Chapter 5
Handling Slender/Thin Geometries with Sharp Edges in Sharp Interface Immersed Boundary Approach

Pradeep Kumar Seshadri and Ashoke De

Nomenclature

$\hat{n}_v$	Angle weighted pseudo-normal
$\hat{n}_e$	Angle weighted edge normal
$\hat{n}_s$	Outward surface normal
$\bar{\alpha}$	Mean incidence
$\Delta\alpha$	Angular amplitude
α	Pitch angle
f^*	Reduced frequency
t^*	Non-dimensional time
U_∞	Free stream velocity

Abbreviations

IBM	Immersed boundary method
NS	Navier–Stokes
FVM	Finite volume method
AMR	Adaptive mesh refinement
LEV	Leading edge vortex

P. K. Seshadri · A. De (✉)
Department of Aerospace Engineering, Indian Institute of Technology Kanpur, Kanpur 208016, India
e-mail: ashoke@iitk.ac.in

S. Roy et al. (eds.), *Immersed Boundary Method*, Computational Methods in Engineering & the Sciences, https://doi.org/10.1007/978-981-15-3940-4_5

5.1 Introduction

5.1.1 General Overview

In recent times, non-boundary conforming approaches like immersed boundary method enjoy widespread popularity for its ability to model flow past arbitrarily complex geometries. A major task in such approaches is to inject the description of immersed object onto the underlying background mesh (Cartesian grids). Based on how the forcing is introduced to satisfy the boundary conditions at the immersed interface, the approach can be broadly classified as continuous forcing (diffused interface) (Kumar et al. 2015; Peskin 2002) or discrete forcing (sharp interface) (Choi et al. 2007; Gilmanov and Sotiropoulos 2005; Kumar and Roy 2016; Udaykumar et al. 2001). In the former, a forcing term is added to continuous Navier–Stokes (NS) equation before discretization, while in the latter, the solution field near the interface is directly reconstructed or the cells that are intercepted by immersed surface is reconstituted into non-rectangular control volumes in order to enforce strict conservation laws.

The forcing term in diffused interface approach ensures the satisfaction of interface boundary condition by using Dirac delta function. This ends up spreading the force term over several neighbouring grid nodes. This results in an increase in the effective width of immersed body. Thus, capturing sharp features of geometry becomes difficult with this approach. On the other hand, the sharp interface approach (solution reconstruction as well as cut cell strategy) allows for the exact imposition of boundary condition. The focus of this study is on the solution reconstruction-based sharp interface approach.

Solution reconstruction-based sharp interface approach because of its non-intrusive character is emerging as an attractive class of immersed boundary approach as it can be implemented on any existing flow solver with very little modification. Unlike the cut cell-based approach which involves highly complex geometric operations (especially with regard to moving body problems as it needs to reconstitute the boundary intercept cell at every time instance), flow reconstruction schemes are much simpler in its implementation and formulation. It does not even lead to a significant increase in computational cost. Usually, the flow is reconstructed along surface normal of the immersed object using various interpolation schemes (depending on the flow physics). This class of approach too encounters issues when handling moving body problems. It suffers from spurious force and pressure oscillations. These are attributed to abrupt forcing point role reversals as the immersed object moves through the background mesh.

5.1.2 Handling Thin/Sharp Bodies

A major difficulty in almost all variants of immersed boundary approach is to ensure accurate representation of complex geometries. For instance, the diffused interface approach discussed earlier will smear out the discontinuity around these sharp edges over a number of grid cells (Zhu and Peskin 2002). This would alter the geometry, change the pressure distribution. Kang et al. (2000) pointed out that such smeared out pressure profiles can cause parasitic currents when it is used to make velocity field divergence-free. Cut cell approach may get into stability-related problems as thin/sharp geometries will lead to arbitrarily small cells. Ensuring conservation laws for such small cells is difficult. The irregularity in flux stencils for such small cells can lead to spurious oscillations of pressure and wall shear stresses. Thus, special treatments like cell merging (Seo and Mittal 2011), cell clustering (Muralidharan and Menon 2018) and hybrid of ghost cell and cut cell algorithms (Ji et al. 2008) are proposed to address some of these issues.

In case of solution reconstruction-based approach, the challenge is twofold in representing thin/sharp geometries,

1. Due to infinite curvature at sharp corners, accurate and consistent inside/outside node classification becomes challenging.
2. Lacks enough number of nodes for accurately reconstructing the flow field around sharp corners. Capturing sharp discontinuities becomes difficult as lack of enough number of points reducing order of accuracy of flux terms. Thus, demands more grid resolution for resolving the flow field.

Several works have tried to address this issue. From the standpoint of accurately representing the immersed surface, works of Gilmanov and Sotiropoulos (2005), Choi et al. (2007), Yang and Stern (2013), Senocak et al. (2015), have provided detailed descriptions regarding algorithms that can be utilized for geometric pre-processing. But most of the research work have tried to address issues from solver's standpoint. For instance, Das et al. (2018) suggest ad hoc corrections around sharp corners, reducing order of accuracy of reconstruction schemes. Ghias et al. (2007) and Onishi et al. (2013) proposed arbitrary dummy cell approach which stores the value of virtual ghost cell points which can be utilized in flux calculations, thereby preserving the order of accuracy of the schemes. Balaras and Vanella (2009) and Liu and Hu (2018) proposed adaptive mesh refinement strategies to improve grid resolution near sharp boundaries.

5.1.3 Objective

The objective of the current study is to present a simple and robust set of procedure that can be followed in order to efficiently handle sharp edges. Through a case study involving dynamic stall in oscillating airfoil, the article tries to highlight the possible

issues arising from the nature of the geometry, limitations of the algorithm (both computational geometry as well as solution reconstruction) and possible errors at the implementation level. To demonstrate the capability of our algorithm and also to highlight issues described earlier, we contrast our results obtained with the help of the algorithm proposed by Gilmanov and Sotiropoulos (2005) in his 2005 work.

5.2 Numerical Details

5.2.1 Flow Solver Details

An in-house density-based finite volume flow solver is used in the study. The 3D unsteady Navier–Stokes equation is solved in generalized curvilinear co-ordinate system using a co-located multiblock grid structure. For simulating incompressible and low Mach number flow, a preconditioning strategy is adopted. Low-diffusion flux-splitting scheme is used for discretizing convective fluxes and central difference scheme for viscous fluxes. Time marching is through a dual time stepping approach. A second-order backward three-point differencing is used for discretizing physical time step, while explicit Euler is used for local pseudo-time stepping. Parallel processors communicate using MPI. Further details about the solver can be found in Das and De (2015).

5.2.2 Immersed Boundary Pre-processing Procedure

Immersed Geometry Description

Immersed body is represented by unstructured triangular meshes. A shared list of vertices and a list of triangular elements storing pointers for the vertices are a common way of representing the triangulations. File formats like STL and Neutral use such element-vertex connectivity data structure. As long as meshes are static, this minimum information is sufficient for most of the geometric operations.

In dealing with moving body and fluid–structure interaction problems where large deformation can lead to bad elements, gaps or cracks, sometime even fragmentation, complexity level of geometric operations increases. These operations often require adjacent queries to be answered, local mesh to be edited to discard bad elements and so on. To perform them in an efficient and robust way, a comprehensive data structure is needed.

Half-edge data structures are the most popular data structure among the available for two reasons: one for its fixed size (no dynamic arrays) and another for its performance regarding all the adjacent related queries in constant time. We use a compact array-based half-edge mesh data structure as proposed by Alumbaugh and

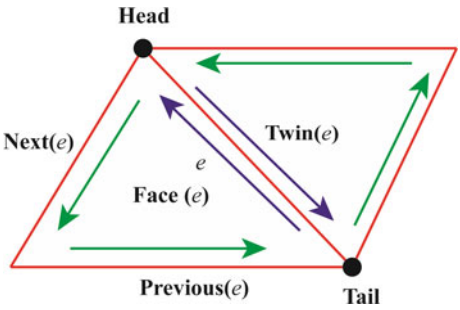

Fig. 5.1 Representation of half-edge data structure

Jiao (2005). This half-edge data structure augments and is constructed efficiently from the standard element-vertex connectivity. Figure 5.1 shows the representation of half-edge data structure. For a given half edge 'e', there is a twin in the adjacent triangle that is oriented in opposite direction. Face containing half edge 'e' has two other half edges, denoted as next and previous. Based on these informations, the following adjacent queries can be answered.

1. For a given vertex, which triangle element uses it?
2. For a given vertex, which edges are incident on it?
3. For a given triangle, what are edges that border it?
4. For a given triangle, what are its adjacent triangles?
5. For a given edge, what are the triangles it shares its edge with?

Some of these queries are needed to efficiently calculate the angle weighted pseudo-normal for vertices and edges which are crucial for treating sharp edges. More on this can be found in Sect. 5.2.3.

Node Classification Algorithm

One of the crucial steps in immersed boundary approach is to accurately classify the nodes as solid, fluid and immersed boundary nodes. In immersed boundary literature, one can find two different approaches to classify the nodes: one, using signed distance function (Choi et al. 2007; Gilmanov and Sotiropoulos 2005; Mittal et al. 2008) and another using ray-casting algorithm (Borazjani et al. 2008; De Tullio et al. 2006). For an immersed body which is closed, smooth and has orientable surfaces, one can use the dot product between line projected from given point 'P' (see Fig. 5.2a) onto the surface (at point 'P_0') and its surface normal. Depending on the sign of the dot product, a given node can be classified as solid or fluid node. But let us assume a situation wherein the immersed boundary has sharp edges as in airfoil (shown in Fig. 5.2a). Consider point A from the shaded region. In order to classify the node, project a line from point 'A' to the surface. Notice that the line falls at the vertex of the surface where the surface normal is discontinuous. The dot product between surface normal $\hat{n}_2$ and the projected line are in the opposite direction, and thus the exterior fluid point will be marked as interior solid point wrongly. The signed distance approach to classify any point in the shaded region shown in Fig. 5.2a will fail. It is worth to note that the triangular meshes are not C^1 continuous at its vertex

and edges, and thus, the normal is undefined at those points. On the other hand, consider the ray-casting approach as shown in Fig. 5.2b, c. In this approach, a ray is cast from a point (say F as in Fig. 5.2b), and the number of intersections it makes with the surface is counted. In this study, we use Moller and Trumbore ray/triangle intersection algorithm (Möller and Trumbore 2005) which solves a linear system of equations to find the barycentric co-ordinates (u, v, w) (see Fig. 5.2c) and the distance from the origin to point of intersection 'P'. As long as the computed value fulfils the barycentric criteria, the intersection point is within the bounds of the triangle. Depending on whether the ray intersects the surface at odd or even number of times, the nodes are classified as exterior or interior. An axis-aligned bounding box (Fig. 5.2b) is implemented to reduce the number of intersection tests as a large number of grid nodes are located outside it. All the nodes outside the bounding box are classified as fluid nodes.

While both signed distance approach and ray-casting approach classifies the nodes as fluid or solid, a separate algorithm is required to tag the immersed boundary nodes. These nodes are the nearest neighbour fluid nodes to the surface on which the solution reconstruction is performed. In the present study, in order to tag immersed boundary nodes, a loop over all solid nodes is performed checking the status of immediate neighbouring nodes. If the immediate neighbour is fluid, then this node is tagged as immersed boundary nodes. Similarly, a loop over all the immersed boundary nodes is performed to identify its immediate solid neighbours. Those solid neighbours are tagged as ghost nodes, which will be used for field extension approach in case of moving body problem.

Closest point computation

Once the classification is done and immersed boundary nodes are identified, an important task is to find the closest surface point to the given immersed node. This is carried out in two-step process. First step involves finding a minimum bounding sphere for triangular element (see Fig. 5.3a) and storing its centre and radius. This radius is then compared with the distance between grid node and centre of this sphere. The one with minimum difference is chosen. In the next step, we use David Eberly's 'distance between point and triangle in 3D' algorithm (Eberly 1999) which defines a squared distance function (Q) for any point on the triangle, T to the point P.

$$Q(u, v, w) = |T(u, v, w) - P|^2 \tag{5.1}$$

This function is a quadratic in barycentric co-ordinates (u, v, w). The closest point is given by the global minimum of Q which occurs when the gradient of Q equals zero. The challenge is to find whether this point is closest to the edge (R2, R3 and R4), vertex (R5, R6, R7) or to the actual face (R1) itself. In all the three cases, finding distance from a point to triangle translates into finding distance to a line, a point or plane, respectively. The index of the closest triangle face, edge or vertex is stored along with point of intersection.

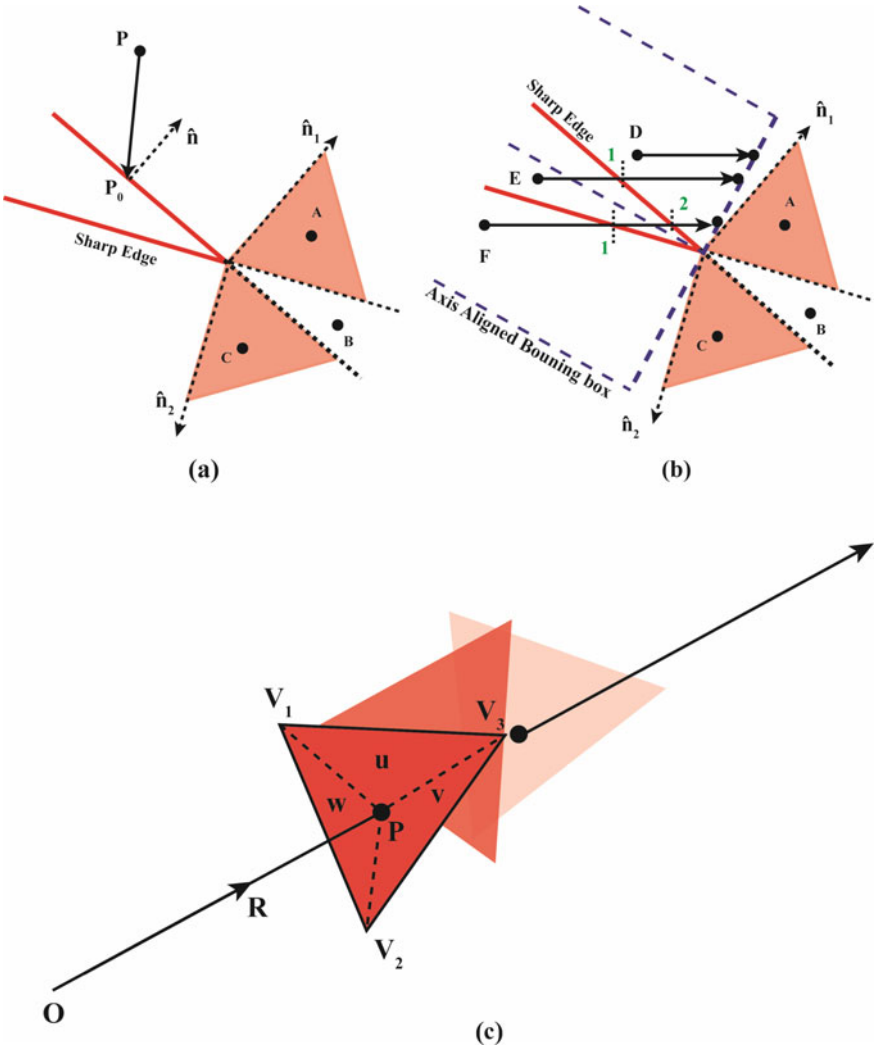

Fig. 5.2 **a** Signed distance calculation for sharp edges. **b** Ray-casting approach for node classification. **c** Ray cast from origin O passes through a number of triangles

5.2.3 Solution Reconstruction

Angle Weighted Pseudo-normals

Before moving further, we would like to re-emphasize few observations from the above discussion.

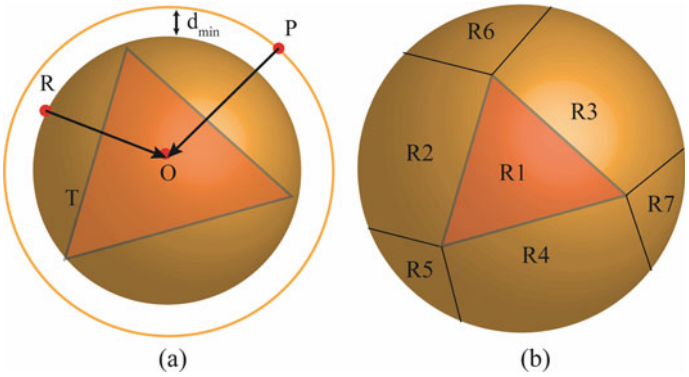

Fig. 5.3 **a** Triangle element (T) bounded by minimum bounding sphere with radius OR and a slightly larger sphere with radius OP. Here, P denotes the nearest neighbour grid node. **b** Seven regions where the projected point from P could lie

1. For a given point, closest distance to a triangle could be a vertex, an edge or face itself.
2. Except for triangle face, vertex or an edge has no well-defined normal as they are not C^1 continuous.
3. Apart from the mesh boundary, if the immersed object itself has concave regions, its exact representation becomes further difficult.

While many of the sharp interface immersed boundary literature (Choi et al. 2007; Yang and Stern 2013; Senocak et al. 2015) makes these observations, most of their concerns are regarding improving the accuracy of node classification. When it comes to solution reconstruction procedure, they apply their reconstruction stencil parallel to surface normal alone irrespective of the fact that the immersed boundary node is close to the edge or vertex. Thus, when sharp-edged regions are encountered, the solution accuracy gets deteriorated as they do not have well-defined normal. Thus, many of these studies tend to focus on improving flux accuracy (Onishi et al. 2013) (by introducing dummy cells around the sharp edge regions), experimenting with different interpolation schemes [linear (Gilmanov et al. 2003), quadratic (Gilmanov and Sotiropoulos 2005), bilinear, trilinear (Mittal and Iaccarino 2005), logarithmic, providing tangential correction (Choi et al. 2007)], adopting local/adaptive mesh refinement (Balaras and Vanella 2009) approaches.

In this study, we define angle weighted pseudo-normal for vertices and edges of all the triangles based on the work of Bærentzen and Aanaes (2005). These pseudo-normal vectors augment the surface normal. Whenever the closest point on the surface is computed, we also store the information regarding which edge/vertex/face is associated with the immersed node with the help of half-edge data structure described earlier in the immersed geometry description of Sect. 5.2.2. While solution reconstruction stencils are applied, they are applied in the direction parallel to associated face/edge/vertex normal.

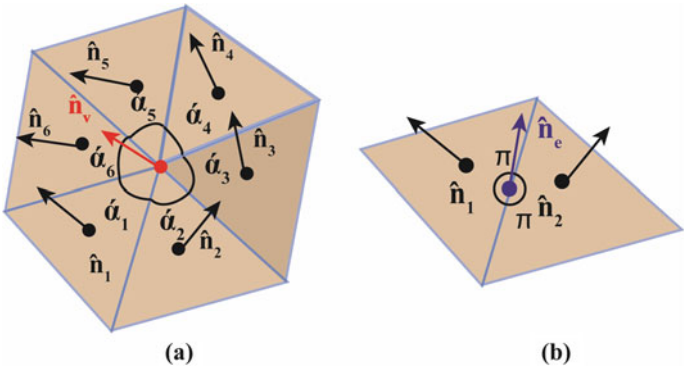

Fig. 5.4 Representation of **a** angle weighted Vertex normal, **b** angle weighted edge normal

Figure 5.4a shows the representation of angle weighted pseudo-normal of a vertex which is defined as

$$\hat{n}_v = \frac{\sum_i \alpha_i \hat{n}_i}{\left\| \sum_i \alpha_i \hat{n}_i \right\|} \tag{5.2}$$

where 'i' denotes the number of incident faces, and α_i is the incident angle.

In case of an edge (see Fig. 5.4b) between face 1 and 2, the angle weighted normal is defined as

$$\hat{n}_e = \pi \hat{n}_1 + \pi \hat{n}_2 \tag{5.3}$$

Direction of Reconstruction Stencil

Figure 5.5 illustrates the direction along which the solution reconstruction stencil is applied. The points P1, P2 and P3 are closest to with vertex V, edge e and face F of the triangular elements, respectively. For reconstructing the solution at node P1, a line parallel to angle weighted vertex pseudo-normal $\hat{n}_v$ projected onto the surface at P1′ is constructed. Similarly for P2, a line parallel to edge normal $\hat{n}_e$ is projected onto the surface at P2′. For P3, a line parallel to surface normal $\hat{n}_s$ is constructed.

Reconstruction Stencils

Flow field variables such as pressure, velocity and temperature are reconstructed at immersed boundary nodes as well as at ghost nodes (in case of moving body problem). A quadratic stencil is applied on the line projected from these nodes to the immersed surface such that it satisfies the boundary conditions at the immersed surface (Seshadri and De 2018). Dirichlet boundary condition is applied for the velocity, and Neumann boundary condition is applied for pressure and temperature.

Sharp interface immersed boundary approach encounters issues of mass conservation and spurious oscillations while modelling moving body problems as the

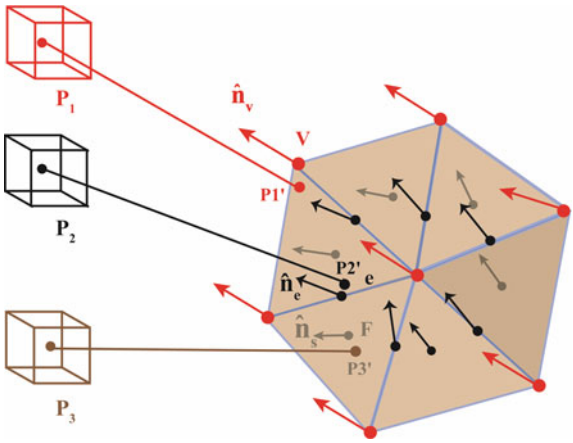

Fig. 5.5 Representation of direction along which reconstruction stencil is applied

approach fails to uphold geometric conservation law. As the immersed object moves through the background Cartesian mesh, the role of these Cartesian grid nodes change suddenly and abruptly at each time instance, i.e. from solid to fluid or fluid to solid. This spatial and temporal discontinuity induces errors resulting in spurious, high-frequency oscillations. In order to address this issue, solid nodes that are immediate neighbour to the immersed surface are marked as ghost nodes, and solutions from the outside field are extrapolated so that when the role change happens, there is a continuity maintained.

5.3 Results and Discussion

In order to demonstrate the efficiency and robustness of our algorithm in handling sharp edges, a detailed comparative study with respect to Gilmanov et al.'s (2005) algorithm is presented in this section. A test case involving dynamic stall of an oscillating airfoil is chosen for this purpose.

5.3.1 Algorithm 1: Gilmanov et al.'s Algorithm

The algorithm is summarized through the following pseudo-code.

1. Input: background Cartesian grid and triangulated surface geometry of immersed body (STL/Neutral format)
2. Determine the face centres of triangular elements and outward surface normal.
3. Locate all Cartesian grid nodes that are in the immediate vicinity of immersed body and within a small prescribed threshold search radius.

4. Gilmanov et al. set search radius approximately equal to the near-body grid spacing. This is found to be inadequate. By examining the grid spacing of the cells that are adjacent to a given node, a maximum of grid spacing is set to be the search radius.
5. For each near-body Cartesian grid nodes, locate the surrounding triangular surface elements. Again within the sphere of search radius prescribed earlier.
6. For a given Cartesian grid node, calculate the signed distance to the face of the associated surface elements.
7. Examine the signs to identify whether the Cartesian grid nodes are inside or outside the immersed body.
8. After the classification of all near-boundary nodes into either immersed nodes or solid nodes, nodes that are interior can also be easily identified by searching along the grid lines. All nodes within two solid nodes will also be solid nodes.
9. For reconstructing solution field, the immersed nodes are projected on to the surface parallel to the surface normal.
10. A quadratic interpolation stencil is imposed along the projected line to obtain the solution field that satisfies the boundary conditions at the interface.
11. For moving body problems, the solution fields are extended inside the solid body by populating the ghost nodes with the information from the fluid region through extrapolation. Again, a quadratic stencil is imposed.

5.3.2 Algorithm 2: Our Present Algorithm

The summary of our algorithm presented here is given below

1. Input: background Cartesian grid and triangulated immersed surface (STL/Neutral format).
2. Based on the element-vertex connectivity information obtained from the surface mesh, establish a half-edge data structure that provides edge connectivity information for robust geometric operations.
3. Determine face centres, surface normal, angle weighted vertex and edge-based pseudo-normal.
4. In order to classify the Cartesian grid nodes as internal or external to the immersed body, first compute bounding box that contains all the vertices defining the immersed surfaces.
5. All nodes that fall outside the bounding box are fluid nodes.
6. Rays are cast from the Cartesian grid nodes that fall within the bounding box to the end of the bounding box in a predefined chosen direction.
7. Check for ray–triangle intersection. If the rays cast from a given Cartesian grid node which does not intersect with triangle, then it is classified as a fluid node.
8. If the ray intercepts with triangle, then the number of interceptions is counted. If the count is odd, it is a solid node and if it is even it is a fluid node.

9. The immersed nodes are identified as fluid nodes that are in the immediate vicinity of solid nodes. The ghost nodes are identified as solid nodes that are in the immediate vicinity of immersed boundary nodes.
10. In order to find the closest surface point to the Cartesian grid node, first determine the distance between Cartesian nodes and face centre of triangular elements. As a first check, this distance is compared to the radius of minimum sphere bounding each of those triangular elements.
11. The triangles which are closer to the nodes are chosen. A quadratic distance function is constructed between the point and triangle. The closest point is obtained by finding global minimum of the quadratic function.
12. The projected point can be close to the edge or vertex and many a times fall on actual face itself. The minimum distance and location of the projected point are stored along with the closest edge, vertex and face information.
13. For solution reconstruction, a line starting from immersed surface passing through immersed node is constructed. This line is constructed such that it is parallel to surface normal or angle weighted vertex or edge normal depending on what is closest to the immersed node.
14. Quadratic interpolation and extrapolation (in case of moving body problems) strategy is followed as described in case 1.

5.3.3 Dynamic Stall of an Oscillating Airfoil

A flow past NACA0012 airfoil pitching about its half chord ($x/c = 0.5$) is chosen as a test case for demonstrating the capabilities of our present algorithm to handle sharp edges even in case of moving body problems. The parameters are chosen from the PIV study of Ohmi et al. (1991). The airfoil begins moving impulsively at $t^* = 0$ (non-dimensional time, $t^* = tU_\infty/c$) and ends at $t^* = 5.0$. The Reynolds number (based on chord length) is 3000. The flow is simulated at a free stream Mach number which is 0.3. The mean incidence $\bar{\alpha}$ and angular amplitude $\Delta\alpha$ of the airfoil are 30° and 15°, respectively. The expression gives the instantaneous angle of attack governing the pitching motion is $\alpha = \bar{\alpha} - \Delta\alpha \cos(2\pi f t)$. The reduced frequency of the pitching oscillation $f^* = fc/2U_\infty$ is 0.1. The Eulerian fluid domain is of size 40c × 30c with 425 × 317 nodes in X–Y plane. The grid is uniformly refined locally near the immersed boundary.

First column in Fig. 5.6 presents the results of Ohmi et al. (1991) obtained by experimental study. The streamline pattern shows the time evolution of unsteady wake past the pitching airfoil. The airfoil impulsively starts its pitching motion from minimum incidence at $t^* = 0$. As the airfoil pitches up, the flow remains attached up to $t^* = 1$. Then, the flow starts separating at the leading edge resulting in the formation of leading edge vortex (LEV). This LEV grows till the airfoil reaches the end of upstroke at $t^* = 2.5$. As the stroke reverses, the growth of LEV stops and it is shed when the airfoil reaches the end of its downstroke at $t^* = 5.0$.

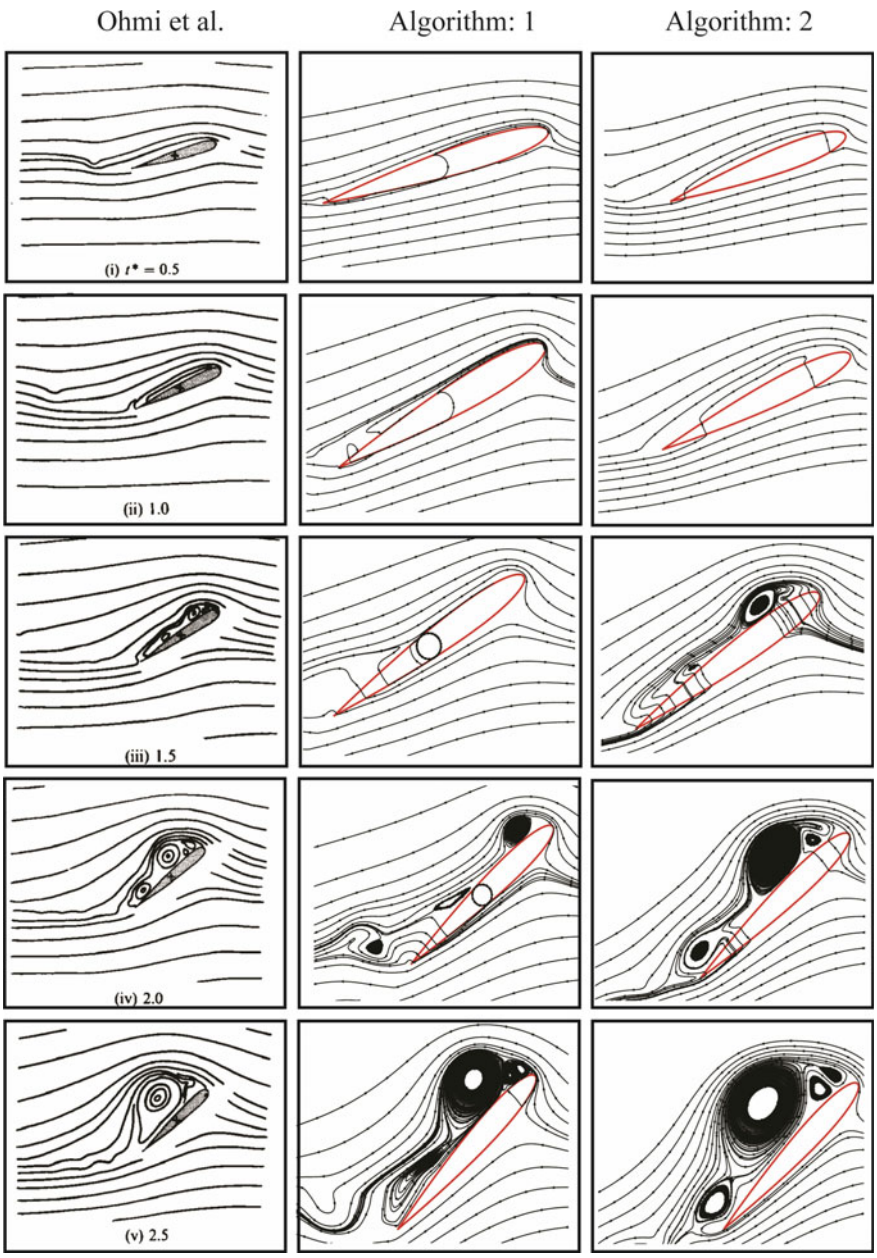

Fig. 5.6 Time evolution of wake past a pitching NACA 0012 airfoil: first column shows PIV results of Ohmi et al. (1991); second column shows Case 1: Gilmanov et al. formulation; third column shows Case 2: present formulation

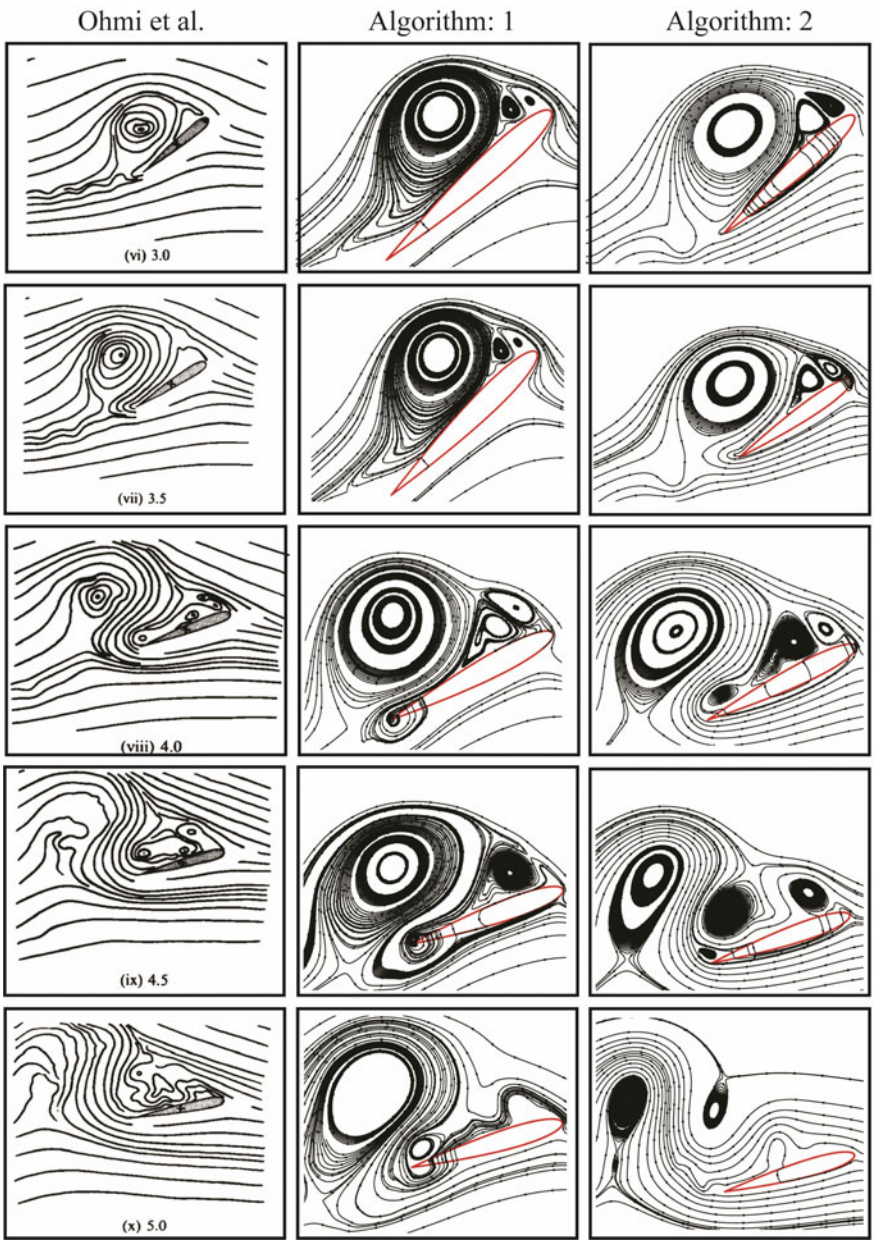

Fig. 5.6 (continued)

Comparing the results from Algorithm 1 and Algorithm 2 with that of Ohmi et al. (1991), one can notice the dynamic stall phenomenon captured by our present algorithm (Algorithm 2) is in excellent agreement with the experimental results. Case 1 algorithm on the other hand fails from the first instance depicted in Fig. 5.6. The flow separates at the trailing edge at $t^* = 0.5$ itself, before it is expected to separate at the leading edge after $t^* = 1$. The flow separation slowly spreads towards upper regions of trailing edge, and from $t^* = 1.5$ onwards, one can notice leading edge vortex which grows till $t^* = 2.5$. As the downstroke begins, the leading edge vortex stops growing and starts shedding. While all the leading edge phenomena are captured well throughout the cycle, one can clearly notice that Algorithm 1 fails to handle the sharp-cornered trailing edge region.

The following sections of the article try to explore the answers to two major questions: where does the Algorithm 1 fail? How does our present algorithm successfully address those issues? The objective is not just to highlight possible sources of errors arising from the limitations of algorithm but also to point out sources of errors arising at the level of implementation.

Immersed Boundary Operations in Interblock Boundaries

Consider earlier observation of Algorithm 1 in Fig. 5.7 where the flow separation at the trailing edge is shown in the instance corresponding to $t^* = 0.5$. A genuine reason for the flow separation could be erroneous handling of sharp corner. But a more careful analysing of the flow field results reveals the flow which separates exactly at the block boundary (as shown in Fig. 5.7a). The U and V contours shown in Fig. 5.7c, d show kinks formed exactly at the block boundary interface. The corresponding plots for the same time instance from Algorithm 2 show that the flow exactly re-attaches at the trailing edge tip. The corresponding U and V contours show smooth distribution without any kinks or disturbances near the trailing edge.

Figure 5.8a shows a streamline plot of pitching airfoil at $t^* = 0.5$ but with fewer block structure. Note that the flow here does not separate at the trailing edge but at the mid-chord as shown in Fig. 5.8b. This observation confirms that the immersed boundary treatment at the block boundaries is the source of the error. In parallel multiblock structured flow solvers, the block boundaries contain layers of dummy cells that exchange and retain information regarding the variables from the adjacent block. This becomes necessary for maintaining order of accuracy of discretization schemes near block boundaries. The number of dummy cell layers depends on the order of discretization schemes involved.

In case of parallel immersed boundary treatment, apart from the information regarding flow variables, additional information regarding tagging, distance function needs to be supplied to these dummy layers. Note the fact that these informations are calculated in every processor that contains blocks that overlap with immersed boundary surface. Each block has access to the information about entire geometry. Although the solution reconstruction is performed only for the physical domain, information from dummy layers are sought for interpolation/extrapolation operations. Thus for an accurate solution reconstruction procedure, the tagging and distance function information provided to these dummy layers should strictly correspond to the values from

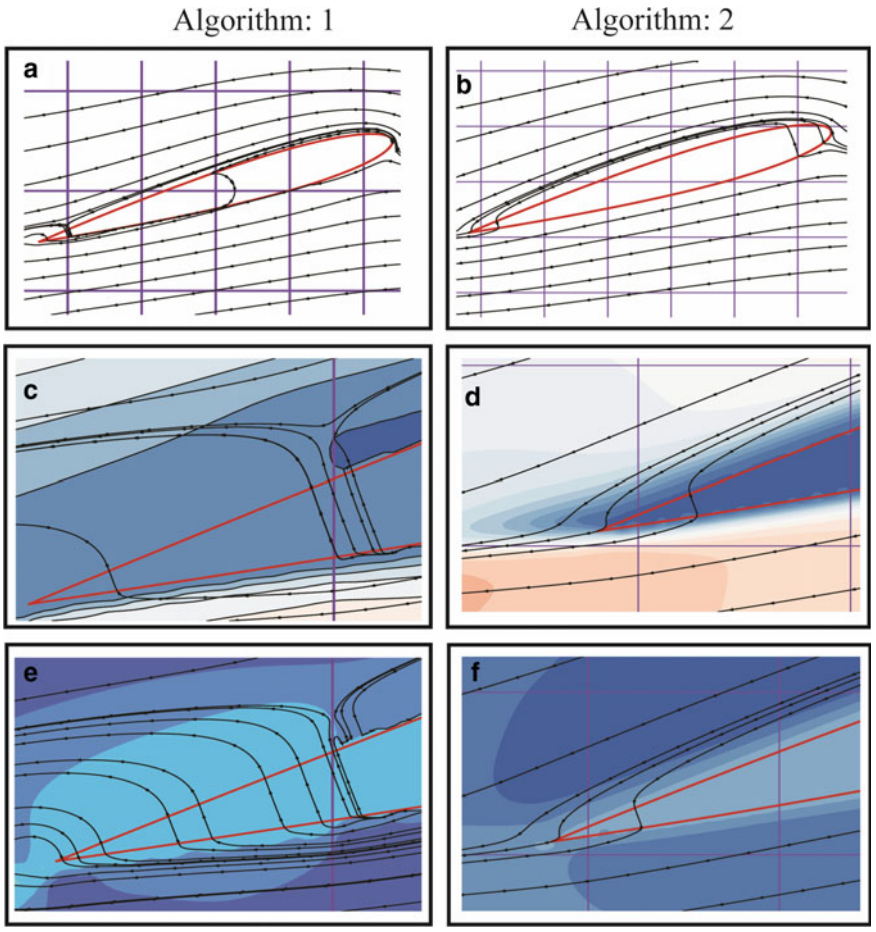

Fig. 5.7 Time evolution of vortices in a pitching NACA 0012 airfoil: Algorithm 1: Gilmanov et al. formulation; Algorithm 2: present formulation

adjacent block whose layer of cells overlaps with that of the dummy layers. Errors from these block boundaries can arise because of two main reasons

1. Number of dummy layers is inadequate for immersed boundary operations which reduce the accuracy of solution near block boundaries.
2. There is inconsistency in tagging, distance function information provided to the dummy cell layers.

In present solver, five layers of dummy cells are shared between the block boundaries which are adequate for implementing fifth-order discretization schemes. Figure 5.8c shows that there is inconsistency in the classification of nodes in dummy layer regions. The four encircled region shows that wherever there is an overlap of adjacent block, there is inconsistency in tagging. Figure 5.9 provides a clearer view of

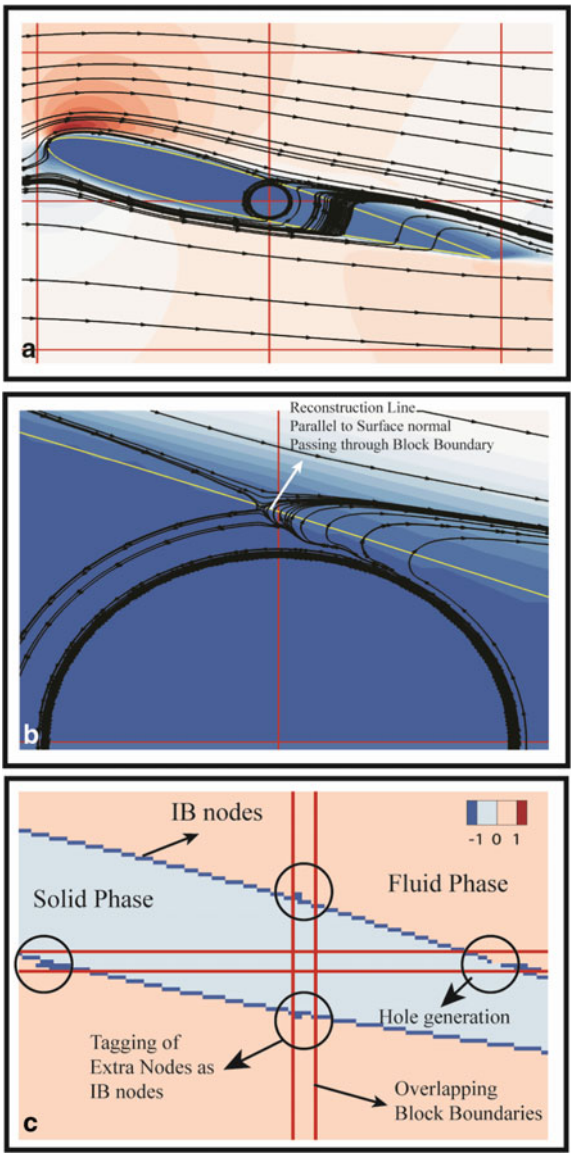

Fig. 5.8 Pitching airfoil at $t^* = 0.5$; **a** fluid Domain with fewer block structure, **b** contour plot showing distribution of nodes, **c** the encircled regions show inconsistent tagging

the region. Note that Fig. 5.9b where Algorithm 1 has tagged fluid nodes as IB nodes and Fig. 5.9c shows the region where the algorithm fails to tag IB node, leading to the hole generation. There could be two possibilities as to why is tagging inconsistent near block boundaries. One, the classification algorithm fails at the block boundaries or the search radius definition used is leading to erroneous tagging.

Figure 5.9d shows that the distribution of search radius near block boundaries is inconsistent with the rest of the physical domain. Figure 5.9e shows 3D contour of

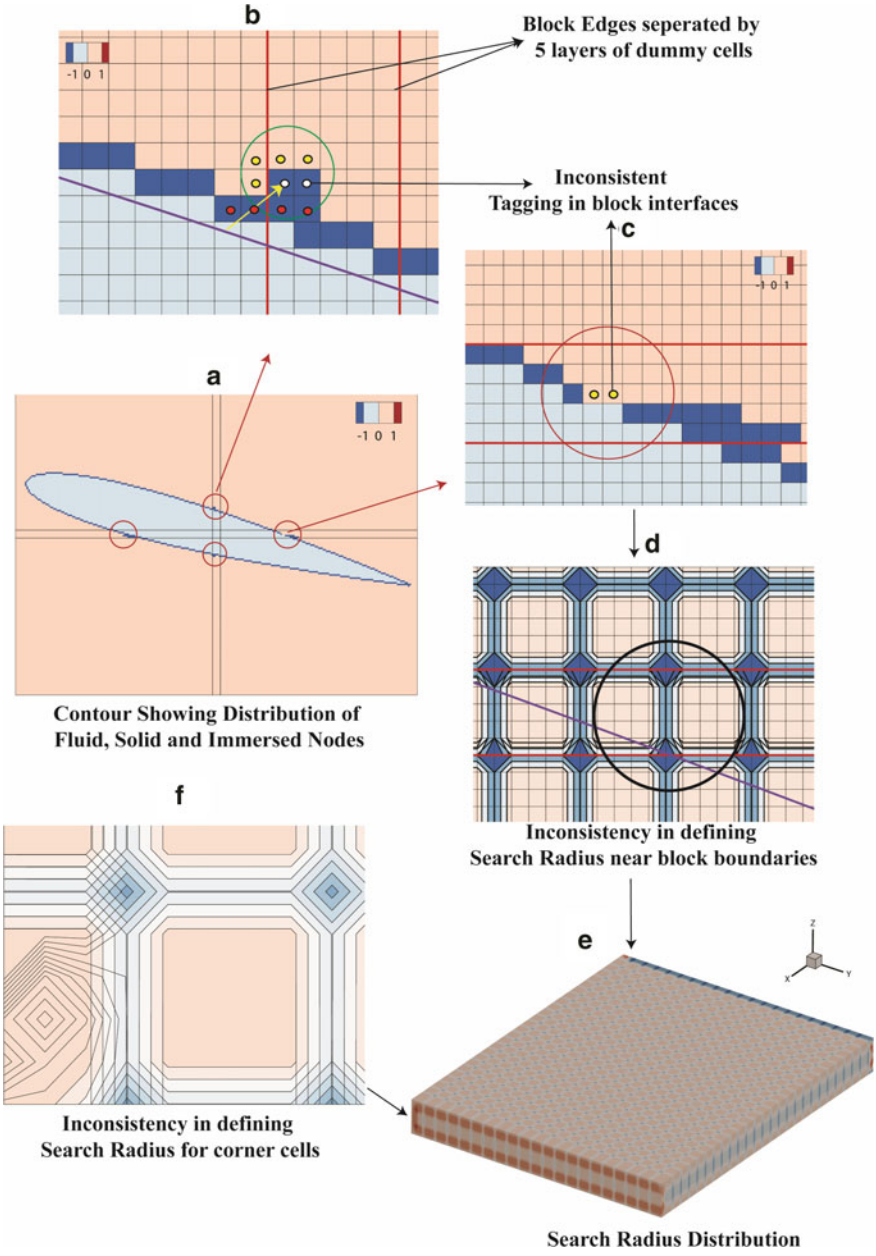

Fig. 5.9 Tagging and search radius distribution in block boundaries

search radius distribution. While distribution is uniform in the rest of the physical domain, near the block boundaries it becomes inconsistent. This is in fact due to the definition of the search radius adopted in Sect. 5.3.1. The number of adjacent cells near block boundaries is less than the interior nodes. Hence, the maximum grid spacing picked by the dummy layers as the search radius would be different from that of the adjacent block layers from the physical domain that overlaps with them. This suggests that search radius distribution in block corners would be more chaotic as there are still lesser adjacent cells. This indeed is shown in Fig. 5.9f. The discussion regarding the classification algorithm is taken up as a separate subsection below. The accurate results from Fig. 5.7b, d, f show that Algorithm 2 which uses minimum bounding sphere strategy to determine the search radius provides consistent definition.

Periodic Boundary Condition

The periodic boundary condition is employed in the solver by rebuilding the connectivity information of the grid such that the periodic faces are considered to be neighbour faces making the periodic boundary treatment implicit. When the block containing periodic face contains the immersed body, as it happens in simulating 2D flows by considering unit span in Z-direction, the immersed boundary treatment on the periodic face becomes not so straight forward. This is mainly because the dummy cell layers populating the periodic faces do not have overlapping immersed body as shown in Fig. 5.10. This is unlike the dummy cell layers that overlap the physical domain where the information regarding the immersed body is available (DC-2 and 3 in Fig. 5.10). A 2D flow demands that information on every section in Z-plane is the same. Thus, the tagging and distance function information calculated for physical boundaries can be copied to the dummy cell layers of periodic faces.

Node Classification Algorithm and Hole Generation

Figure 5.11a shows the flow field of a pitching airfoil at $t^* = 1.5$. Apart from early flow separation, the solution accuracy away from the trailing edge too is deteriorated. Figure 5.11c shows that fluid nodes near trailing edge are being tagged as solid nodes incorrectly. Figure 5.11e provides enlarged view of the tail section. Remember that Algorithm 1 suggests that if for at least one node within the search radius, the computed scalar dot product between surface normal and line drawn from itself to immersed surface is greater than zero, then the node is outside the surface. But this logic fails near sharp edges as discussed in Sect. 5.2.2. The line drawn from a given point to the triangular mesh surface can fall on edge or vertex where the surface normal is not continuous. Especially near sharp corners, one can have the same distance to two triangles from a given point, but the direction of surface normal need not be the same. This makes the sign of dot product, which is calculated with surface normal, ambiguous. On the other hand, Algorithm 2 with its Moller–Trumbore ray-casting algorithm along with winding algorithm provides a robust node classification. Figure 5.11b shows a smooth flow field. The tagging distribution corresponding to that time instance shown in Fig. 5.11d, f is clean without any holes or gaps.

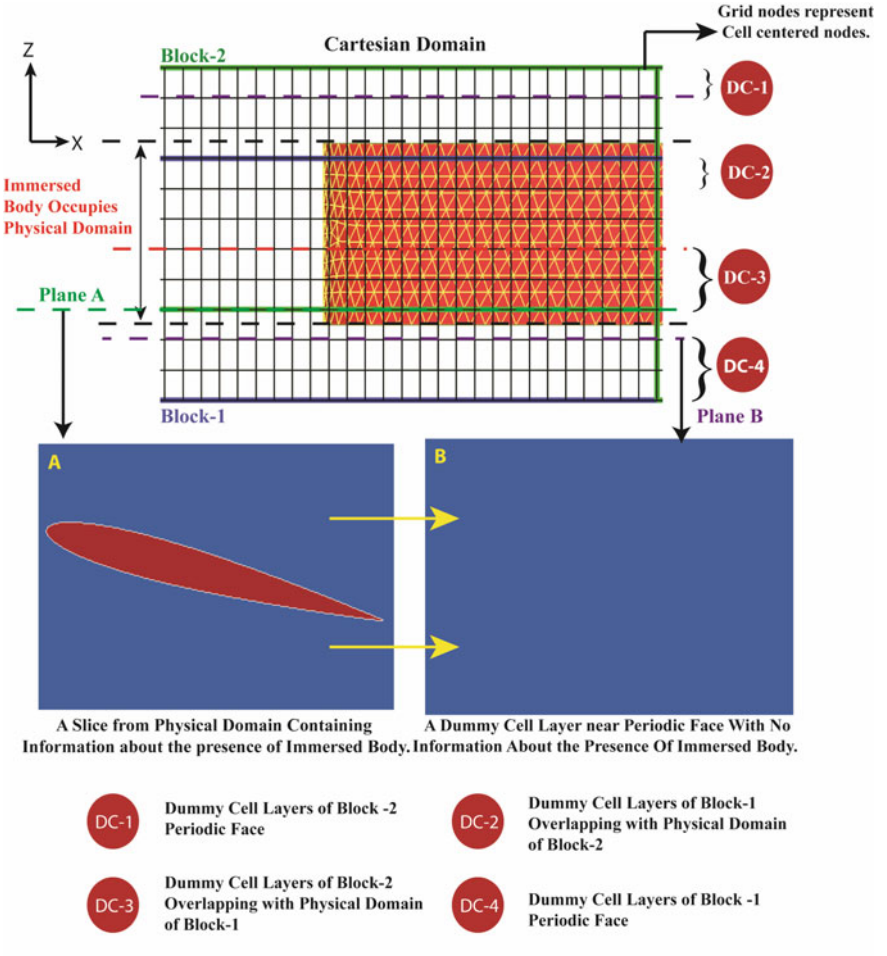

Fig. 5.10 Schemetics showing implementation of periodic boundary conditions

Direction adopted for Solution Reconstruction

To emphasize on the importance of the direction adopted for imposing reconstruction stencil, the pitching airfoil test case is simulated again now with an improvised Algorithm 1. Improvisation is done on two aspects.

1. Without changing the definition of the search radius, the variable is shared across its block boundary from physical domain of adjacent block to the dummy layers of a given block like any other flow variable which is shared in a parallel environment.
2. A strict conservative bounding box is imposed to avoid any hole or gap formation near its trailing edge.

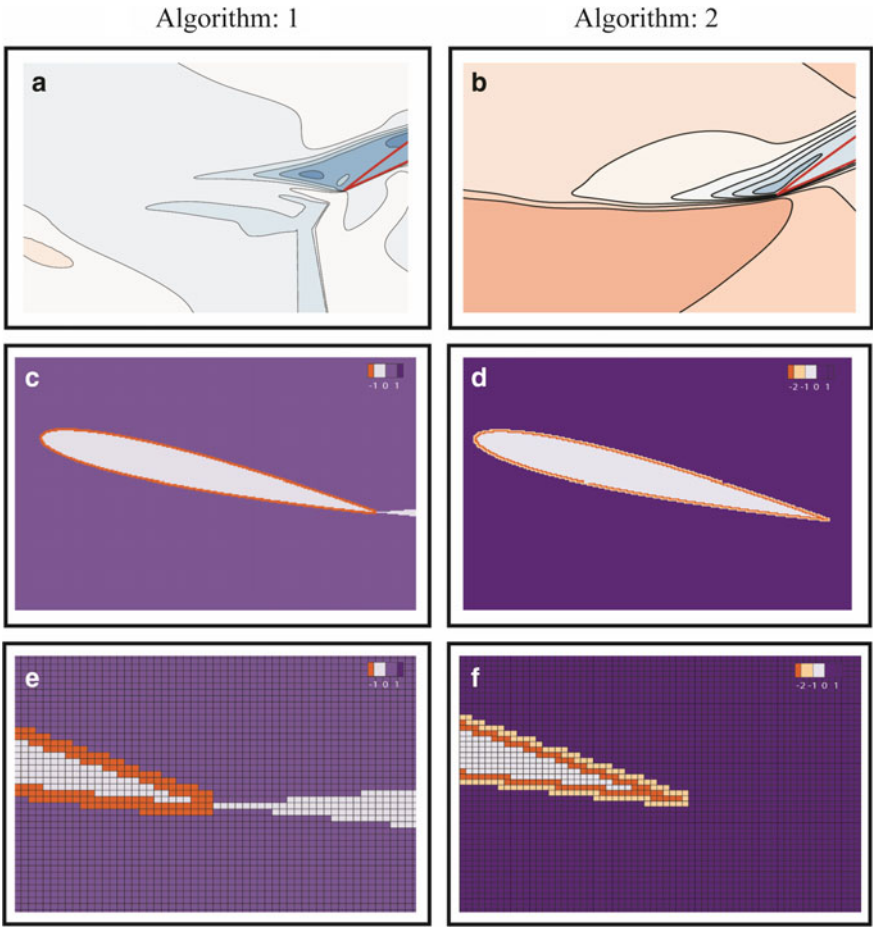

Fig. 5.11 Contrasting signed distance node classification (Algorithm 1) with ray-casting approach (Algorithm 2)

The results from the improvised algorithm are compared with the literature at three instances $t^* = 1.5$, 2.5 and 4.0. Figure 5.12 corresponding to time instance $t^* = 1.5$ shows that the improvised Algorithm 1 provides much better results compared to previous results. It is able to predict the formation of trailing edge vortex just below the mid-chord region. This result is almost identical with Algorithm 2 and Ohmi et al.'s (1991) experimental results. Also, these improvised results are better in its capturing of trailing edge vortex formation than Kumar and Roy (2016) whose results are based on sharp interface IB approach using incompressible flow solver or Akbari and Price (2003) work which is based on boundary confirming approach.

At time instance $t^* = 2.5$ (shown in Fig. 5.13), it is expected from Ohmi et al.'s (1991) results that trailing edge vortex and leading edge vortex almost coalesces into a big vortex when it reaches its peak amplitude. The results from improvised

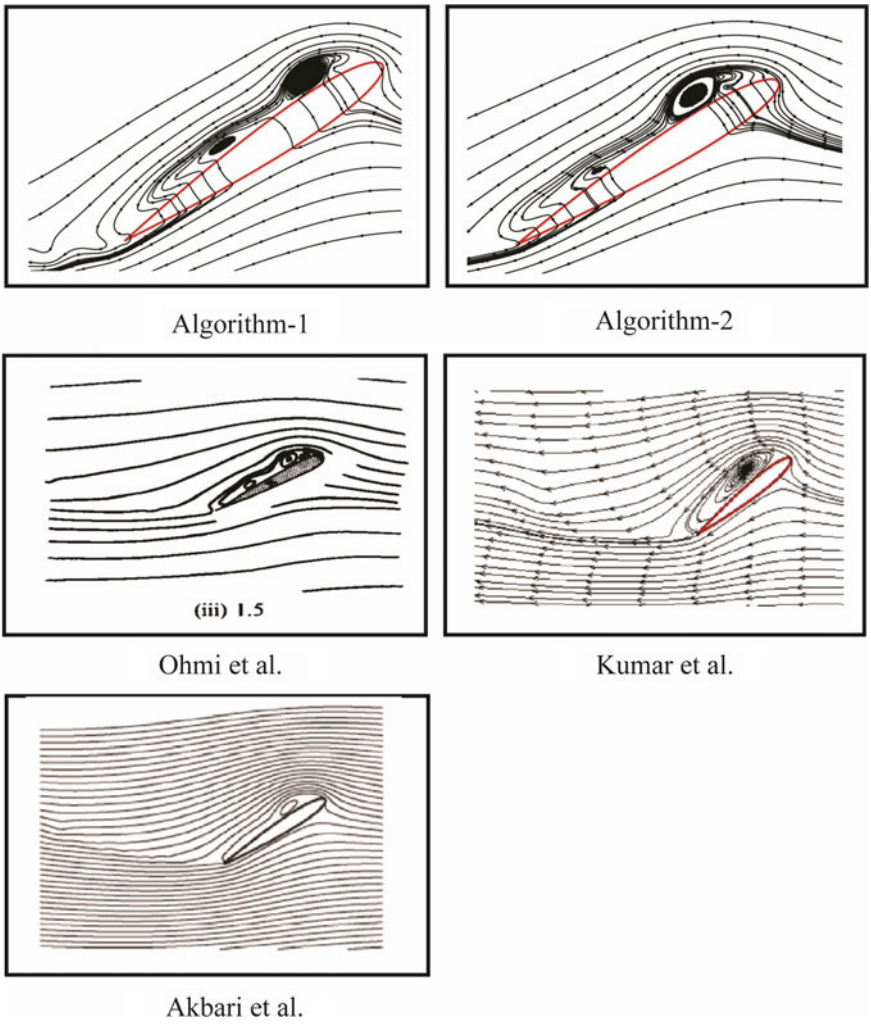

Fig. 5.12 Streamline plots of pitching airfoil at $t^* = 1.5$

Algorithm 1 show the trailing edge vortex which is still strong and has not started interacting with the leading edge vortex. Algorithm 2 shows that the trailing edge vortex is almost engulfed by the leading edge vortex. Since no trailing edge vortex is formed in Kumar and Roy (2016), the leading edge vortex formed grows to occupy the entire chord. Akbari and Price (2003) results show the presence of trailing edge vortex.

At the time instance $t^* = 4.0$ (shown in Fig. 5.14), the downstroke motion has led to the shedding of leading edge vortex. From Ohmi et al.'s (1991) results, one can notice the presence of leading edge vortex, a triangular vortex at mid-chord and

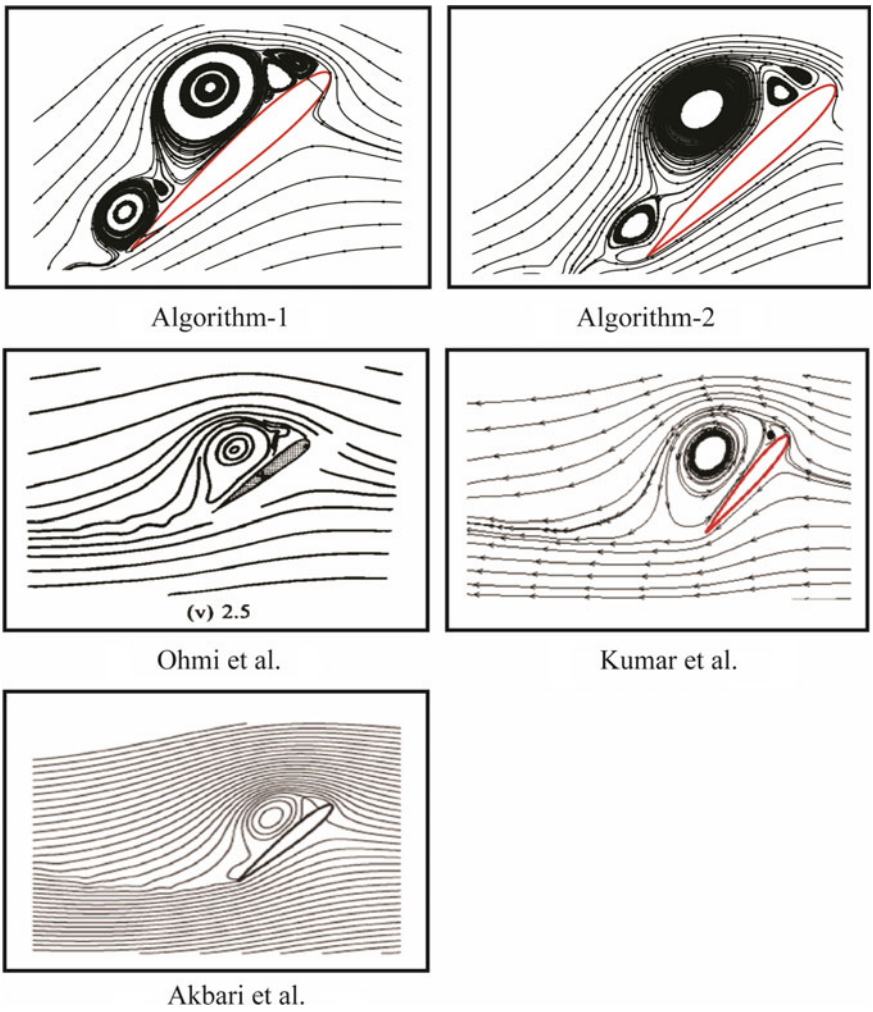

Fig. 5.13 Streamline plots of pitching airfoil at $t^* = 2.5$

a trailing edge vortex. Algorithm 2 and Akbari and Price (2003) results agree well with the experimental observations. But the results of improvised Algorithm 1 do not. The trailing edge vortex is too small. The mid-chord vortex has moved away from the surface and is coalescing with the shed vortex. The leading edge vortex is much broader. Kumar and Roy (2016) results on the other hand show the presence of mid-chord vortex and leading edge vortex. But the vortex near the trailing edge is not on the surface but in the wake.

Figure 5.15 shows the velocity contour plot corresponding to the three time instances we discussed above. At $t^* = 1.5$ and $t^* = 2.5$, the results from improvised Algorithm 1 show that the wake region is chaotic even though the flow field

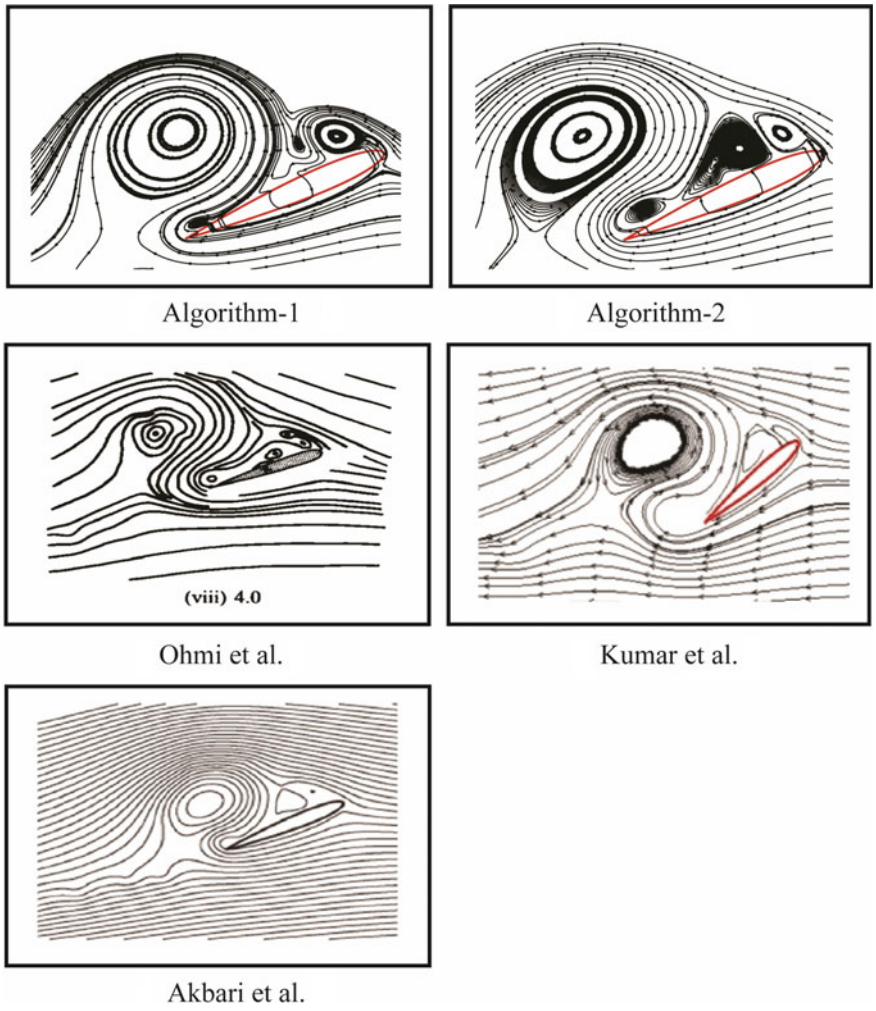

Fig. 5.14 Streamline plots of pitching airfoil at $t^* = 4.0$

around the body is captured accurately. At $t^* = 1.5$, Fig. 5.15a shows an unphysical patch of low-velocity region at the trailing edge tip. Flow field corresponding to $t^* = 4.0$ (Fig. 5.15e) is comparable with the results from Algorithm 2 (Fig. 5.15f).

Figure 5.16 shows the pressure co-efficient distribution around the leading edge of the airfoil at $t^* = 2.5$. Results of Algorithm 1 show that airfoil surface is not sharply represented just like its trailing edge part. The immersed body is actually larger than the actual geometry of the airfoil. In case of Algorithm 2, the geometry is sharply represented. Though the solution reconstruction stencil adopted is the same for both the algorithm, there is a drastic difference in the quality of solutions obtained between these two algorithms. This is true not just in the near-body region but also in wake

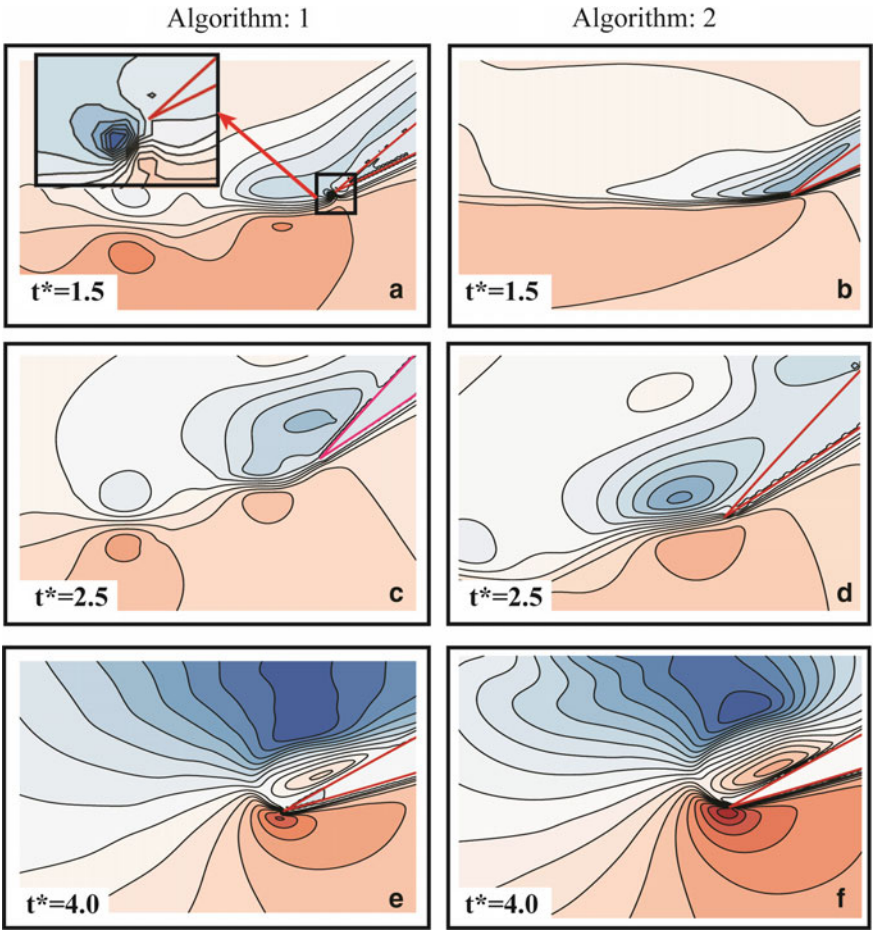

Fig. 5.15 U-V contour plot at time instances $t^* = 1.5$, 2.5 and 4.0

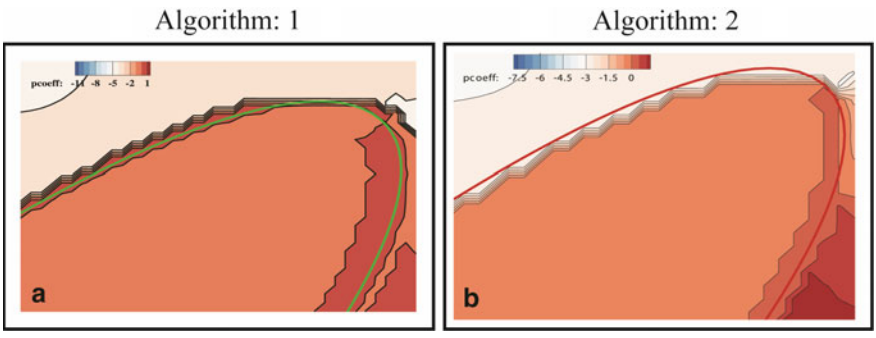

Fig. 5.16 Pressure contour plot corresponding to $t^* = 2.5$

region. This difference is attributed to the direction along which those reconstruction stencils are adopted. In case of Algorithm 1, it is parallel to the direction of surface normal. In case of Algorithm 2, it can be parallel to the direction of surface or vertex or edge normal depending on where the closest point in triangle lie. This enhances the accuracy of geometry representation especially in modelling sharp edges.

5.4 Conclusion

In this study, we have presented our simple and robust algorithm to handle the representation of sharp edges while adopting sharp interface immersed boundary approach. By appreciating the nature of sharp-edged geometries as well as the inherent limitations of representing immersed body with triangular meshes, we have adopted a set of computational geometry procedures that takes care of even the extreme situations when dealing with node classification or exact close point in the triangle, without any ambiguity. With the help of a pitching airfoil test case, we have systematically presented the important role played by such geometry processing algorithms. Their role is not just restricted to geometry pre-processing step as is the case in most of the immersed boundary approach. They play a crucial role in the solution reconstruction procedure as well. The versatile nature of our algorithm is successfully demonstrated by comparing the test case results with the algorithm adopted by Gilmanov et al.

Acknowledgements The authors would like to acknowledge the IITK computer centre (www.iitk.ac.in/cc) for providing support to perform the computation work, data analysis and article preparation.

References

Akbari MH, Price SJ (2003) Simulation of dynamic stall for a NACA 0012 airfoil using a vortex method. J Fluids Struct 17:855–874

Alumbaugh TJ, Jiao X (2005) Compact array-based mesh data structures. In: Proceedings of the 14th international meshing roundtable, Springer, pp 485–503

Bærentzen JA, Aanaes H (2005) Signed distance computation using the angle weighted pseudonormal. IEEE Trans Vis Comput Graph 11:243–253

Balaras E, Vanella M (2009) Adaptive mesh refinement strategies for immersed boundary methods. In: 47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition, pp 162

Borazjani I, Ge L, Sotiropoulos F (2008) Curvilinear immersed boundary method for simulating fluid structure interaction with complex 3D rigid bodies. J Comput Phys 227:7587–7620

Choi J-I, Oberoi RC, Edwards JR, Rosati JA (2007) An immersed boundary method for complex incompressible flows. J Comput Phys 224:757–784

Das P, De A (2015) Numerical investigation of flow structures around a cylindrical afterbody under supersonic condition. Aerosp Sci Technol 47:195–209

Das S, Panda A, Deen NG, Kuipers JA (2018) A sharp-interface immersed boundary method to simulate convective and conjugate heat transfer through highly complex periodic porous structures. Chem Eng Sci 191:1–18

De Tullio M, Christallo A, Balaras E, Pascazio G, Iaccarino G, Napolitano M (2006) Recent advances in the immersed boundary method

Eberly D (1999) Distance between point and triangle in 3D. Magic Softw http://www.magic-software.com/Documentation/pt3tri3.pdf

Ghias R, Mittal R, Dong H (2007) A sharp interface immersed boundary method for compressible viscous flows. J Comput Phys 225:528–553

Gilmanov A, Sotiropoulos F (2005) A hybrid cartesian/immersed boundary method for simulating flows with 3D, geometrically complex, moving bodies. J Comput Phys 207:457–492

Gilmanov A, Sotiropoulos F, Balaras E (2003) A general reconstruction algorithm for simulating flows with complex 3D immersed boundaries on Cartesian grids. J Comput Phys 191:660–669

Ji H, Lien FS, Yee E (2008) A robust and efficient hybrid cut-cell/ghost-cell method with adaptive mesh refinement for moving boundaries on irregular domains. Comput Methods Appl Mech Eng 198:432–448

Kang M, Fedkiw RP, Liu XD (2000) A boundary condition capturing method for multiphase incompressible flow. J Sci Comput 15:323–360

Kumar M, Roy S (2016) A sharp interface immersed boundary method for moving geometries with mass conservation and smooth pressure variation. Comput Fluids 137:15–35

Kumar SP, De A, Das D (2015) Investigation of flow field of clap and fling motion using immersed boundary coupled lattice Boltzmann method. J Fluids Struct 57:247–263

Liu C, Hu C (2018) An adaptive multi-moment FVM approach for incompressible flows. J Comput Phys 359:239–262

Mittal R, Iaccarino G (2005) Immersed boundary methods. Annu Rev Fluid Mech 37:239–261

Mittal R, Dong H, Bozkurttas M, Najjar FM, Vargas A, Von Loebbecke A (2008) A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. J Comput Phys 227:4825–4852

Möller T, Trumbore B (2005) Fast, minimum storage ray/triangle intersection. In: ACM SIGGRAPH 2005 courses, ACM, pp 7

Muralidharan B, Menon S (2018) Simulation of moving boundaries interacting with compressible reacting flows using a second-order adaptive cartesian cut-cell method. J Comput Phys 357:230–262

Ohmi K, Coutanceau M, Daube O, Loc TP (1991) Further experiments on vortex formation around an oscillating and translating airfoil at large incidences. J Fluid Mech 225:607–630

Onishi K, Obayashi S, Nakahashi K, Tsubokura M (2013) Use of the immersed boundary method within the building cube method and its application to real vehicle cad data. In: 21st AIAA computational fluid dynamics conference, pp 2713

Peskin CS (2002) The immersed boundary method. Acta Numer 11:479–517

Senocak I, Sandusky M, DeLeon R, Wade D, Felzien K, Budnikova M (2015) An immersed boundary geometric preprocessor for arbitrarily complex terrain and geometry. J Atmos Ocean Technol 32:2075–2087

Seo JH, Mittal R (2011) A sharp-interface immersed boundary method with improved mass conservation and reduced spurious pressure oscillations. J Comput Phys 230:7347–7363

Seshadri PK, De A (2018) Assessment of pressure reconstruction schemes in sharp interface immersed boundary method. In: AIP conference proceedings, AIP Publishing, pp 030002

Udaykumar H, Mittal R, Rampunggoon P, Khanna A (2001) A sharp interface cartesian grid method for simulating flows with complex moving boundaries. J Comput Phys 174:345–380

Yang J, Stern F (2013) Robust and efficient setup procedure for complex triangulations in immersed boundary simulations. J Fluids Eng 135:101107

Zhu L, Peskin CS (2002) Simulation of a flapping flexible filament in a flowing soap film by the immersed boundary method. J Comput Phys 179:452–468