algorithms

MDPI

Article

Efficient Local Refinement Near Parametric Boundaries Using kd-Tree Data Structure and Algebraic Level Sets

Tao Song, Huanyu Liao and Ganesh Subbarayan *

School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907, USA; taso@microsoft.com (T.S.); liao85@purdue.edu (H.L.)
* Correspondence: ganeshs@purdue.edu

Abstract: In analysis of problems with parametric spline boundaries that are immersed or inserted into an underlying domain, the discretization on the underlying domain usually does not conform to the inserted boundaries. While the fixed underlying discretization is of great convenience as the immersed boundaries evolve, the field approximations near the inserted boundaries require refinement in the underlying domain, as do the quadrature cells. In this paper, a kd-tree data structure together with a sign-based and/or distance-based refinement strategy is proposed for local refinement near the inserted boundaries as well as for adaptive quadrature near the boundaries. The developed algorithms construct and utilize implicit forms of parametric Non-Uniform Rational B-Spline (NURBS) surfaces to algebraically (and non-iteratively) estimate distance as well as sign relative to the inserted boundary. The kd-tree local refinement is demonstrated to produce fewer sub-cells for the same accuracy of solution as compared to the classical quad/oct tree-based subdivision. Consistent with the kd-tree data structure, we describe a new a priori refinement algorithm based on the signed and unsigned distance from the inserted boundary. We first demonstrate the local refinement strategy coupled with the the kd-tree data structure by constructing Truncated Hierarchical B-spline (THB-spline) "meshes". We next demonstrate the accuracy and efficiency of the developed local refinement strategy through adaptive quadrature near NURBS boundaries inserted within volumetric three-dimensional NURBS discretizations.

Keywords: immersed boundary analysis; Non-Uniform Rational B-splines; kd-tree data structure; adaptive quadrature; algebraic level sets

Citation: Song, T.; Liao, H.; Subbarayan, G. Efficient Local Refinement Near Parametric Boundaries Using kd-Tree Data Structure and Algebraic Level Sets. *Algorithms* **2022**, *15*, 245. https://doi.org/10.3390/a15070245

Academic Editors: Devendra Kumar, Jesper Jansson and Frank Werner

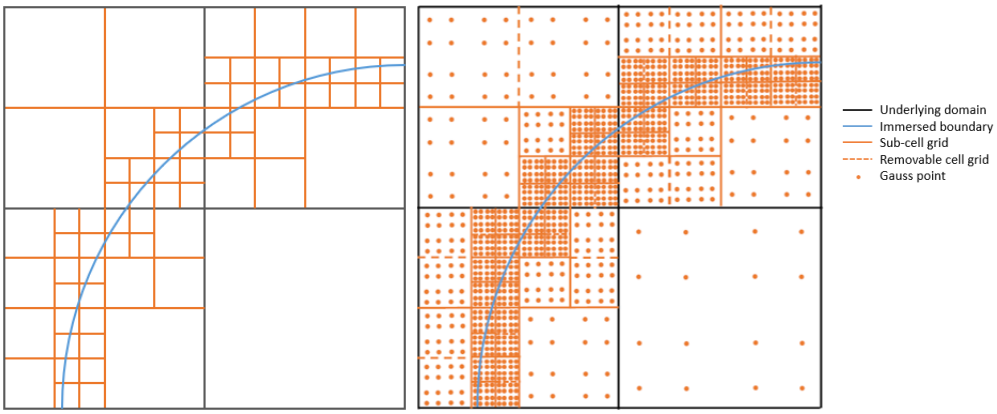
Received: 27 April 2022
Accepted: 12 July 2022
Published: 13 July 2022

1. Introduction

The use of fixed underlying discretizations on which an immersed boundary evolves is a common approach in both finite element analysis and isogeometric analysis (IGA [1,2]) to address many moving boundary problems, including fluid–structure interaction [3,4], phase evolution [5–8], and crack propagation [9–11]. In these problems, the behavioral fields may vary rapidly or even discontinuously across the boundaries, thus necessitating refinement locally near the inserted boundary. There are many well-established approaches to realizing local refinement of cells. A simple and natural subdivision scheme is the space tree illustrated in Figure 1a, that is, a quad-tree in two-dimensions (2D) or oct-tree in three-dimensions (3D).

Subdivision surfaces that converge to smooth limit surfaces are one solution to local refinement of complex free-form parametric surfaces, first proposed by Catmull and Clark [12]. Subdivision surfaces were later used for finite element analysis [13,14] to recursively refine the initial mesh in order to achieve local refinement as well as good continuity at vertices. In the isogeometric framework, the tensor product Non-Uniform Rational B-spline (NURBS) surfaces can be refined (*h*-refinement [15]) through knot insertion. Anisotropic refinement has been studied in [16]. However, these methods do not address local refinement. Extensions to tensor product representations, such as the

Hierarchical B-splines [17], T-splines [18,19], and Truncated Hierarchical B-splines (THB-splines) [20] allow for local refinement. These geometry modeling techniques have been used for isogeometric analysis as well [20–22].



(**a**) Local refinement (**b**) Quatrature points

Figure 1. Illustration of (**a**) refinement of fields and (**b**) their associated quadrature points near immersed boundaries.

In general, the quadrature cells are intersected by the boundaries regardless of the field refinement near the immersed boundaries, and therefore cell-wise numerical integration will be inaccurate. As illustrated in Figure 1b, the immersed boundaries necessitate a separate integration on each side of the boundary; however, an accurate integration over such cut cells is challenging. This inaccuracy is independent of any decision as to the optimal choice of the number of quadrature points discussed in reference [23]. In general, an adaptive quadrature strategy in which the quadrature cells are locally sub-divided in accordance with the immersed boundaries is necessary. Renken and Subbarayan [24] described an adaptive quadrature scheme on trimmed regions of a parametric NURBS surface. Space tree quadrature cell refinement has been extensively used in isogeometric analysis [10,25–30]. However, quad/oct-tree subdivision introduces redundant quadrature cells and significantly increases the computational time during matrix assembly. For example, in the quad-tree quadrature cells shown in Figure 1b, the total number of quadrature cells increases by 276% after a three-level subdivision. The computational performance is even worse if more cells are intersected by the immersed boundaries or if a higher level of subdivision is desired.

To refine cut quadrature cells, adaptive quadrature algorithms that rely on subdivision have been developed to generate boundary-fitted quadrature sub-cells in the cut elements. Rüberg and Cirak [31] planarized the immersed surfaces and then subdivided the hexahedral elements into boundary-fitted tetrahedral sub-cells. Cheng and Fries [32] approximated the boundary segment within each cut element with a polynomial, followed by a subdivision of the cut element into a few serendipity elements. Kudela et al. [33] developed a directly integrable blending formulation for triangles with curved sides or quadrilaterals resulting from the cut elements. While the curved element methods yields very few quadrature sub-cells and preserves geometric accuracy, a robust implementation accounting for all special cases remains non-trivial. Furthermore, these methods are not easily extendable to three-dimensional problems. Thus, all of the existing adaptive quadrature schemes either introduce redundant quadrature points, or require special rules.

In this paper, algorithms are proposed to enhance the efficiency of local refinement and adaptive quadrature near parametric boundaries, including:

1. Novel a priori local mesh refinement algorithms using signed and unsigned distance measures from parametric NURBS surfaces [34,35]. These algorithms construct and utilize implicit forms of parametric NURBS to algebraically (and non-iteratively) identify intersection between cells and immersed parametric boundaries.

2. An adaptive quadrature algorithm using the above refinement strategy coupled with the *k*-dimensional tree (referred to as kd-tree) data structure to efficiently eliminate unnecessary quadrature points and special rules for cut cells.

2. kd-TreeData Structure for Cell Subdivision

The broad idea of kd-tree subdivision is motivated by the process shown in Figure 2, in which the magnified bottom right 2×2 elements in Figure 1a are shown. It can be observed that many adjacent non-intersected quad-tree sub-cells (marked with dashed lines) can be combined into single cells. By removing these dashed lines, the number of sub-cells is reduced without loss of integration accuracy, as demonstrated later in this paper. Thus, for certain cells a complete subdivision into four (2D) or eight (3D) sub-cells is not necessary. A partial subdivision can be carried out by either splitting the cells direction-wise or combining the sub-cells during post-processing. As shown in Figure 2, the total number of quadrature sub-cells decreases by 17% with partial subdivisions.

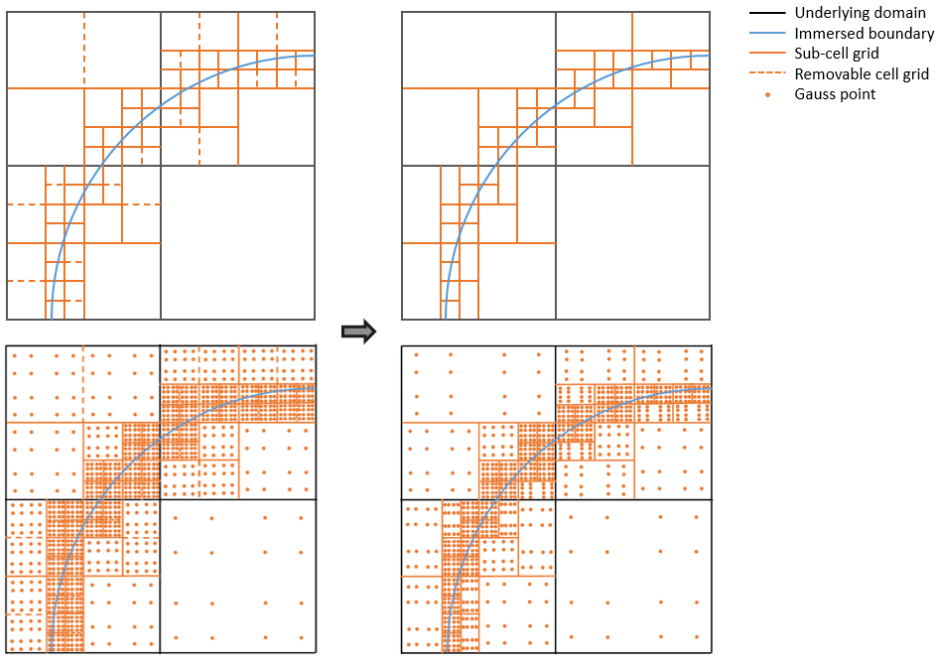


Figure 2. Illustration of sub-cell coalescence when removing non-intersected cell grids.

2.1. Algorithm

kd-tree is a binary space tree that partitions the space dimension-wise, and is a superset of the quad-tree and oct-tree. The kd-tree structure has a long tradition in computational geometry [36] in the context of solving range searching problems. kd-tree parallelization has been extensively investigated to increase construction speed [37,38]. Specifically, given a set of spatial points, a kd-tree can store the points in an orderly manner in order to achieve quick range searching. In this work, the kd-tree is used to store the sub-cells instead of spatial points. A two-dimensional example is shown in Figure 3.

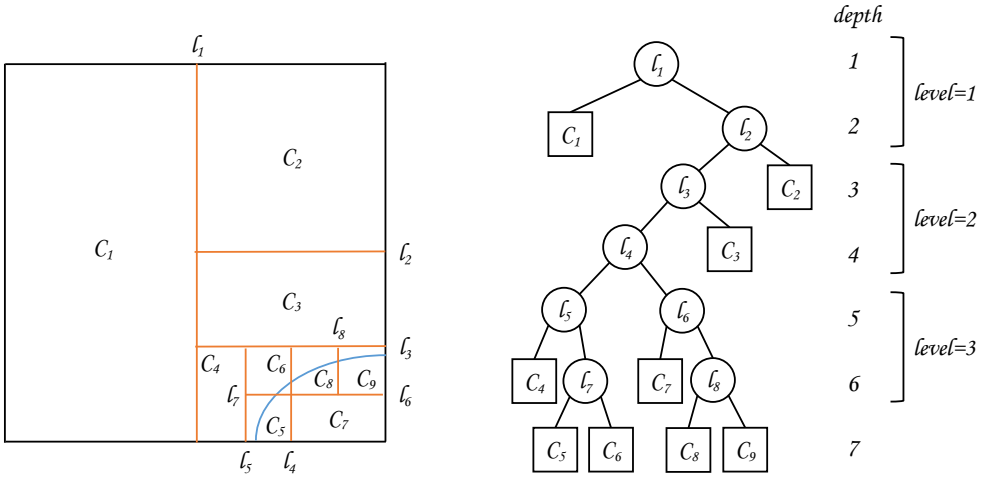


Figure 3. Illustration of kd-tree based sub-division. The maximum level illustrated here is three. Each level consists of two depths that represent different splitting directions; l_i (stored in nodes) and C_i (stored in leaves) denote a splitting line and a sub-cell, respectively.

In a standard kd-tree, each level has k depths, i.e., k-times splitting (e.g., a 2D tree splits in the x and y directions successively, whereas a 3D tree splits in the x, y, and z directions). In the present work, extra flexibility is provided in the splitting procedure by enabling an arbitrary order of the splitting direction within each level. First, we explore all possible splitting directions at each depth, and choose the direction in which the splitting line/plane l does not intersect with the boundary. If the intersection cannot be avoided in any direction, a default splitting direction is chosen. Each split produces two new sub-cells. If any of the sub-cells is not intersected by the boundary, a corresponding leaf is created in the kd-tree. Otherwise, the split continues until the maximum level is reached. The pseudocode of the proposed algorithm is described in Algorithm 1.

Remarks:

1. After the kd-tree is constructed for a given cut element, a tree traversal algorithm, such as Depth First Search (DFS) or Breadth First Search (BFS), can be employed to extract all the leaves of the kd-tree and obtain the sub-cells stored therein. The tree traversal step is generally much faster than tree construction.
2. The intersection between the splitting line/plane and the boundary can be identified by checking the signed algebraic level sets [34,35] between them, as discussed in Section 4. A line or plane is assumed to intersect the boundary if the bounding vertices of the line/plane have opposite signs. The algebraic level sets are a monotonic measure of distance to the NURBS boundary, enabling accurate judgment of the intersection. This intersection check does not introduce additional cost compared with a quad/oct-tree structure, as it is required in the latter as well.
3. In the included numerical examples, the cell is cut in the middle whenever an intersection is detected; this can be further enhanced by unequal splitting of the cells, albeit with the additional cost of determining an accurate intersection location.

Algorithm 1 Algorithm for building kd-tree data structure.

Input: Cell C, depth of tree $depth$, splitting direction vector dir and auxiliary flag $nocheck$
Output: kd-tree v containing quadrature sub-cells in leaves

function BUILDKDTREE(C, $depth$, $dirs$, $nocheck$)
  if $depth \ mod \ Dim = 0$ **then** ▷ Reinitialize at the first depth of each level
    $dir \leftarrow [x, y](2D)$ or $[x, y, z](3D)$
    $nocheck \leftarrow false$
  end if
  $SplitDir \leftarrow dir(1)$ ▷ Set the first element in dir as the default splitting direction
  if $nocheck = false$ **then**
    if C not intersect boundary *or* $depth = MaxDepth$ **then**
      return leaf containing C
    else if $length(dir) \geq 2$ **then** ▷ No further check is needed if only one direction is available
      for $i \leftarrow 1, length(dir)$ **do**
        $nocheck \leftarrow true$
        if Splitting line(2D)/plane(3D) in $dir(i)$ not intersect boundary **then**
          $nocheck \leftarrow false$
          $SplitDir \leftarrow dir(i)$
        end if
      end for
    end if
  end if
  $C'_{half}, C''_{half} \leftarrow$ Split C in $SplitDir$
  $dir_{new} \leftarrow$ Exclude $SplitDir$ from dir
  $v_{left} \leftarrow$ BuildKdTree(C'_{half}, $depth$+1, dir_{new}, $nocheck$)
  $v_{left} \leftarrow$ BuildKdTree(C''_{half}, $depth$+1, dir_{new}, $nocheck$)
  return v with two branches v_{left} and v_{right}
end function

2.2. Comparison with Quad/Oct-Tree

Here, the kd-tree structure is compared with quad-tree and oct-tree in the following three aspects:

1. *Number of Quadrature Sub-cells Generated*: Table 1 summarizes the worst-case and best-case ratios of the number of sub-cells generated by kd-tree to that generated by quad/oct-tree. In the worst-case scenario, the immersed boundary is sufficiently complex that all of the cells have to be completely subdivided in each level. Therefore, the number of sub-cells generated by kd-tree subdivision is the same as that generated by quad-tree and oct-tree. The best-case ratio is proven in Theorem 1.

Table 1. Worst-case and best-case ratios of the number of sub-cells generated by kd-tree, quad-tree, and oct-tree.

Problem Dimension	Worst Case	Best Case
2D ($N_{\text{2d-tree}}/N_{\text{quad-tree}}$)	1	2/3
3D ($N_{\text{3d-tree}}/N_{\text{oct-tree}}$)	1	3/7

Theorem 1. *Given an arbitrary immersed boundary, the number of sub-cells generated by kd-tree subdivision is no smaller than two-thirds of that generated by quad-tree, and three-sevenths of that generated by oct-tree.*

Proof. The minimum kd-tree splits of a 2D and a 3D cell are illustrated in Figure 4. A complete single-level 2D subdivision operation increases the number of sub-cells

by $4-1=3$, whereas an incomplete single-level 2D subdivision only increases it by $3-1=2$. Assume that the total number of complete and incomplete subdivision operations is given by S_c and S_i, respectively. The total number of sub-cells in a 2D tree is provided by

$$N_{\text{2d-tree}} = 1 + 3S_c + 2S_i. \tag{1}$$

Next, assume that the total number of subdivisions is S_{total}. If $S_c = S_{\text{total}}$ and $S_i = 0$, the kd-tree degenerates to a quad-tree, i.e.,

$$N_{\text{quad-tree}} = 1 + 3S_{\text{total}}. \tag{2}$$

Given an arbitrary immersed boundary, the $N_{\text{2d-tree}}/N_{\text{quad-tree}}$ ratio is provided by

$$\frac{N_{\text{2d-tree}}}{N_{\text{quad-tree}}} = \frac{1+3(S_{\text{total}}-S_i)+2S_i}{1+3S_{\text{total}}} \geq \frac{1+2S_{\text{total}}}{1+3S_{\text{total}}} > \frac{2}{3}. \tag{3}$$

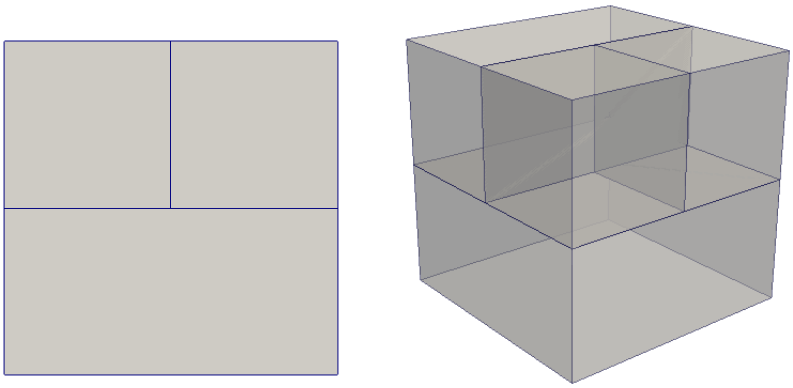
Likewise, a complete single-level 3D subdivision operation adds $8-1=7$ sub-cells. However, the number of sub-cells added by an incomplete 3D subdivision may vary from 3 to 6. If a real number $c \in [3,6]$ is used to represent the average sub-cell increment per subdivision, the total number of sub-cells in a 3D tree can be expressed as

$$N_{\text{3d-tree}} = 1 + 7S_c + cS_i. \tag{4}$$

The $N_{\text{3d-tree}}/N_{\text{oct-tree}}$ ratio can then be obtained as follows:

$$\frac{N_{\text{3d-tree}}}{N_{\text{oct-tree}}} = \frac{1+7(S_{\text{total}}-S_i)+cS_i}{1+7S_{\text{total}}} \geq \frac{1+cS_{\text{total}}}{1+7S_{\text{total}}} > \frac{c}{7} \geq \frac{3}{7}. \tag{5}$$

□



(**a**) Minimum 2D tree splits. (**b**) Minimum 3D tree splits.

Figure 4. Minimum kd-tree splits within a single level: (**a**) 2D tree and (**b**) 3D tree splits, producing three and four sub-cells, respectively.

2. *Computational Cost in Tree Construction*: Given a quadrature cell to be subdivided, the vertices that need to be checked in the kd-tree are a subset of those in a quad/oct-tree. Furthermore, the kd-tree yields a lower number of sub-cells in each level (see Figure 4). Therefore, kd-tree results in a faster algorithm than quad/oct-tree.
3. *Aspect Ratio of Generated Sub-Cells*: Table 2 lists the worst-case and best-case aspect ratios of the sub-cells generated by quad-tree, oct-tree, and kd-tree subdivisions. Because the cell splits are symmetric and the splitting direction changes at different depths of each level, no aspect ratio worse than 1:2 (2D tree) or 1:1:2 (3D tree) can occur.

Table 2. Aspect ratios of the sub-cells generated by quad-tree, oct-tree, and kd-tree subdivision. The initial element shape is assumed to be square (2D) or cubic (3D).

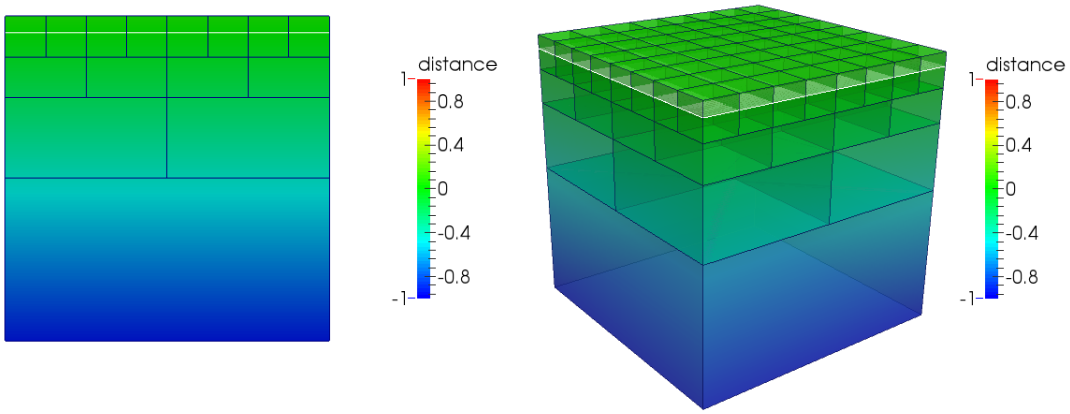
Tree Type	Worst Case	Best Case
Quad-tree	1:1	
Oct-tree	1:1:1	
2D tree	1:2	1:1
3D tree	1:1:2	1:1:1

2.3. Numerical Examples

Two numerical examples are presented below to demonstrate kd-tree sub-division.

2.3.1. Hyper-Planar Boundary

The first example, as shown in Figure 5, involves a hyper-plane immersed boundary (a line in 2D and a plane in 3D). A hierarchical sub-cell structure can be observed in this example. The number of generated sub-cells is listed in Table 3.



(**a**) Immersed line at $y = 0.95$. (**b**) Immersed plane at $z = 0.95$.

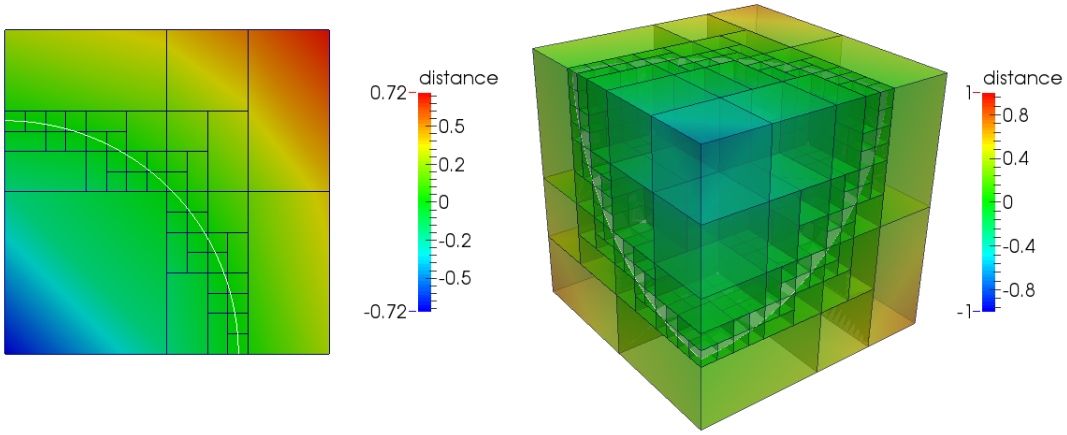
Figure 5. kd-tree subdivision of a unit cell in the presence of (**a**) a 2D immersed line and (**b**) a 3D immersed plane. The maximum level is three in both examples. The domain color represents a signed distance to the immersed boundary.

Table 3. Comparison of the kd-tree and quad/oct-tree subdivision in the presence of a hyper-planar boundary as shown in Figure 5.

Tree Type	$N_{subcell}$	Ratio of $N_{subcell}$
2D tree	15	0.682
Quad-tree	22	
3D tree	85	0.574
Oct-tree	148	

2.3.2. Hyper-Spherical Boundary

The second example is illustrated in Figure 6, where a hyper-spherical boundary is embedded in the domain. The corresponding numbers of created sub-cells are summarized in Table 4.



(**a**) Immersed circle ($R = 0.72$) (**b**) Immersed spherical surface ($R = 0.72$)

Figure 6. kd-tree subdivision of a unit cell in the presence of (**a**) a quadrant and (**b**) a one-eighth spherical surface. The hyper-spheres are centered at a corner and have a radius of R. The maximum level is four in both examples. The domain color represents a signed distance to the immersed boundary.

Table 4. Comparison of kd-tree and quad/oct-tree subdivision in the presence of a hyper-spherical boundary as shown in Figure 6.

Tree Type	$N_{subcell}$	Ratio of $N_{subcell}$
2D tree	48	0.787
Quad-tree	61	
3D tree	521	0.722
Oct-tree	722	

3. kd-Tree Data Structure for THB-Spline Refinement

The popular THB splines [20] are chosen here to demonstrate local refinement using kd-tree sub-division. We first briefly review THB splines before demonstrating kd-tree data structure for THB splines.

3.1. Brief Review of Truncated Hierachical B-Splines

Define a sequence of k-variate B-spline bases $B^l, l = 0, 1, \ldots, L-1$ satisfying the following nesting relation:

$$\operatorname{span} \mathcal{B}^0 \subset \operatorname{span} \mathcal{B}^1 \subset \cdots \subset \operatorname{span} \mathcal{B}^{L-1}. \tag{6}$$

Thus, any lower-level B-spline basis function $N_i^l \in \mathcal{B}^l, l = 0, 1, \ldots, L-2$ can be represented by a linear combination of higher-level B-spline basis functions $N_j^{l+1} \in \mathcal{B}^{l+1}$ as follows:

$$N_i^l = \sum_{\operatorname{supp} N_j^{l+1} \subset \operatorname{supp} N_i^l} \alpha_j N_j^{l+1}. \tag{7}$$

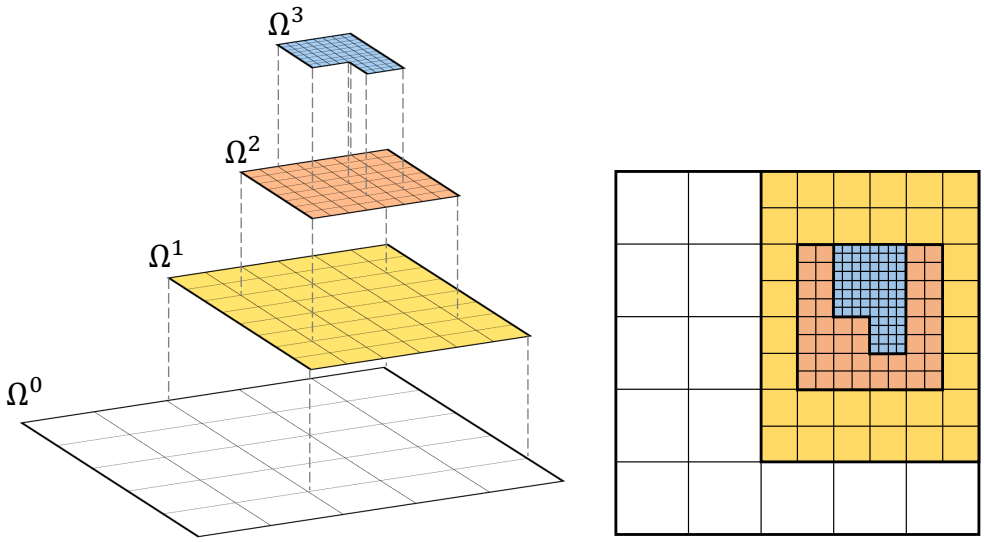
For the case of one-dimensional dyadic refinement (i.e., the knot vectors are uniform and each knot span is halved from V^l to V^{l+1}), Equation (7) takes the form:

$$N_{i,p} = 2^{-p} \sum_{j=0}^{p+1} \binom{p+1}{j} N_{2i-1+j,p}^{l+1} \tag{8}$$

where p is the degree of the B-spline. We further define a sequence of nested domains

$$\Omega^0 \supseteq \Omega^1 \supseteq \cdots \supseteq \Omega^{L-1} \tag{9}$$

where each $\Omega^l \in \mathbb{R}^k$ represents a k-dimensional refinement region at level l and its boundary $\partial\Omega^l$ is aligned with the knot grid of $\mathcal{B}^{l-1}$. A two-dimensional hierarchical mesh example is shown in Figure 7.



(**a**) Nested domains (**b**) Overlaid hierarchical mesh

Figure 7. A two-dimensional four-level dyadic hierarchical mesh: (**a**) the nested domains contain level-wise sub-meshes; (**b**) the sub-meshes are overlaid to generate the hierarchical mesh.

The hierarchical B-spline basis $\mathcal{H}$ is defined as

$$\mathcal{H} = \bigcup_{l=0}^{L-1} \left\{ N^l \middle| N^l \in \mathcal{B}^l \wedge \operatorname{supp} N^l \subseteq \Omega^l \wedge \operatorname{supp} N^l \nsubseteq \Omega^{l+1} \right\}. \tag{10}$$

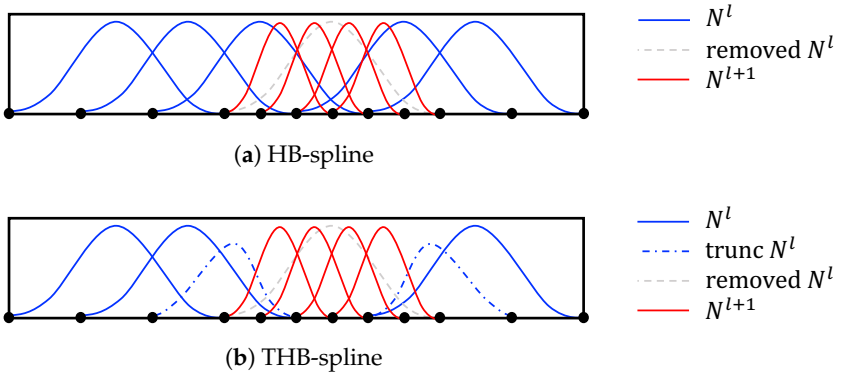
As illustrated in Figure 8a, the hierarchical B-spline basis functions do not satisfy the partition of unity property. Giannelli et al. [20] proposed a truncation operator to remove the higher-level B-spline components from a current-level basis function:

$$\operatorname{trunc}^{l+1} N_i^l = N_i^l - \sum_{\substack{\operatorname{supp} N_j^{l+1} \subset \operatorname{supp} N_i^l \\ \operatorname{supp} N_j^{l+1} \subseteq \Omega^{l+1}}} \alpha_j N_j^{l+1} \qquad \text{(Subtractive Representation)} \tag{11a}$$

$$= \sum_{\substack{\operatorname{supp} N_j^{l+1} \subset \operatorname{supp} N_i^l \\ \operatorname{supp} N_j^{l+1} \nsubseteq \Omega^{l+1}}} \alpha_j N_j^{l+1} \qquad \text{(Additive Representation)}. \tag{11b}$$

The general THB-spline basis $\mathcal{T}$ can be constructed as follows:

$$\mathcal{T} = \bigcup_{l=0}^{L-1} \left\{ \widehat{N}^l \middle| \widehat{N}^l = \operatorname{trunc}^{L-1} \cdots \operatorname{trunc}^{l+2} \operatorname{trunc}^{l+1} N^l, \right.$$
$$\left. N^l \in \mathcal{B}^l \wedge \operatorname{supp} N^l \subseteq \Omega^l \wedge \operatorname{supp} N^l \nsubseteq \Omega^{l+1} \right\}. \tag{12}$$



(**a**) HB-spline

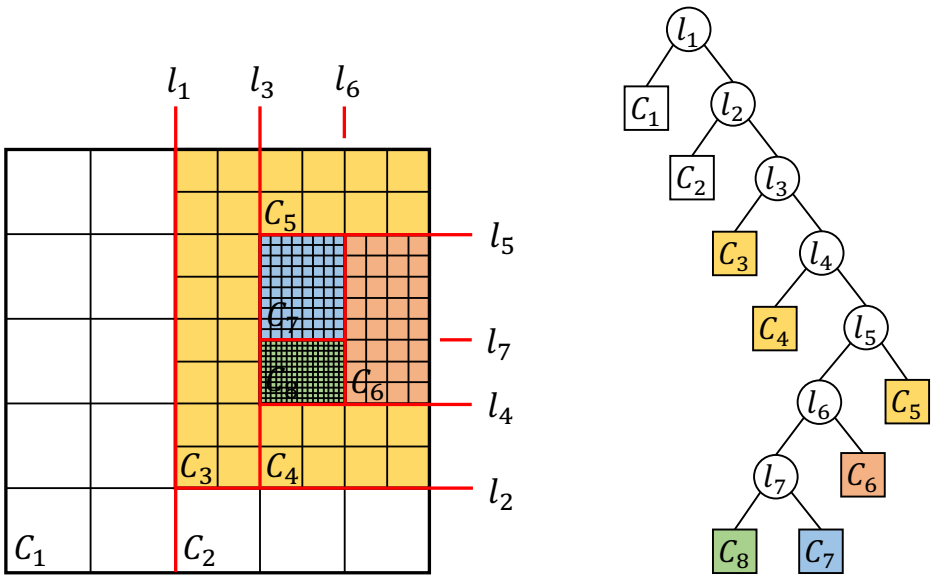
(**b**) THB-spline

Figure 8. Illustration of the basis functions of a two-level (**a**) hierarchical B-spline and (**b**) truncated hierarchical B-spline. The N^l in the dashed line can be represented by a linear combination of N^{l+1}, and is therefore removed to avoid linear dependence.

3.2. kd-Tree-Based Mesh Representation

The THB splines are defined over a sequence of nested domains which form a hierarchical mesh. An efficient representation of the hierarchical mesh along with a robust mesh generator plays an important role in the performance of the THB spline-based local refinement.

Kiss et al. [39] proposed a quad-tree data structure to represent two-dimensional THB spline meshes. As will be shown later, a quad-tree results in exponential space complexity in the worst case. Recently, an alternative binary tree data structure has been utilized [40] for multi-dimensional meshes. The internal nodes of the tree store the splitting lines during domain subdivision, whereas the leaves of the tree contain homogeneous pieces of the domain. The nature of this new data structure is in fact a kd-tree. Figure 9 illustrates an example of kd-tree representation for a two-dimensional hierarchical mesh.



(**a**) (**b**)

Figure 9. kd-tree representation of a two-dimensional hierarchical mesh. (**a**) Splitting lines and cells in the hierarchical mesh; (**b**) kd-tree representation. In (**a**), the domain is subdivided into homogeneous cells, with each belonging to only one hierarchical level, while in (**b**) the splitting lines and the cells are stored in the internal nodes and leaves of a kd-tree data structure, respectively.

A significant advantage of kd-tree representation over quad/oct-tree is that the former approach enables unequal subdivision of the domain, which results in fewer splits during tree construction. Consequently, the number of leaves generated by kd-tree subdivision is smaller than that generated by quad-tree or oct-tree subdivision. Figure 10 illustrates an instance of successive stripe refinement on a square domain. As the boundaries of the refinement regions are not aligned with bisection planes, a quad-tree representation will contain $3 \cdot 2^n - 3$ internal nodes and $9 \cdot 2^n - 8$ leaves, leading to a space complexity of $O(2^n)$. In contrast, its kd-tree counterpart only needs n internal nodes and $n + 1$ leaves, of which the space complexity is $O(n)$.

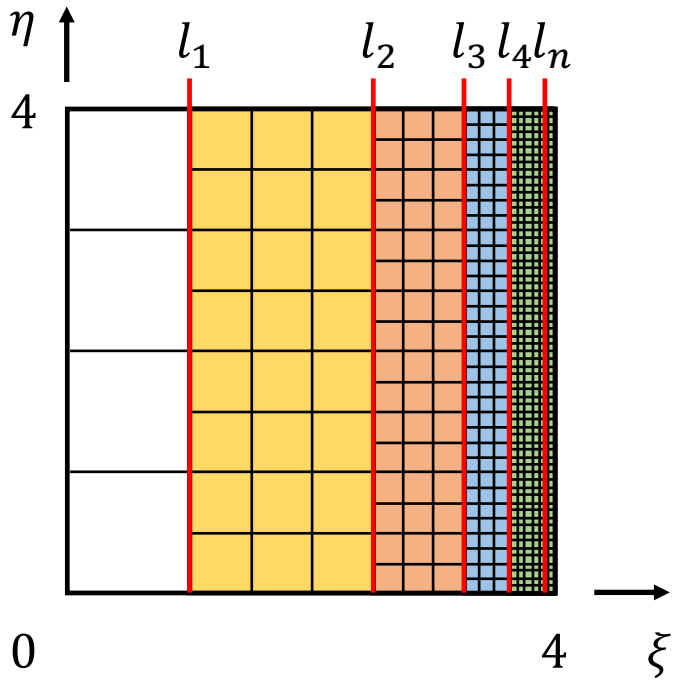


Figure 10. n-stripe refinement on a 4×4 domain. The left boundary of each refinement region Ω^i is provided by $\xi = 4 - 3 \cdot 2^{1-i}$ and the right boundaries coincide at $\xi = 4$. The splitting lines during kd-tree subdivision are labeled with $l_i, i = 1, 2, \cdots, n$.

4. Local Refinement Near Immersed Boundaries Using Algebraic Level Sets

We briefly review signed algebraic level sets before describing the local refinement algorithm using the algebraic level sets.

4.1. Brief Review of Algebraic Level Sets

Here, we propose using unsigned and signed distance from parametric boundaries for local refinement near immersed boundaries. In general, estimating the shortest distance to a parametric boundary requires numerical iteration using the Newton–Raphson method. Such iterations are generally non-robust, as Newton–Raphson iterations yield non-unique foot (nearest) points on the parametric surface near regions of large curvature. In order to circumvent these challenges, we have recently proposed techniques founded on algebraic geometry to estimate unsigned and signed distance measures from parametric boundaries, termed algebraic level sets [34,35], which have been further developed for point projection [41] and phase merging [42]. Algebraic level sets are briefly reviewed below prior to describing mesh refinement algorithms based on algebraic level sets.

The main idea is to convert a parametric NURBS entity into its implicit form using the resultant theory and to use the level set of the "implicitized" function as a measure of distance. Pre-processing by decomposing the NURBS entity into constituent Bezier patches and post-processing by blending using R-functions are utilized to generate the level sets from complex NURBS entities. The algebraic level sets have the following properties:

1. Exact locally near the surface
2. Monotonic function of exact distance
3. Sufficiently smooth for engineering applications
4. Efficiently obtained without numerical iterations

Thus, algebraic level sets can serve the role of a distance measure from the parametric boundary. Therefore, we refer to the algebraic level sets interchangeably as algebraic distance fields. We briefly review the computation of algebraic distance fields and illustrate the procedure through simple examples. Although, the examples illustrated here are of NURBS curves, the procedure is naturally applicable to parametric surfaces, as demonstrated in [34,35].

4.1.1. Implicitization of a Parametric Curve

Given a rational parametric curve $\boldsymbol{C}(X(u), Y(u), W(u))$ of degree p with $x = \frac{X(u)}{W(u)}$, $y = \frac{Y(u)}{W(u)}$, we can construct two auxiliary polynomials:

$$g_1(x, u) = W(u)x - X(u) = 0 \tag{13a}$$
$$g_2(y, u) = W(u)y - Y(u) = 0 \tag{13b}$$

The above polynomial equations can be rearranged in descending power of u, as follows:

$$g_1(u) = a_p u^p + a_{p-1} u^{p-1} + \cdots + a_1 u + a_0 \tag{14a}$$
$$g_2(u) = b_p u^p + b_{p-1} u^{p-1} + \cdots + b_1 u + b_0 \tag{14b}$$

From the above, the following resultant system can be obtained through algebraic manipulations [43]:

$$\begin{bmatrix} (a_p b_{p-1}) & \cdots & (a_p b_0) \\ \vdots & \ddots & \vdots \\ (a_p b_0) & \cdots & (a_1 b_0) \end{bmatrix} \begin{pmatrix} u^{p-1} \\ u^{p-2} \\ \vdots \\ 1 \end{pmatrix} = \left[\mathbf{M}^B\right]_{p \times p} \begin{pmatrix} u^{p-1} \\ u^{p-2} \\ \vdots \\ 1 \end{pmatrix} = 0 \tag{15}$$

where $(a_i b_j) = a_i b_j - a_j b_i$, $\mathbf{M}^B$ is a Bezout matrix and is a function of x and y with the following important property:

$$\mathbf{M}^B(x, y) = \mathbf{M}^B_x x + \mathbf{M}^B_y y + \mathbf{M}^B_w \tag{16}$$

where $\mathbf{M}^B_x, \mathbf{M}^B_y$ and $\mathbf{M}^B_w$ depend on control point coordinates and weights. Therefore, these matrices can be pre-computed for a given rational parametric curve and re-used given any new physical point $\boldsymbol{x}$. The determinant, $\det(\mathbf{M}^B(\boldsymbol{x}))$, is defined as the Bezout resultant. As all allowable parameter values u for curve $\boldsymbol{C}(X(u), Y(u), W(u))$ are roots of the system presented in Equation (15), $\det(\mathbf{M}^B(\boldsymbol{x})) = 0$ provides the equation for the implicit version of the curve. Thus, the algebraic level sets corresponding to a rational parametric curve (e.g., a Bezier curve) are provided by

$$\Gamma(\boldsymbol{x}) = \det(\mathbf{M}^B(\boldsymbol{x})). \tag{17}$$

An example of algebraic level sets is shown in Figure 11.

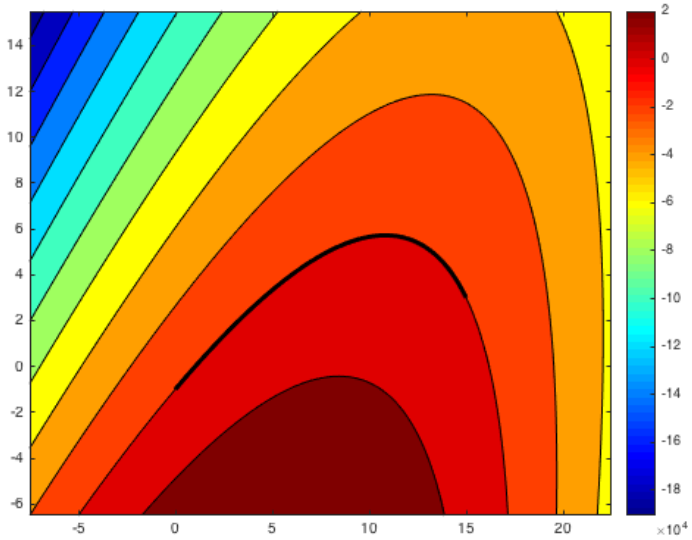


Figure 11. Implicitization of a quadratic Bezier segment. Level set $\Gamma(\boldsymbol{x}) = \det(\mathbf{M}(\boldsymbol{x}))$ can be used as a measure of distance.

4.1.2. Boolean Operations by R-Functions

As observed in Figure 11, the direct implicitization extends the parametric curve beyond its end points, yielding an invalid distance measure in the extended region. Therefore, it is desirable to trim the curve $\boldsymbol{C}(X(u), Y(u), W(u))$ within its parameter range $u \in [a, b]$. In related prior work, Biswas and Shapiro [44] constructed an approximate distance from a line segment as

$$g = \sqrt{\Gamma^2 + \frac{(|\phi| - \phi)^2}{4}} \tag{18}$$

with Γ being the normal distance from the line and ϕ a set of points that are positive in a region formed by a circle circumscribing the line and negative outside of it. This form yields a smooth distance function across the boundary $\phi = 0$. Upreti et al. [34] extended the above idea by carrying out boolean operations on fields obtained on individual segments of an arbitrarily shaped parametric curve and an enclosing convex region using R-functions [45,46]. The R-functions enable a smooth and purely algebraic boolean operation, resulting in a continuous distance measure. Two specific R-functions used in this study are:

1. R-conjunction, equivalent to point-wise set operation of Boolean intersection:

$$g_1 \wedge g_2 = g_1 + g_2 - \sqrt{g_1^2 + g_2^2} \tag{19}$$

2. R-disjunction, equivalent to point-wise set operation of Boolean union:

$$g_1 \vee g_2 = g_1 + g_2 + \sqrt{g_1^2 + g_2^2} \tag{20}$$

Upreti et al. [34] used the convex hull property of Bezier and NURBS curves to provide a natural convex region bounded by control points for curve trimming. They then normalized the Bezier segments before composing them using the R-conjunction operation (Equation (19)), thereby generating the desired unsigned algebraic level set. An example of such an unsigned algebraic level set is shown in Figure 12.

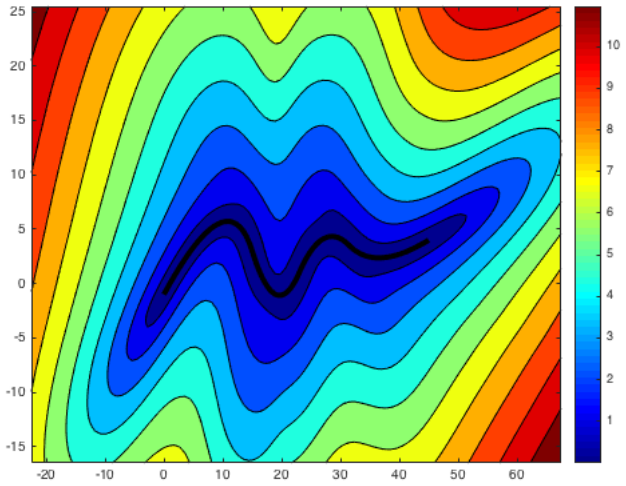


Figure 12. Algebraic level sets from an open quadratic NURBS curve. The generated algebraic level sets ensure the smoothness of the field.

4.1.3. Signed Algebraic Level Sets

The algebraic sign calculation of closed domains [35] is based on point containment queries. The algorithmic procedure is to first construct a concave/convex bounding box for the closed boundary by combining the convex hull of each Bezier patch. The candidate point is queried to see whether it lies within this bounding box. If the point is outside the bounding box, then the algorithm stops and the point is determined as being outside the bounded solid. However, if it is inside the box, point containment queries are performed in the next step in the regions between each Bezier segment and its convex hull. If the queried point lies inside one of these regions, then the algorithm stops and the point is again determined to lie outside the bounded solid; otherwise, the query point is classified as lying within the solid. A positive value of the algebraic sign function implies that the point is contained in the enclosed region, while zero implies a boundary point. An example of a signed algebraic level set from a closed NURBS curve is shown in Figure 13.

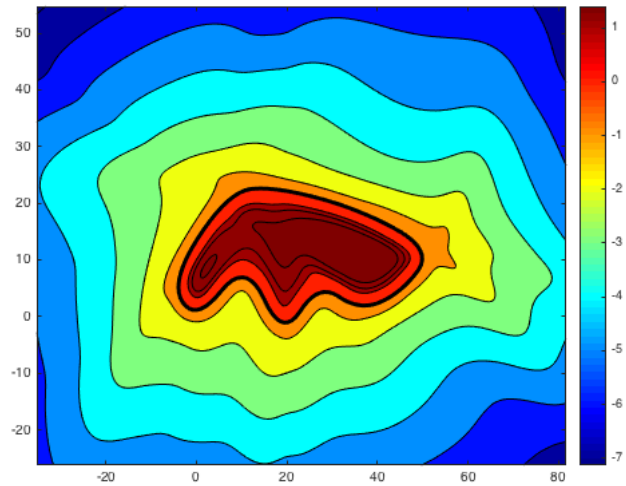


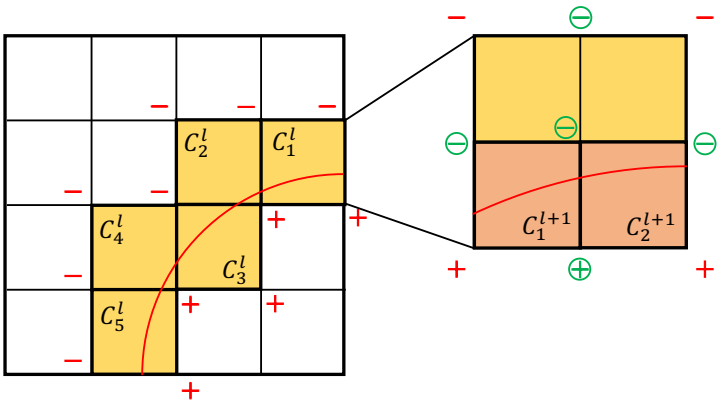
Figure 13. Algebraic level sets from a closed quadratic NURBS curve.

4.2. Mesh Refinement Algorithms Based on Algebraic Level Sets

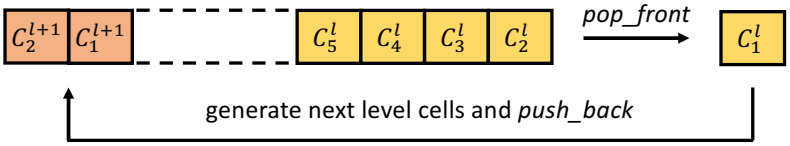
The classical mesh refinement schemes are based on a posteriori error estimators [47,48], which require a trial solution before adaptive refinement of the mesh is carried out. In the context of immersed boundary problems, it is expected that the behavioral fields will vary greatly near the boundaries. Therefore, in light of the position of immersed bound-

aries, a priori mesh refinement is possible. Two adaptive refinement algorithms based on signed [35] and unsigned [34] distance fields are proposed here, as follows:

1. *Sign-based Refinement Algorithm* (SRA): The cut cells and their neighbors are recursively subdivided until the maximum level is reached. The cut cells are identified by checking the signed distance of the cell vertices to the immersed boundaries. As illustrated in Figure 14, a cell is marked as a cut cell if its vertices have opposite signs. To minimize the number of sign checkings, the vertex signs of the cut cells (ones not circled in Figure 14a) are stored in memory. As the subdivision continues, the cut cells in the next level can be determined by an additional check of the signs of the edge, face, and cell centers (circled in Figure 14a). A first-in-first-out (FIFO) stack queue (Figure 14b) is utilized to keep track of the cells. The cut cells in the current level are first popped out from the front of the stack. After sign checking, the cut cells in the next level are generated and then pushed to the back of the stack. The checking of signed distance and new cell generation each stop when the maximum chosen level of refinement is reached.



(**a**) Sign-based cell subdivision with respect to a circular boundary



(**b**) Cell stack corresponding to (**a**)

Figure 14. Illustration of the sign-based refinement algorithm: (**a**) the cut cells in level l are subdivided into appropriate sub-cells. The new cut cells in level $l+1$ are identified by the newly calculated (circled) signs and the previously obtained (not circled) signs, and then (**b**) pushed to the back of the stack to wait for the next level subdivision.

2. *Distance-based Refinement Algorithm* (DRA): This algorithm is similar to the SRA except that the cut cells are determined by checking the magnitude of the distance of the cell vertices instead of the sign. This is particularly useful when the feature size d_{feature} of the immersed boundaries is smaller than the cell size d_{cell}, in which case the sign-based algorithm may fail to detect a cut cell (see Figure 15a). In distance-based refinement, a cell is marked as a cut cell if

$$\max_i |d_i| < \sqrt{k} d_{\text{cell}} \tag{21}$$

where k is the dimension of the domain and d_i is the distance of the ith vertex of the cell to the closest boundary. As shown in Figure 15b, cells that would be missed by SRA can be identified by DRA. In general, the DRA criterion is stricter than that of the SRA, and can therefore cause relatively more cells to be subdivided.

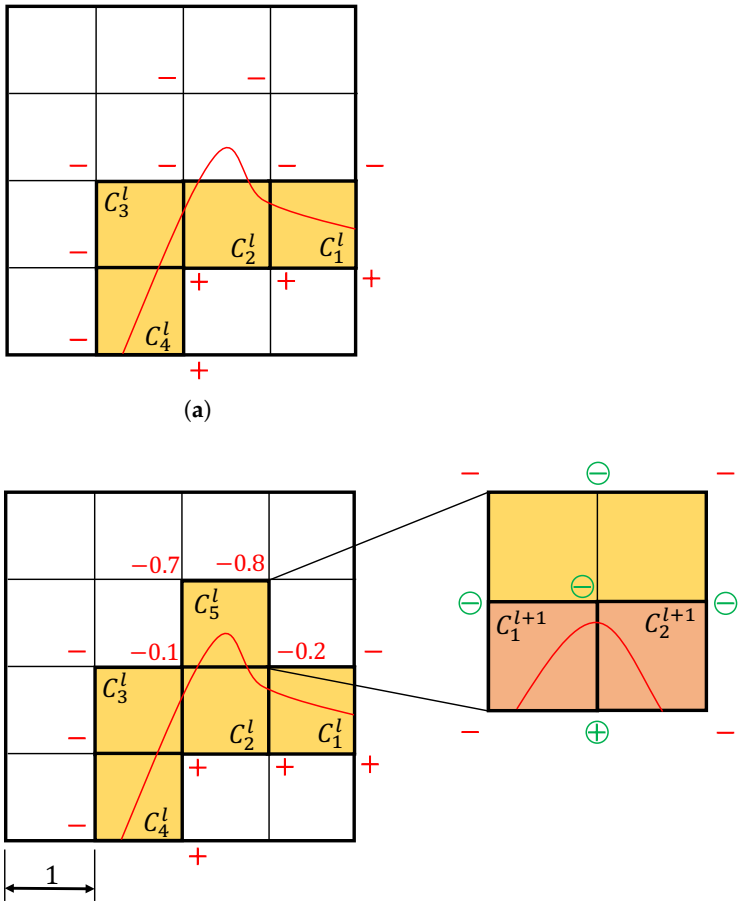
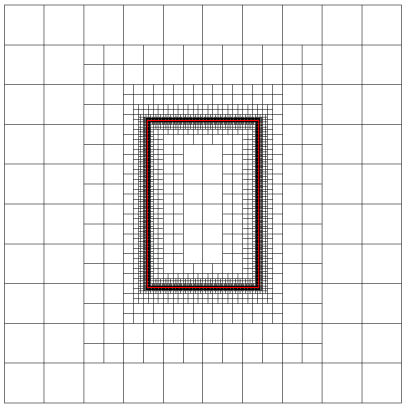


Figure 15. (**a**) Determination of level l cut cells using the sign-based criterion. A cut cell is missed due to the small feature size. (**b**) Determination of level l cut cells using the distance-based criterion, followed by sign-based subdivision to generate the level $l+1$ cut cells. For a cell size $d_{\text{cell}} = 1$, the distances of the C_5^l vertices to the boundary are annotated in the figure.

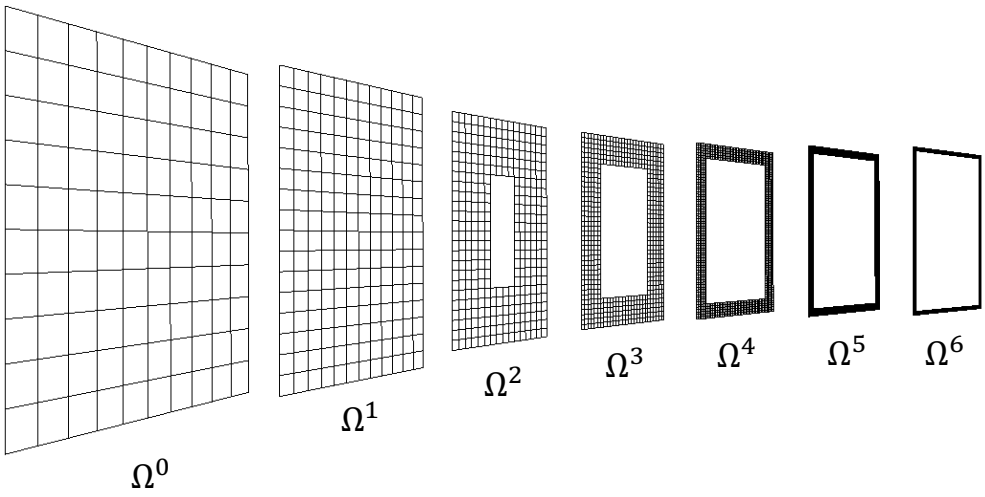
One possible strategy combining the above two algorithms is to dynamically switch between the two based on d_{cell}. The distance-based criterion is chosen if $d_{\text{cell}} > d_{\text{feature}}$; otherwise, the sign-based criterion is preferred. The inset magnified picture of Figure 15b shows a sign-based subdivision in the cut cell detected by the distance-based algorithm.

4.3. Numerical Examples

The efficiency and robustness of the proposed algorithms are demonstrated below through several numerical examples. Figure 16 shows a seven-level hierarchical refinement of a square domain with respect to a rectangular immersed boundary. The computational cost as a function of the number of levels is listed in Table 5. The hierarchical mesh generation only takes tens of microseconds on a desktop personal computer, even for a very large number of levels.



(**a**)



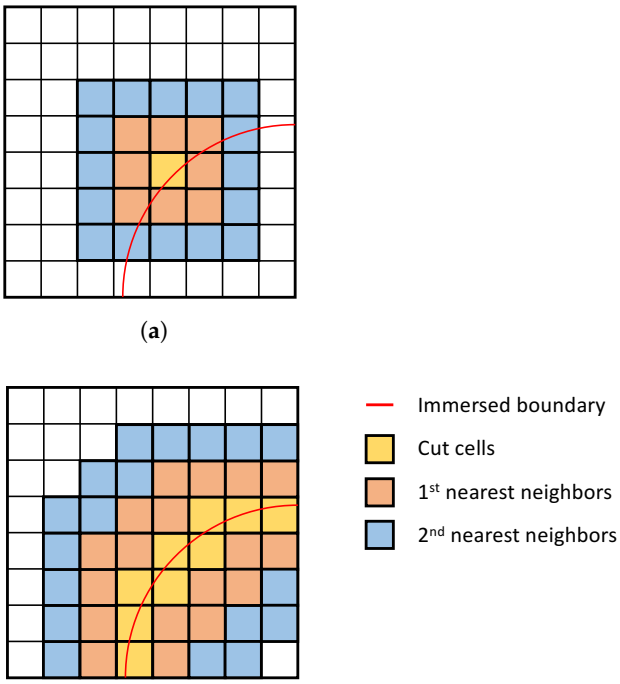
(**b**)

Figure 16. A two-dimensional seven-level hierarchical refinement of a square domain with respect to a rectangular immersed boundary: (**a**) hierarchical mesh and (**b**) sub-meshes in each level.

Table 5. Computer time on a personal computer required for hierarchical refinement of the geometry shown in Figure 16.

Number of Levels	1	2	3	4	5	6	7
Time Cost (s)	0.1	0.7	1.4	3.2	6.8	14.8	36.7

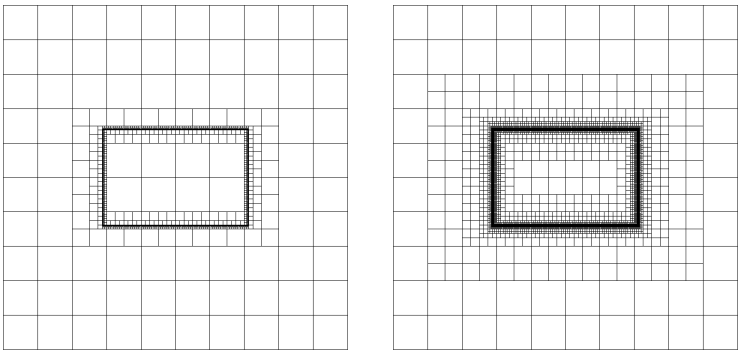
In order to include a larger local refinement region, it may be desirable to refine both the cut cells and their ith neighbors. As illustrated in Figure 17, the order i controls the bandwidth of a single level mesh. In contrast to level-wise hierarchical refinement, refinement within a level is referred to as *horizontal refinement*. A two-dimensional example of horizontal refinement is shown in Figure 18.



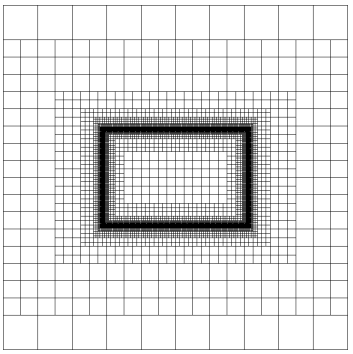
(**a**)

(**b**)

Figure 17. Schematic of horizontal refinement: (**a**) the neighbors of a cut cell provide a larger refinement region, while (**b**) the union of the neighbors forms a refinement band.



(**a**) $i = 0$ (**b**) $i = 1$



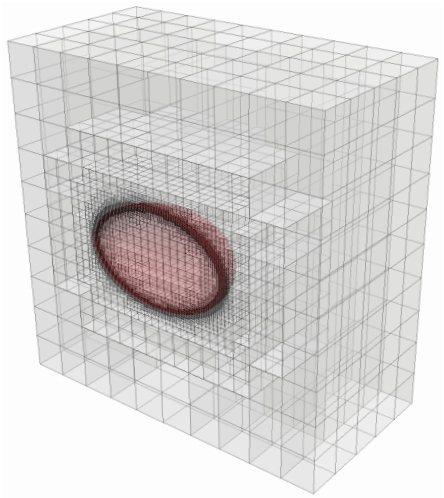
(**c**) $i = 2$

Figure 18. Horizontal refinement with different bandwidth. (**a**) Only the cut cells are refined. (**b**) First-order refinement: the cut cells and their first nearest neighbors are refined. (**c**) Second-order refinement: the cut cells and up to their second nearest neighbors are refined.

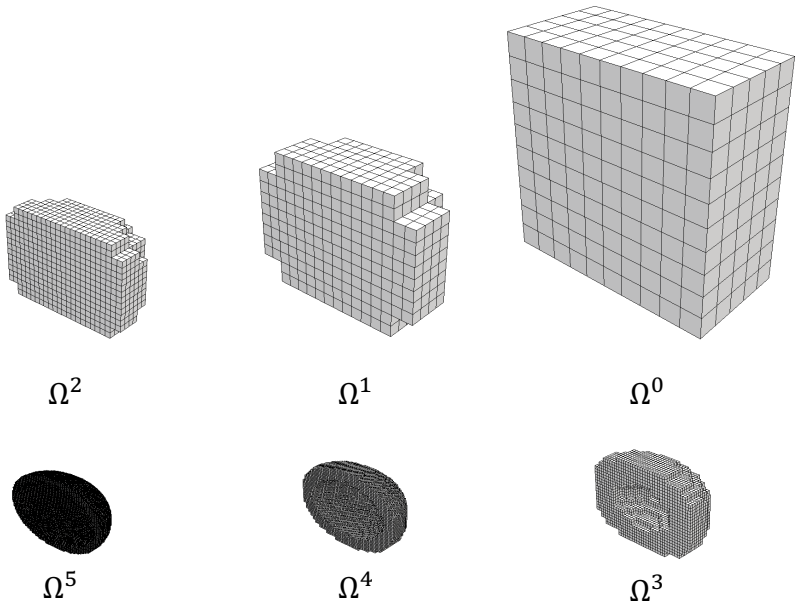
The mesh refinement algorithms and kd-tree data structure can be directly extended to three-dimensional problems thanks to their dimension-independent nature. Figure 19 shows a six-level hierarchical refinement of a cubic domain in the presence of an ellipsoidal boundary. The computational cost of refinement is summarized in Table 6. Compared to the time cost during matrix assembly and system solution generation, the sub-second mesh generation time is almost negligible.

Table 6. Computer time on a personal computer required for hierarchical refinement of the geometry shown in Figure 19.

Number of Levels	1	2	3	4	5	6
Time Cost (s)	0.1	2.4	8.6	47.9	166.3	434.8



(**a**) Hierarchical mesh around the immersed boundary



(**b**) Exploded view of the nested sub-meshes

Figure 19. A three-dimensional six-level hierarchical refinement of a cubic domain with respect to an ellipsoidal boundary: (**a**) hierarchical mesh and (**b**) sub-meshes in each level.

5. Adaptive Quadrature Near Immersed Boundaries

As mentioned earlier, accurate quadrature is challenging near inserted boundaries. The quadrature cells that are cut by the parametric boundary need to be refined adaptively in order to improve the accuracy of the quadrature. In this section, we utilize the kd-tree data structure as well as the previously described distance-based refinement strategy to demonstrate adaptive quadrature and to solve for the behavioral unknowns in three-dimensional geometrical examples. To compare the developed refinement strategy against oct-tree subdivision of the quadrature cells, a uniform background "mesh" of NURBS basis functions is used, into which the parametric NURBS boundaries are immersed. Unlike the THB refinement shown in Figure 19, where the refinement doubles as quadrature cells, the uniform background mesh allows us to compare different adaptive quadrature strategies while holding the number of "nodal" unknowns constant.

5.1. Single Inclusion with Spherical Boundary

The single-inclusion geometry shown in Figure 20 is used to verify the accuracy and convergence of the kd-tree data structure by comparing the estimated stress concentration factor against its theoretical value. The spherical inclusion is immersed in a finite domain with a uniform fraction of unit magnitude, as shown in Figure 20. The theoretical limit of the stress concentration factor (SCF) near the equator for a single spherical void in an infinite underlying domain is $\frac{\sigma_{max}}{\sigma} = 2$ [49]. A two-dimensional view of the quadrature points obtained using a kd-tree cell division is shown in Figure 21, revealing hierarchical refinement of quadrature cells near the immersed boundaries. In Figure 22, the contours of the Von Mises stress around the equator of the spherical void are shown. The convergence of the stress concentration factor is plotted in Figure 23 and listed in Table 7. Convergence to the theoretical limit is obtained with a decrease in the size ratio $\frac{L}{d}$, provided that sufficient degrees of freedom (control grids of $20 \times 20 \times 20$ or $30 \times 30 \times 30$) are used.

Table 7. Maximum von Mises stress (near equator) with different size of inclusion ($30 \times 30 \times 30$ control points) in Figure 23.

Diameter of Inclusion *d*	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Length Ratio *L*/*d*	1.25	1.43	1.67	2	2.5	3.3	5
Stress Concentration Factor	3.93	3.10	2.72	2.46	2.29	2.29	2.28

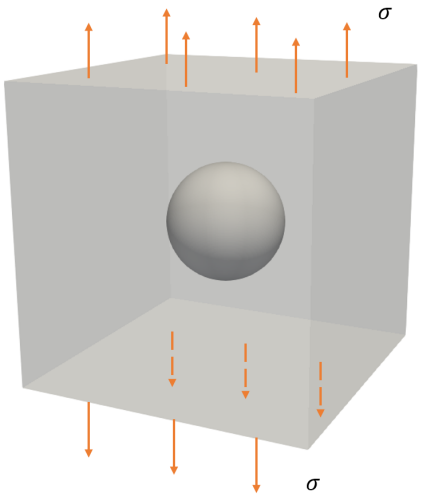


Figure 20. Single spherical void under uniform fraction of unit magnitude.

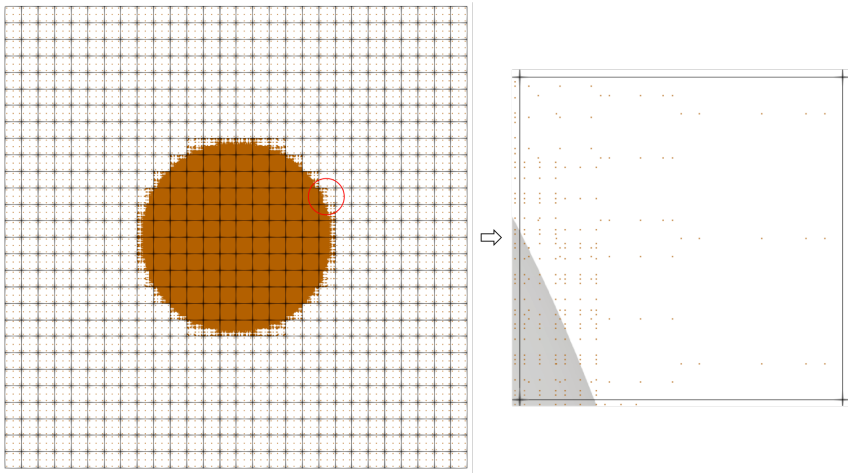


Figure 21. Two-dimensional view (xy-plane) of quadrature points in the domain obtained using kd-tree cell division. A control point grid of $30 \times 30 \times 30$ was used.

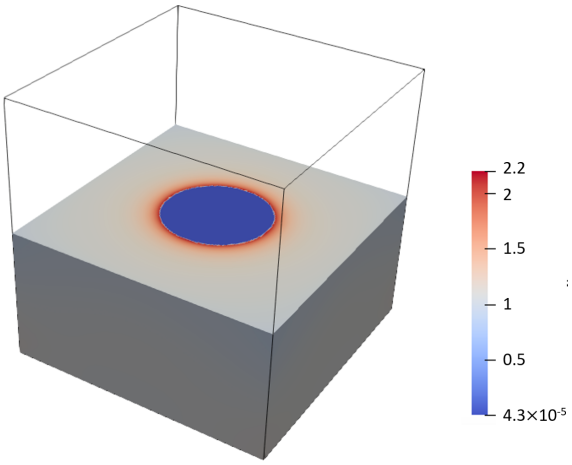


Figure 22. Von Mises stress and stress concentration behavior around the equator of the spherical inclusion. The sphere has a diameter of 0.2 and is placed at the center of a $1 \times 1 \times 1$ cubic domain.

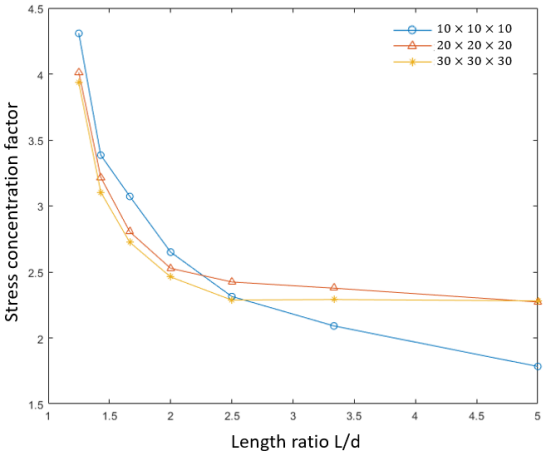
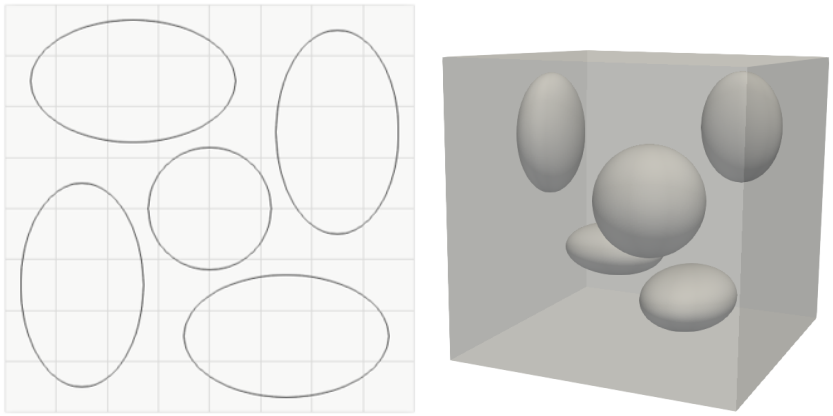


Figure 23. Convergence of the stress concentration factor with refinement as a function of the scale ratio $\frac{L}{d}$ (where L is the size of the underlying cubic domain and d is the diameter of spherical void). The plots correspond to $10 \times 10 \times 10$, $20 \times 20 \times 20$, and $30 \times 30 \times 30$ control grids.

5.2. Multiple Inclusions with Hyper-Spherical Boundary

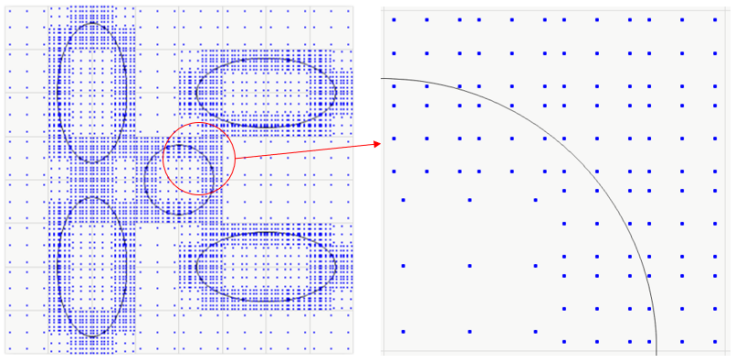
The second set of examples are similar in spirit and contain a complex geometry with multiple elliptical/ellipsoidal voids in two dimensions (Figure 24a) and in three dimensions

(Figure 24b). The elliptical/ellipsoidal voids are immersed in the underlying domain, and both kd-tree and quad/oct-tree quadrature cell refinement are implemented (see Figures 25 and 26) to estimate the volume of the solid region. The estimated volume listed in Tables 8 and 9 indicates that kd-tree cell division achieves the same degree of accuracy as quad/oct-tree sub-division, with both methods resulting in an error of 1.70×10^{-3} in the two-dimensional example and 4.77×10^{-5} in the three-dimensional example. For two-level refinement, in the two-dimensional example kd-tree uses 3276 quadrature points, while quad-tree uses 4032 quadrature points. In the three-dimensional example, kd-tree uses 118,476 quadrature points, nearly one-third fewer than oct-tree subdivision, which results in 161,244 quadrature points.

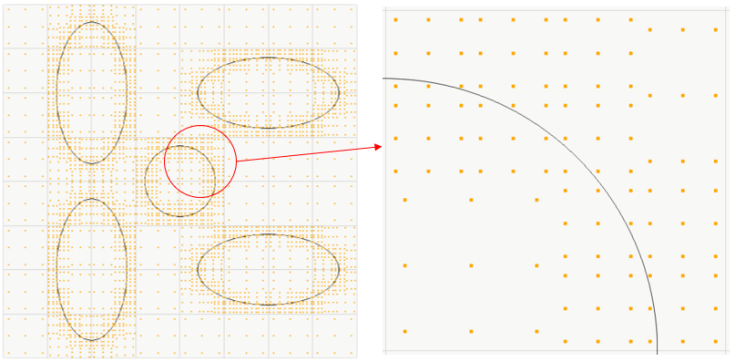


(**a**) two-dimensional (**b**) three-dimensional

Figure 24. Unit cell with five (**a**) elliptical and (**b**) ellipsoidal inclusions.



(**a**) quad-tree



(**b**) kd-tree

Figure 25. Quadrature points generated for the example in Figure 24a using (**a**) quad-tree and (**b**) kd-tree data structure with two sub-levels.

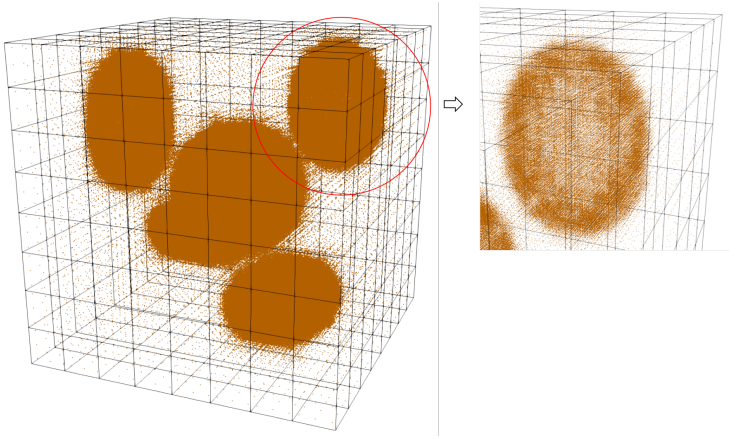


Figure 26. Quadrature points generated for the example in Figure 24b using a kd-tree data structure with two sub-levels.

Table 8. Comparison of kd-tree and quad-tree cell division for quadrature cell refinement in the presence of multiple inclusions with elliptical boundary (shown in Figure 24a).

Tree Type	N_{qp}	Ratio of N_{qp}	V_{act}	V_{approx}	Error
2D tree	3276	0.771	0.7173	0.7160	1.7000×10^{-3}
Quad-tree	4032			0.7160	1.7000×10^{-3}

Table 9. Comparison of kd-tree and oct-tree cell division for quadrature cell refinement in the presence of multiple inclusions with hyper-spherical boundary (shown in Figure 24b).

Tree Type	N_{qp}	Ratio of N_{qp}	V_{act}	V_{approx}	Error
3D tree	118,476	0.735	0.9162	0.9618	4.7714×10^{-5}
Oct-tree	161,244			0.9618	4.7714×10^{-5}

5.3. Particulate System with Spherical Fillers

In the third example, a more complex geometry is analyzed to show the efficiency of kd-tree over oct-tree cell division. The number of quadrature points used in the example is on the order of 1×10^7. A particulate system was generated by random packing of 84 spherical fillers in matrix to achieve a volume loading of 43%, as shown in Figure 27. The net volume of solid spherical fillers was then evaluated separately using adaptive quadrature based on kd-tree as well as oct-tree cell division. The estimated volume listed in Table 10 indicates that kd-tree cell division achieves the same degree of accuracy as oct-tree subdivision after two levels of refinement. kd-tree results in a savings of 5.53×10^6 quadrature points relative to oct-tree subdivision, which amounts to 26% savings in the number of integration points for the same accuracy.

Table 10. Comparison of kd-tree and oct-tree cell division for quadrature cell refinement in a particulate system with 84 spherical fillers.

Tree Type	N_{qp}	Ratio of N_{qp}	V_{act}	V_{approx}	Error
3D tree	15,753,744	0.740	0.4256	0.4256	3.3451×10^{-5}
Oct-tree	21,287,043			0.4256	1.9108×10^{-5}

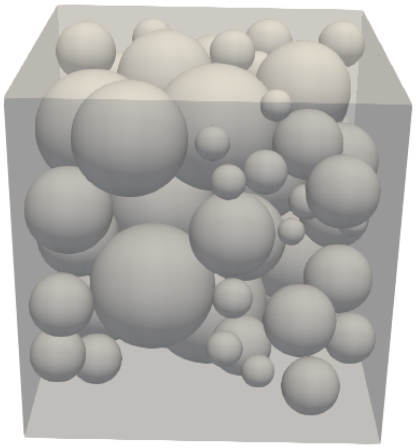


Figure 27. Particulate system with 84 spherical fillers.

6. Summary

Three important computational stategies for efficient analysis of immersed boundary problems have been demonstrated in this paper. First, kd-tree sub-division was proposed and proven to be more efficient through theoretical analysis as compared against quad-tree and oct-tree subdivision. The number of sub-cells generated by kd-tree subdivision may be as few as two-thirds that generated by a quad-tree or three-sevenths that generated by an oct-tree. Second, we described an efficient local refinement strategy near NURBS boundaries immersed in the domain based on sign and distance relative to the immersed boundary. The signs and distance relative to the boundary were efficiently obtained using algebraic level sets. Last, we evaluated the use of kd-tree cell division for adaptive quadrature of NURBS-discretized domains near immersed boundaries. The use of kd-tree-based adaptive quadrature to estimate the volume of a problem with multiple hyper-spherical voids resulted in the same degree of accuracy as oct-tree subdivision while requiring only a third as many quadrature points. Thus, significant reduction in degrees of freedom and quadrature points can be achieved with efficient local refinement using kd-tree cell division.

Author Contributions: Conceptualization and methodology, T.S. and G.S.; validation, T.S. and H.L.; writing, T.S., H.L. and G.S.; visualization, T.S. and H.L.; supervision, G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hughes, T.; Cottrell, J.; Bazilevs, Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Eng.* **2005**, *194*, 4135–4195. [CrossRef]
2. Natekar, D.; Zhang, X.; Subbarayan, G. Constructive solid analysis: A hierarchical, geometry-based meshless analysis procedure for integrated design and analysis. *Comput.-Aided Des.* **2004**, *36*, 473–486. [CrossRef]
3. Hsu, M.C.; Kamensky, D.; Bazilevs, Y.; Sacks, M.S.; Hughes, T.J. Fluid–structure interaction analysis of bioprosthetic heart valves: significance of arterial wall deformation. *Comput. Mech.* **2014**, *54*, 1055–1071. [CrossRef] [PubMed]
4. Xu, F.; Schillinger, D.; Kamensky, D.; Varduhn, V.; Wang, C.; Hsu, M.C. The tetrahedral finite cell method for fluids: Immersogeometric analysis of turbulent flow around complex geometries. *Comput. Fluids* **2016**, *141*, 135–154. [CrossRef]
5. Ji, H.; Chopp, D.; Dolbow, J. A hybrid extended finite element/level set method for modeling phase transformations. *Int. J. Numer. Methods Eng.* **2002**, *54*, 1209–1233. [CrossRef]

6. Chessa, J.; Smolinski, P.; Belytschko, T. The extended finite element method (XFEM) for solidification problems. *Int. J. Numer. Methods Eng.* **2002**, *53*, 1959–1977. [CrossRef]
7. Gómez, H.; Calo, V.M.; Bazilevs, Y.; Hughes, T.J. Isogeometric analysis of the Cahn–Hilliard phase-field model. *Comput. Methods Appl. Mech. Eng.* **2008**, *197*, 4333–4352. [CrossRef]
8. Song, T.; Upreti, K.; Subbarayan, G. A sharp interface isogeometric solution to the Stefan problem. *Comput. Methods Appl. Mech. Eng.* **2015**, *284*, 556–582. [CrossRef]
9. Dolbow, J.; Moës, N.; Belytschko, T. Discontinuous enrichment in finite elements with a partition of unity method. *Finite Elem. Anal. Des.* **2000**, *36*, 235–260. [CrossRef]
10. Tambat, A.; Subbarayan, G. Isogeometric enriched field approximations. *Comput. Methods Appl. Mech. Eng.* **2012**, *245*, 1–21. [CrossRef]
11. Tambat, A.; Subbarayan, G. Simulations of arbitrary crack path deflection at a material interface in layered structures. *Eng. Fract. Mech.* **2015**, *141*, 124–139. [CrossRef]
12. Catmull, E.; Clark, J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Comput.-Aided Des.* **1978**, *10*, 350–355. [CrossRef]
13. Kobbelt, L.; Hesse, T.; Prautzsch, H.; Schweizerhof, K. Iterative mesh generation for FE-computations on free form surfaces. *Eng. Comput.* **1997**, *14*, 806–820. [CrossRef]
14. Cirak, F.; Ortiz, M.; Schröder, P. Subdivision surfaces: A new paradigm for thin-shell finite-element analysis. *Int. J. Numer. Methods Eng.* **2000**, *47*, 2039–2072. [CrossRef]
15. Piegl, L.; Tiller, W. *The NURBS Book*, 2nd ed.; Springer: New York, NY, USA, 1997.
16. Da Veiga, L.B.; Cho, D.; Sangalli, G. Anisotropic NURBS approximation in isogeometric analysis. *Comput. Methods Appl. Mech. Eng.* **2012**, *209*, 1–11. [CrossRef]
17. Forsey, D.; Bartels, R. Hierarchical B-spline refinement. *ACM Siggraph Comput. Graph.* **1988**, *22*, 205–212. [CrossRef]
18. Sederberg, T.; Zheng, J.; Bakenov, A.; Nasri, A. T-splines and T-NURCCs. *ACM Trans. Graph. (TOG)* **2003**, *22*, 477–484. [CrossRef]
19. Sederberg, T.; Cardon, D.; Finnigan, G.; North, N.; Zheng, J.; Lyche, T. T-spline simplification and local refinement. *Acm Trans. Graph.* **2004**, *23*, 276–283. [CrossRef]
20. Giannelli, C.; JüTtler, B.; Speleers, H. THB-splines: The truncated basis for hierarchical splines. *Comput. Aided Geom. Des.* **2012**, *29*, 485–498. [CrossRef]
21. Bazilevs, Y.; Calo, V.; Cottrell, J.; Evans, J.; Hughes, T.; Lipton, S.; Scott, M.; Sederberg, T. Isogeometric analysis using T-splines. *Comput. Methods Appl. Mech. Eng.* **2010**, *199*, 229–263. [CrossRef]
22. Vuong, A.V.; Giannelli, C.; Jüttler, B.; Simeon, B. A hierarchical approach to adaptive local refinement in isogeometric analysis. *Comput. Methods Appl. Mech. Eng.* **2011**, *200*, 3554–3567. [CrossRef]
23. Hughes, T.J.R.; Reali, A.; Sangalli, G. Efficient quadrature for NURBS-based isogeometric analysis. *Comput. Methods Appl. Mech. Eng.* **2010**, *199*, 301–313. [CrossRef]
24. Renken, F.; Subbarayan, G. NURBS-based solutions to inverse boundary problems in droplet shape prediction. *Comput. Methods Appl. Mech. Eng.* **2000**, *190*, 1391–1406. [CrossRef]
25. Zhang, X.; Rayasam, M.; Subbarayan, G. A meshless, compositional approach to shape optimal design. *Comput. Methods Appl. Mech. Eng.* **2007**, *196*, 2130–2146. [CrossRef]
26. Rayasam, M.; Srinivasan, V.; Subbarayan, G. CAD inspired hierarchical partition of unity constructions for NURBS-based, meshless design, analysis and optimization. *Int. J. Numer. Methods Eng.* **2007**, *72*, 1452–1489. [CrossRef]
27. Düster, A.; Parvizian, J.; Yang, Z.; Rank, E. The finite cell method for three-dimensional problems of solid mechanics. *Comput. Methods Appl. Mech. Eng.* **2008**, *197*, 3768–3782. [CrossRef]
28. Kim, H.J.; Seo, Y.D.; Youn, S.K. Isogeometric analysis for trimmed CAD surfaces. *Comput. Methods Appl. Mech. Eng.* **2009**, *198*, 2982–2995. [CrossRef]
29. Kim, H.J.; Seo, Y.D.; Youn, S.K. Isogeometric analysis with trimming technique for problems of arbitrary complex topology. *Comput. Methods Appl. Mech. Eng.* **2010**, *199*, 2796–2812.
30. Schillinger, D.; Dede, L.; Scott, M.; Evans, J.; Borden, M.; Rank, E.; Hughes, T. An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces. *Comput. Methods Appl. Mech. Eng.* **2012**, *249*, 116–150. [CrossRef]
31. Rüberg, T.; Cirak, F. A fixed-grid b-spline finite element technique for fluid–structure interaction. *Int. J. Numer. Methods Fluids* **2014**, *74*, 623–660. [CrossRef]
32. Cheng, K.; Fries, T.P. Higher-order XFEM for curved strong and weak discontinuities. *Int. J. Numer. Methods Eng.* **2010**, *82*, 564–590. [CrossRef]
33. Kudela, L.; Zander, N.; Bog, T.; Kollmannsberger, S.; Rank, E. Efficient and accurate numerical quadrature for immersed boundary methods. *Adv. Model. Simul. Eng. Sci.* **2015**, *2*, 1. [CrossRef]
34. Upreti, K.; Song, T.; Tambat, A.; Subbarayan, G. Algebraic distance estimations for enriched isogeometric analysis. *Comput. Methods Appl. Mech. Eng.* **2014**, *280*, 28–56. [CrossRef]
35. Upreti, K.; Subbarayan, G. Signed algebraic level sets on NURBS surfaces and implicit Boolean compositions for isogeometric CAD–CAE integration. *Comput.-Aided Des.* **2017**, *82*, 112–126. [CrossRef]

36. De Berg, M.; Van Kreveld, M.; Overmars, M.; Schwarzkopf, O. *Computational Geometry*; Springer: Berlin/Heidelberg, Germany, 2000.
37. Karras, T. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics, Paris, France, 25–27 June 2012; pp. 33–37.
38. Hu, L.; Nooshabadi, S.; Ahmadi, M. Massively parallel KD-tree construction and nearest neighbor search algorithms. In Proceedings of the 2015 IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, Portugal, 24–27 May 2015; pp. 2752–2755.
39. Kiss, G.; Giannelli, C.; Jüttler, B. Algorithms and Data Structures for Truncated Hierarchical B–splines. In *Mathematical Methods for Curves and Surfaces: Proceedings of the 8th International Conference, MMCS 2012, Oslo, Norway, 28 June–3 July 2012*; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2014; pp. 304–323.
40. Giannelli, C.; Jüttler, B.; Kleiss, S.; Mantzaflaris, A.; Simeon, B.; Špeh, J. THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis. *Comput. Methods Appl. Mech. Eng.* **2016**, *299*, 337–365. [CrossRef]
41. Liao, H.; Vaitheeswaran, P.K.; Song, T.; Subbarayan, G. Algebraic point projection for immersed boundary analysis on low degree NURBS curves and surfaces. *Algorithms* **2020**, *13*, 82. [CrossRef]
42. Vaitheeswaran, P.; Subbarayan, G. Improved Dixon Resultant for Generating Signed Algebraic Level Sets and Algebraic Boolean Operations on Closed Parametric Surfaces. *Comput.-Aided Des.* **2021**, *135*, 103004. [CrossRef]
43. Sederberg, T. Implicit and Parametric Curves and Surfaces for Computer Aided Geometric Design. Ph.D. Thesis, Purdue University, West Lafayette, IN, USA , 1983.
44. Biswas, A.; Shapiro, V. Approximate distance fields with non-vanishing gradients. *Graph. Model.* **2004**, *66*, 133–159. [CrossRef]
45. Rvachev, V. On the analytical description of some geometric objects. *Rep. Ukr. Acad. Sci.* **1963**, *153*, 765–767.
46. Shapiro, V. *Theory of R-Functions and Applications: A Primer*; Technical Report TR91-1219; Department of Computer Science, Cornell University: Ithaca, NY, USA, 1991.
47. Babuška, I.; Rheinboldt, W. Error estimates for adaptive finite element computations. *SIAM J. Numer. Anal.* **1978**, *15*, 736–754. [CrossRef]
48. Ainsworth, M.; Oden, J. A posteriori error estimation in finite element analysis. *Comput. Methods Appl. Mech. Eng.* **1997**, *142*, 1–88. [CrossRef]
49. Jaeger, J.C.; Cook, N.G.; Zimmerman, R. *Fundamentals of Rock Mechanics*; John Wiley & Sons: Hoboken, NJ, USA, 2009.