Immersed NURBS for CFD Applications

Jeremy Veysset, Ghina Jannoun, Thierry Coupez, and Elie Hachem

Abstract We present a new immersed method for Computational Fluid Dynamics applications. It is based on the use of Non Uniform Rational B-Splines (NURBS). The distance function to an immersed solid is computed directly from its Computer Aided Design (CAD) description. This allows to bypass the generation of surface meshes and to obtain accurate levelset functions for complex geometries. Combined with a metric based anisotropic mesh adaptation and stabilized Finite Elements Method (FEM), it allows a novel, efficient and flexible approach to deal with a wide range of fluid structure interaction problems. The metric field is computed directly at the node of the mesh using the length distribution tensor and an edge based error analysis. Several 2D and 3D numerical examples will demonstrate the applicability of the proposed method.

1 Introduction

Immersed methods for Fluid Structure Interaction (FSI) are gaining popularity in many scientific and engineering applications. Different approaches can be found such as the embedded boundary method [1], the immersed boundary method [2], the fictitious domain [3], the immersed volume method [4–7] and the cartesian method [8]. All these methods are attractive because they simplify a number of issues in Fluid-Structure applications such as meshing the fluid domain, using a fully Eulerian algorithm, problems involving large structural motion and deformation [9] or topological changes [10].

However they use non-body fitted grids which require special interface treatments. Indeed recent developments are focusing on issues related to the immersion of a surface mesh for complex 3D geometries [7], the detection and the intersection algorithms for the interface and finally the transmission of boundary conditions

J. Veysset • G. Jannoun • T. Coupez • E. Hachem (✉)
MINES ParisTech, Center for Materials Forming (CEMEF), UMR CNRS 7635,
BP 207, 06904 Sophia-Antipolis, France
e-mail: jeremy.veysset@mines-paristech.fr; ghina.jannoun@mines-paristech.fr;
thierry.coupez@mines-paristech.fr; elie.hachem@mines-paristech.fr

S. Perotto, L. Formaggia (eds.), *New Challenges in Grid Generation and Adaptivity for Scientific Computing*, SEMA SIMAI Springer Series 5,
DOI 10.1007/978-3-319-06053-8_7

between the solid and the fluid regions. In particular these methods appear to be limited by the quality and the accuracy of the surface mesh description of a given immersed solid.

In this work, we present a new immersion technique that simplifies and bypasses the generation of a surface mesh. It is based on the use of Non Uniform Rational B-Splines (NURBS) curves or surfaces, representing simple or complex geometries. We compute the distance function from any point in the fluid mesh to these NURBS, thus representing the immersed solid by the zero iso-value of this function. The computation of the distance mainly relies on patching the NURBS functions [11] and using a Newton method [12]. Although, many methods and techniques have been already developed to compute the distance to NURBS functions, none of them has been used to compute level-set functions for immersed objects needed to solve FSI problems. Therefore instead of relying on the resolution of the surface mesh, the proposed method uses directly the Computer Aided Design (CAD) definition which keeps the quality of its analytical description. In practice, it eliminates the cost of the surface mesh generation step and reduces the complexity to set up a Fluid-Structure application.

Combined with anisotropic mesh adaptation, it provides an attractive immersed framework. Therefore, for the mesh adaptation, we retain the use of a metric constructed directly at the nodes of the mesh without any direct information from the elements, neither considering any underlying interpolation [13–15]. It is performed by introducing a statistical concept: the length distribution function. First, we use a second order tensor to approximate the distribution of lengths defined by gathering the edges at the node. Then we compute the error along and in the direction of each edge. Finally we extend the approach to deal with multicomponent fields (tensors, vectors, scalars). It uses a single metric to account for different fields such as the levelset function of the immersed solid and all components of the velocity field. Note also that the proposed algorithm is implemented in the context of adaptive meshing under the constraint of a fixed number of nodes. With such an advantage, we can provide a very useful tool for practical FSI problems and avoid a drastic increase in the number of nodes. The paper is structured as follows. Section 2 presents the details of the new immersion technique. Section 3 describes the used error estimator for anisotropic mesh adaptation. In Sect. 4 several numerical examples are used to highlight the capability of the approach. Finally conclusions and perspectives are given in Section 5.

2 NURBS Immersion

Let us first recall some notations and definitions of the NURBS functions. All the steps that constitute the computation of the distance function to an immersed solid defined by NURBS functions will be outlined. First, we highlight the relation between the computation of the distance function and the geometrical problem. Based on the principle of an elimination criterion, we obtain the needed initial guess

for the Newton resolution. Finally a simple algorithm is used to sign the obtained distance function, positive in the solid domain and negative outside.

2.1 Definition of NURBS Functions

NURBS or Non-Uniform Rational B-Spline functions are piecewise-polynomial parametric functions. They were introduced in the 1950s [16, 17] in the industrial engineering field to represent complicated geometries like ship hulls and aircraft exterior surfaces. They are now widely implicated in the CAD field and used in many designing softwares (CATIA, Pro Engineer, SolidWorks...). With such mathematical functions, it is possible to represent any geometry with different levels of complexity. Their main advantage is that they can be locally modified by just moving control points without affecting the rest of the geometry. Figure 1 shows an example of a NURBS curve with the corresponding control points and knots. The definition of a NURBS curve c is as follows:

$$c(u) = \frac{\sum_{i=1}^{n} N_{i,p}(u)\omega_i P_i}{\sum_{i=1}^{n} N_{i,p}(u)\omega_i} \tag{1}$$

where p is the degree of the curve, $N_{i,p}$ the basis functions, P_i the control points, n the number of control points, ω_i the weights and u the parameter taking its values in the knot vector U. The knot vector U has $n + p + 1$ knots. The first and last knots have multiplicity $p + 1$ ($U = \{\underbrace{u_0, \ldots, u_0}_{p+1}, u_1, \ldots, u_{n-1}, \underbrace{u_n, \ldots, u_n}_{p+1}\}$). The basis functions are defined by the Cox-De Boor recursion formula [18, 19]:

$$N_{i,0}(u) = \begin{cases} 1 \; if \; u_i \leq u < u_{i+1} \\ 0 \; otherwise \end{cases} \tag{2}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u), \; with \; p \in \mathbb{N}^*. \tag{3}$$

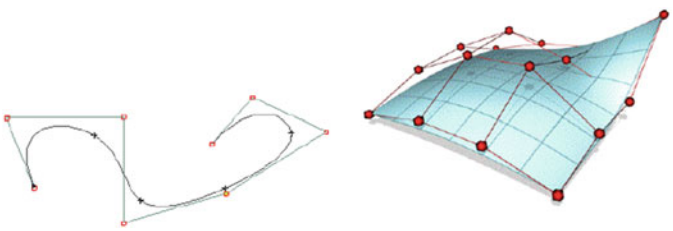

Fig. 1 Example of a NURBS curve and a NURBS surface, their control points and knots

Following the definition given by (1), a NURBS surface is defined as follows:

$$s(u,v) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} N_{i,p}(u)N_{j,q}(v)\omega_{ij}P_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} N_{i,p}(u)N_{j,q}(v)\omega_{ij}} \tag{4}$$

where p and q are the polynomial degrees in the u and v directions, $N_{i,p}$ and $N_{j,q}$ the basis functions in the u and v directions, P_{ij} the control points, ω_{ij} the weights and u and v the parameters taking their values in the U and V knot vectors. The latters are constructed in the same way as mentioned previously in the NURBS curve definition.

2.2 The Closest Point Problem

The objective is to compute the level-set of the immersed objects involved in the simulations directly from their CAD definition, i.e. their CAD files. Indeed, in these files, each object is commonly characterized by NURBS curves or surfaces. Let Ω, Ω_f, Ω_s and Γ represent respectively the whole domain, the fluid domain, the solid domain and the interface verifying:

$$\begin{cases} \Omega_f \bigcup \Omega_s = \Omega \\ \Omega_f \bigcap \Omega_s = \Gamma \end{cases}. \tag{5}$$

Then for each node X of the computational domain Ω, the level-set function α which is the signed distance from the interface reads:

$$\alpha(X) \begin{cases} > 0 \text{ if } X \in \Omega_s \\ = 0 \text{ if } X \in \Gamma \\ < 0 \text{ if } X \in \Omega_f \end{cases}. \tag{6}$$

The immersed solid is implicitly defined by the zero iso-value of this function α. In what follows, we describe the algorithm to compute the minimum distance between the nodes of the computational mesh and the surface of the immersed object. This can be achieved by solving the closest point problem, which can be seen as a root finding problem [20]. In fact, if we consider a point P and a NURBS curve c, the projection of the point P on the curve c is mathematically equivalent to finding the parameter u^* such that:

$$(P - c(u^*)).c'(u^*) = 0. \tag{7}$$

This kind of problem can be solved by using a Newton method. It requires a good starting value in order to obtain fast and accurate results. Different approaches are proposed in the literature. In [12], the authors make a sampling of points on the curve and take as an initial value the closest one. However, this method has been described as time consuming. We will adopt here a more efficient way to find a good initial value [21]. It consists first in selecting the part of the curve containing the root. Then the initial value is taken on this part of the curve and the Newton method is performed only on this part. Therefore, we subdivide the NURBS curve into rational Bezier segments as a preparation phase. We recall that a rational Bezier curve c of degree p is defined by:

$$c(u) = \frac{\sum_{i=0}^{p} B_{i,p}(u)\omega_i P_i}{\sum_{i=0}^{p} B_{i,p}(u)\omega_i} \tag{8}$$

where P_i are the control points, ω_i the weights and $B_{i,p}$ the Bernstein polynomials defined by the following formula:

$$B_{i,p}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i}. \tag{9}$$

Then we eliminate the Bezier segments that do not satisfy a certain criterion (this criterion is detailed thereafter). Finally we use a Newton-Raphson method to solve the point projection problem (7) on the remaining rational Bezier segments. Analogously, a NURBS surface can be decomposed into a set of rational Bezier surfaces. Then the same scheme is performed in order to find the minimum distance relatively to the NURBS surface.

$$s(u,v) = \frac{\sum_{i=0}^{p}\sum_{j=0}^{q} B_{i,p}(u)B_{j,q}(v)\omega_{ij}P_{ij}}{\sum_{i=0}^{p}\sum_{j=0}^{q} B_{i,p}(u)B_{j,q}(v)\omega_{ij}} \tag{10}$$

where s is the rational Bezier surface, p and q the degrees of s, P_{ij} the control points and ω_{ij} the weights.

Different alternatives are proposed in the literature. For instance, in [22] and [23], the Bezier segments are subdivided until the created control polygons become simple and convex or until a flatness condition is reached. In our case, the Bezier segments are not subdivided. In [24], the authors prefer to use a geometric criterion for the elimination of the rational Bezier curves based on computing the tangent cone of every rational Bezier curve. In [25], they introduce an algebraic function instead of subdividing the NURBS geometry. Consequently, Eq. (7) is transformed into a polynomial equation and the roots of this new equation are extracted using a Sturm method.

Algorithm 1 Closest Extremity

if $\forall$ i $\in$ [1,n] $P_1P_i.PP_1 \geq 0$ **then**
  P_1 is the closest point
else if $\forall$ i $\in$ [1,n] $P_nP_i.PP_n \geq 0$ **then**
  P_n is the closest point
else
  Go to Algorithm 2
end if

Algorithm 2 Segment Elimination

if $\forall$ i $\in [1, p+1]$ $P_{k,1}P_{k,i}.PP_{k,1} \geq 0$ **then**
  $P_{k,1}$ is the closest point and the Bezier patch B_k is eliminated
else if $\forall$ i $\in [1, p+1]$ $P_{k,n}P_{k,i}.PP_{k,n} \geq 0$ **then**
  $P_{k,n}$ is the closest point and the Bezier patch B_k is eliminated
else
  Apply Newton method
end if

2.3 Outline of the Algorithm

Inspired by all the above described works, we present a new modified algorithm, adapted mainly from [21]. We first subdivide the curve into a set of rational Bezier segments B_k as a preparation phase. Then we check if one of the extremities of the NURBS curve is the closest point (Algorithm 1).

If the closest point is not an endpoint, we eliminate all the subcurves B_k whose closest point from point P is an extremity of B_k (Algorithm 2).

$P_{k,i}$ being the control points of the Bezier segment B_k. If all the subcurves B_k have been suppressed, then the curve has got at least a cust and the closest point is one of these custs (point of multiplicity equal to p, p being the degree of the curve). Thus we compute the distance for all the singular points and check which one is the closest. Otherwise we look for the closest point with a Newton method on the remaining sub segments.

Analogously, we transpose the algorithm to compute the closest distance between a point and a NURBS surface. We first subdivide the surface into a set of rational Bezier patches (i.e. surfaces) B_k. Then, as for NURBS curves, we check if one of the corners of the NURBS surface is the closest point (Algorithm 3). If none of the corners of the NURBS surface is selected as the closest point of the query point P, we eliminate the rational Bezier patches B_k whose closest point from point P is a corner of B_k (Algorithm 4). If all the sub patches B_k have been suppressed, then the surface has got at least a cust and the closest point is one of these custs (point of multiplicity equal to p and q, p and q being the degrees of the curve respectively in the u and v directions). Thus we compute the distance for all the singular points and check which one is the closest. Otherwise we look for the closest point using a Newton method on the remaining sub-patches.

Algorithm 3 Closest Corner

if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{11}P_{ij}.PP_{11} \geq 0$ **then**
  P_{11} is the closest point.
else if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{m1}P_{ij}.PP_{m1} \geq 0$ **then**
  P_{11} is the closest point.
else if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{1n}P_{ij}.PP_{1n} \geq 0$ **then**
  P_{11} is the closest point.
else if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{mn}P_{ij}.PP_{mn} \geq 0$ **then**
  P_{11} is the closest point.
else
  Go to Algorithm 4
end if

Algorithm 4 Patch Elimination

if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{k,11}P_{k,ij}.PP_{k,11} \geq 0$ **then**
  $P_{k,11}$ is the closest point and the Bezier patch B_k is eliminated
else if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{k,m1}P_{k,ij}.PP_{k,m1} \geq 0$ **then**
  $P_{k,m1}$ is the closest point and the Bezier patch B_k is eliminated
else if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{k,1n}P_{k,ij}.PP_{k,1n} \geq 0$ **then**
  $P_{k,1n}$ is the closest point and the Bezier patch B_k is eliminated
else if $\forall$ i $\in$ [1,m] and $\forall$ j $\in$ [1,n] $P_{k,mn}P_{k,ij}.PP_{k,mn} \geq 0$ **then**
  $P_{k,mn}$ is the closest point and the Bezier patch B_k is eliminated
else
  Apply Newton method
end if

2.4 The Newton Method Resolution

The last part of the algorithm consists in solving, using the Newton method, the point inversion problem (7) on the selected segments or patches of the NURBS function. Since the corners of the NURBS function have been treated in the previous subsection, the distance between a point and a NURBS function can now simply be expressed as an orthogonal point projection problem [c.f. Eq. (7)]. The segment constituted by the query point and the closest point on the curve is orthogonal to the derivative of the curve at this closest point. From the Taylor expansion of Eq. (7), we can state that the parameter of the curve in the Newton algorithm is computed as follows:

$$u_{i+1} = u_i - \frac{(c(u_i) - P).c'(u_i)}{(c(u_i - P).c''(u_i) + \|c'(u_i)\|^2}. \tag{11}$$

The algorithm is performed as far as the parameter value does not change significantly or until Eq. (7) is satisfied under a given precision. Analogously, the problem statement for finding the distance between a point and a NURBS surface is the following, find the parameters u and v such that:

$$\begin{cases} a(u,v) = (s(u,v) - P).s_u(u,v) = 0 \\ b(u,v) = (s(u,v) - P).s_v(u,v) = 0 \end{cases} \tag{12}$$

where s_u and s_v are the partial derivatives respectively in the u and v directions of the NURBS surface s. The problem is transformed by solving iteratively the following system:

$$\begin{bmatrix} a_u(u_i, v_i) a_v(u_i, v_i) \\ b_u(u_i, v_i) b_v(u_i, v_i) \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} -a(u_i, v_i) \\ -b(u_i, v_i) \end{bmatrix} \tag{13}$$

where a_u, a_v, b_u and b_v are the partial derivatives respectively in the u and v directions of a and b. Replacing (12) in (13) gives:

$$J_i . \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} -(s(u_i, v_i) - P).s_u(u_i, v_i) \\ -(s(u_i, v_i) - P).s_v(u_i, v_i) \end{bmatrix} \tag{14}$$

$$\text{with } J_i = \begin{bmatrix} \|s_u(u_i, v_i)\|^2 + (s(u_i, v_i) - P).s_{uu}(u_i, v_i) s_u(u_i, v_i).s_v(u_i, v_i) \\ \quad + (s(u_i, v_i) - P).s_{uv}(u_i, v_i) \\ s_u(u_i, v_i).s_v(u_i, v_i) + (s(u_i, v_i) - P).s_{vu}(u_i, v_i) \|s_v(u_i, v_i)\|^2 \\ \quad + (s(u_i, v_i) - P).s_{vv}(u_i, v_i) \end{bmatrix}. \tag{15}$$

Finally the parameters are computed by the following equation:

$$\begin{bmatrix} u_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} + \begin{bmatrix} u_i \\ v_i \end{bmatrix}. \tag{16}$$

The method is performed iteratively until the u and v parameters do not change significantly or both equations in (12) are satisfied under a given precision.

2.5 *Computing the Sign of the Distance*

Now that the distance has been obtained with the detailed algorithm, we need to sign it in order to check whether the point lies inside or outside the object. If the point is outside the object, then the distance will take a negative sign and vice versa. We propose two methods for signing the distance. The first one consists in defining a point O lying inside the object and computing the scalar product $\overrightarrow{P_p P}.\overrightarrow{P_p O}$, P being the query point and P_p the closest point of P on the object boundary (Fig. 2).

If the sign of the obtained scalar product is negative, then it means that the point P is outside of the object and the distance takes a negative sign. This method is efficient and easy to implement but its main drawback lies in the fact that it works only for convex objects. The second method is more generic and works for any type of objects. It consists in computing the number of intersections between the edge constituted by the query point P and the inside point O and the object's boundary

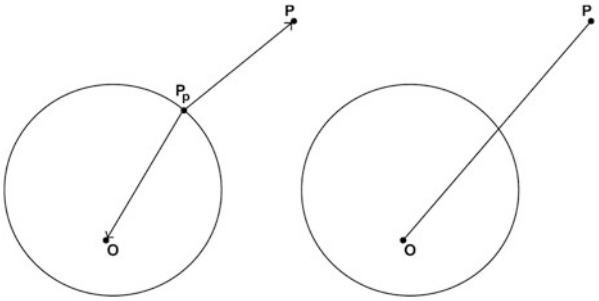

Fig. 2 Scalar product signing method (*left*) and intersection signing method (*right*)

(Fig. 2). If the number of intersections is odd, then the distance takes a negative sign. The outline of the new implemented algorithm takes finally the following form:

1. The NURBS curve (respectively surface) is subdivided into rational Bezier segments (respectively patches).
2. Then we check if one of the corner of the NURBS function is the closest point.
3. If it is the case, go to step 6.
4. Eliminate the rational Bezier segments (respectively patches) that do not contain the closest point.
5. Compute the closest point with a Newton method on the remaining segment (respectively patch).
6. Sign the distance.

3 Construction of an Anisotropic Mesh

In this section, we recall important features of the anisotropic meshing approach relying on the length distribution tensor approach and the associated edge based error analysis as developed in [13].

3.1 Edge Based Error Estimation

We consider $u \in \mathscr{C}^2(\Omega) = \mathscr{V}$ and $\mathscr{V}_h$ a simple P^1 finite element approximation space:

$$\mathscr{V}_h = \left\{ w_h \in \mathscr{C}^0(\Omega), w_h|_K \in P^1(K), K \in \mathscr{K} \right\}$$

where $\Omega = \bigcup_{K \in \mathscr{K}} K$ and K is a simplex (segment, triangle, tetrahedron, ...).

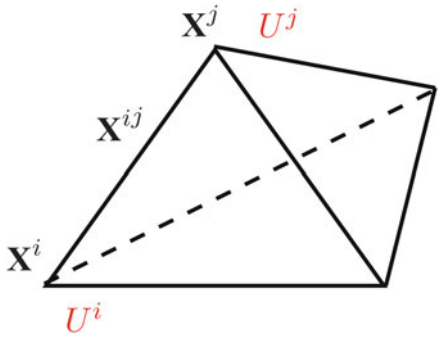

Fig. 3 Length $\mathbf{X^{ij}}$ of the edge joining nodes i and j

We define $\mathbf{X} = \left\{\mathbf{X}^i \in \mathbb{R}^d,\ i = 1, \cdots, N\right\}$ as the set of nodes of the mesh and we denote by U^i the nodal value of u at $\mathbf{X}^i$ and we let Π_h be the Lagrange interpolation operator from $\mathscr{V}$ to $\mathscr{V}_h$ such that:

$$\Pi_h u(\mathbf{X}^i) = u(\mathbf{X}^i) = U^i\ ,\ \forall i = 1, \cdots, N.$$

As shown in Fig. 3, we define the set of nodes connected to node i by $\Gamma(i) = \left\{j\ ,\ \exists^i K \in \mathscr{K}\ ,\ \mathbf{X}^i, \mathbf{X}^j \text{ are nodes of } K\right\}$.

By introducing the following notation: $\mathbf{X}^{ij} = \mathbf{X}^j - \mathbf{X}^i$ and using the analysis carried in [13], we can set the following results:

$$\nabla u_h \cdot \mathbf{X}^{ij} = U^{ij}\,, \tag{17}$$

$$||\underbrace{\nabla u_h \cdot \mathbf{X}^{ij}}_{U^{ij}} - \nabla u(X^i) \cdot \mathbf{X}^{ij}|| \leq \max_{Y \in [X^i, X^j]} |\mathbb{H}(u)(Y)\mathbf{X}^{ij} \cdot \mathbf{X}^{ij}|\,, \tag{18}$$

where $\mathbb{H}(u) = \nabla^{(2)} u$ is the associated Hessian of u. Recall that taking $u \in \mathscr{C}^2(\Omega)$ we obtain $\nabla u \in \mathscr{C}^1(\Omega)$.

Applying the interpolation operator on ∇u and using (17) we obtain a definition of the projected second derivative of u in terms of only the values of the gradient at the extremities of the edge:

$$\nabla g_h \mathbf{X^{ij}} \cdot \mathbf{X^{ij}} = g^{ij} \cdot \mathbf{X^{ij}} \tag{19}$$

where $\nabla g_h = \Pi_h \nabla u$, $g^i = \nabla u(\mathbf{X^i})$ and $g^{ij} = g^j - g^i$.

Using a mean value argument, we set that:

$$\exists y \in [x^i, x^j] | g^{ij} \cdot \mathbf{X^{ij}} = \mathbb{H}(u)(Y)\mathbf{X}^{ij} \cdot \mathbf{X}^{ij}\,.$$

We use this projection as an expression of the error along the edge:

$$e_{ij} = g^{ij} \cdot \mathbf{X^{ij}}. \tag{20}$$

However this equation cannot be evaluated exactly as it requires that the gradient of u be known and continuous at the nodes of the mesh. For that reason, we resort to a gradient recovery procedure.

3.2 Gradient Recovery

Based on an optimization analysis, the author in [13] proposes a recovery gradient operator defined by:

$$G^i = (\mathbb{X}^i)^{-1} \sum_{j \in \Gamma(i)} U^{ij} \mathbf{X}^{ij} \tag{21}$$

where $\mathbb{X}^i = \frac{d}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} \mathbf{X}^{ij} \otimes \mathbf{X}^{ij}$ is what we call the length distribution tensor at node $\mathbf{X}^i$. Note that this construction preserves the second order:

$$\left|\left(G^i - g^i\right) \cdot \mathbf{X^{ij}}\right| \sim \left(\mathbb{H}(u)\mathbf{X}^{ij} \cdot \mathbf{X}^{ij}\right)$$

where G^i is the recovery gradient at node i [given by (21)] and g^i being the exact value of the gradient at node i.

The error is evaluated by substituting G by g in (20):

$$e_{ij} = G^{ij} \cdot \mathbf{X^{ij}}.$$

3.3 Metric Construction from the Edge Distribution Tensor

Taking into account this error analysis, we construct the metric for the unit mesh as follows:

$$\mathbb{M}^i = \left(\frac{d}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} \mathbf{X}^{ij} \otimes \mathbf{X}^{ij} \right)^{-1}.$$

For a complete justification of this result, the reader is referred to [13].

3.4 Error Behavior due to Varying the Edge Length

In this section, we introduce a new way to enforce the number of nodes N and we propose a novel approach to compute the stretching factor without using the

dimensional parameter p as was proposed in [13]. First, we start by examining how the error behaves when we change the length of the edges by stretching coefficients

$$\mathscr{S} = \left\{ s_{ij} \in \mathbb{R}^+ \, , \, i = 1, \cdots, N \, , \, j = 1, \cdots, N \, , \, \Gamma(i) \cap \Gamma(j) \neq \phi \right\}.$$

In order to obtain a new metric depending on the error analysis, one has to calculate first a new length for each edge and then to use it for rebuilding the length distribution tensor. An interesting way of linking the error variations to the changes in edge lengths is by introducing a stretching factor $s \in \mathbb{R}$ such that

$$\begin{cases} \widetilde{\mathbf{X}_{ij}} = s\mathbf{X}_{ij} \\ ||\widetilde{e_{ij}}|| = s^2||e_{ij}|| = s^2||G^{ij} \cdot \mathbf{X}_{ij}|| \end{cases} \tag{22}$$

where $\widetilde{e_{ij}}$ and $\widetilde{\mathbf{X}_{ij}}$ are the target error at edge ij and its associated edge length.

Following the lines of [13] we can simply define the metric associated with $\mathscr{S}$ by:

$$\widetilde{\mathbb{M}^i} = \frac{1}{d} \left(\widetilde{\mathbb{X}^i} \right)^{-1} \tag{23}$$

where

$$\widetilde{\mathbb{X}^i} = \frac{1}{\Gamma(i)} \sum_{j \in \Gamma(i)} s_{ij}^2 \mathbf{X}^{ij} \otimes \mathbf{X}^{ij}$$

is the length distribution tensor. Let n_{ij} be the number of created nodes in relation with the stretching factor s_{ij} and along the edge ij. When scaling the edges by a factor s_{ij}, the error changes quadratically so that the number of created nodes (number of sub-edges as shown in Fig. 4) along the edge ij is given by:

$$n_{ij} = \left(\frac{\widetilde{e_{ij}}}{e_{ij}} \right)^{\frac{1}{2}} = s_{ij}^{-1}.$$

Here $\widetilde{e_{ij}}$ denotes the induced error for edge $\widetilde{\mathbf{X}^{\mathbf{ij}}}$.

Giving the number of nodes (or sub-edges) created along the current edge, it is possible now to build a tensor of distribution of nodes in all directions by solving

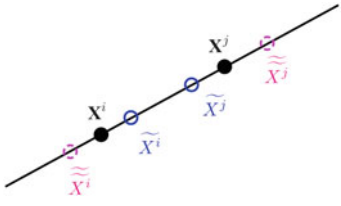

Fig. 4 Varying the edge in its own direction

the following optimization problem:

$$\min_i \sum_{j \in \Gamma(i)} |N^i \cdot \mathbf{X}^{ij} - n_{ij}\mathbf{X}^{ij} \otimes \mathbf{X}^{ij}|^2$$

where

$$N^i = \det(N^i) = \det\left((\mathbb{X}^i)^{-1} \sum_{j \in \Gamma(i)} n_{ij}\mathbf{X}^{ij} \otimes \mathbf{X}^{ij} \right).$$

By considering the averaging process of the number of nodes distribution function, the total number of nodes in the adapted mesh is given by

$$N = \sum_i N^i.$$

Assuming a uniform totally balanced error along the edge, $\widetilde{e_{ij}} = e$ is constant, we get a direct relation between N and e as follows:

$$N^{ij}(e) = s_{ij}^{-1}(e) = \left(\frac{\widetilde{e_{ij}}}{e_{ij}} \right)^{+\frac{1}{2}}.$$

For a node i we have

$$N^i(e) = \det\left(\left(\frac{1}{d} \right) (X^i)^{-1} \sum_{j \in \Gamma(i)} N_{ij}(e)\mathbf{X}^{ij} \otimes \mathbf{X}^{ij} \right)$$

with

$$N^i(e) = e^{\frac{2}{d}} N^i(1)$$

so that

$$N = e^{\frac{2}{d}} \sum_i N^i(1).$$

Hence, the global induced error for a given total number of nodes N can be determined by:

$$e(N) = \left(\frac{N}{\sum_i N^i(1)} \right)^{-\frac{4}{d}}.$$

Therefore the corresponding stretching factors under the constraint of a fixed number of nodes N are given by:

$$s_{ij} = \left(\frac{\widetilde{e_{ij}}}{e(N)} \right)^{-\frac{1}{2}}.$$

3.4.1 Extension to Multi-Component Field

Here we propose to construct a unique metric directly from a multi-component vector field containing, for instance, all the components of the velocity field and/or different levelset functions of the immersed solids. Consequently, we do not need to intersect several metrics but construct it using the following error vector: $\mathbf{e_{ij}} = \left\{ e_{ij}^1, e_{ij}^2, \cdots, e_{ij}^n \right\}$.

Let us introduce $u = \{u_1, u_2, \cdots, u_n\}$,

$$\mathbb{Z} = \mathscr{V} \times \mathscr{V} \times \cdots \times \mathscr{V}$$

and

$$\mathbb{Z}_h = \mathscr{V}_h \times \mathscr{V}_h \times \cdots \times \mathscr{V}_h.$$

In the view of constructing a unique metric, we choose to apply the above theory for each component of u. It comes out immediately that the error is now a vector given by the following expression:

$$\overrightarrow{e_{ij}} = \left\{ e_{ij}^1, e_{ij}^2, \cdots, e_{ij}^n \right\}$$

and then

$$s_{ij} = \left(\frac{||\widetilde{e_{ij}}||}{||\overrightarrow{e_{ij}}||} \right)^{-\frac{1}{2}}.$$

Here, the norm can be L_2, L_1 or L_∞. In the following numerical experiments, we used the L^2 case to compute the error.

3.5 Application to the Velocity Field and to the Levelset Function

Let $v_h(X^i) = V^i \in \mathbb{R}^d, d = 2, 3$ the finite element solution of the Navier-Stokes equations. Introduce the vector field $\mathscr{Y} = \left(\frac{v}{|v|}, |v|, \alpha \right)$ made of $d + 1$ components

vector fields. Recall that α is the level set function used to localize an immersed body. We obtain then for every node i,

$$\Pi_h \mathscr{Y}(X^i) = \left\{ \frac{V^i}{|V^i|}, |V^i|, \alpha \right\} = \mathscr{Y}^i.$$

Obviously the case $|v| = 0$ must be accounted for by using $\frac{V^i}{\max(|v^i|,\varepsilon)}$ with $\varepsilon \approx 10^{-6}$ chosen as a small value so that $\mathscr{Y}_k^i = 0$ when $|v^i| = 0$.

Using the vector $\mathscr{Y}^i$, the adaptivity will now take into account, using one unique metric, the variations in the velocity directions, the velocity norm and the levelset functions. Indeed, the adaptivity will focus mainly on the change of direction rather than the intensity of the velocity. Consequently, and as presented by the numerical results in the following section, even the small vortices developed by the solution will be very well captured. What is even more interesting is the capability of the method to automatically detect the boundary layers at the fluid-solid interfaces due to the anisotropically adapted mesh exhibiting highly stretched elements. Finally, we recall that we use a mesh technique (MTC) based on the local modification and the conformity control through the theorem for minimal volume preserving. This was introduced in [26] and extended to anisotropic mesh adaptation in [13, 27].

4 Applications

The performance of the new NURBS immersed method will be assessed using several 2D and 3D examples. First we show that combining the new immersed method with anisotropic mesh adaptation can lead to a novel, efficient and flexible immersed framework able to handle simple and very complex geometries. Then, we combine it with flow solvers based on a stabilized three-fields velocity-pressure-stress finite element formulation, designed for the computation of rigid bodies in an incompressible Navier-Stokes flow at high Reynolds number. Indeed, this formulation consists of considering the whole domain as a single one, meshed by a single grid, and solved with an Eulerian framework. Continuity at the fluid-solid interface is then obtained naturally and there is no need to enforce it. Then, it imposes the use of an appropriate constitutive equation describing both the fluid and the solid domain. For instance, the presence of the solid is taken into account as an extra stress in the Navier-Stokes equation [6, 28]. The results show that the method is very efficient and robust in particular at high Reynolds numbers using anisotropic meshes with highly stretched elements.

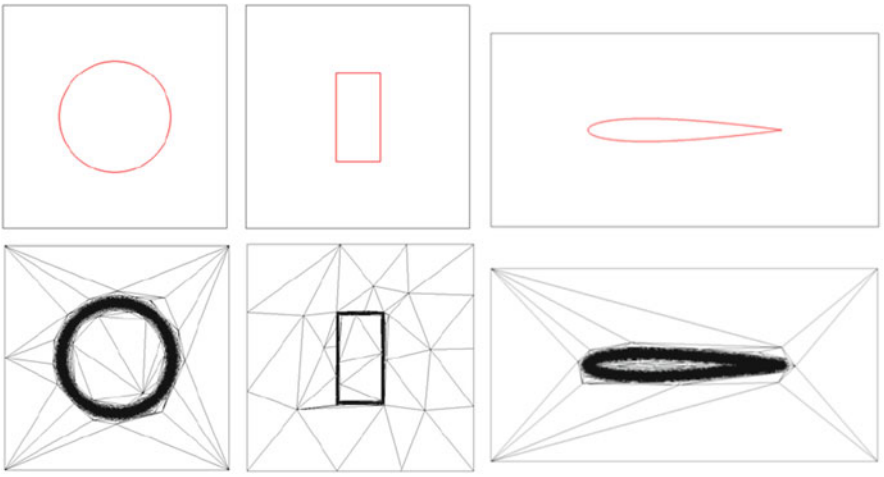

Fig. 5 2D applications of the immersed NURBS method: level-set zero iso-value (*top*); adapted meshes (*bottom*)

4.1 Immersed 2D and 3D Simple Geometries

First we test the method by immersing simple objects. Indeed, the distance function for the circle and the rectangle can be obtained easily using analytical functions. Therefore, they will be used first to test the implemented algorithm, in particular in the presence of curvatures, sharp angles and singularity. We immerse the CAD descriptions of a circle, a rectangle and a NACA profile in 2D, a sphere and cube in 3D. We use the computed levelset functions as the mesh criterion.

Figure 5 presents the zero isovalues of the immersed objects inside the computational domain. As expected, it reflects the sharp capture of the geometries and the right orientation and deformation of the mesh elements (longest edges parallel to the boundary). This yields a great reduction of the number of triangles and consequently a reduction in the computational costs. These first results show that the method works properly and that the obtained results are accurate and respect well the geometry of the objects.

The extension of the method to deal with 3D objects described this time by NURBS surfaces is tested on a sphere and a cube immersed inside a larger domain. Figure 6 shows the zero-isovalues of the computed levelset functions and several cut in the planes highlighting the obtained meshes at the interfaces. Once again the results prove that the implemented method works well and shows that combining the new immersed method with anisotropic mesh adaptation lead to a very practical tool for immersed methods.

Taking a closer look at the mesh near the interfaces, we can detect the good orientation of the elements with the stretching in the right direction. This demonstrates

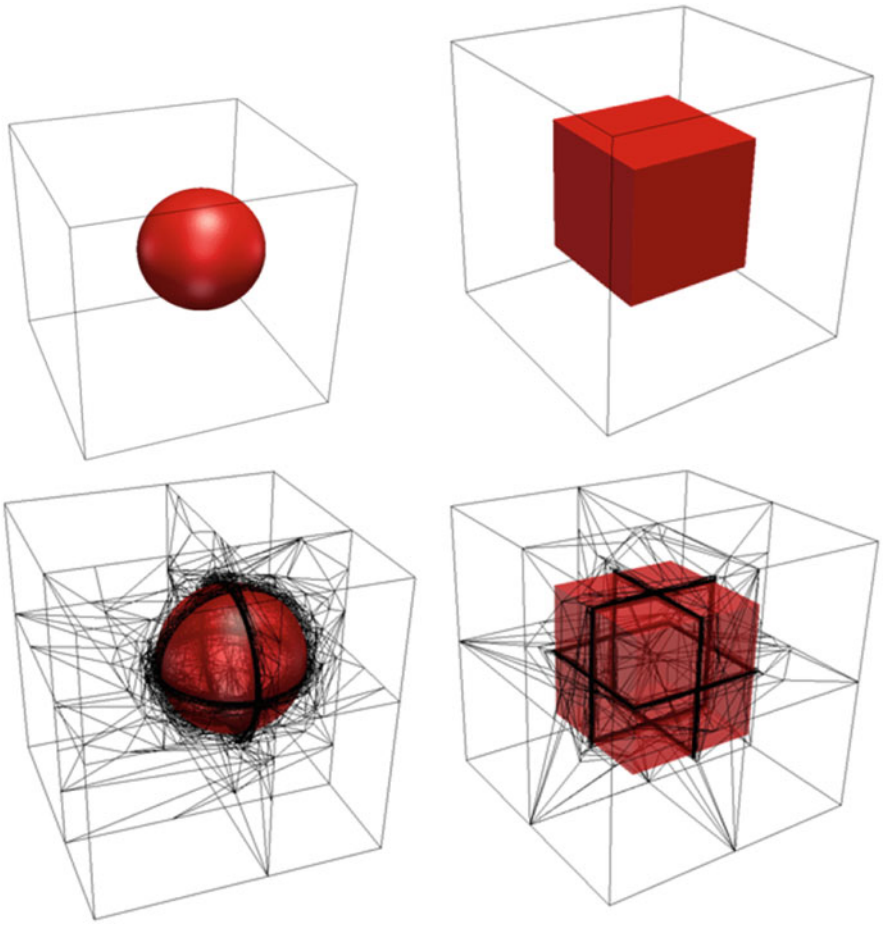

Fig. 6 3D applications of the immersed NURBS method: level-set zero iso-value (*top*); adapted meshes (*bottom*)

the ability of the algorithm to work under the constraint of a fixed number of nodes and to effectively control the elements sizes, orientations and locations.

4.2 Immersed 3D Complex Geometries

In this section, we test the immersed method on complex geometries: a ship hull and a large airship. Two difficulties must be underlined. The first is clearly the edge of the ship hull while the second is the presence of the hole all along the airship. Note also that both geometries are described this time by several NURBS surfaces.

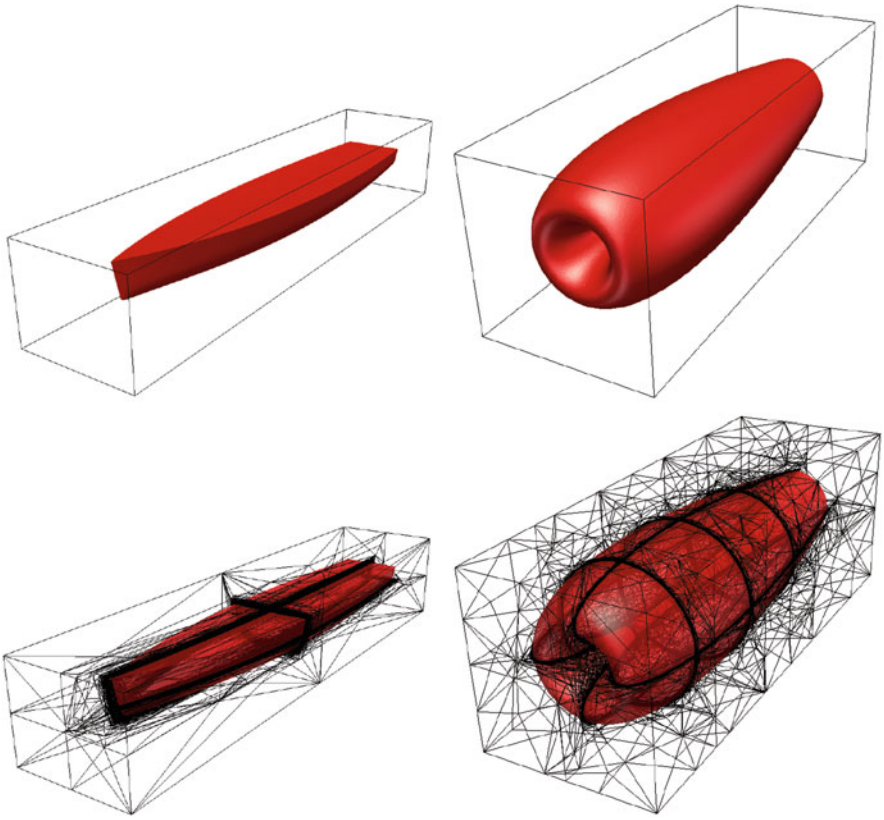

Fig. 7 3D applications of the immersed NURBS method: level-set zero iso-value (*top*); adapted meshes (*bottom*)

The same algorithm is applied iteratively on both geometries: (1) distance function computation using NURBS, (2) sign determination and (3) anisotropic mesh adaptation. The obtained results are shown in Fig. 7. As expected, the algorithm progressively detects and refines the mesh at the interfaces leading to a well respected shape in terms of curvature, angles, etc. All the small details in the given geometries are captured accurately. These observations reflect the ability of the anisotropic mesh adaptation algorithm to automatically adjust the shape and orientation of the elements while optimizing their numbers. For instance, the singularity of these edges could not be recovered without an accurate distance computation and anisotropic refined mesh adaptation.

It is worth mentioning that both the use of NURBS and anisotropic mesh adaptation are complementary. As mentioned previously, immersed objects are usually surface meshes. Therefore the anisotropic mesh adaptation can be limited by the facetization of the object, i.e. the accuracy of the surface mesh file. By immersing

Table 1 Computational time in seconds of the distance calculation of the ship hull immersed with an IGES file, a STL file and the transportation method

n cores	NURBS	Surface mesh	NURBS + transport
1	138.10	13.37	2.72
2	70.92	6.99	2.23
4	43.14	3.53	2.12
8	22.30	2.03	0.70

NURBS objects we overcome this issue as the object geometry is kept analytical. Thus the anisotropic mesh adaptation reaches its full potential.

We present in Table 1 the computational time taken to compute the distance function of the ship hull. We compare several techniques and we use different number of cores (1, 2, 4 and 8) also to test the implementation in a parallel environment. First, we notice that the algorithm works well in parallel and shows a good scalability. Note that we did not extend further this study since it is not in the scope of this paper. Secondly, we compare the present method to the computation of the distance function obtained by immersing a surface mesh (i.e. STL file). Even though the comparison is not fair since the execution time to obtain the surface mesh is not counted and the quality of the surface mesh remains unclear, the purpose of this comparison still gives us an idea on the potential of the method and the possibilities for improvement. However, to make the comparisons fair, we immersed first the ship hull inside a smaller domain using the NURBS, and then we transport the obtained distance function on this refined mesh to the larger computational domain. In the latter case, the cost of this method referred as NURBS + Transport becomes negligible and interesting for practical CFD applications.

4.3 CFD Applications

The objective of this test case is to show the utility of the immersed NURBS method. Indeed, combined with flow solvers it allows to easily and accurately deal with complex fluid structure interaction problems. Therefore, we consider a turbulent flow past an immersed large scale airship. This 3D computations have been obtained using 64 2:4 GHz Opteron cores. The air movement around the airship is quite complex and interesting; i.e. it allows the study of the influence of different airfoils and their positions to optimize the aerodynamic design. A number of vortices between the objects and the surroundings can be observed due to the turbulence dissipation. All these observations are highlighted by the streamlines in Fig. 8. Moreover, we can clearly see on the vertical planes cutting through the airships that the solid region satisfies the zero velocity and, hence, the no-slip condition on the extremely refined interface is also verified. The airship slows down the air circulation on the surface and influences the main air circulation along the hole.

Note also in Fig. 9 the concentration of the resolution not only along all the boundary layers but also at the detachment and in the wake regions. This reflects

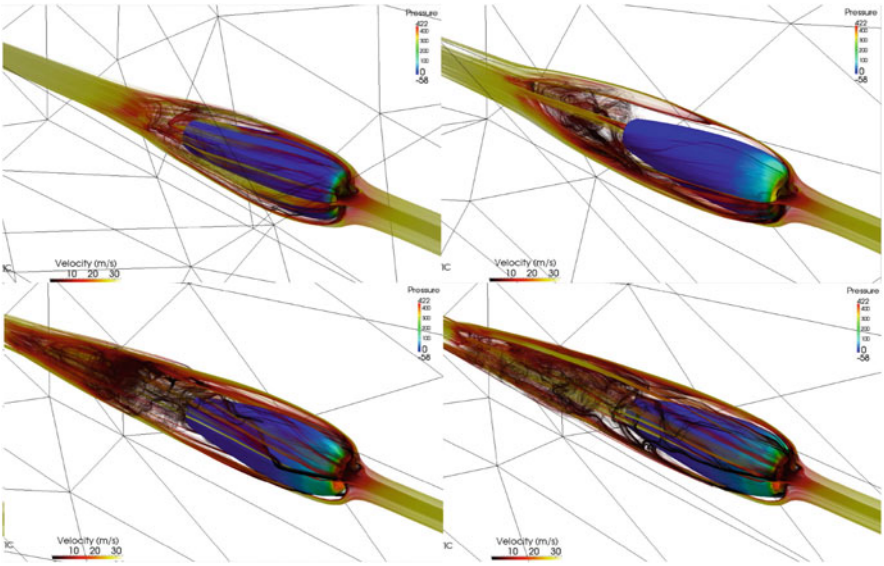

Fig. 8 Snapshots of the streamlines around an airship described by NURBS surfaces

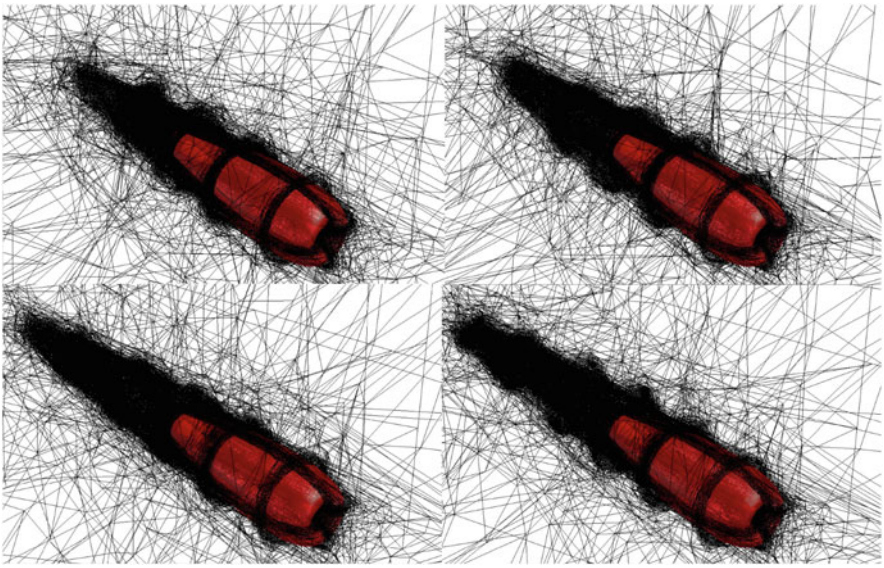

Fig. 9 Snapshots of the adapted mesh around an airship described by NURBS surfaces

well the anisotropy of the solution caused by the discontinuity of the boundary conditions and the nature of the flow. The elements far from the immersed solid are mostly isotropic and increase in size as the velocity gradient decreases. Again, this reflects and explains why, for a controlled number of nodes, the mesh is naturally and automatically coarsened in that region with the goal of reducing the mesh size around the boundaries and in the wake regions.

5 Conclusions

We present a new NURBS immersed method for Computational Fluid Dynamics applications. This method is an extension of the standard Immersed Volume method and more accurate. The immersion of an object described by surface meshes is replaced by the direct use of the CAD definition keeping the quality of its analytical description. The distance computation is performed using a modified algorithm based on the decomposition of the NURBS functions in sub-curves or surfaces and a selection criterion. The Newton method is presented and used to solve the distance problem. The numerical examples show that combined with anisotropic mesh adaptation and flow solver, it leads to a novel, accurate and efficient method to deal with complex fluid structure interaction problems. The natural extension of this work is to optimize and accelerate the implemented algorithm.

Acknowledgements The authors gratefully acknowledge the help of the Agence Nationale de la Recherche Scientifique (ANR), France, under the project ANR-10-REALISTIC-0065.

References

1. Kreiss, H., Petersson, A.: A second order accurate embedded boundary method for the wave equation with dirichlet data. SIAM J. Sci. Comput. **27**, 1141–1167 (2006)
2. Peskin, C.: Flow patterns around heart valves: a numerical method. J. Comput. Phys. **10**, 252–271 (1972)
3. Glowinski, R., Pan, T., Kearsley, A., Periaux, J.: Numerical simulation and optimal shape for viscous flow by a fictitious domain method. Int. J. Numer. Methods Fluids **20**, 695–711 (2005)
4. Hachem, E., Digonnet, H., Massoni, E., Coupez, T.: Immersed volume method for solving natural convection, conduction and radiation of a hat-shaped disk inside a 3d enclosure. Int. J. Numer. Methods Heat Fluid Flow **22**, 718–741 (2012)
5. Hachem, E., Kloczko, T., Digonnet, H., Coupez, T.: Stabilized finite element solution to handle complex heat and fluid flows in industrial furnace using the immersed volume method. Int. J. Numer. Methods Fluids **68**, 99–121 (2012)
6. Hachem, E., Feghali, S., Codina, R., Coupez, T.: Anisotropic adaptive meshing and monolithic variational multiscale method for fluid-structure interaction. Comput. Struct. **122**, 88–100 (2013)
7. Hachem, E., Feghali, S., Codina, R., Coupez, T.: Immersed stress method for fluid structure interaction. Int. J. Numer. Methods Eng. **94**, 805–825 (2013)

8. Johansen, H., Colella, P.: A cartesian grid embedded boundary method for Poisson's equation on irregular domains. J. Comput. Phys. **147**, 60–85 (1998)
9. Farhat, C., Rallu, A., Wang, K., Belytschko, T.: Robust and provably second-order explicit-explicit and implicit-explicit staggered time-integratorsfor highly nonlinear fluid-structure interaction problems. Int. J. Numer. Methods Eng. **84**(1), 73–107 (2010)
10. Farhat, C., Maute, K., Argrow, B., Nikbay, M.: Shape optimization methodology for reducing the sonic boom initial pressure rise. AIAA J. Aircraft **45**, 1007–1018 (2007)
11. Piegl, L., Rajab, K., Smarodzinava, V., Valavanis, K.: Point-distance computations: a knowledge-guided approach. Comput. Aid. Des. Appl. **5**(6), 855–866 (2008)
12. Piegl, L., Tiller, W.: The NURBS Book, 2nd edn. Springer, Berlin Heidelberg (1996)
13. Coupez, T.: Metric construction by length distribution tensor and edge based error for anisotropic adaptive mesing. J. Comput. Phys. **230**, 2391–2405 (2011)
14. Coupez, T., Hachem, E.: Solution of high-reynolds incompressible flow with stabilized finite element and adaptive anisotropic meshing. Comput. Methods Appl. Mech. Eng. **267**, 65–85 (2013)
15. Coupez, T., Jannoun, G., Nassif, N., Nguyen, H., Digonnet, H., Hachem, E.: Adaptive time-step with anisotropic meshing for incompressible flows. J. Comput. Phys. **241**, 195–211 (2013)
16. De Casteljau, P.: Outillages methodes calcul. Tech. rep., A. Citro n, Paris (1959)
17. Bezier, P.: Definition numerique des courbes et surfaces I. Automatisme **XI**, 625–632 (1966)
18. Cox, M.: The numerical evaluation of B-splines. Tech. rep., National Physics Laboratory DNAC4 (1971)
19. De Boor, C.: On calculating with B-splines. J. Approx. Theory **6**, 50–62 (1972)
20. Schneider, P., Eberly, D.: Geometric Tools for Computer Graphics. Morgan Kaufmann, San Francisco (2003)
21. Selimovic, I.: Improved algorithms for the projection of points on NURBS curves and surfaces. Comput. Aid. Geom. Des. **23**, 439–445 (2006)
22. Ma, Y., Hewitt, W.: Point inversion and projection for NURBS curve and surface: control polygon approach. Comput. Aid. Geom. Des. **20**, 79–99 (2003)
23. Dyllong, E., Luther, W.: Distance calculation between a point and a NURBS surface. In: Curve and Surface Design. Saint-Malo, pp. 55–62 (1999)
24. Cohen, E., Johnson, D.: Distance extrema for spline models using tangent cones. In: Proceedings of the Graphics Interface 2005 Conference, Victoria, pp. 169–175, 9–11 May 2005
25. Chen, X.: Improved algebraic algorithm on point projection for Bezier curves. In: Second International Multisymposium on Computer and Computational Sciences, pp. 158–169 (2007)
26. Coupez, T.: A mesh improvement method for 3D automatic remeshing. In: Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, pp. 615–626. Pineridge Press, Swansea (1994)
27. Gruau, C., Coupez, T.: 3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric. Comput. Methods Appl. Mech. Eng. **194**, 4951–4976 (2005)
28. Hachem, E., Rivaux, B., Kloczko, T., Digonnet, H., Coupez, T.: Stabilized finite element method for incompressible flows with high reynolds number. J. Comput. Phys. **229**, 8643–8665 (2010)