopted the GTS format as defined in the GNU Triangulated Surface Library [12]. The GTS Library is an Open Source Free Software library, which provides a set of useful functions to deal with 3D surfaces meshed with interconnected triangles. GTS file format is very simple and a GTS file consists of the numbers of vertices, edges, and faces (triangles) followed by the lists of vertices, edges (vertex indices), and faces (edge indices). The edge connectivity information extra to the two formats discussed above turns out to be very critical for robust geometric operations, as will be clear in the following parts. The package also provides useful conversion utilities between GTS format and other widely used CAD formats such as STL and DXF. There are file formats for surface triangulations with complete vertex, edge, and face adjacency information. However, for IB applications, GTS format seems to be a good choice as more information is not required for the geometric operations involved in the setup procedure.

Inside/Outside Status Determination

In this study, we only consider immersed objects that are closed, watertight manifold and well resolved by the underlying grid. For "dirty" geometries with holes and/or cracks, a preprocessing step that fixes these geometric problems is required. ADMesh can be used to repair STL files by adding facets to fill holes and matching edges to remove cracks. It is important to check the fixed geometries to prevent any undesirable changes to the original ones. For unclosed geometries and thin membranes, other approaches that require no inside/outside status have to be applied.

The most frequently used algorithm is the so-called "point in polyhedron" test, that is, to cast a ray from a point and count the number of intersections of this ray with the triangulation. Therefore, the core geometric operation will be the "segment-triangle intersection" test in 3D as discussed in [7]. For Cartesian grids, all grid lines align with the coordinate directions, thus grid points along a grid line can share the same ray for this purpose [9]. Also, the "segment-triangle intersection" can be greatly simplified to a "line-triangle intersection" test and the end point checks are not necessary any more.

In this study, we further simplify the "line-triangle intersection" test to a 2D "point-in-triangle" test. In [9], the "line-triangle intersection" test was used to calculate the spatial position of the intersection of the line with the triangle. Since a coordinate-aligned line is to be used as a ray, for the intersection of the line with the plane that the triangle lies within, two of three coordinates are already specified. By utilizing the linear property of a triangle, the third coordinate can be evaluated using the barycentric coordinates of the point (line/ray in 3D) with regard to the 2D projection of the 3D triangle. Compared with the 3D "line-triangle intersection" test, the 2D "point-in-triangle" test only uses one third floating operations, which represents a substantial savings in computational cost for cases with high resolution Cartesian grids and fine surface triangulations.

One major issue that affects the robustness of geometric algorithms is the inaccuracies in floating point arithmetic. For instance, a point very close to an edge of a triangle may be determined as *outside*, but it could be calculated as *outside* for the neighboring triangle that shares the same edge with the former triangle due to the floating point error. To prevent problems of this type in our algorithm, the barycentric coordinates for a triangle are calculated based on the three edges, i.e., each edge and the third vertex produces one determinant. For two triangles sharing the same edge, the query of a point will be evaluated using exactly the same function for the determination of its side with regard to that edge. For more related discussion about robust floating point usage, the reader is referred to [13]

One additional issue is that for cases with a point coincid-

ing with a vertex or on one of the edges, the intersection may be countered multiple times for all triangles sharing the same vertex and twice for two triangles sharing the same edge. In [9], up to 20 random rays were casted for grid points that inside/outside status could not be determined from the coordinate-aligned rays. Thus 3D "segment-triangle intersection" became a necessity in their algorithm. In this work, a different approach is developed by slightly changing the position of the point. Basically, a 2D point is represented by two floating point numbers for its two coordinates. For each floating point number, the nearest floating point number is located and the difference between them is calculated. Then a perturbation of $10 \sim 10^3$ times of this difference is added to the corresponding coordinate, which represents a negligible change of the point position. Combined with the robust barycentric coordinate calculation algorithm discussed above, the 2D "point-in-triangle" test can give an accurate count of intersections. In the rare situation that some collapses still happen, further disturbance can be added until all rays get even (includes zero) number of intersections with the triangulation. Then for each grid point, the number of intersections by the ray through this grid point are separated into left and right intersections. For a grid point inside the solid, the numbers of left and right intersections should be both odd numbers, whereas for a grid point outside the solid, both are even numbers. This step further provides posterior check of the inside/outside status determination process.

After the inside/outside status of all grid points is determined, further grid point classification for immersed boundary treatment can be easily achieved. As shown in Fig. 1, a grid point inside the solid body is identified as a *solid point*; if a grid point is in the fluid phase and one or more grid line segments connecting its immediate neighboring grid points are intersected by the fluid-structure interface, then this grid point is defined as an *interface point* or *IB point* as used in some other methods; all the rest grid points are defined as *fluid points*. In our direct forcing immersed boundary method, a forcing term is imposed on a grid point in the *solid point* and *interface point* categories to represent the effect of an immersed rigid body on the fluid flow. Therefore, the grids from both categories are collectively defined as *forcing points* for convenience.

Closest Surface Point Computation

After all the interface points are identified, the subcell position of an interface point has to be located for the accurate imposition of boundary conditions at the immersed surface. This step is critical as it distinguishes the IB method from the zeroth-order stepwise approximation of the immersed boundary. In [5], one-dimensional scheme was adopted and only the subcell position along a grid line was required. This is particularly convenient for grid-line aligned ray casting techniques, as the ray-triangle intersections are exactly the necessary data. However, Multiple dimensional schemes are more widely accepted [14] and these schemes can be described as a problem of find the closest point (or usually normal projection) on the immersed surface for an interface point.

An intuitive implementation of this task is to loop over all triangles and calculate the closest point on the triangle if this interface point is within a slightly enlarged bounding box of the triangle under consideration. This approach is very simple and works fine for immersed objects with small number of triangles. However, for high-resolution simulations with complex geometries, triangulations with millions of faces could be encountered. This approach will be extremely expensive and the situation will be worse if fluid-structure interactions with large-amplitude body motions are considered.

In this study, an efficient algorithm is developed for obtaining the closest point information for all interface points in one single loop of all triangles. In general, for each triangle, a slightly enlarged bounding box is used similar to what given in the previous part. For all grid points within the bounding box, only those tagged as interface points will be further processed with closest point calculation. The local shortest distance to the triangle will be saved and the minimum value will be obtained after the loop over all triangles is completed. The global closest point on the triangulation corresponding to the global shortest distance will be saved for further usage.

The 3D "closest point on triangle to point" algorithm used in this study is adopted from [13]. The Voronoi feature regions of a triangle is used to determine which feature the orthogonal projection of the interface point is in, then the closest point is only evaluated with the corresponding feature region. The algorithm in [13] has been optimized to give high performance and can be directly used.

With the closest point on triangulation to an interface point available, the interpolation stencil can be constructed as discussed in [14, 15].

EXAMPLES

In this part, several examples are used to demonstrate the robustness and efficiency of our present algorithm. The first two examples are used to show the treatment of special cases that rays coincide with vertices and go through edges. The third example is used to verify the algorithm complexity.

Cube

In this example, a cube is used to check the robustness of our algorithm. The cube is discretized using 12 triangles and the different views of the triangulation is shown in Fig. 2. The GTS file for this triangulation consists of eight vertices:

```
-5 5 -5
5 5 -5
```

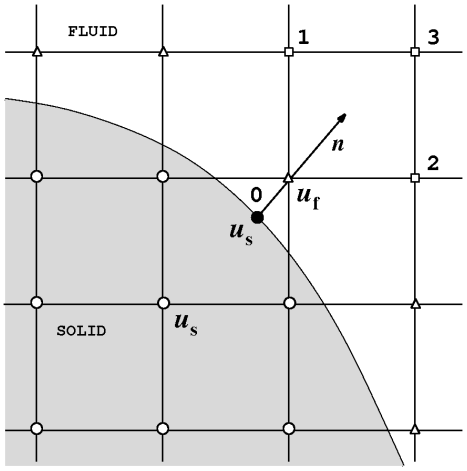


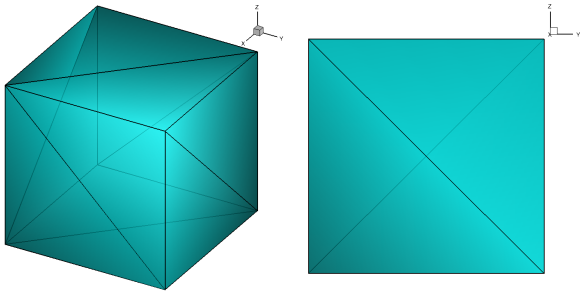
FIGURE 1. IMMERSED BOUNDARY TREATMENT.

```
5 -5 -5
-5 -5 5
-5 -5 -5
-5 5 5
5 5 5
5 -5 5
```

The first domain tested is $[-10,10]\times[-10,10]\times[-10,10]$, and the grid is uniform with $20\times20\times20$ (in the x, y, and z directions, respectively) points. We use rays in the x direction, thus these rays coincides with the grid points in the $y-z$ plane as shown in Fig. 3a. Due to the staggered grid arrangement, there are 40 intersections (10×4edges) with the cube edges for the u grid, and 40 intersections (10×4edges) with the cube edges on the two $y-z$ cube faces) for the v grid; and for the w grid, 40 intersections with the diagonal edges on the two $y-z$ cube faces.

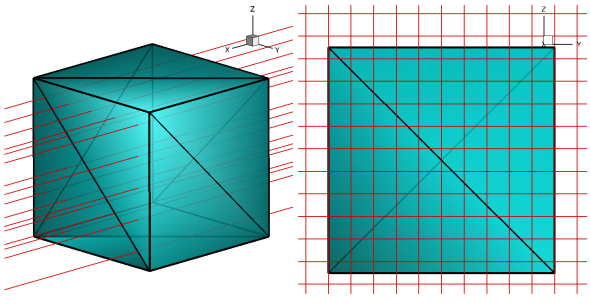
If we slightly change the domain to $[-10,10]\times[-10.5,9.5]\times[-10,10]$, and keep the grid uniform with the same number of points in each direction. Again, there are 40 intersections (10×4edges) with cube edges for the u grid (Fig. 3b), and 40 intersections (10×4edges) with the diagonal edges on the two $y-z$ cube faces) for the v grid (Fig. 3c). However, for the w grid, there are 12 intersections (3×4triangles) with cube vertices and 108 intersections (9×12edges) with all the edges on the two $y-z$ cube faces (Fig. 3d).

With our present robust algorithm, the rays involved in these intersections are repositioned by a very tiny amount (order of 10^{-13} for double precision number near 1). Then a second loop over all triangles gives no such intersections any more, and the inside/outside status determination is not affected by the tiny disturbance introduced. Without these treatments, an intersection with an edge might be counted twice and that with a vertex might be counted even more times. Also, for the diagonal edges on the $y-z$ cube faces, a ray might be computed as outside for both

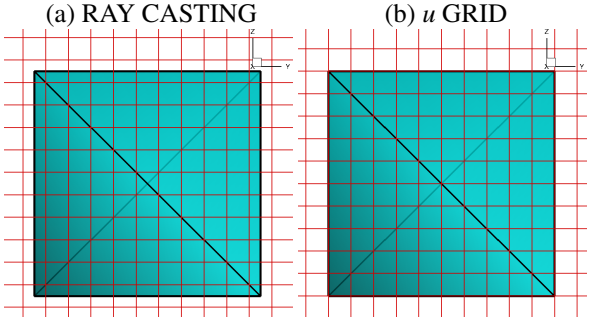


(a) OBLIQUE VIEW (b) $y-z$ VIEW

FIGURE 2. TRIANGULATION OF A CUBE.



(a) RAY CASTING (b) u GRID



(c) v GRID (d) w GRID

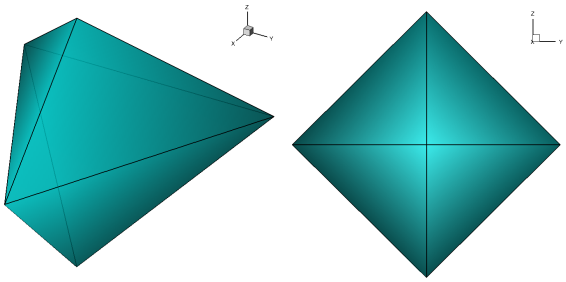
FIGURE 3. RAY CASTING AND SOLID-GRID CONFIGURATIONS FOR THE CUBE CASE.

triangles on one cube face, i.e., leaked through the "crack" between two triangles, but computed corrected on the other cube face, then the inside/outside status of the grid points on this ray would be determined incorrectly.

Triangular Bipyramid

In this example, a triangular bipyramid is used to further check our algorithm. This geometry is constructed by joining two tetrahedra along one face. It has 6 triangles and the two different views of it are shown in Fig. 4. The GTS file for this triangulation consists of five vertices:

```
0 0 -5
```



(a) OBLIQUE VIEW (b) $Y-Z$ VIEW

FIGURE 4. TRIANGULATION OF A TRIANGULAR BIPYRAMID.

```
-5  -5  0
5   0   0
-5  5   0
0   0   5
```

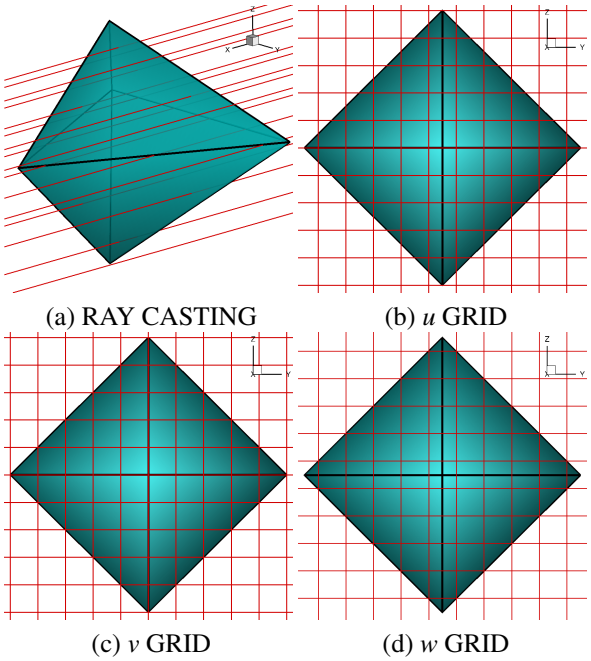
The first domain tested is $[-10,10]\times[-10,10]\times[-10,10]$, and the grid is uniform with $20\times20\times20$ (in the x, y, and z directions, respectively) points. We use rays in the x direction, thus these rays coincides with the grid points in the $y-z$ plane as shown in Fig. 5a. Due to the staggered grid arrangement, there are 40 intersections (5×4edges) with the edges for the u grid, and 20 intersections with the edges in the $x-z$ plane for the v grid; and for w grid, 40 intersections with edges in the $x-y$ plane.

If we slightly change the domain to $[-10,10]\times[-10,10]\times[-10.5,9.5]$, and keep the grid uniform with the same number of points in each direction. Again, there are 40 intersections with edges for the u grid (Fig. 5b), and 40 intersections with the edges for the w grid (Fig. 5d). However, for the v grid, there are 18 intersections with triangle vertices and 82 intersections with all edges (Fig. 5c).

It is evident that some rays go through vertices/edges for the whole triangulation only once, with the inside/outside determination rules, many grid points would be categorized incorrectly. Therefore, it is required to use a different ray to avoid the intersections with vertices and edges. If a random ray not along the grid line direction is used, complicated 3D "ray-triangle intersection" test has to be applied; in addition, all grid points at the grid lines intersecting with vertices/edges have to be checked individually, which would make the algorithm very complicated and expensive as each ray from each point has to be checked against every triangle.

Sphere

In this example, a unit sphere (diameter $D=1$) is used to check the efficiency of our algorithm. As listed in Tab. 1, eight triangulations ranging from 80 to 1310720 faces give a wide spectrum of surface resolution. Fig. 6 shows the first six triangulations, the last two are too dense to distinguish surface elements and not shown here. Each fine triangulation is obtained from dividing one triangle in the coarser one into four pieces and the edge length, Δl, is halved. The edge length ranges from 0.2129 to 0.0017. It is evident that the coarsest triangulation with 80 triangles gives a very crude approximation of the sphere surface and its volume is about 13% less than the analytical value. But the triangulation with 1280 faces turns out to be a good approximation with only 1% less volume. If the surface elements are directly used for evaluating force distribution on the surface, the surface resolution is usually required to be not too different from the local grid spacing. And this may result in a surface triangulation much finer than what needed for resolving the surface itself. For instance, in the first example, a cube can be perfectly described by 12 triangles, but apparently using these triangles to approximate the surface force distribution is not accurate. One possibility is to use two triangulations, with a coarser one for IB setup and the fine one for surface force calculation. Of course, this comes at a price of complicated data structures and more memory consumption.



(a) RAY CASTING (b) u GRID

(c) v GRID (d) w GRID

FIGURE 5. RAY CASTING FOR THE TRIANGULAR BIPYRAMID CASE.

Since the test is only to examine the efficiency, a domain of $[-0.64,0.64]\times[-0.64,0.64]\times[-0.64,0.64]$ is adopted. As listed in Tab. 2, seven uniform grids are chosen for domain discretization, with 20, 40, 80, 160, 320, 640, and 1280 points for each coordinate direction, respectively. And the grid spacing ranges from 0.064 to 0.001. The combination of surface triangu-

TABLE 1. TRIANGULATIONS OF A UNIT SPHERE ($V = 0.5236$).

	Vertices	Edges	Faces	Volume	Δl
1	42	120	80	.4573	.2129
2	162	480	320	.5049	.1065
3	642	1920	1280	.5191	.0532
4	2562	7680	5120	.5225	.0266
5	10242	30720	20480	.5233	.0133
6	40962	122880	81920	.5235	.0067
7	163842	491520	327680	.5236	.0033
8	655362	1966080	1310720	.5236	.0017

TABLE 2. SEVEN UNIFORM GRIDS.

Grid	1	2	3	4	5	6	7
npoint	20	40	80	160	320	640	1280
Δh	.064	.032	.016	.008	.004	.002	.001

lations and grids give us a comprehensive parametric space for our test.

As discussed in the previous section, the algorithm for inside/outside status determination is divided into two parts. The first part is a loop over all triangulation faces. This loop may repeat once if there are rays going through vertices and/or edges. The second part is a loop over each grid point to determine its inside/outside status based on the intersections on the ray passing this point. The latter doesn't change with the resolution of the surface triangulation and has a linear algorithm complexity $O(N^3)$ with N the number of grid points in one grid direction ($npoint$ in Tab. 2 and the figures). For part one, Fig. 7 shows the ray casting CPU time as a function of $npoint$ for all eight triangulations. In general, as the surface triangulations become refined ($nface$ increases from 80 to 1310720), the algorithm complexity changes from $O(npoint^2)$ to $O(npoint)$.

Figure 8 shows the ray casting CPU time as a function of $nface$ for all seven grids. One coarser grids, as the surface triangulation is refined, the CPU time increases almost linearly. On the finest grid, the CPU time increases very slow until the triangulation becomes finer than what is necessary for resolving the surface (triangulation 5 gives a adequate resolution as it volume is 99.9% of the analytical value).

After each grid point has been categorized into either inside or outside type, an interface point is determined by checking its neighboring points for IB crossing. As this step is performed for each grid point, it has a linear algorithm complexity, i.e., $O(npoint^3)$.

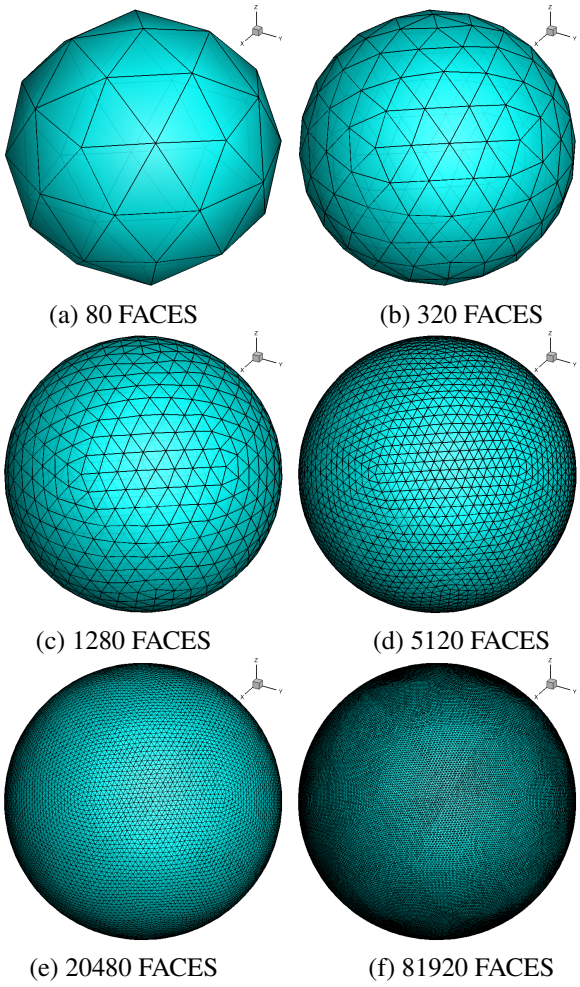

(a) 80 FACES (b) 320 FACES (c) 1280 FACES (d) 5120 FACES (e) 20480 FACES (f) 81920 FACES

FIGURE 6. TRIANGULATIONS OF A UNIT SPHERE.

For the closest point computation algorithm, it is different from the ray casting algorithm as it builds up a 3D bounding box for each triangle and check for interface points within the bounding box. As the grid is refined, the number of grid points to be checked increases rapidly. Figure 9 shows the CPU time as a function of number of grid points in one coordinate direction. It is evident that for coarser surface triangulations, the bounding boxes are larger and involves much more grid points; thus the CPU time increases more rapidly than that for finer surface triangulations. In general, the algorithm complexity changes from $O(npoint^{2.7})$ to $O(npoint^{1.5})$ as the surface triangulation increases from 80 facets to 1310720 facets. Similar to the ray casting algorithm, with a very fine underlying Cartesian grid a

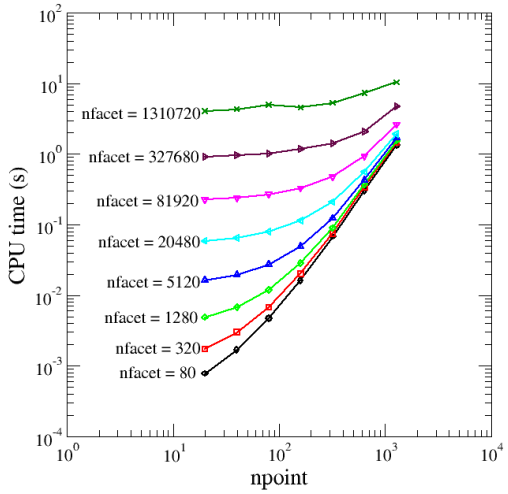


FIGURE 7. RAY CASTING CPU TIME AS A FUNCTION OF *npoint* FOR ALL TRIANGULATIONS.

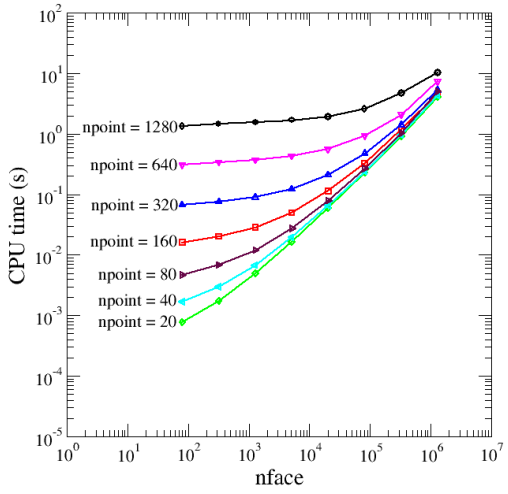


FIGURE 8. RAY CASTING CPU TIME AS A FUNCTION OF *nface* FOR ALL GRIDS.

very coarse surface triangulation doesn't result in much CPU time savings.

On the other hand, Figure 10 shows the CPU time as a function of number of surface facets for all seven grids. On coarser grids, the algorithm complexity is almost linear, e.g., proportional to the number of facets. As the grid is refined,

The last step of the immersed boundary setup is the construction of interpolation stencils for all interface points. The computational cost of this step is proportional to the number of interface points, and depends on the configuration of the immersed object and the underlying grid, instead of the resolution of the surface triangulations. Nonetheless, with the same underlying grid, a refined triangulation for a curved immersed surface does produce some more interface points than a coarse triangulation.

CONCLUSIONS

In this study, we have developed a simple and robust algorithm for setting up immersed objects represented by surface triangulations in immersed boundary simulations. First, we have reviewed approaches in the literature and analyzed their disadvantages. Then we have discussed our algorithm in detail. It has two major components, i.e., ray-casting algorithm for the inside/outside status determination of a grid point with regard to the solid object, and the computation of the closest point on the surface triangulation for an interface point. The former algorithm is particularly robust compared with previous approaches. It uses exactly the simple formulation for evaluating the relative position of a ray with respect to an edge shared by two triangles; also the extreme cases such as ray passing vertices and/or edges are handled without any ambiguity. The algorithm completes with a couple of loops over all surface facets, which is very simple and efficient. For the closest point computation, a very robust algorithm computing the exact closest point to the surface facets including edges and vertices is adopted. It is also simple and efficient as the computation completes in a single loop of all surface facets.

Several cases have been performed to demonstrate the efficiency and robustness. The first two examples are used to show the treatment of special cases that rays coincide with vertices and go through edges. The third example demonstrates the CPU time complexity of different components of the algorithm.

ACKNOWLEDGMENT

This work was sponsored by the Office of Naval Research (ONR) under grant N000141-01-00-1-7, with Dr. Patrick Purtell as the program manager.

REFERENCES

[1] Gilmanov, A., Sotiropoulos, F., and Balaras, E., 2003. "A general reconstruction algorithm for simulating flows with complex 3d immersed boundaries on cartesian grids". *Journal of Computational Physics,* ***191***(2), pp. 660 – 669.

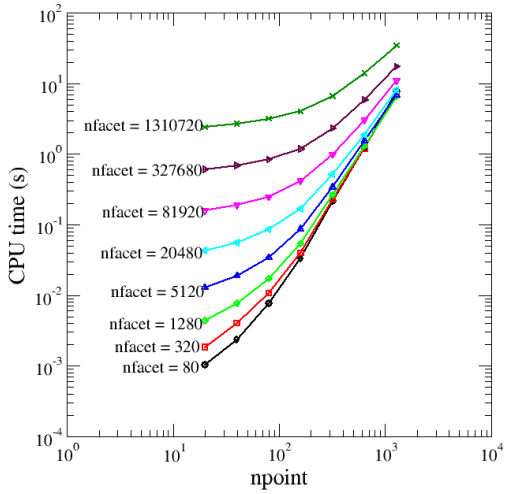


FIGURE 9. CLOSEST POINT COMPUTATION CPU TIME AS A FUNCTION OF *npoint* FOR ALL TRIANGULATIONS.

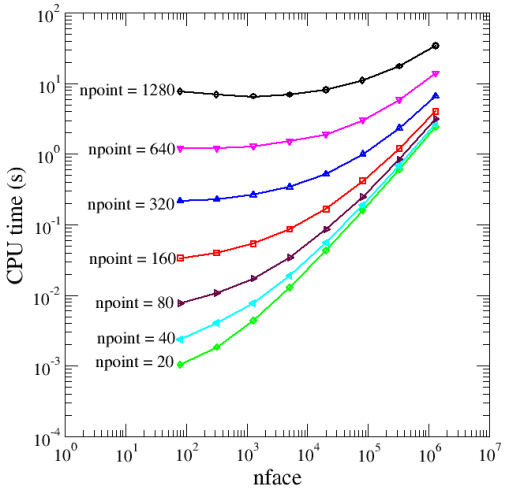


FIGURE 10. CLOSEST POINT COMPUTATION CPU TIME AS A FUNCTION OF *nfacet* FOR ALL GRIDS.

[2] Gilmanov, A., and Sotiropoulos, F., 2005. "A hybrid cartesian/immersed boundary method for simulating flows with 3d, geometrically complex, moving bodies". *Journal of Computational Physics,* ***207***(2), pp. 457 – 492.

[3] Choi, J.-I., Oberoi, R. C., Edwards, J. R., and Rosati, J. A., 2007. "An immersed boundary method for complex incompressible flows". *Journal of Computational Physics,* ***224***(2), pp. 757 – 784.

[4] Martin, A. D., 1998. ADMesh version 0.95. https://sites.google.com/a/varlog.com/www/admesh-htm. [Online; accessed 1-January-2012].

[5] Fadlun, E. A., Verzicco, R., Orlandi, P., and Mohd-Yusof, J., 2000. "Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations". *Journal of Computational Physics,* ***161***(1), pp. 35–60.

[6] Mittal, R., Dong, H., Bozkurttas, M., Najjar, F., Vargas, A., and von Loebbecke, A., 2008. "A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries". *Journal of Computational Physics,* ***227***(10), pp. 4825 – 4852.

[7] O'Rourke, J., 1998. *Computational Geometry in C*, 2 ed. Cambridge University Press, New York, NY.

[8] Borazjani, I., Ge, L., and Sotiropoulos, F., 2008. "Curvilinear immersed boundary method for simulating fluid structure interaction with complex 3d rigid bodies". *Journal of Computational Physics,* ***227***(16), pp. 7587 – 7620.

[9] Iaccarino, G., and Verzicco, R., 2003. "Immersed boundary technique for turbulent flow simulations". *Applied Mechanics Reviews,* ***56***(3), pp. 331–347.

[10] Aftosmis, M. J., Berger, M. J., and Melton, J. E., 1998. "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry". *AIAA Journal,* ***36***, June, pp. 952–960.

[11] Tecplot, Inc., 2011. Tecplot 360®2011 Data Format Guide, Release 2. http://download.tecplot.com/360/current/dataformat.pdf. [Online; accessed 1-January-2012].

[12] Popinet, S., 2011. The GNU Triangulated Surface Library. http://gts.sourceforge.net/. [Online; accessed 1-January-2012].

[13] Ericson, C., 2005. *Real-Time Collision Detection*. Morgan Kaufmann, San Francisco.

[14] Balaras, E., 2004. "Modeling complex boundaries using an external force field on fixed cartesian grids in large-eddy simulations". *Computers & Fluids,* ***33***(3), pp. 375–404.

[15] Yang, J., and Balaras, E., 2006. "An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries". *Journal of Computational Physics,* ***215***(1), pp. 12–40.