ACCURATE APPROXIMATION OF UNSTRUCTURED GRID INTO REGULAR GRID WITH COMPLEX BOUNDARY HANDLING

by

Dana El-Rushaidat

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Xavier M. Tricoche, Chair
Department of Computer Science

Dr. Christoph M. Hoffmann
Department of Computer Science

Dr. Bharat Bhargava
Department of Computer Science

Dr. David F. Gleich
Department of Computer Science

Approved by:
Dr. Kihong Park

My humble effort is dedicated to the memory of my father and my father-in-law, two strong men who believed in me and taught me to believe in myself.

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my advisor who guided me through this long journey, my thesis committee for their insightful comments, my friend and sister Raine for never giving up on me, my mother and my sisters for their continuous support, my husband and my two boys for their patience, help, and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Unstructured grids, commonly used in computational fluid dynamics (CFD) simulations, allow for flexible point distribution and arbitrary boundary geometry. This property makes it straightforward to model complex domains and apply adaptive mesh refinement techniques. However, from a post-processing perspective, unstructured grids are expensive to process, have high storage costs, and do not support smooth data approximation.

Regular grids, in contrast, are grids with implicit connectivity and geometry. They have a reduced memory footprint while supporting efficient spatial queries and smooth approximation. However, regular grids lack the flexibility to represent complex geometry due to the rigidity imposed by their structured nature.

This thesis investigates the problem of creating accurate approximations of unstructured grids on regular grids. For that, a regular grid must be selected that can accommodate the varying spatial resolution of the original dataset while being as small as possible. The choice of the regular grid directly affects the computational and storage cost of the approximation. The approximation procedure must lend itself to parallel and distributed implementation to prove effective in the context of the large-scale datasets produced by high-performance computing. For unstructured grids with solid boundaries, the approximation grid needs to account for the solid boundary to yield an accurate approximation close to the body.

A customized octree with hybrid refinement criteria is used to obtain the optimal rectilinear grid. The approximation of the unstructured data, solved in a least-squares fashion, typically results in an ill-posed system due to the highly non-uniform distribution of the points in the original unstructured grid. We propose a variable regularization to resolve the ill-posedness in the approximation system while maintaining low approximation error.

A solid boundary handling is achieved using an approximation of the boundary geometry inside each regular grid cell cut by the boundary. The research demonstrates this using linear kernels and high-order B-spline kernels. The proposed approximation can adapt to different kernels based on the need of the visualization application. We test the method using visualization algorithms that rely heavily on interpolation. We demonstrate the effectiveness of this method using several 2D and 3D unstructured simulation datasets with solid

boundaries. The evaluation shows that we can achieve qualitatively and quantitatively high accuracy in representing the original unstructured data, at least six times faster query time, and dramatically reduced file size and memory footprint. The boundary-aware rectilinear grid approximation can produce accurate visualization near the solid boundary. Results show that the approximation method proposed in this thesis provides a scalable, accurate approximation for an unstructured grid into a regular grid. Various visualization applications can benefit from the proposed approximation since the approximation can be easily integrated into almost any visualization algorithm. The proposed approximation provides high accuracy while having a smaller file size, allowing for the visualization of complex and large datasets using devices with limited storage and processing power. Naive rectilinear grids without special boundary handling capabilities suffer from a poor ability to present complex geometries. In contrast, the proposed boundary-aware approximation grid can approximate the solid boundary inside the regular cells, thus providing support for complex geometry handling that other regular grids can't provide. The interesting part of the phenomenon presented in the simulation happens close to the solid boundary. Therefore, the ability to visualize data near the boundary is essential for achieving a high-quality visualization analysis.

1. INTRODUCTION

Numerical simulations often produce datasets defined over complex unstructured meshes, also referred to as unstructured grids. Two and three-dimensional meshes are essential means to represent the numerical simulation results, which we are in this thesis interested in visualizing. The computational domain is discretized into individual elements (*e.g.*, triangles, prisms, etc.).

Unstructured meshes have no inherent structure in the mesh representation, allowing mesh points to be added, deleted, and/or displaced. In contrast, mesh connectivity is re-configured to enhance accuracy. When a solid body is present in the simulation data, the mesh is typically refined close to the body, and a high portion of the mesh resolution is invested around the body to correctly resolve high gradients associated with the so-called *boundary layer*. Though unstructured grids are flexible in data representation and amend-able to complex geometry, the high storage cost and the complex post-processing analysis of the unstructured grids made their users' tasks more complicated.

This thesis seeks an alternative representation for the original unstructured data to address the existing significant limitations of the post-processing unstructured grids faced by end-users, whether engineers or visualization analysts. Processing would be much simpler for the end-user if the dataset could be reliably approximated to a much smaller form, much simpler geometrically, much faster to process for all sorts of visualization tasks, and supports much more smooth derivative computation.

The choice for this approximation of unstructured datasets is regular grids. Regular grids also referred to as rectilinear grids, are axis-aligned grids. In contrast to unstructured grids, regular grids are highly space-efficient, faster in locating a cell in the grid and provide high-order derivatives. Still, they are limited in their flexibility in representing complex geometry.

We aim to minimize the reconstruction error at the input data point, and we need a smooth approximation field elsewhere. The approximation problem in hand is solved using the least-square method. Unfortunately, this optimization system is often large due to the size of the unstructured grid and ill-posed due to the lack of constraints that comes from the inherent nature of the unstructured grid with a highly non-uniform distribution of the original data point. The proposed approximated solution is a global one.

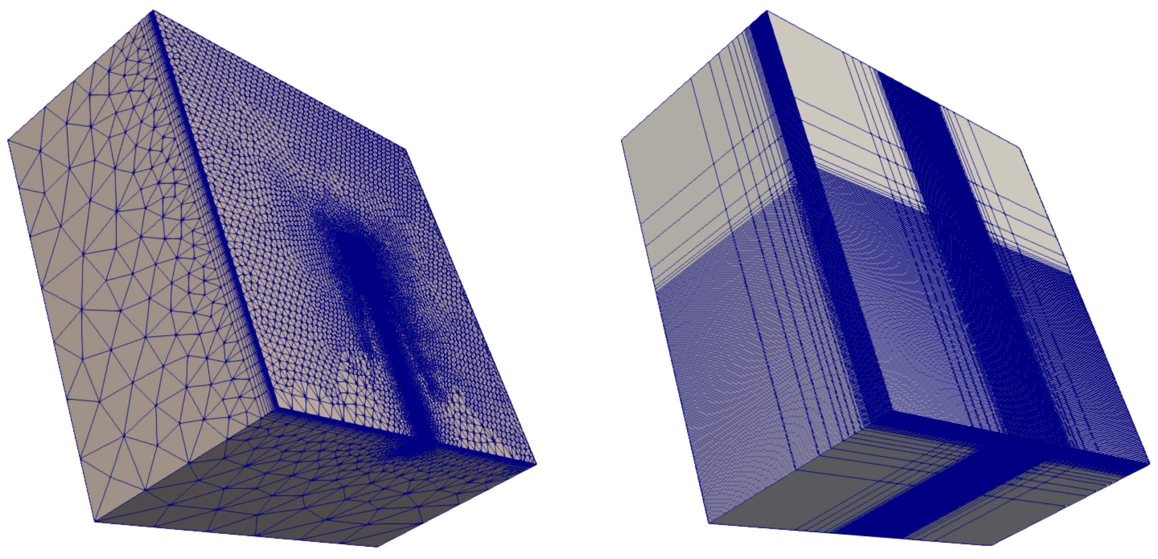
The choice of the kernel to be used for the fitting problem is crucial. A low-order kernel will generate a less smooth result and have limited derivatives, resulting in a more straightforward system to solve and interpolate. A high-order interpolation kernel will provide a smoother approximation, less rank deficiency since the system will become less sparse but on the cost of a denser system to solve and a slightly higher interpolation cost. The approximation can adapt to kernels with different orders. The choice of the kernel can be seen as an application-specific choice.

The approximation system uses block-solving to overcome the large dataset size; therefore, the size of the approximation system to solve. We solve the system in blocks while maintaining the smoothness and the accuracy of the approximation.

The proposed approximation is endowed with boundary handling capabilities, That allow for accommodating challenging boundaries.

The thesis focuses on the unstructured grids resulting from computational fluid dynamics (CFD) simulations. The datasets used for testing the accuracy of the proposed solution come from CFD simulations. The visualization tools used throughout the thesis are related to flow visualization, the group of visualization algorithms that study the flow behavior in datasets resulting from CFD simulations.

The investigated solution in this thesis discusses the following: given an unstructured grid (with arbitrary point distribution and often including a solid boundary), construct an approximation of the data representation by imposing a regular grid over the initial unstructured dataset. The approximated solution aims to minimize the approximation error of the given initial data points, and provide an alternative to the original data that is more memory efficient, easier to interpolate, can provide high-quality derivatives, and is flexible in geometry representation. Figure 1.1 shows a side-by-side comparison between a 3D original unstructured grid and its rectilinear approximation grid.



(a) Unstructured grid dataset. (b) Approximation into rectilinear grid.

Figure 1.1. Example of a 3D unstructured grid and its approximation.

1.1 Significance

The accurate approximation of a large unstructured dataset with solid boundary into a regular grid that is computationally feasible and has boundary awareness will benefit many visualization algorithms that rely heavily on accurate interpolation, particularly close to the solid boundary. Examples of such applications are isosurface extraction [1], [2], volume rendering [3], shading [4]–[6], ridge and valley manifolds extraction [7]–[9], and feature detection [10], [11].

The path to creating an accurate rectilinear approximation faces many challenges. We are providing an approximation for complex datasets with highly non-uniform point distribution. The solid boundaries in the dataset make the problem more complex and challenging. Some of the challenges that face the approximation include the choice of the approximation grid, the ill-posedness arising from the dataset's inherent complexity, how to achieve scalability given the large size of the data, and solid boundaries' accurate processing. We give more clarification of those challenges and introduce the proposed solutions in the next section.

1.2 Research Questions and Proposed Solutions

As listed above, the unstructured grid suffers from high storage cost, slow interpolation, and inefficient high-order derivative calculation. This thesis discusses the approximation of a large unstructured dataset with solid boundary into a regular grid with special boundary handling. The proposed solution addresses the major bottlenecks faced while using unstructured grids. The approximated regular grid provides fast interpolation, storage cost-saving, and accurate high-order derivatives. The solution investigates the poor representation of the regular grid to represent complex geometry compared to an unstructured grid. We provide additional tools to the regular grid that makes boundary-awareness a property for the approximation grid.

To obtain the resulting approximation, we faced and answered many research questions, and here we list the major ones together with the general proposed solutions:

1. **How to generate the best rectilinear grid that provides high approximation accuracy and low storage cost?**

   This thesis focuses on the approximation of scientific simulation data in the form of an unstructured grid. Such simulation result is commonly known for the highly non-uniform point distribution. The unstructured grid flexibility allows for the accurate presentation of such results. Thus, The straightforward solution of using a uniform grid for the approximation would be a wrong choice. Therefore, we seek in this thesis for a grid that can replicate the original data, be sufficient for accurate approximation of the original data points, and be as small as possible. We came up with a customized quadtree/octree that will guide the design of the adaptive grid. The carefully designed adaptive approximation grid allows accurate reproduction of the phenomena associated with the simulation, together with enjoying the best memory usage.

2. **How to resolve the approximation problems related to the rank-deficient system where the commonly known regularization is not sufficient to constrain?**

The approximation problem aims to find the best fit of the approximation values assigned to the rectilinear grid that reduces the error norm between the scattered point value and the interpolated value obtained from the approximated rectilinear grid in a least-square sense, and the fitting problem results in the highly sparse large rectangular system. The non-uniform distribution of scattered points results in a lack of constraints for the fitting problem.

We resolve the problem by introducing smoothing/regularization to the fitting system, adding additional constraints, and solving the approximation system's rank deficiency. Unfortunately, the standard regularization was not adequate to resolve the issues encountered in the approximation system. We propose an improvement to the uniform regularization technique commonly used to resolve ill-posedness in approximation systems. The regularization approach generates a smooth approximation that does not suffer from either over-smoothing or under-smoothing.

3. **How to provide accurate interpolation close to the solid boundary?** The main drawback of using regular grids is its lack of flexibility in representing complex geometries. Though the lack of boundary flexibility problem exists when using a low-order approximation kernel, the visual artifacts close to the solid boundary become more apparent when using a high-order approximation kernel. In this thesis, we aim to improve the accuracy of interpolation close to the solid boundary. Two methods are used to aid us in achieving this goal. First, we use the cut-cell method, which allows for identification and special treatments to the cells cut by the solid boundary to retain the correct flow close to the solid boundary. Second, the marching cube algorithm is an algorithm that is mainly used to find the isosurfaces in scalar polygonal meshes by defining a code for each cell based on it being inside, outside, or partially inside the isosurface. We use the leading techniques from the marching cube algorithm to approximate the solid boundary geometry inside the rectilinear grid cells. Together we equip the resulting approximation grid with additional information that allows for the on-the-spot generation of the triangulation of the boundary inside the rectilinear grid cut-cell. The additional information is stored in the same rectilinear grid as

an extra data array. A boundary-aware rectilinear grid can use the same standard rectilinear grid with only special knowledge of all data arrays provided. Together all those methods end up with accurate handling of the cells cut by the solid boundary.

1.3 Thesis Structure

The contents of this thesis are organized as follows. We review the mathematical background and the related work in Chapter 2. In Chapter 3, we introduce the design choice of the rectilinear grid that we will use for the approximation. Then we demonstrate the bi-level smoothing and its effect on the approximation quality. Later in the thesis in Chapter 4, the boundary-aware rectilinear grid is introduced, and the details of the boundary-awareness added to the rectilinear grid are discussed. Each chapter ends with a discussion of the results obtained from each Chapter. We end the thesis with the conclusion and future work in Chapter 5.

2. MATHEMATICAL BACKGROUND

This thesis uses tools from the approximation theory, computational geometry, and scientific visualization. This chapter covers the mathematical concepts, basic definitions, and terminologies necessary to understand the rest of our work presented in later chapters of this thesis. Our thesis is based on approximating a grid type into another; we start with an overview of the grid types and their properties in Section 2.1. In Section 2.2 and Section 2.3, we review linear and B-spline kernels that we will use as our approximation kernels.

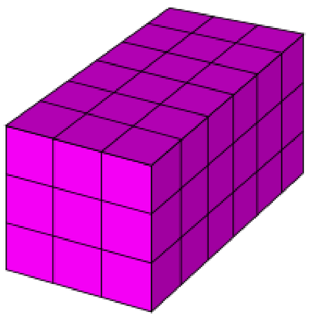
Our work is closely related to a family of approximation tools, namely scattered data approximation. We introduce this type of approximation and how it is tailored for our purposes in Section 2.4. To solve the approximation problem in this thesis, we will use the least-square scheme together with regularization. We introduce them in Section 2.5 and Section 2.6, respectively.

For testing our approximation accuracy numerically against the original data, we use a list of error measures that we explain in Section 2.7. Section 2.8 introduces a heavily used visualization application in visualization literature that we will use throughout the thesis to measure the performance and quality of our approximation results.

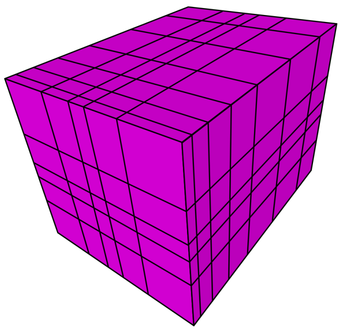
2.1 Grid Types

The data provided by simulations can be in different data representations, also referred to as mesh or grid interchangeably. The data representation can be structured or unstructured. The organization of the grid vertices and the connectivity between the vertices can categorize the mesh type. The mesh type controls its respective properties, such as the storage cost and interpolation method. We can roughly classify the grids into the following: uniform (Image), rectilinear, curvilinear, and unstructured. Figure 2.1 shows the different types of meshes.

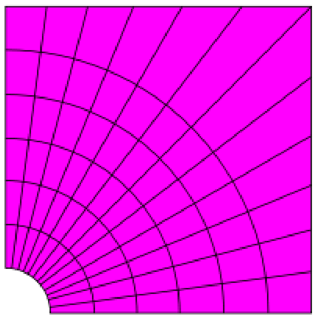
1. **Uniform grid**: An example of a uniform grid can be seen in Figure 2.1a. The uniform grid enjoys a structured point arrangement with uniform spacing along each axis, making an implicit relationship between the grid vertices and implicit cell definition.



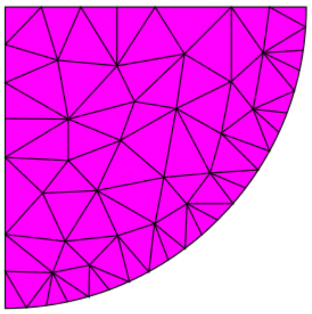
(a) Uniform grid.



(b) Rectilinear grid.



(c) Curvilinear grid.



(d) Unstructured grid.

Figure 2.1. Example grid types; structured:(a),(b),(c) [12], and unstructured:(d) [13]

The storage requirements for a uniform grid are the origin of the mesh and the spacing (a single value per axis). No explicit storage of the grid vertex coordinates or the connectivity (cell information) is needed. Given the point index, origin, and grid spacing, the point coordinates can be calculated on the fly. For any query point, it takes $\mathcal{O}(1)$ constant time to find the cell enclosing this query point, regardless of where the query point lies in the grid. The uniform grid is the most commonly used data representation for images and datasets with uniform point spacing.

2. **Rectilinear grid**: A rectilinear grid in Figure 2.1b is a type of grid shared with a uniform grid all properties of being structured but different in that this grid has variable spacing per axis. This allows for more flexibility in data representation, but with some extra cost in storage and query processing time.

For storage, rectilinear grids need to store the same information as uniform grids except that the grid spacing is not a single uniform value, so axis grid vertices need to be stored.

3. **Curvilinear grid**: the curvilinear grid shown in Figure 2.1c is similar to unstructured geometry in that the points are not axis aligned like the last two grid types, but it still maintains implicit connectivity between vertices. In a curvilinear grid, the coordinate lines may be curved. A one-to-one mapping between the curved coordinates and a corresponding rectilinear grid can be performed. After the mapping is performed, the interpolation is done in the same manner as any rectilinear grid.

4. **Unstructured grid**: Figure 2.1d shows an unstructured grid with arbitrary points and various cell types. Though this offers high flexibility and adaptivity in data representation, this comes at the cost of having extra storage overhead and expensive query time compared to the previous structured grid types. Compared to structured grids, the unstructured grid has high storage overhead due to the explicit storage of the points coordinates and the explicit storage of the connectivity information. Those are implicit in the structured grids.

   Interpolation in an unstructured grid requires finding the cell enclosing the query point; this is an expensive operation due to the arbitrary distribution of points in the unstructured grid. Since many visualization algorithms entail heavy interpolation, this is considered a visualization issue, and it is called point location problem.

   To overcome this problem, spatial data structures are needed to narrow down the number of test cells candidates to enclose the query point. The query time in the unstructured grid then is dependent on the performance of those data structures, and this measure varies based on the grid complexity and the size of the cell. Octrees have been used to resolve the point location problem [14]. In addition, other methods used ray tracing where spatial data structures are used to find the nearest cell and the adjacency list is used to walk to the cell enclosing the point [15]. However, such methods could fasten the point search in the unstructured grid; they are still memory intensive due to duplicate cells saving in different branches in those trees that can reach ten times the number of original cells. One of the state-of-the-art methods is cell-trees [16], a data structure built to find the cell location enclosing a point using bounding interval hierarchy [17], [18]. The use of the hierarchy reduces the storage overhead and reduces the number of cells to be tested for inclusion. This cell-tree reduces the build time of the tree and the memory overhead compared to the other data structures. Throughout the thesis, we will compare the performance of our approximation result with the unstructured grid using a cell-tree.

2.2 Trilinear Interpolation

In this section, we introduce linear interpolation in the context of rectilinear grid.

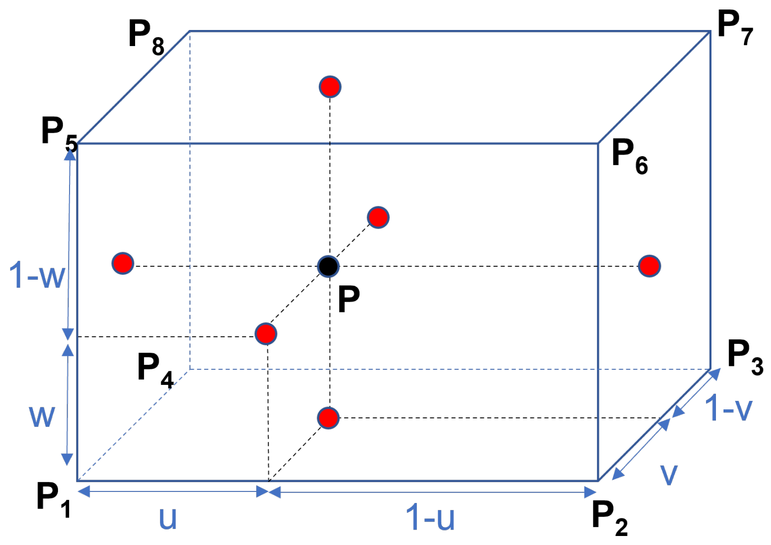


Figure 2.2. Interpolation of the query point P in the cuboid cell of the rectilinear grid.

Given a query point in a rectilinear grid, we can find the cell enclosing the query point in $\mathcal{O}(\log(\max(gr_x, gr_y, gr_z)))$, where gr_x, gr_y, and gr_z respectively designate the grid resolution for x, y, and z axis. The search for the cell is a binary search, given that the grid values per axis are sorted. The search per axis can be done in parallel, so the search time for the cell enclosing a query point equals the binary search in the axis with the highest grid resolution

gr. Within the rectilinear grid cuboid cell enclosing the query point P, trilinear interpolation is used to interpolate within the cell using the equation:

$$\begin{aligned}P = (1-u)(1-v)(1-w)P_1 + u(1-v)(1-w)P_2 + uv(1-w)P_3 + \\ (1-u)v(1-w)P_4 + (1-u)(1-v)wP_5 + u(1-v)wP_6 + uvwP_7 + (1-u)vwP_8\end{aligned} \tag{2.1}$$

where P_i is the values at the ith rectilinear grid cell vertex. u, v, and w are the relative query point position at the x, y, and z axes, respectively. Figure 2.2 shows a rectilinear grid cell and the trilinear interpolation coefficients.

2.3 B-Spline

Throughout the chapters, we use B-spline kernel with different degrees as an approximating kernel to solve the problem of approximating unstructured data into a regular grid. Figure 2.3 shows the basis for linear to cubic B-spline kernels. Below we explain the basis of B-spline: Given $m+1$ control points $P_0, P_1, \ldots, P_m$ and a knot vector $U = \{u_0, u_1, \ldots, u_{m+p+1}\}$, the B-spline curve of degree p defined by these control points and knot vector U is:

$$C(u) = \sum_{\mathrm{i}=0}^{m} N_{\mathrm{i},p}(u) P_\mathrm{i} \tag{2.2}$$

where, clamping is used for the knot vector $U = \{u_0 = \ldots = u_p, u_{p+1}, \ldots, u_{m+1} = \ldots = u_{m+p+1}\}$, and $N_{\mathrm{i},p}$ is the p^{th} degree polynomial function given by the recursive equation:

$$N_{\mathrm{i},p}(u) = \frac{u - u_\mathrm{i}}{u_{\mathrm{i}+p} - u_\mathrm{i}} N_{\mathrm{i},p-1}(u) + \frac{u_{\mathrm{i}+p+1} - u}{u_{\mathrm{i}+p+1} - u_{\mathrm{i}+1}} N_{\mathrm{i}+1,p+1}(u) \tag{2.3}$$

with the base case

$$N_{\mathrm{i},0}(u) = \begin{cases} 1, & u_\mathrm{i} \leqslant u \leqslant u_{\mathrm{i}+1} \\ 0, & otherwise \end{cases} \tag{2.4}$$

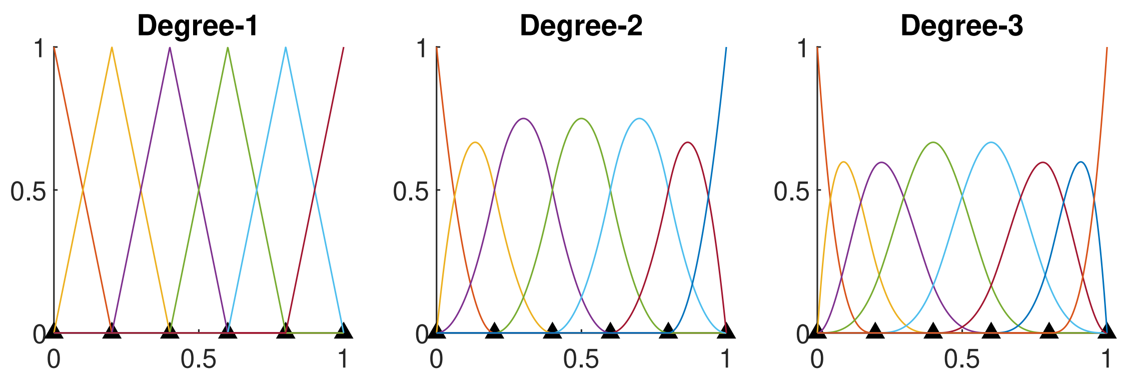


Figure 2.3. 1D B-spline basis of different degrees using the same uniform knots shown with ▲.

The 1D B-spline basis can be extended to higher dimensions by tensor product. The general tensor product formula for The multivariate basis function $N_{\mathrm{i},p} : \mathbb{R}^D \to \mathbb{R}$ is the tensor product of the 1D basis functions and is given by:

$$N_{\mathrm{i},p}(\boldsymbol{u}) = \prod_{d=1}^{D} N^{(d)}_{\mathrm{i}_d,p}(u_d) \qquad \boldsymbol{u} = [u_1, \ldots, u_D] \tag{2.5}$$

Here the index i goes from 1 to $\prod_{d=1}^{D} n_d$ and covers all combinations of $\mathrm{i}_d = 1, \ldots, n_d$ for $d = 1, \ldots, D$.

In a B-spline function, the computation of derivatives is straightforward [19], and is given in Equation (2.6).

$$C'(u) = \sum_{\mathrm{i}=0}^{m-1} N_{\mathrm{i}+1,p-1}(u) \frac{P_{\mathrm{i}+1} - P_{\mathrm{i}}}{u_{\mathrm{i}+p+1} - u_{\mathrm{i}+1}} \tag{2.6}$$

2.4 Scattered Data Approximation Using Rectilinear Grid

The work in this thesis is closely related to scattered data approximation. We consider the problem of approximating the per-point input data values of an unstructured input dataset through reconstruction kernels over a rectilinear grid. In this section, we introduce the basics of the approximation used in the coming chapters.

Let $\mathcal{S}$ be the set of input data points, which consist of n data points, $\{(\boldsymbol{x}_{\mathrm{i}}, \boldsymbol{f}_{\mathrm{i}})\}_{\mathrm{i}\in I}$, where $\{\boldsymbol{x}_{\mathrm{i}}\}_{\mathrm{i}\in I}$ are coordinates in $\mathbb{R}^d$, $\{\boldsymbol{f}_{\mathrm{i}}\}_{\mathrm{i}\in I}$ are the associated input data values in the data space $\mathbb{R}^q$, and I is the index set of the input data points. In the following, we consider, without loss of generality, the case of three-dimensional scalar data ($q = 1$, $d = 3$) and vector data ($q = 3$, $d = 3$). Other attribute types (*e.g.*, tensor data) can be handled similarly.

Let $\{\boldsymbol{x}_{\mathrm{i}}\}_{\mathrm{i}\in I}$ admit an axis-aligned bounding box $(\boldsymbol{a}, \boldsymbol{b})$. Let $\mathcal{G}$ be an axis-aligned nonuniform rectilinear grid with resolution $\{gr_d\}_{1\leq d\leq 3} \in \mathbb{N}$ spanning the spatial region delimited by $\boldsymbol{a}$ and $\boldsymbol{b}$. The spacing of the rectilinear grid along each axis is defined by the grid coordinates $\{rr^d_{\mathrm{i}}\}_{\mathrm{i}=1,\ldots,gr_d}$ for each dimension d. $\mathcal{G}$ contains $gr = \prod_{d=1}^{3} gr_d$ grid vertices. The vertices are defined as $\{\boldsymbol{p}_k\}_{k\in K}$, where $\boldsymbol{p}_k$ are the coordinate of each grid vertex, defined as the Cartesian product of $\{rr^d_{\mathrm{i}}\}$ Here K designates the 3-dimensional index set corresponding to $\mathcal{G}$. Each cell of the rectilinear grid is an axis-aligned hexahedron (cuboid) with boundaries defined by the grid coordinates. The interpolation equation given any kernel and a point coordinate $\boldsymbol{x}$ is given by:

$$\psi(\boldsymbol{x}) = \sum_{k\in K} \boldsymbol{g}_k \varphi_k(\boldsymbol{x}) \tag{2.7}$$

where φ_k is the approximation kernel for the k^{th} grid vertex with coordinate $\boldsymbol{p}_k$. The approximation problem then becomes to find the function values $\boldsymbol{g}_k$ corresponding to the rectilinear grid vertices $\boldsymbol{p}_k$.

To use B-splines as an approximation kernel with the rectilinear grid, we consider the B-spline kernel from Equation (2.2). We define the unique knots of each b-spline basis to be the rectilinear grid coordinates per axis, that is rr^d_{i}, namely, $\{u_p, u_{p+1}, \ldots, u_{m+1}\}$ the unique knot vector without clamping corresponding to the rectilinear grid $\{rr^d_0, rr^d_1, \ldots, rr^d_{rg}\}$ per axis.

The choice of the B-spline degree p determines the system's complexity, the smoothness of the solution, and its differentiability. If $p = 1$ the kernel is equivalent to the trilinear interpolation in the rectilinear grid, and the number of control points matches the number of rectilinear grid vertices $\boldsymbol{p}_k$. If $p = 3$, the approximation problem involves cubic B-splines. The d-dimensional B-spline is the tensor product of d p-degree (one-dimensional) B-spline basis functions.

2.5 Least-Squares Approximation

The thesis shows that we aim to acquire an approximation onto the rectilinear grid using interpolation and regularization. Our approximation should provide accurate results compared to the original data. The least-square method fits our needs in minimizing the difference between the original data and the interpolated results. In this section, we introduce the least-squares method and explain how this fits with our requirements in this thesis.

The least-square method minimizes the sum of the squares of the difference between the interpolation values of the scattered point $\psi(\boldsymbol{x}_\mathrm{i})$ and the input data value $\boldsymbol{f}_\mathrm{i}$ and is given by

$$\operatorname{argmin} \sum_{\mathrm{i} \in I} \left\| \psi(\boldsymbol{x}_\mathrm{i}) - \boldsymbol{f}_\mathrm{i} \right\|^2 . \tag{2.8}$$

Equation (2.8) yields a rectangular linear system with n rows (the number of scattered points) and m columns (the number of unknowns). Each row of the linear system expresses the fitting constraint of an input data point in terms of a linear function of the unknown coefficients $\boldsymbol{c}_\mathrm{j}$. In practice, this system will often be over-constrained ($n > m$), extremely sparse, and can be solved in the least-squares sense.

2.6 Regularization

Rank deficiency can arise when attempting to solve the least-squares system where the resolution of the rectilinear grid locally exceeds the density of input data points. This is closely related to the nature of the rectilinear grid and is a natural byproduct of the refinement procedure used to construct the output rectilinear grid.

Though the extension of the rectilinear grid throughout the domain might be considered a downside of the rectilinear grid vs. octrees. The extension that can be considered as wasteful allows for better reconstruction properties. In addition, when the waste of storage or computational cost in the rectilinear grid due to this extension through the domain is measured, it is considered very limited compared to the reconstruction properties gained. A common technique to mitigate this rank deficiency problem is to add a regularization term that assigns additional constraints to the linear system [20]. In this work, we add smoothness constraints that we formulate as zero discrete Laplacian of the unknown rectilinear grid point values (*i.e.*, $\Delta \boldsymbol{g} = \mathbf{0}$) [21]. These constraints define so-called *harmonic functions*, which have desirable properties in the context of our data approximation problem. In particular, harmonic functions are smooth functions entirely determined by their boundary conditions and whose values remain within the range of the boundary values. Here, the role of the boundary conditions is played by the unknowns for which enough interpolation constraints are available. Equation (2.9) shows the discrete Laplacian used for the regularization:

$$\Delta \boldsymbol{g} = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} + \frac{\partial^2 g}{\partial z^2} = 0 \tag{2.9}$$

The second derivative for the laplacian is obtained using finite difference using 3 stencils per axis, using the given equation for x-axis:

$$\frac{\partial^2 g}{\partial x^2} = \frac{2}{(h_1 + h_2)h_2} g_{x-1} + \frac{2}{(h_1 + h_2)} (\frac{1}{h_1} + \frac{1}{h_2}) g_x + \frac{2}{(h_1 + h_2)h_1} g_{x+1} \tag{2.10}$$

where h_1 is the spacing between the grid points g_x and g_{x+1}, and h_2 is the spacing between the grid points g_{x-1} and g_x. Same apply for the other two axes. Practically, we control the relative contribution of the smoothness constraints to the overall error measure through a weighting coefficient λ. With the addition of smoothness constraints, we obtain the following minimization problem:

$$\underset{\{\boldsymbol{g}_k\}_{k \in K}}{\operatorname{argmin}} \sum_{\mathrm{i} \in I} \left\| \psi(\boldsymbol{x}_\mathrm{i}) - \boldsymbol{f}_\mathrm{i} \right\|^2 + \lambda \sum_{k \in K} \left\| \Delta \boldsymbol{g}_k \right\|^2 \tag{2.11}$$

where $\Delta \boldsymbol{g}_k$ corresponds to the discrete Laplacian at the grid vertex $\boldsymbol{g}_k$.

It is important to note that the corresponding matrix is highly sparse due to the local support of both the interpolation kernel and the finite difference Laplacian. The suitable sparse solver can therefore solve the system of equations. We selected for that purpose the sparse QR factorization solution implemented in SuiteSparseQR [22] and available in Matlab, which efficiently leverages the available parallelism to compute the solution.

2.7 Error Measures

Throughout the thesis, we use different measures to assess our approximation result, including visual and numerical results. We measure approximation error using normalized root mean squared error (RMS), normalized average error (AVG), and normalized maximum error (NME):

$$\mathrm{RMS} = \frac{1}{n\, f_{\mathrm{rng}}} \sqrt{\sum_{\mathrm{i} \in I} \left(\psi(\boldsymbol{x}_{\mathrm{i}}) - f_{\mathrm{i}}\right)^2} \tag{2.12}$$

$$\mathrm{AVG} = \frac{1}{n\, f_{\mathrm{rng}}} \sum_{\mathrm{i} \in I} \left|\psi(\boldsymbol{x}_{\mathrm{i}}) - f_{\mathrm{i}}\right| \tag{2.13}$$

$$\mathrm{NME} = \frac{1}{f_{\mathrm{rng}}} \max_{\mathrm{i} \in I} \left|\psi(\boldsymbol{x}_{\mathrm{i}}) - f_{\mathrm{i}}\right| \tag{2.14}$$

where $f_{\mathrm{rng}} = \max_{\mathrm{i} \in I}(f_{\mathrm{i}}) - \min_{\mathrm{i} \in I}(f_{\mathrm{i}})$ is the range of the input data values.

2.8 Flow Map and Finite-Time Lyapunov Exponent

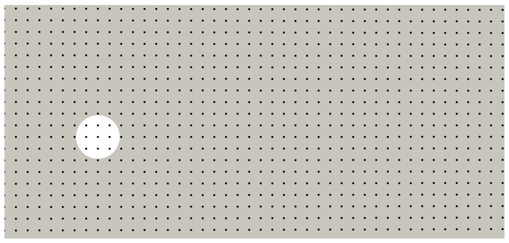
To assess the quality of our approximation of the original data, we use numerical and visual results. We listed the numerical measures in Section 2.7.

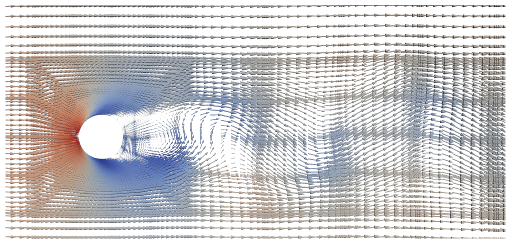
We also look for a visualization application that relies heavily on interpolation, thus serving as a diagnostic tool for our method's approximation quality. We chose for that purpose the finite-time Lyapunov exponent (FTLE), which is widely used in the visualization literature to extract flow structures [23]–[29].

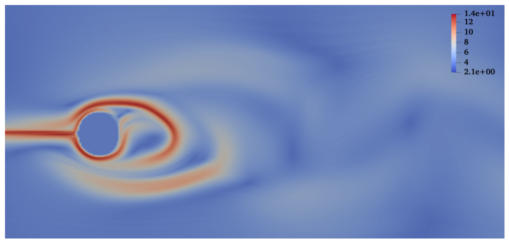
FTLE [30]–[33] evaluates at each location the rate of separation of nearby trajectories over a finite time interval under the action of the fluid flow. FTLE is an important tool to study the flow and its transport behavior. It owes its popularity to its ability to reveal the presence of so-called *Lagrangian coherent structures* (or LCS), which act as moving material boundaries that form barriers to particle transport [34].

A time-dependent vector field $\mathbf{v}$ defines a *dynamical system* through following equation [35].

$$\dot{\mathbf{x}}(t,\, t_0,\, \mathbf{x_0}) = \mathbf{v}(t,\, \mathbf{x}(t,\, t_0,\, \mathbf{x}_0)), \tag{2.15}$$


(a) Grid over which the FTLE values will be obtained.


(b) Velocity data for the Flow About a Cylinder dataset.



(c) The resulting FTLE.

Figure 2.4. 2D FTLE generation for Flow about a Cylinder data.

where the dot designates derivation with respect to the time variable t, and $\mathbf{x}(t, t_0, \mathbf{x}_0)$ is the position at time t of a point starting at $\mathbf{x}_0$ at time t_0. The path $\mathbf{x}(\cdot, t_0, \mathbf{x}_0)$ corresponding to the solution of Equation 2.15 forms an *integral curve* or streamline. The map $\phi_\tau : (t_0, \mathbf{x}_0) \mapsto \mathbf{x}(t_0 + \tau, t_0, \mathbf{x}_0)$, which maps each point $\mathbf{x}_0$ at t_0 to its new position after time τ, is called *flow map*.

The variations of the flow map around a given initial position $\mathbf{x}_0$ are locally determined by its spatial derivative, the matrix $\mathbf{J}(\tau, t_0, \mathbf{x}_0) = \nabla_{\mathbf{x}_0}\phi_\tau(t_0, \mathbf{x}_0)$. FTLE corresponds to the average exponential separation rate $\Lambda(\tau, t_0, \mathbf{x}_0)$, for positive or negative τ and is defined as

$$\Lambda(\tau, t_0, \mathbf{x}_0) = \frac{1}{2|\tau|} \log ||\mathbf{J}(\tau, t_0, \mathbf{x}_0)^T \mathbf{J}(\tau, t_0, \mathbf{x}_0)||. \tag{2.16}$$

Here $||.||$ designates the matrix norm corresponding to the maximum eigenvalue. Ridges[1] of Λ for forward (resp. backward) advection correspond to unstable (resp. stable) Lagrangian coherent manifolds that strongly repel (resp. attract) nearby particles. An example of FTLE computation is illustrated in Figure 2.4.

[1]↑Ridges can be thought of as lines and surfaces along which the considered scalar quantity – here FTLE – is locally maximal.

3. FRAMEWORK FOR REGULARIZED APPROXIMATION OF UNSTRUCTURED DATA WITH RECTILINEAR GRID

3.1 Introduction

This chapter considers the problem of approximating complex unstructured datasets with highly non-uniformly distributed points onto a rectilinear grid.

Our method falls in the general category of the so-called *scattered data* approximation problems. Nonetheless, we are primarily interested in the ability of the resulting global approximation to accurately approximate the native, cell-wise interpolation of the original unstructured grid.

The choice of the grid is a crucial aspect of the accuracy of the approximation. Given the vertices and associated input data values of an unstructured grid and a linear approximation kernel, we look for the most efficient grid that allows for accurate approximation.

We aim for a grid that accurately captures the main aspects of the simulation and is as small in size as possible.

We generate an adaptive rectilinear grid using a customized hybrid octree method that balances the point density and the local error to provide the most efficient rectilinear grid.

We aim for an approximation that provides a global solution. Our approximation minimizes the reconstruction error at the input data points while yielding a globally smooth approximated field.

We solve for the coefficients of the approximation kernel of the rectilinear grid by solving an optimization least-squares problem. Unfortunately, this optimization system is often ill-posed caused by the lack of constraints due to the non-uniform distribution of the original data. We introduced regularization to resolve the rank deficiency issue by adding smoothness constraints to the system. While regularization helps condition the ill-posedness in the approximation system, naive regularization can cause over-smoothing in the part of the system that is already well constrained and not enough smoothing where the system needs the smoothing constraint the most. Therefore, we introduced variable smoothing, which assigns adequate localized smoothing while providing an accurate approximation.

To make the approximation solution scalable and work with large unstructured datasets, we use parallel block-solving. The approximation system is split into blocks by domain. The size and number of blocks are chosen to maintain a system with a roughly equal number of unknowns to be solved in parallel to balance the workload of each thread.

We should note that this chapter limits our work to linear approximation kernel and does not consider solid boundary recognition and handling. We will demonstrate our framework with a higher-order kernel and add boundary-aware capabilities in the next chapter.

We provide a thorough evaluation of the hybrid octree method used for the meshing choice. The evaluation involves comparing the approximation quality of other meshing methods and our hybrid meshing. The approximation method is evaluated over 2D and 3D datasets. We also provide evaluation for our regularized approximation numerically and visually using interpolation intensive visualization algorithms. We manage to dramatically increase the performance of visualization post-processing and obtain a visual quality of the results that faithfully convey the properties of the original dataset.

The chapter is organized as follows. We start with Section 3.2, where we review various existing scattered data approximation methods. Section 3.3 explains the approximation problem we are dealing with and the various challenges involved. In Section 3.4 we explain the regularization used to constrain the rank-deficient system we are trying to solve. Section 3.5 introduces our variable regularization scheme. Later in Section 3.6, we discuss the choice of the grid to be used for the approximation. We illustrate in Section 3.7 how parallel block-solving enables solving large systems without deterioration in the approximation quality. Finally, an evaluation of our methodologies can be found in Section 3.8.

3.2 Related Work

Many solutions to the problem of scattered data interpolation can be found in the literature. These solutions can be categorized as mesh-based or mesh-free techniques.

Among the mesh-free techniques, radial basis functions (RBF) [36] construct an interpolating function as a weighted sum of basis functions centered at the input data points. Unfortunately, the basis functions that possess the best theoretical guarantees of approximation quality (*e.g.*, thin plate) have global support, yielding a dense linear system intractable for large datasets. RBF functions with local support avoid these problems, but they are challenging to apply to highly non-uniform point distributions, typical in unstructured computational meshes with adaptive refinement. In addition, while RBF enables a globally smooth approximation, their evaluation at arbitrary query points can be computationally expensive.

An alternative mesh-based approach, more closely related to the present work, consists in interpolating a set of arbitrarily distributed data points on a uniform grid. Arge *et al.* [37] compute a cell-wise fit over uniform grids, whereby only the vertices of cells containing enough data points are fitted through a linear least-squares system. In contrast, the value of the rest of the points is subsequently obtained using global extrapolation techniques. Arigovindan *et al.* [38] propose a variational approximation solution expressed as a regularized optimization problem that combines data fitting and smoothness constraints to accommodate non-uniform data points distribution. This setup leads to a large sparse linear least-squares system for which the authors introduce a multi-resolution formulation that lends itself to a solution via multigrid iterations. The method is restricted to B-spline kernels and 2D image approximation. Subsequent work by Vuçini *et al.* [39], [40] extends this method to the 3D setting and considers its application in scientific visualization. In particular, it introduces a scheme to construct the solution on a block-wise basis, which achieves smoothness across blocks at the expense of a higher approximation error along with their interface. Vuçini and Kropatsch [41] present a strategy to select a target resolution for the approximation grid based on a per-cell statistical study of the error distribution. Most recently, Francis *et al.* [42]. Introduced an efficient solution based on roughness minimization and digital filtering for 2D point sets.

Vuçini and Kropatsch [40], [41] studied the uniform grid resolution that will reduce the approximation error. They investigate the strong correlation between the approximation error and the average standard deviation of points values in the cells. They presented a strategy that selects a target uniform resolution for the approximation grid based on a per-cell statistical study of the error distribution. This study, done on a uniform grid, motivates us to add a local error measure to aid in obtaining low approximation error in the rectilinear grid while maintaining the grid size as small as possible.

The common limitation of all these methods is their restriction to uniform grids and specific interpolating kernels. In contrast, our solution allows for an accurate approximation over an automatically determined rectilinear grid through arbitrary kernels. As a result, we can achieve similar or better error bounds with a significantly lower resolution than prior work.

A different class of numerical analysis methods called adaptive mesh refinement (AMR) [43] uses multiple levels of data refinement. Regions that need higher precision are locally refined in subsequent more refined level patches. This dynamic focus of memory usage allows AMR to accurately represent mesh data without incurring the cost of a globally fine mesh. Structured Adaptive Mesh Refinement (SAMR) [44], [45] is an AMR approach where the computational grid is implemented as a collection of uniform grid components. It takes advantage of the simplicity of the uniform grid. However, such methods present several challenges, especially for parallel computing, such as load balancing, complex synchronization, maintaining continuity between levels, etc.

To facilitate interpolation in unstructured grids, efficient solutions to the point location problem have been devised in the visualization literature [16], [46], [47]. These methods use hierarchical tree data structures to accelerate the cell search process. Despite the significant speed-up they achieve, these data structures require extra storage, and interpolation in unstructured grids remains significantly more expensive than in rectilinear grids. Finally, a large body of research investigates various solutions to the data reduction problem. These solutions include lossless and lossy data compression techniques, as well as mesh reduction techniques. Li *et al.* [48] offer a recent overview of the relevant work from a visualization perspective. In addition, Hoang *et al.* propose a framework to study the trade-off between precision vs. resolution [49].

3.3 Scattered Data Approximation

Let $\boldsymbol{S}$ be the unstructured input grid, and let $\boldsymbol{G}$ be the rectilinear grid to be used for the approximation. Each input vertex $\boldsymbol{x}_{\mathrm{i}}$ in $\boldsymbol{S}$ is associated with a input data value $\boldsymbol{f}_{\mathrm{i}}$. The corresponding data approximation problem consists in finding the best least-squares approximation of the data points $(\boldsymbol{x}_{\mathrm{i}}, \boldsymbol{f}_{\mathrm{i}})$ in $\boldsymbol{S}$ through a given kernel-based approximation over $\boldsymbol{G}$ as discussed in Chapter 2, Equation 2.7. In this chapter, we are using linear fitting for simplicity. The kernel indicates that the fitting provides an approximation function value assigned to each rectilinear grid point. Our approximation method is not limited to linear fitting, and other kernels can be used. We will explore other kernel types in the next chapter. the least-squares problem in hand is given by Equation (2.8) in Chapter 2. Given the linear kernel to be used for the approximation. We are trying to solve for the fitting values for the rectilinear grid that will minimize the approximation error, which is the difference between the original function value and the interpolated value from the approximation. The approximation problem is solving the following linear problem:

$$\mathbf{A}\boldsymbol{x} \approx \boldsymbol{b} \tag{3.1}$$

$\boldsymbol{b}$ is the function values f_{i} that we are trying to minimize the 2-norm difference with the interpolation value given the specified kernel. The matrix $\mathbf{A}$ is rectangular with the number of rows (*i.e.*, the number of input data points) typically widely exceeding the number of columns (*i.e.*, the number of control point unknowns). The matrix $\mathbf{A}$ is sparse, and the sparsity depends on the choice of the kernel. The higher the order of the approximation kernel, the less sparse the $\mathbf{A}$ matrix would be.

3.4 Regularized Smoothing

In practice, this linear system is generally rank-deficient, owing to the unavoidable discrepancies between the spatial distributions of the input data points and that of the rectilinear control points. A common technique to mitigate this rank deficiency problem is to add a regularization term that assigns additional constraints to the linear system [20]. In this work, we add smoothness constraints that we formulate as zero discrete Laplacian of the unknown rectilinear grid point values (*i.e.*, $\Delta \boldsymbol{g} = \mathbf{0}$) [21]. These constraints define so-called *harmonic functions*, which have desirable properties in the context of our data approximation problem. In particular, harmonic functions are smooth functions entirely determined by their boundary conditions and whose values remain within the range of the boundary values. Here, the role of the boundary conditions is played by the unknowns for which enough interpolation constraints are available. Practically, we control the relative contribution of the smoothness constraints to the overall error measure through a weighting coefficient λ. With the addition of smoothness constraints, we obtain the following minimization problem:

$$\underset{\{\boldsymbol{g}_\mathrm{j}\}_{\mathrm{j}\in J}}{\operatorname{argmin}} \sum_{\mathrm{i}\in I} \|\psi(\boldsymbol{x}_\mathrm{i}) - \boldsymbol{f}_\mathrm{i}\|^2 + \lambda \sum_{\mathrm{j}\in J} \left\|\Delta \boldsymbol{g}_\mathrm{j}\right\|^2 \tag{3.2}$$

where $\Delta \boldsymbol{g}_k$ corresponds to the discrete Laplacian at the grid point $\boldsymbol{g}_k$.

The introduction of the regularization to the initial linear system in Equation (3.1) described above will provide the $\mathbf{A}$ matrix composed of an interpolation part and regularization part. For the right-hand side, we will have b corresponding to the interpolation rows and zeros for the rest of the rows given we aim at minimizing the Laplacian. After the addition of the regularization the initial linear system shown in Equation (3.1) will become the following:

$$\begin{bmatrix} \mathbf{A} \\ \Delta \boldsymbol{g} \end{bmatrix} \boldsymbol{x} \approx \begin{bmatrix} \boldsymbol{b} \\ \mathbf{0} \end{bmatrix} \tag{3.3}$$

Where $\Delta \boldsymbol{g}$ corresponds to the discrete Laplacian, obtained by discretizing over the grid using 2nd order finite differences.

It is important to note that the corresponding matrix is extremely sparse due to the local support of both the interpolation kernel and the finite difference Laplacian. The system of equations can therefore be solved with a suitable sparse solver. We selected for that purpose the sparse QR factorization solution implemented in SuiteSparseQR [22] and available in MATLAB, which efficiently leverages the available parallelism to compute the solution. QR factorization is a decomposition of a matrix $\mathbf{A}$ into a product $\mathbf{A} = \mathbf{QR}$ of an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$. QR decomposition is often used to solve the linear least-squares problem and is the basis for a particular eigenvalue algorithm [20].

3.5 Bi-Level Smoothing

In Chapter 2, we introduced regularization. The regularization or smoothing term of Equation (3.2) effectively plays two roles. First, it enforces smoothness across the domain in the solution. Second, it adds constraints on the rectilinear grid point values in the linear system, which mitigates the potential rank deficiency of the fitting constraints. However, it can be challenging to find a single regularization weight that proves suitable across the whole approximation domain. For grid points with a sufficient number of associated interpolation constraints, an extreme value of λ causes an over-smoothing effect, resulting in loss of detail and increased approximation error. On the other hand, an extremely low λ value will not be sufficient to resolve the ill-posedness in the system that we introduced the regularization to resolve.

This observation led to the use of bi-level smoothing, where each approximation grid point is assigned one of several possible smoothing coefficients λ_k. Equation (3.2) is modified to be

$$\underset{\{\boldsymbol{g}_k\}_{k\in K}}{\operatorname{argmin}} \sum_{\mathrm{i}\in I} \|\psi(\boldsymbol{x}_{\mathrm{i}}) - \boldsymbol{f}_{\mathrm{i}}\|^2 + \lambda_k \sum_{k\in K} \|\Delta \boldsymbol{g}_k\|^2 \tag{3.4}$$

Each grid point uses a suitable λ_k value that corresponds to the grid point ill-posedness criteria.

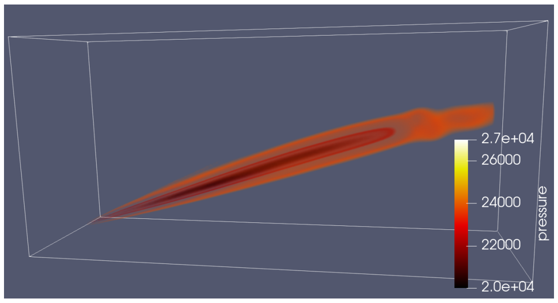
Practically, we use two values of λ: A high value λ_{high} at grid points that lack sufficient constraints due to few or no surrounding input data points, and a lower value λ_{low} for all other well-constraint grid points.

Two criteria are used to determine whether a low or high value λ should be used at any given grid point: 1) The number of input data points surrounding the grid point is considered to mitigate rank deficiency. Each input data point within the kernel support provides a constraint to the unknown in the system. With a low uniform smoothing, the grid points with no interpolating constraints cause rank deficiency. Given a linear kernel, we consider a grid point with less than eight input data points within its kernel support in 3D

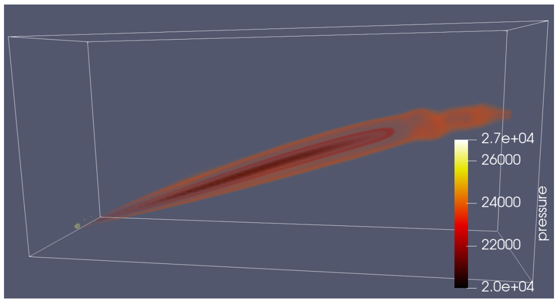
(4 in 2D) to be *insufficiently constrained* and assign it a high smoothing coefficient. 2) The distance of the closest input data points to the grid point is used to judge whether the system is ill-conditioned or not. Grid points with a sufficient number of input data points within the kernel support can still cause an ill-conditioned system if those input data points are on (or close to) the edge of the kernel support where the kernel tapers off. In that case, the column of the linear system associated with the grid point will have a minimal max norm, resulting in significant artifacts in the approximation solution. If the linear kernel basis value of the closest data point is less than 0.4, we consider the grid point to have insufficient interpolation constraints and apply a higher smoothing coefficient. The specific values of λ_{high} and λ_{low} are dependent on the dataset and the approximation kernel and must be chosen based on several practical considerations. λ_{high} must be high enough to prevent visible artifacts in regions with sparse data points, but not too high not adversely to affect the approximation accuracy. Similarly, the value of λ_{low} must be chosen low enough to achieve an accurate fit in densely populated regions.

Using our bi-level smoothing scheme, we can achieve an accurate approximation while maintaining the desired level of overall smoothness in the solution. Figure 3.1 shows the volume rendering comparison between uniform smoothing and our bi-level smoothing constructed with the same rectilinear grid. Our bi-level smoothing accurately captures the vortices without generating artifacts. In contrast, uniform smoothing either produces spurious structures in low-constraint regions for low values of λ or filters out relevant features of the dataset for large λ. For reference, the average error and normalized max error AVG / NME (Equations (2.13) and (2.14)) in this particular case are 0.0066 / 3.4303 for uniform low λ, 0.0376 / 0.6980 for uniform high λ, and 0.0065 / 0.4777 with our bi-level smoothing scheme. We can find the same observation in Figure 3.2.

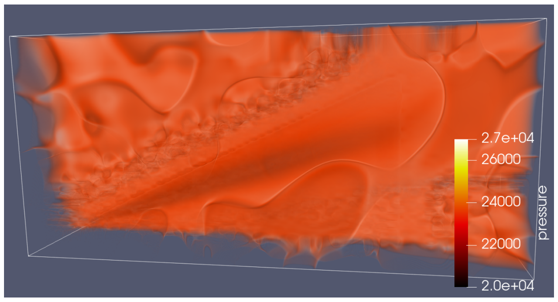
To allow for a fixed pair of smoothing coefficients to produce high-quality results, irrespective of the specific scale and spatial resolution of the considered dataset, we perform a normalization of the finite-difference Laplace equation associated with each unknown. Specifically, we express the Laplace equation using a finite-difference formula for the second-order derivative suitable for arbitrarily spaced samples [50] and then normalize the resulting equation. The coefficient of the considered unknown is 1. With this normalization, the same



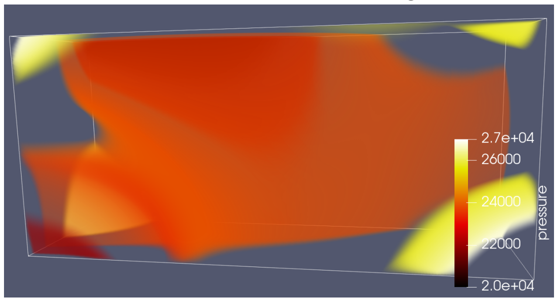
(a) The original unstructured grid dataset.



(b) Approximation using our bi-level smoothing with $\lambda_{\text{low}} = 1 \times 10^{-9}$ and $\lambda_{\text{high}} = 1$.



(c) Approximation using uniform smoothing with $\lambda = 1 \times 10^{-9}$.



(d) Approximation using uniform smoothing with $\lambda = 1$.

Figure 3.1. Volume rendering of a vortex in the original Delta Wing dataset and its approximation. Our bi-level smoothing scheme produces a much more accurate approximation compared to uniform smoothing.

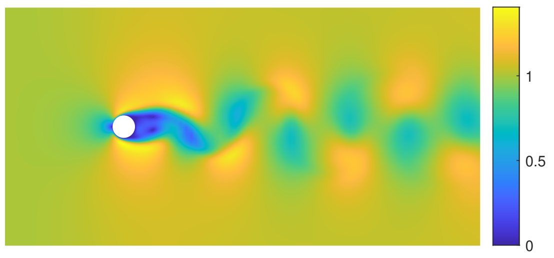
λ values can be used across datasets. In Section 3.8 we present the results for the three datasets that were produced using our bi-level normalized smoothing and the same $\lambda_{\text{high}} = 1$ and $\lambda_{\text{low}} = 1 \times 10^{-9}$.

3.6 Adaptive Grid Design

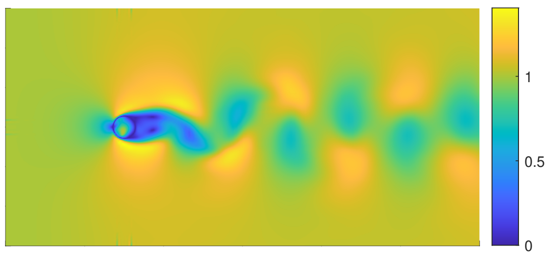
Our solution's key aspect consists of creating a rectilinear mesh upon which we can compute an accurate approximation of the underlying irregularly distributed data.

A good approximation needs an approximation grid that is fine enough to capture the essential features in the original data without unnecessary high grid resolution. The solving and storage become overly expensive.

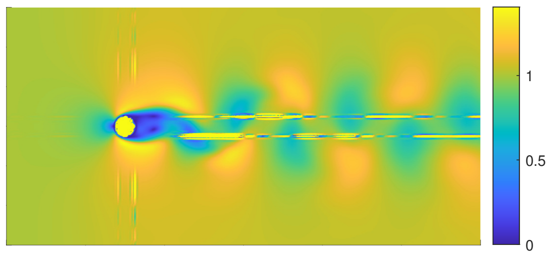
We present a hybrid grid design to be used for the approximation. The grid is adaptive and automatically determined using an octree that is refined based on the original input



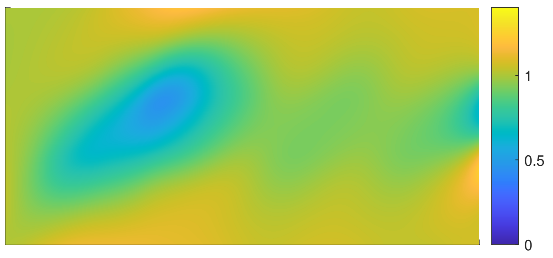
(a) The original unstructured dataset.



(b) Approximation using bi-level smoothing with RMS error $= 7.9\text{e}-2$.



(c) Approximation with uniform smoothing $\lambda = 1\text{e}-9$ and RMS error $= 7.88\text{e}-2$.



(d) Approximation using uniform smoothing with $\lambda = 1$ and RMS error $= 1.76\text{e}-1$.

Figure 3.2. The Flow About a Cylinder original dataset and different approximations using different smoothing methods. Our bi-level smoothing scheme produces a much more accurate approximation compared to uniform smoothing.

point density distribution. In addition, motivated by [40], [41] where the authors use the standard deviation of the points inside the uniform grid cell as a criterion for how fine the uniform grid of their choice should be. We follow a similar approach, but with the local fit error. We refine the grid where the error is high due to the high variation among the points function values, but we use an adaptive grid instead of a uniform one. As a result, we can prevent unnecessary refinement and obtain a smaller rectilinear grid.

We considered several approaches for this purpose. First, we explain why We adopted an adaptive solution instead of a uniform one, which is more straightforward. In the following, we show our findings before presenting the final adaptive resolution that we adopted for the approximation.

3.6.1 Uniform Grid / Image

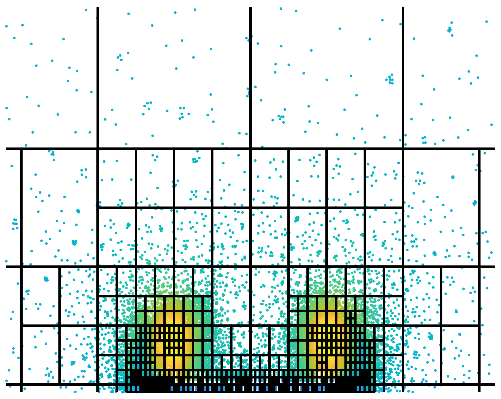
The simplest type of rectilinear grid is a uniform grid, an axis-aligned grid with constant spacing along each axis. Unfortunately, uniform grids are ill-suited to approximate most unstructured datasets, which tend to have a highly non-uniform point distribution. Indeed increasing the resolution often results in a marginal decrease in the approximation error. Hence, one needs to reach a prohibitively fine uniform resolution to achieve a prescribed approximation error threshold, which is inefficient and numerically problematic. To overcome the issues faced when using a uniform grid, we look into a rectilinear grid as an alternative that provides an axis-aligned grid but allows for the flexibility of variable grid spacing. The right choice of grid placements plays an essential role in the approximation quality. The following section discusses different designs for the rectilinear grid and how we reached our hybrid octree design.

3.6.2 Spatial Distribution Driven Meshing

In this approach, the goal is to select the grid resolution and the per-axis point distribution in such a way as to achieve a nearly uniform number of data points per cell. Since this is usually impossible, an alternative objective is to keep the number of data points per cell under a given threshold. While this approach yields a lower approximation error than the uniform one, it proves ineffective since high point density per cell does not necessarily imply high approximation error and vice versa. This occurs when dense regions in the unstructured grid are smooth yet cause grid refinement without any improvement in the error. Conversely, regions with few data points but with a vast variance in the per-point value can benefit from a higher grid resolution, even though the number of the points may be less than the threshold. Hence a density-based refinement might yield a high resolution that is not necessarily needed.

3.6.3 Error-Driven Meshing

The idea of error-driven meshing consists in computing the global approximation error at intermediate refinement stages and using this information to prioritize the refinement of regions associated with the more significant error. This method, by design, reduces the error much more effectively than both previous approaches. Unfortunately, this solution requires solving the data approximation problem for each intermediate resolution, which amounts to a costly procedure.


(a) Quadtree generated using our hybrid method. The quadtree is shown in black lines, and the input points are shown as dots for reference.

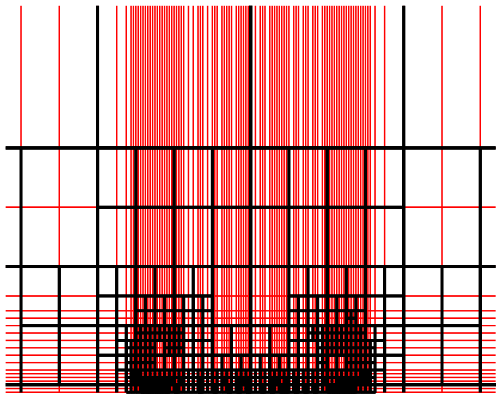

(b) The resulting rectilinear grid from extending the quadtree, shown in red.

Figure 3.3. The quadtree was generated using our hybrid method and the corresponding rectilinear grid.

3.6.4 Proposed Hybrid Solution

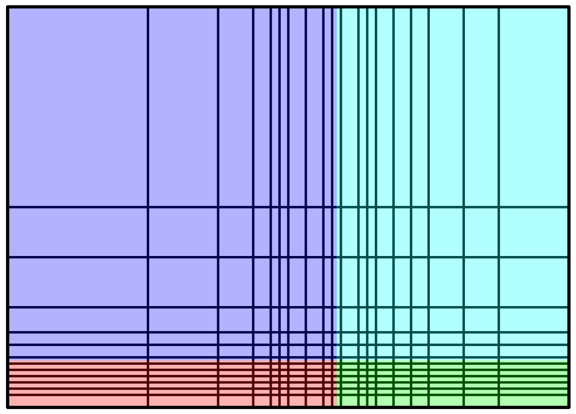
We concluded from the observations above that a more efficient and effective way to adaptively refine a rectilinear grid consists in adopting a hybrid solution that jointly considers the spatial distribution of the data points and the local approximation error.

Practically, we create an octree (or quadtree for 2D datasets). The recursive subdivision of each leaf is controlled by two thresholds, namely the number of points in the leaf and the maximum allowed approximation error over the corresponding cell. In particular, this procedure will prevent the unnecessary refinement of a dense region if it exhibits a low approximation error. In this context, the approximation error is evaluated only locally by solving a small least-squares system corresponding to the trilinear fitting of the data points located inside the leaf cell. Indeed, using a low-order (multi-linear) kernel in this meshing phase offers a conservative estimate of the approximation error that we can subsequently achieve with a potentially higher-order kernel in the global approximation phase. If the number of points per leaf is less than a threshold, the points available are not sufficient for generating a local fit, and the subdivision will stop. Otherwise, the error of the local fit is calculated, and if it exceeds a chosen error tolerance, the leaf cell is further subdivided.

Once the octree is constructed, it is converted into a rectilinear grid by extending the bounding box of each leaf onto the individual coordinate axes. The rectilinear grid generated by the hybrid method is fine enough where needed. This would result in a smaller grid that will be more efficient in solving, storing, and post-processing. An example is shown in Figure 3.3, where a zoom on the slice of the Delta Wing dataset with quadtree generated in Figure 3.3a with the scattered points drawn for reference. In Figure 3.3b the rectilinear grid is generated by extending each quadtree leaf.

3.7 Block-Based Solving

After acquiring the rectilinear grid resolution and axis coordinates, the next step is to solve Equation (3.3) for the approximated values at each grid point. For 3D approximations, as the input data size and approximation grid resolution increase, the memory consumption of the sparse QR solver follows a cubic growth $\mathcal{O}(n^3)$, and can become intractable. To overcome this issue, we partition the rectilinear grid domain into independent blocks by splitting the cells, such that each block contains about the same number of rectilinear grid points, *i.e.*, the same number of unknowns in Equation (3.3). The partitioning into blocks is done over the rectilinear grid obtained by extending the octree as illustrated in Section 3.6. Each block can then be approximated independently, which effectively distributes the storage and computational effort. Layers of ghost grid points are added to the boundary of each block to ensure continuity and smoothness of the solution across block boundaries while preserving the accuracy of the approximation. The number of layers of ghost grid points


(a) The rectilinear grid without blocks division.

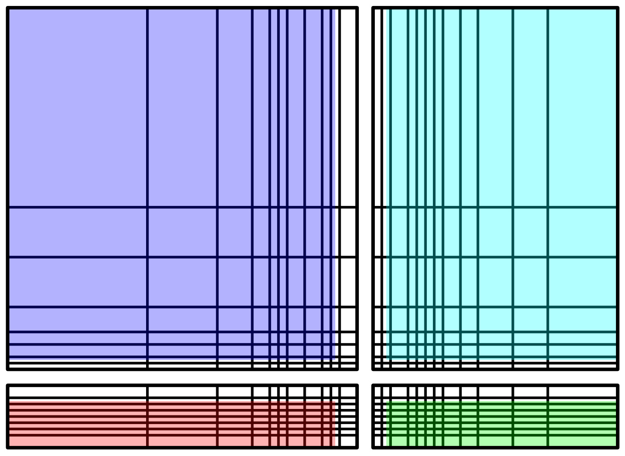

(b) Splitting the rectilinear grid to the left into 4 subdomains with two layers of ghost grid points in white.

Figure 3.4. Example of block-solver divisions.

added depends on the footprint of the kernel used. For multi-linear interpolation, we find that two layers of ghost grid points are sufficient to enforce smoothness.

Each block results in a smaller linear system that is solved independently of the other blocks. In particular, no communication between the blocks is necessary. After obtaining the approximation solution for each block, we seamlessly join the blocks into a global solution. The values associated with the ghost points of each block are averaged at the merge stage. Figure 3.4 shows an example of partitioning a 2D domain into 4 blocks of equal rectilinear resolution. The block method enables the parallel and distributed computation of the approximation solution while maintaining its accuracy. This division of labor into independent blocks dramatically reduces the computation time. For instance, on a 4-core computer, we parallelize the approximation using MATLAB's parallel for-loops (`parfor`) with 4 workers, each processing a series of blocks. Figure 3.5 shows that increasing the number of blocks reduces the parallel approximation time without a noticeable difference in the RMS error for 1M approximation grid points on the Delta Wing dataset. Unsurprisingly, the block decomposition only decreases the computation time up to a point since the relative overhead associated with the ghost points increases with the number of blocks.

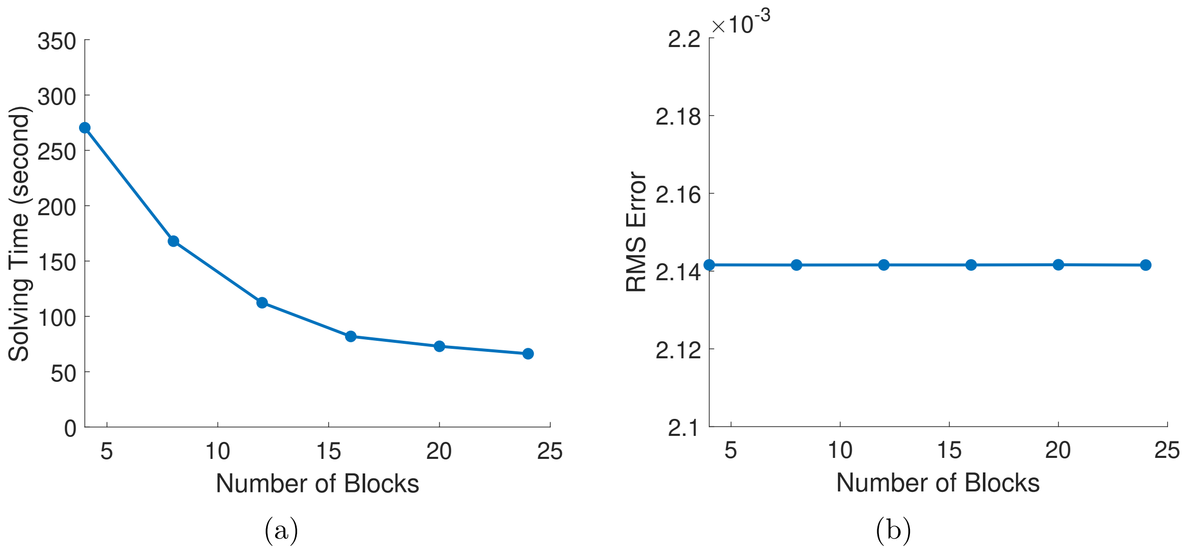
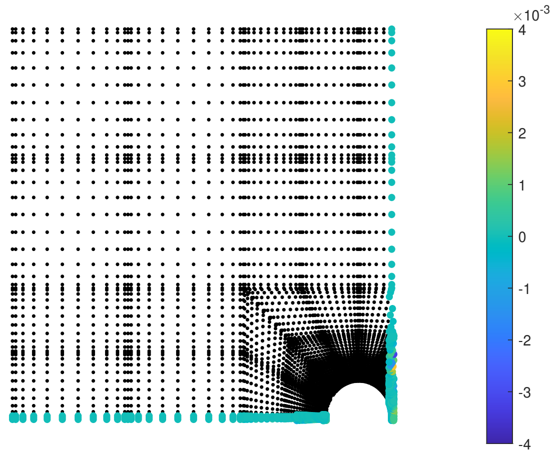
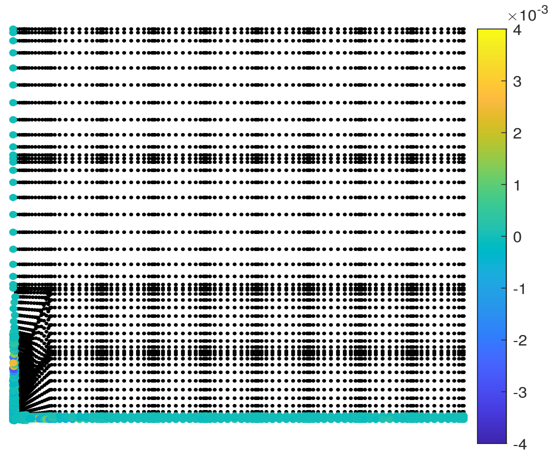


Figure 3.5. An increase in the number of blocks reduces the approximation time required (a) without affecting the RMS error (b).

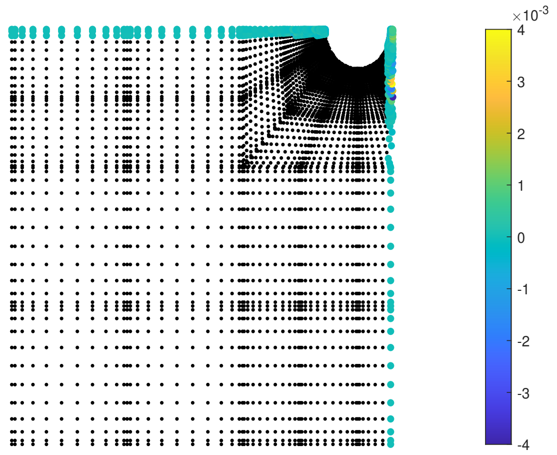
It is worth mentioning that when we split the system into blocks, we are solving different least-square systems, given that the system is fully coupled. We extend the block to several layers of overlapping cells relative to the approximating kernel local support. The addition of overlapping layers minimizes any discontinuity between the blocks resulting from the system split by using the same interpolation fitting at the merging locations. In the case of the linear kernel used here, we extend the block to two voxels (we would use 4 voxels for a cubic approximation kernel). We expect a slight change in approximation error for the points in the overlapped region between the blocks compared to the error in the merged solution. We need to make sure that this difference is small and doesn't affect the reconstruction quality. Figure 3.6 shows the error difference for the overlapping points between the error within the block and the merged solution. We also show in Figure 3.7a the histogram of the distribution of the error difference between the point-wise error in the block vs. the error in the merged solution. The figure shows that a very small percentage of the points are affected by the merge, while most did not see any significant error change. The other two histograms in Figure 3.7b and 3.7c show the error for the overlapping points in the merged block solution and in the solution with no blocking, which reveals that the error is in fact a result of the approximation, not a side effect of the block decomposition. It also shows that the error



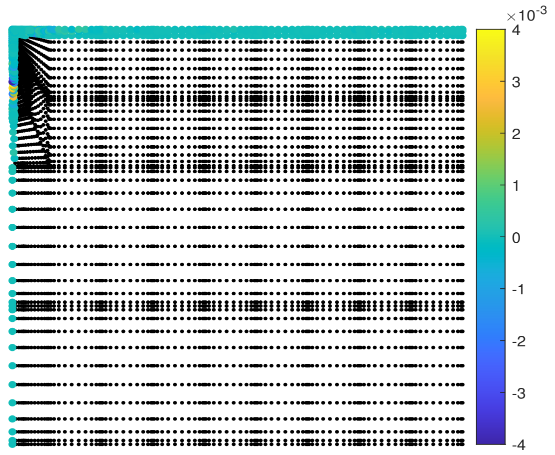
(a) Block1



(b) Block2

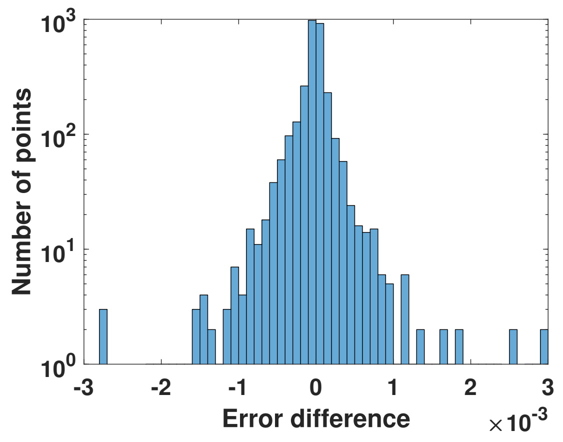


(c) Block3

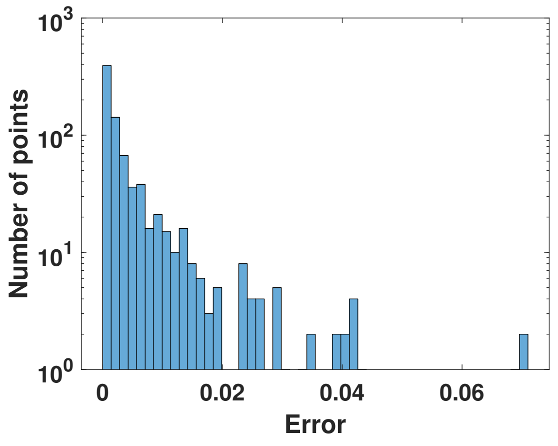


(d) Block4

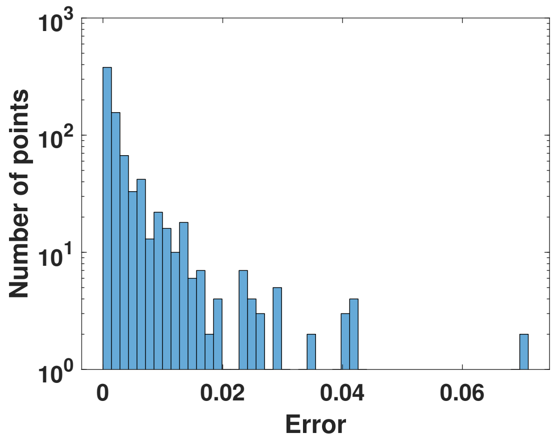
Figure 3.6. The point-wise error-difference between single blocks and the merged blocks for the points in the overlapped region. The original data points for the flow around a cylinder dataset are shown in black for reference.



(a) Error difference histogram



(b) Error histogram for the merged blocks



(c) Error histogram for the no block solution

Figure 3.7. Three histograms of the overlapping points between the blocks.

variations caused by the merge can be neglected given their very low values compared to the approximation error.

3.8 Evaluation

We use 2D and 3D datasets for evaluating our resulting rectilinear approximation. We also compare our hybrid grid design with other grid designs by comparing the approximation quality of various grid designs. Below are short descriptions of the four datasets used to evaluate our hybrid grid design and bi-level regularized approximation.

1. **Flow around a cylinder**: 2D viscous flow around a solid cylinder boundary. Fluid is injected to the left of a channel bounded by solid walls with a slip boundary condition.

2. **Delta Wing**: Navier-Stokes simulation of wind flow around a delta-shaped wing at low speed and high angle of attack. The dataset is comprised of 3M points and 12M cells. The point distribution was adaptively refined to resolve multiple vortices present above the wing, with high point density near the wing and sparse point distribution elsewhere. This makes it a challenging dataset to process due to the extreme non-uniformity of the point distribution.

3. **ICE Train**: Simulation of a high-speed train traveling at a velocity of about 250 km/h with the wind blowing from the side at an angle of 30 degrees. The vortices resulting from the wind are used to study the effects of the train's track holding. The data consists of 1.1M points and 2.7M cells. The point distribution for this dataset is adaptively refined around the train body.

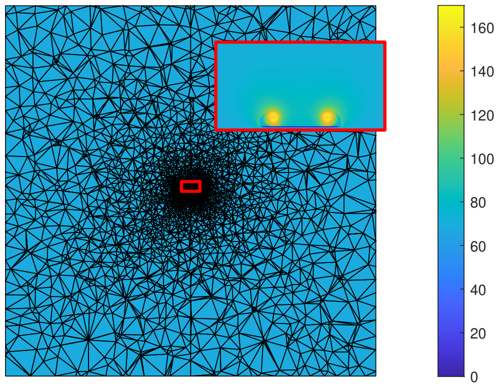
4. **Fish Tank**: Simulation of turbulent mixing of hot and cold air in a box-shaped domain, generated by the Nek5000 solver [51]. The dataset contains 22M points and 23M cells. The spatial distribution of the data points is relatively uniform and close to structured.

3.8.1 Adaptive Grid Design Evaluation

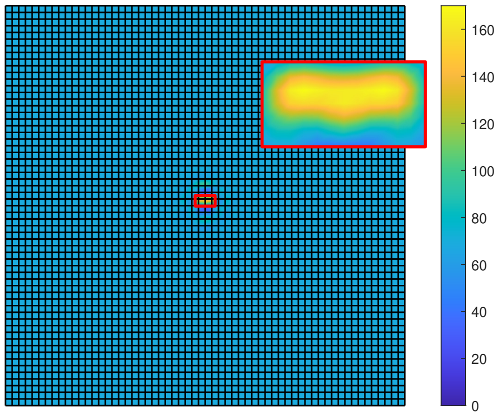
Using our proposed bi-level regularized approximation, we test the various designs for the adaptive grid.

Due to the highly non-uniformly distributed nature of the simulation data, the approximation results achieved with a uniform grid are significantly inferior to those produced with a rectilinear grid. Figures 3.8, 3.9, and 3.10 show that using a uniform grid with highly non-uniformly distributed data points is not the optimal solution. The resolution will be wasted where it is not needed and missing where it is required.

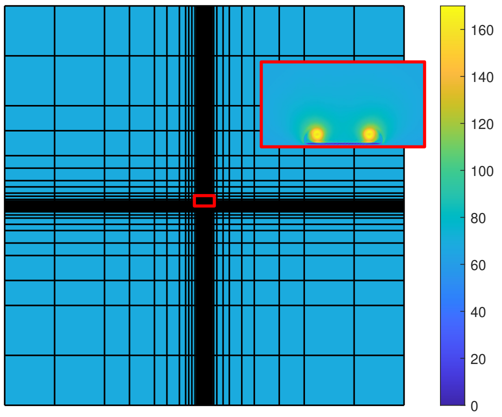
For the given observation, one considers the more general case of rectilinear grids, in which the cells are axis-aligned, but the spacing along each axis can be arbitrary.



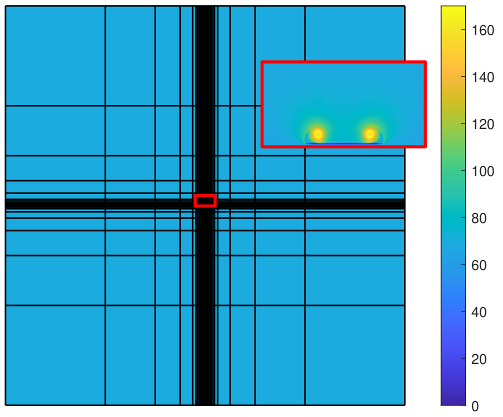
(a) The original Delta Wing dataset showing the two vortices, with 23K points.



(b) Approximation on a uniform grid with resolution of 61×61 ($= 3721$ points) and RMS error of 2.93×10^{-1}.

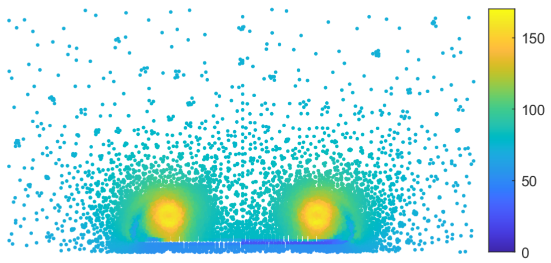


(c) Approximation using rectilinear grid created with spatial method only with a resolution of 123×49 ($= 6027$ points) and RMS error of 6.82×10^{-2}.

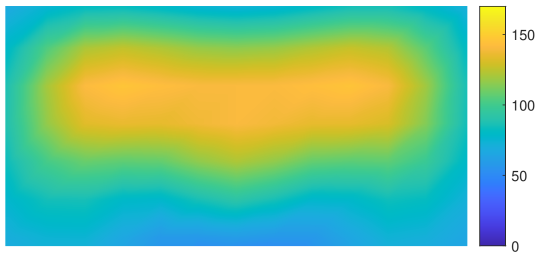


(d) Approximation on rectilinear grid created with our hybrid adaptive resolution with 107×34 ($= 3638$ points) and RMS error of 6.82×10^{-2}.

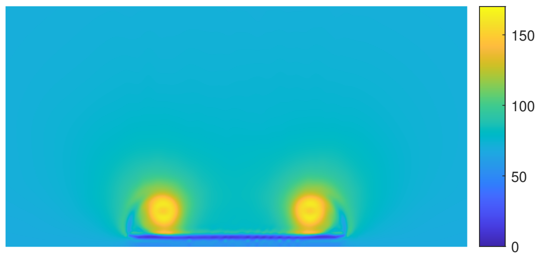
Figure 3.8. 2D slice of the Delta Wing original data and its approximations using various grids, colored by velocity magnitude and zoomed at the vortices. The rectilinear grid generated using our hybrid method provides the best visual result and low approximation error while maintaining the smallest grid size.



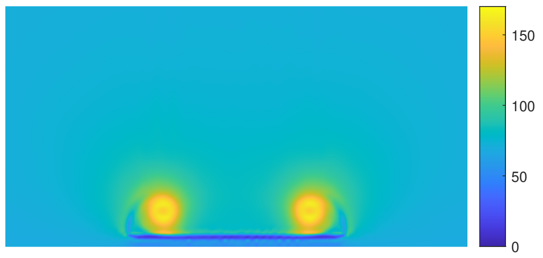
(a) The original dataset scattered points.



(b) Approximation using uniform resolution (RMS error 2.93e − 01).



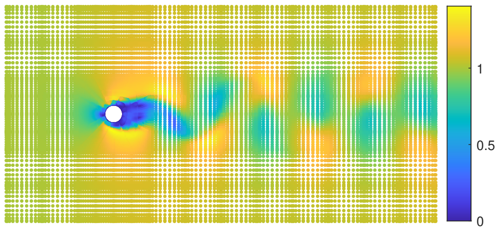
(c) Approximation using only spatial point distribution (RMS error 6.82e − 02).



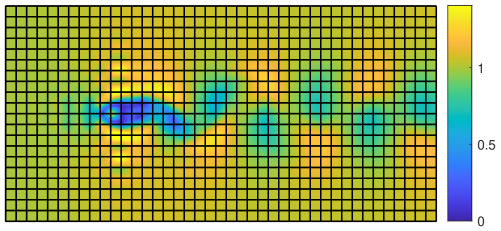
(d) Approximation using the hybrid method (RMS error 6.82e − 02).

Figure 3.9. Zoom at vortices of a 2D slice of Delta Wing dataset and its approximations using various grids, colored by velocity magnitude. The corresponding grids can be seen in Figure 3.8.

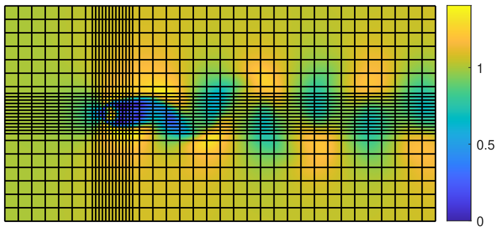
The use of the hybrid solution stops the refinement if the local error is low, even if the points are dense. This occurs when the variation in the function values of the dense points is slight. We add the depth of the quadtree/octree as other criteria to stop the subdivision. Hence, you obtain a reasonable resolution and prevent the grid from over-refinement, which would lead to issues related to the rank deficiency that we will discuss in the next chapter. The hybrid method allows for obtaining a smaller grid while maintaining the same approximation quality. Figure 3.11 shows that the number of points is small, the max level for the quadtree was reached, or the local error is low. This last criterion leads to a smaller grid without deteriorating the approximation quality. The main benefit of the hybrid method is generating a smaller grid and, therefore, faster to solve and interpolate while maintaining visually and numerically the same approximation quality as shown in Figure 3.9. The resulting grid from the hybrid method has resolution 107×34 ($= 3,638$ points), and the grid resolution resulting from only using the spatial point distribution is



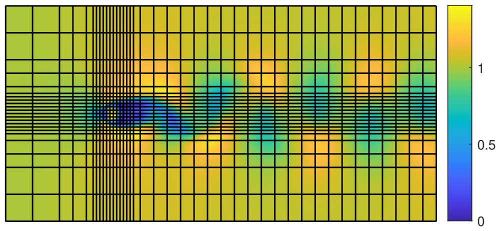
(a) The original dataset, with around 22K points.



(b) Approximation on a uniform grid with 42×23 (= 966 points) and RMS error of 6.9×10^{-2}.



(c) Approximation on rectilinear grid created with spatial method only 43×27 (= 1161 points) and RMS error of 1.71×10^{-2}.

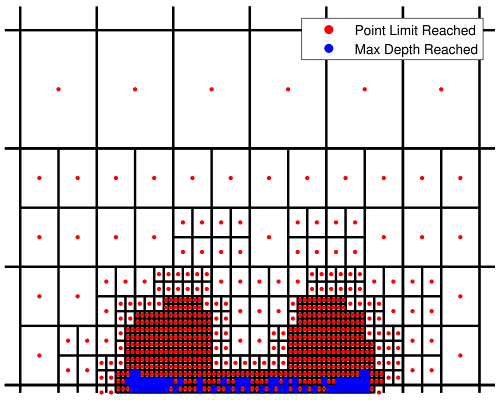


(d) Approximation on rectilinear grid created with our hybrid adaptive resolution with 40×23 (= 920 points) and RMS error of 1.71×10^{-2}.

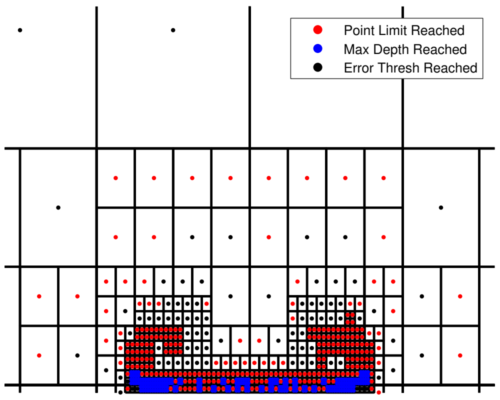
Figure 3.10. The flow around a cylinder original dataset and its approximations using various grids, colored by velocity magnitude.

123×49 (= 6,027 points), about 40% (1 : 1.6) fewer points to solve while the approximation maintained the same RMS error level.

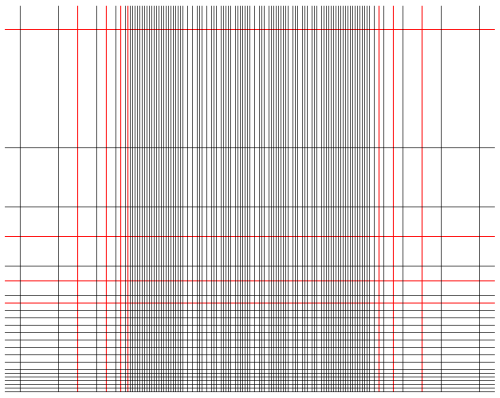
The same observation can be drawn from Figure 3.10. The Figure shows the original 2D flow dataset on the top 3.10a. The bottom images of the Figure compare the approximation results using a uniform grid, an adaptive grid without a hybrid method, and an adaptive grid using the hybrid method. Using the hybrid method, we can obtain an 18% smaller grid (920 vertices without hybrid octree: 1,161 vertices with hybrid octree) while maintaining the same RMS error. Notice that the Delta Wing dataset has a lower reduction in the grid resolution size due to the higher sparsity in the point distribution compared to the flow around a cylinder dataset.



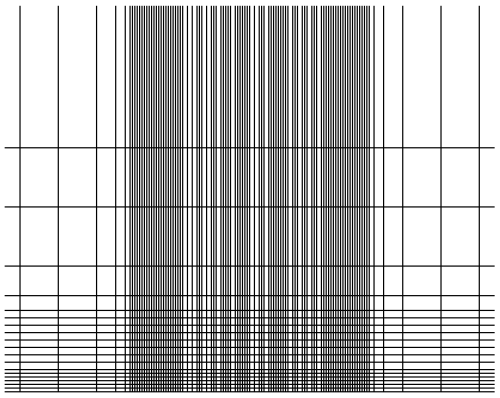
(a) Octree using only the spatial distribution.



(b) Octree with our hybrid method.



(c) The resulting grid using only the spatial distribution(123×49) of input points. The red lines represent the extra resolution in the resulting grid using spatial meshing compared to our hybrid method.



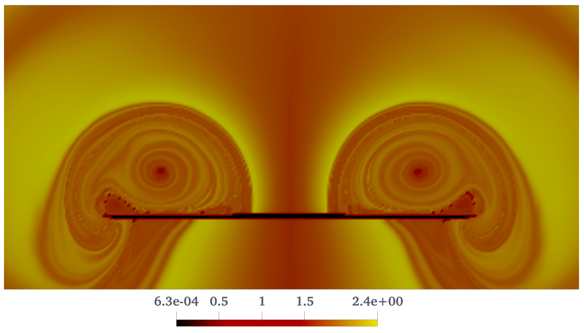
(d) The resulting grid using our hybrid method (107×34).

Figure 3.11. Comparison between the octree with and without the hybrid method and the resulting rectilinear grid from the octrees. The rectilinear grid using the spatial method is larger than the rectilinear grid resulting from the hybrid method.

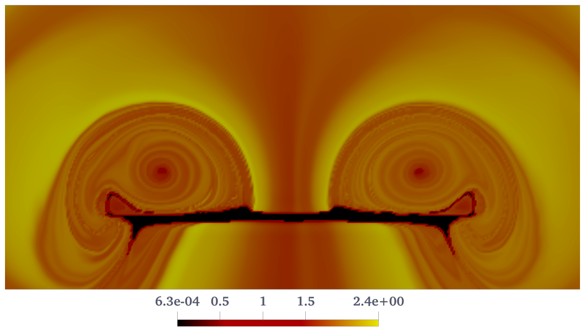
3.8.2 Bi-level Regularization Approximation Evaluation

Qualitative Evaluation

Our evaluation of the Delta Wing and the ICE Train datasets focus on the flow map's computation, in other words, the flow-induced transport, which is the computational basis of a wide range of highly popular flow visualization algorithms. A fundamental challenge in the approximation of fluid flow datasets is that this integration-based data processing leads



(a) The original unstructured grid.



(b) Our approximation.

Figure 3.12. Slice for the forward integration of the flow map for the Delta Wing original data and our approximation.

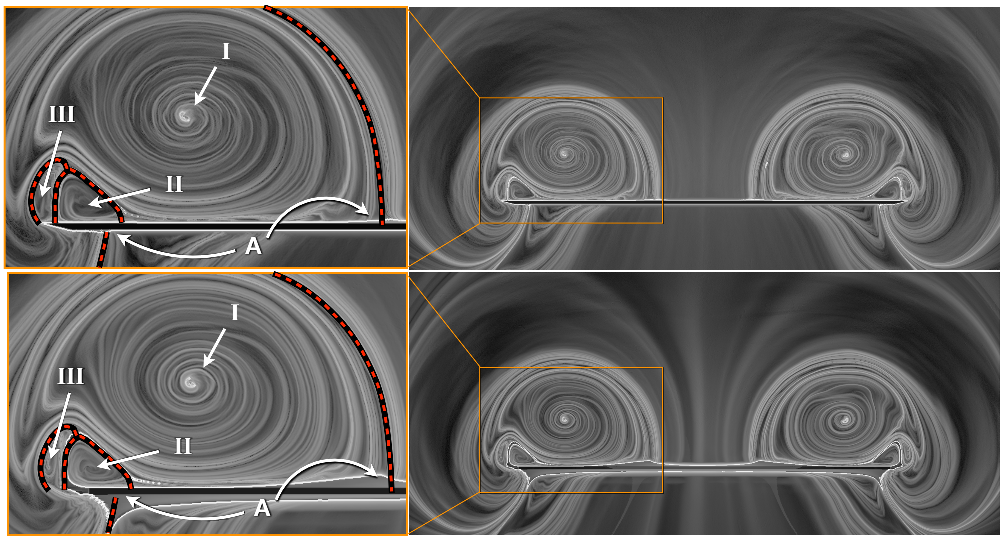
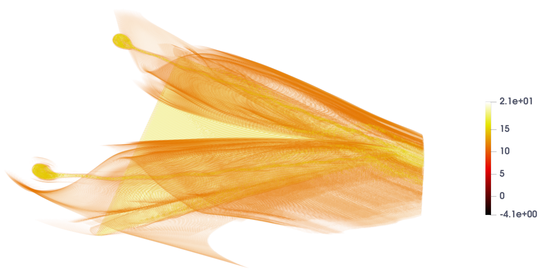


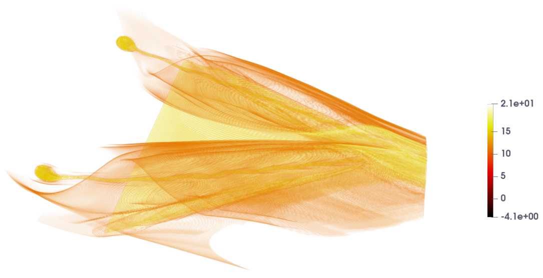
Figure 3.13. Slice of the forward FTLE field derived from the flowmap for the original unstructured grid (top) and the rectilinear approximation (bottom). Left: close-up view of the vortical structures present on the left side of the wing: I: primary, II: secondary, III: tertiary vortex, A: attachment.

to an accumulation of the local approximation error over thousands of integration steps. Therefore, we are interested in studying the impact of this cumulative error on the quality of the results that we can achieve with our approximation.

The timing for flow map generation was performed on a computer with Intel i7-3930K 3.20GHz 6 cores processor, with 12MB cache and 32GB RAM. We did other timing exper-

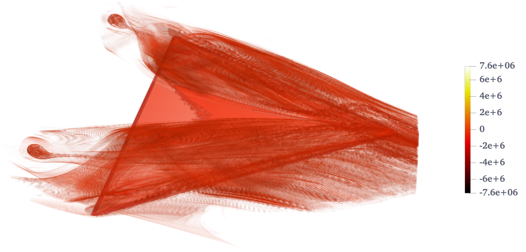


(a) The original unstructured grid.

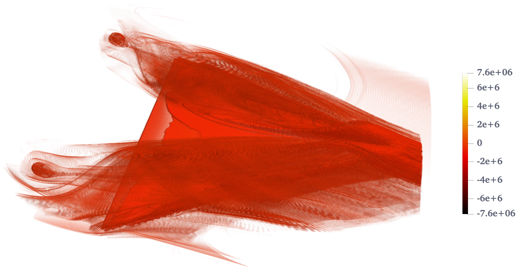


(b) Our approximation.

Figure 3.14. Volume rendering of the FTLE field flow map or $\Delta T = 10$ time steps / 0.1s in the original Delta Wing dataset and our approximation.



(a) The original dataset.



(b) Our approximation.

Figure 3.15. Volume rendering of LCS in flow map $\Delta T = 10$ time steps / 0.1s for the original Delta Wing unstructured grid and our approximation.

iments on a laptop computer with an Intel i7-8550U 1.80 GHz 4 core processor with 8MB cache and 16GB RAM.

Specifically, the flow map is computed in the original unstructured datasets and their rectilinear approximation. The Delta Wing dataset is known to exhibit a *vortex breakdown* phenomenon [52] characterized by the presence of *recirculation bubbles* on each of the primary vortices. Starting from a dense set of initial positions distributed over an axis-aligned box around the wing, trajectories are integrated over space and time using the Dormand-Prince 4/5 Runge-Kutta scheme [53]. Practically, the numerical integration was performed over 10 time steps for a corresponding integration time of 0.1s of the simulation, during which flow particles seeded at the front of the wing are advected past the unstable portion of the primary vortex. Figure 3.12 shows a slice of the flow map corresponding to the Delta Wing dataset. Using the computed flow map, we calculate the corresponding finite-time Lyapunov

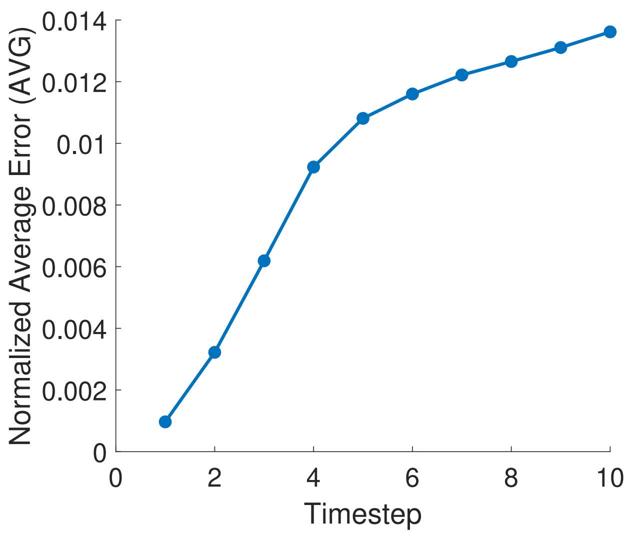


Figure 3.16. The progression of the normalized mean difference between the flow map for the unstructured grid of Delta Wing and the rectilinear approximation.

exponent [30] (FTLE), which measures the separation rate of neighboring particles and the flow. The significance of FTLE in the analysis of fluid flows stems from its ability to reveal so-called Lagrangian coherent structures (LCS), which are surfaces that act as material boundaries and control the behavior of the flow. Practically, LCS can be identified as ridge surfaces of FTLE, whereby ridges correspond to extremal surfaces that form the skeleton of the FTLE field. Their characterization involves the first and second-order derivative of FTLE. Figure 3.13 compares a cross-section of the FTLE field that corresponds to the cross-section of the flow map shown in Figure 3.12. As can be seen in the closeup views on the left, the main flow structures (primary, secondary, and tertiary vortices and boundary layer attachment below and above the wing) are accurately reproduced by the proposed data approximation method. Figure 3.14 offers a three-dimensional view of the FTLE field by volume rendering, and Figure 3.15 shows the volume rendering of LCS in the flow map. Essential features such as the recirculation bubbles and the vortex boundaries are captured in the approximation dataset as clearly as the original.

Figure 3.16 plots the progression of the normalized flow map approximation error as a function of the integration time. As expected, the error increases as the integration pro-

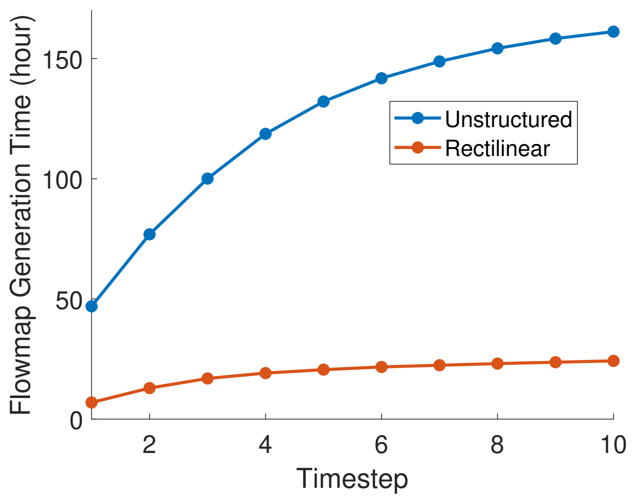


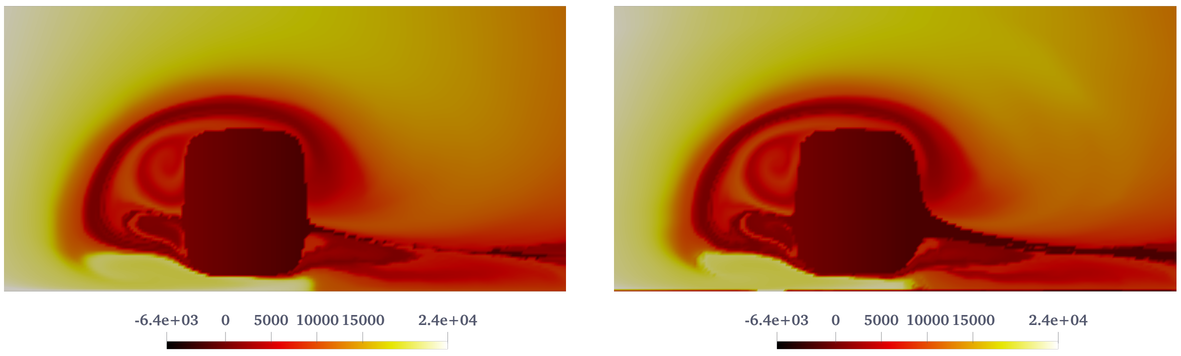
Figure 3.17. The cumulative time required to generate the flow map on the original unstructured Delta Wing grid and the rectilinear grid.

gresses. Interestingly, however, the slope of the error evolution decreases noticeably after 6-time steps, down to 0.06% per time step. Another important observation is that the considered integration length (10 time steps, 0.1s) amounts to the longest sensible advection time in this dataset since the flow transports tracers from the wing close to the domain outer boundary during that time. As such, the maximum measured error of 1.4% can be seen as a rather satisfactory upper bound.

A timing comparison for the generation of flow maps between the original unstructured grid and the rectilinear approximation is shown in Figure 3.17. The total integration time for 10-time steps for the unstructured grid is 161.2 CPU hours, compared to 24.2 CPU hours with our rectilinear grid approximation, achieving a 6.7× speedup.

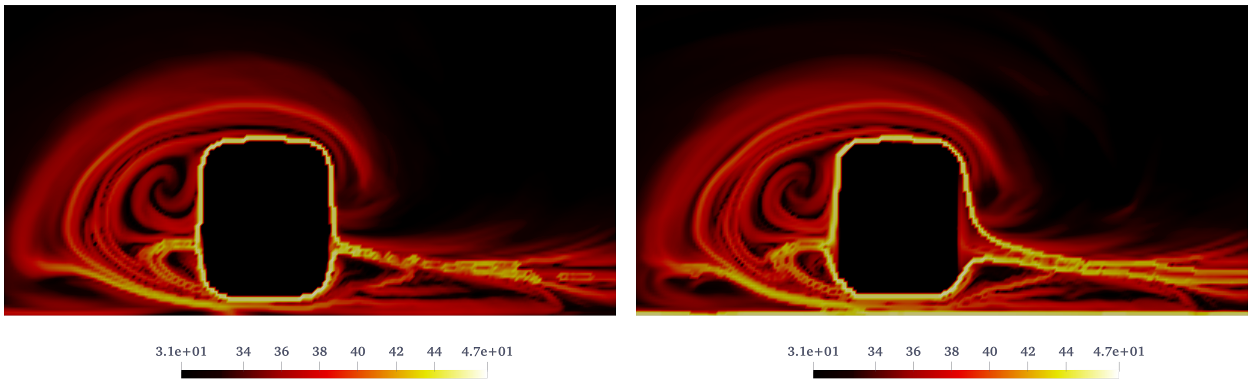
Generating the rectilinear approximation for the 10 time steps took 19.3 minutes. Our rectilinear approximation significantly reduced computational time while preserving the salient features in the flow map visualization.

For the ICE Train dataset, we compare our rectilinear approximation with the original unstructured dataset using the same visualization tools that we applied to the Delta Wing dataset, namely the visualization of the flow map and the corresponding FTLE field. Though the ICE Train dataset is not a time-dependent dataset (it corresponds to a Reynolds-averaged numerical simulation), we can use the flow map to study the flow structure and its



(a) The original unstructured grid. (b) Our approximation.

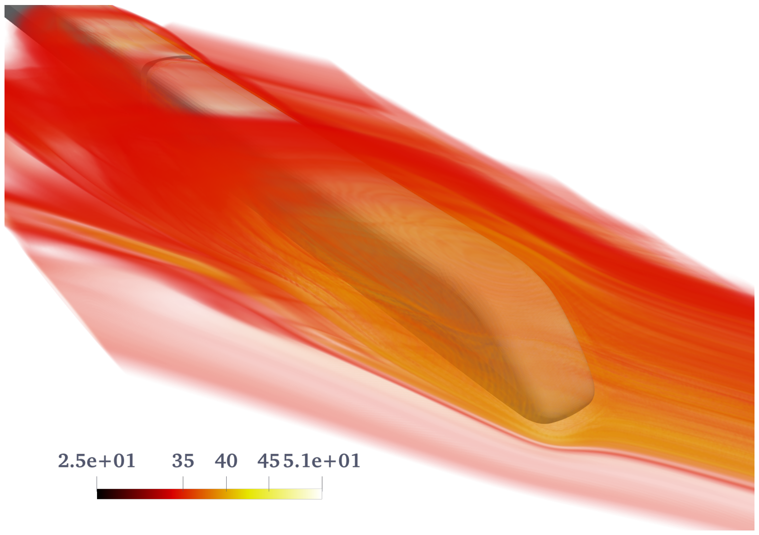
Figure 3.18. Slice of the flow map generated by forwarding integration for the original ICE unstructured grid and our approximation.



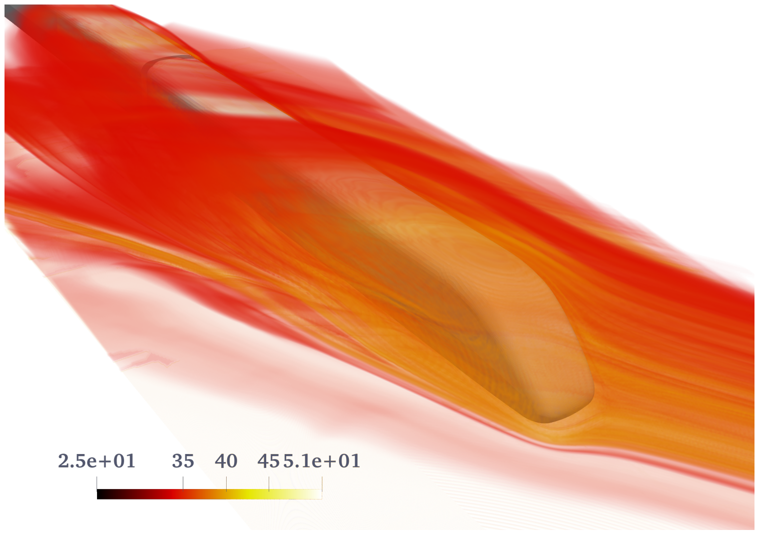
(a) The original unstructured grid. (b) Our approximation.

Figure 3.19. Slice of the FTLE generated from the forward integration of the flow map for the original ICE unstructured data and our approximation.

interaction with the train's body. In particular, we are interested in the vortices known to form on the lee side of the train. These vortices are associated with regions of low pressure that can affect the lateral stability of the train. Figure 3.18 shows a slice of the resulting flow map. The FTLE field calculated from the flow map reveals the presence of several vortices, as expected. Figures 3.19 and 3.20 show a slice of the FTLE field and a volume rendering, respectively, corresponding to the original unstructured dataset and our approximation. While the resulting flow map and FTLE of our approximations capture the major vortex structures of interest, there are visible inaccuracies around and leading from the boundary of the train.



(a) The original unstructured dataset.



(b) Our approximation.

Figure 3.20. Volume rendering of the FTLE generated from the forward integration of the flow map for the original ICE unstructured data and our approximation.

Table 3.1. AVG, RMS, NME, and approximation time

Data	Num. Unstr. Points	Approx. Resolution	Num. Grid Points	RMS	AVG	NME	Solving Time
Delta Wing	3.0M	$30 \times 31 \times 28$	26K	5.7×10^{-2}	3.4×10^{-2}	9.3×10^{-1}	18.2s
		$44 \times 48 \times 39$	82K	4.9×10^{-2}	2.8×10^{-2}	6.0×10^{-1}	28.2s
		$66 \times 76 \times 56$	281K	4.2×10^{-2}	2.1×10^{-2}	5.5×10^{-1}	35.0s
		$108 \times 130 \times 73$	1.02M	3.4×10^{-2}	1.5×10^{-2}	5.1×10^{-1}	118s
Fish Tank	20M	$57 \times 63 \times 57$	205K	4.6×10^{-2}	2.2×10^{-2}	6.4×10^{-1}	180s
		$101 \times 120 \times 71$	860K	3.5×10^{-2}	1.9×10^{-2}	5.7×10^{-1}	210s
		$146 \times 186 \times 72$	1.96M	3.0×10^{-2}	1.7×10^{-2}	5.5×10^{-1}	280s
ICE Train	1.1M	$16 \times 41 \times 19$	59K	2.0×10^{-2}	1.5×10^{-2}	3.9×10^{-1}	11s
		$142 \times 49 \times 30$	208K	1.4×10^{-2}	6.6×10^{-3}	3.8×10^{-1}	41s
		$170 \times 66 \times 54$	606K	8.9×10^{-3}	3.5×10^{-3}	3.2×10^{-1}	176s

Such inaccuracies can also be seen in the Delta Wing dataset close to the boundary of the wing. Those inaccuracies directly result from the geometric limitation of rectilinear grids compared to the flexibility of unstructured grids for defining arbitrary boundary shapes. As described in the conclusion section later, one of our ongoing researches is to capture these types of boundaries more accurately while maintaining the advantages of using rectilinear grids such that these boundary inaccuracies are reduced.

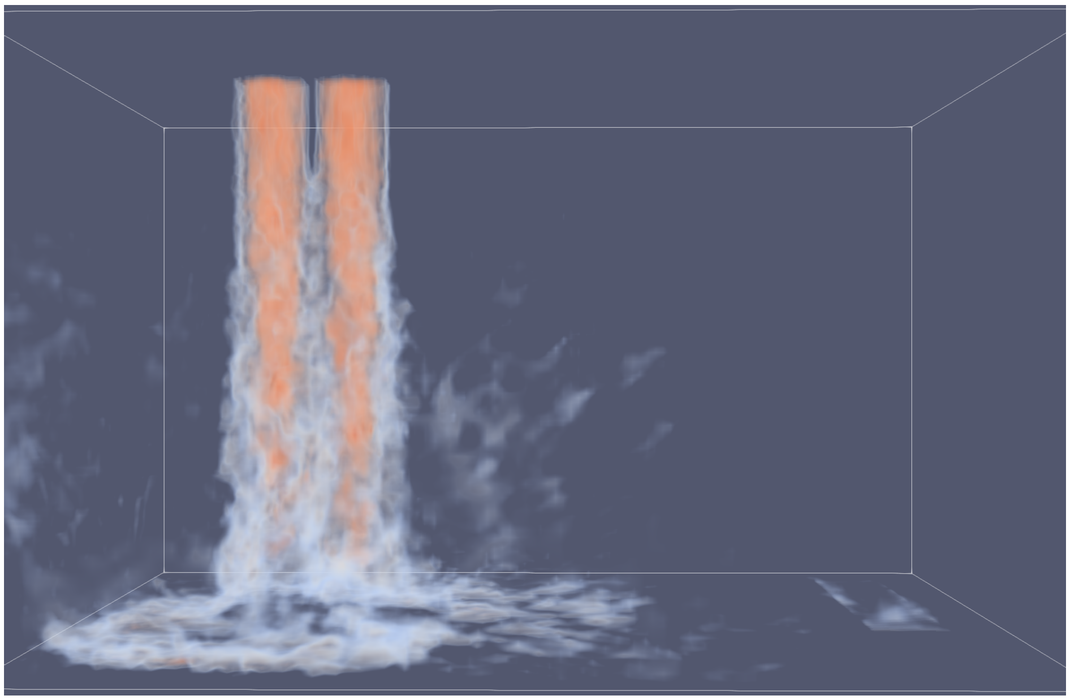
The approximation of the ICE Train took 176 seconds. The calculation of the flow map over the unstructured grid took 171.32 CPU hours. In comparison, the flow map generation over our rectilinear approximation took 31.04 CPU hours, which is a speedup of 5.5× compared to the unstructured flow map calculation.

For the Fish Tank dataset, we compare the volume rendering result of the original unstructured grid and the rectilinear approximation in Figure 3.21. As can be seen, despite the highly turbulent nature of the flow, the results are virtually indistinguishable.

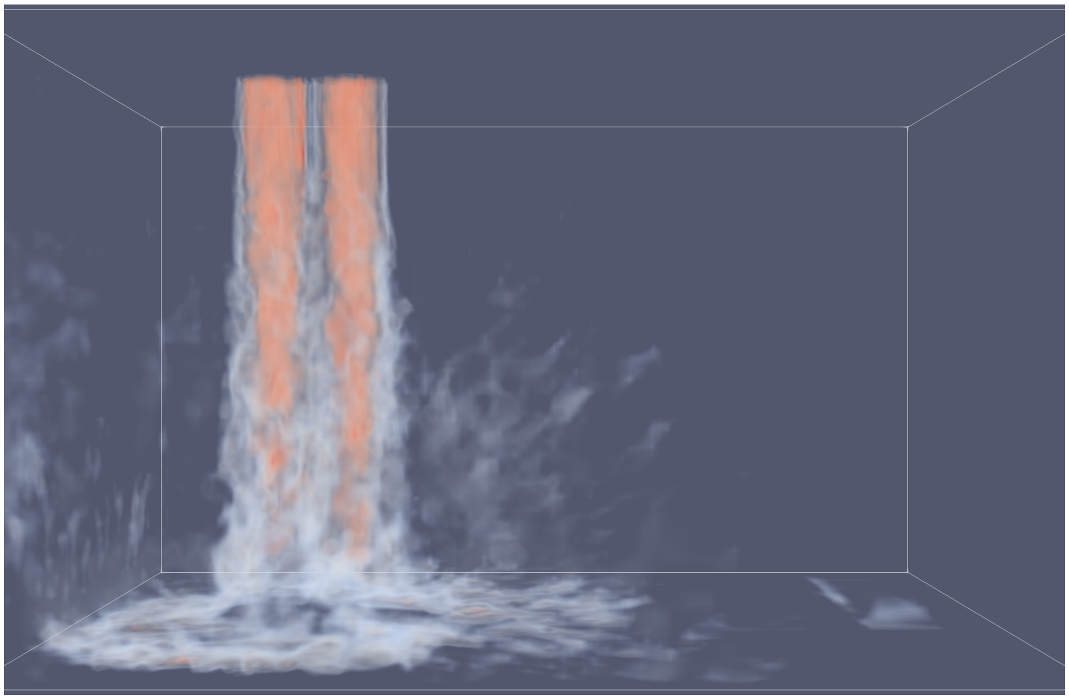
Quantitative Evaluation

In this subsection, we evaluate the accuracy of our approximation. Table 3.1 summarizes the resulting approximation error of the 3D test datasets. Figure 3.22 shows the decreasing trend of RMS error of the Delta Wing dataset with the increase in approximation resolution.

Our approximation achieves low RMS and AVG errors. While the normalized max error (NME) of our approximation tends to be high for these datasets, all the highest error values



(a) The unstructured original dataset.



(b) Our approximation.

Figure 3.21. Volume rendering for the unstructured Fish Tank dataset and our approximation.

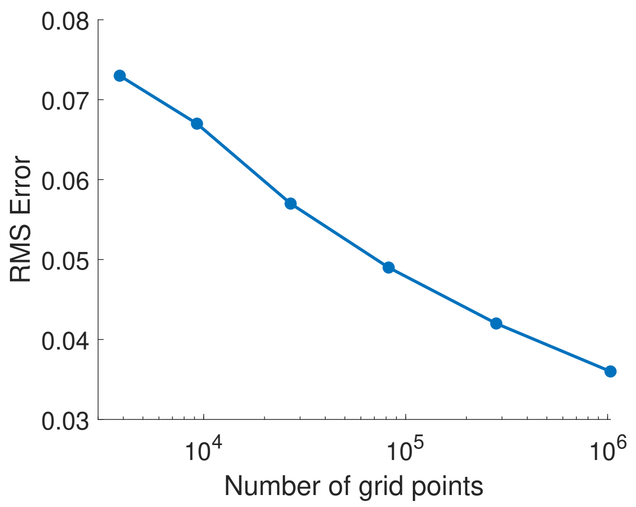
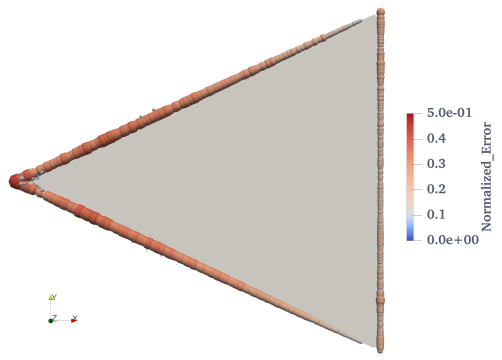
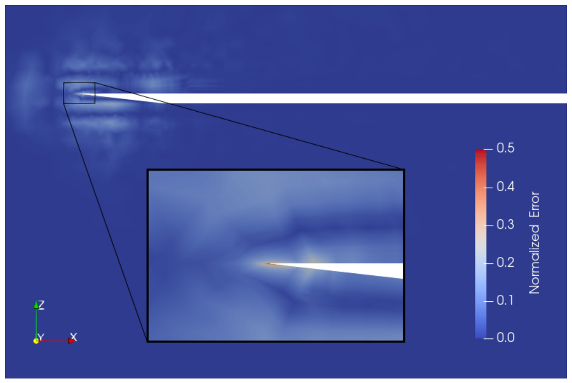


Figure 3.22. The RMS error decreases with the increasing number of grid points used in the approximation for the Delta wing dataset.



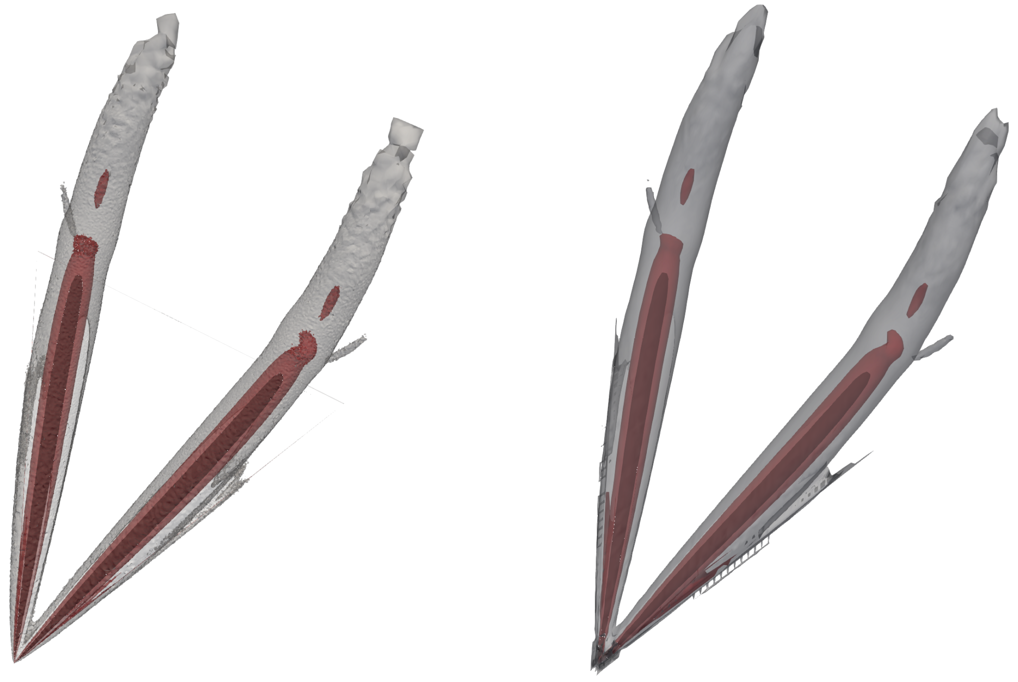
(a) Points with normalized error above 0.1. The size of the points are scaled by its error for visibility.



(b) A vertical planar slice across the location of the highest error.

Figure 3.23. Two views for the max error in the Delta Wing approximation, which occurs at the sharp corners of the wing and is very localized.

occur near the boundaries of the unstructured grid due to the data values changing rapidly to adhere to the boundary conditions. For the Delta Wing dataset, the highest error occurs at the corner of the wing where the pressure value changes dramatically from above to below the wing and where the mesh boundary condition at the wing causes a rapid change in the velocity vector. Cells at the wing boundary are tiny and flat, making these boundary values challenging to capture in the approximation with the enforced smoothness of the rectilinear grid without requiring prohibitively high resolution. Figure 3.23 shows at the left the location



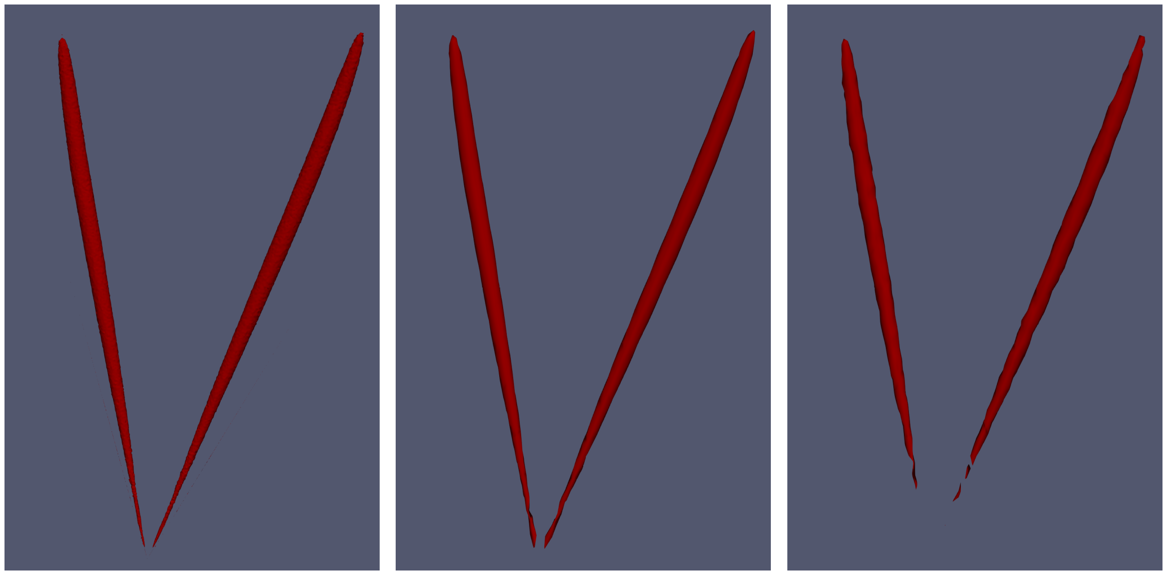
(a) The original unstructured grid. (b) The approximation.

Figure 3.24. Isosurfaces with different iso-values of the vorticity in the Delta Wing dataset and the approximation.

of all the points with the normalized error above 0.1, which constitutes 2.8% of the total points in the dataset. The error value scales the size of each point for better visibility. As the Figure shows, high error points are confined to the sharp corner of the wing. Figure 3.23 shows a planar slice of the Delta Wing approximation error at the right, cutting through the point with the highest normalized error (0.51). As can be seen, the error stays in the thin cell and is very localized.

Similarly, the Fish Tank dataset has the highest error occurring at the mesh boundary where the tank and the pipe boundaries are defined. The boundary conditions cause the data values to decrease to zero sharply.

However, these localized high errors do not noticeably affect the results of the visualization, and they tend to correspond to a smoothing effect. Figures 3.24 compare 3 isosurfaces of the vorticity between the original unstructured grid and the approximation. The points with extreme values that cause significant approximation errors can be seen in the isosur-



(a) The original unstructured grid. (b) The approximation. (c) Naive interpolation

Figure 3.25. Three Isosurfaces of the vorticity in the Delta Wing dataset using original data, our approximation, and naive interpolation.

face constructed for the original unstructured grid (left) as the faint line that delineates the triangular shape of the Delta Wing. It is not present in the isosurface generated from the rectilinear approximation (right). Refer to Subsection 3.8.2 for additional comparisons.

The naive approach to sampling onto a grid is to use the interpolation of the unstructured mesh to determine values at the vertices of the rectilinear grid. Using the same rectilinear grid, this naive approach results in a higher error and noticeably worse visualization results when compared to our solution. We use MATLAB's `griddata` function to interpolate data onto a Cartesian grid linearly. We try this on the Delta Wing dataset and compare interpolating grid values with our least-square approximation, using the same grid generated using our octree method. RMS error of the interpolated grid is 0.0036, while our solution gives an RMS error of 0.0013. We also compare the isosurface visualization in Figure 3.25, where the isosurface generated from naive interpolation of the grid (right) fails to capture the thin vortices near the front of the wing due to the loss of detail in the interpolation process. In contrast, our method (middle) successfully captures those thin vortices. In summary, our

Table 3.2. Performance of cell location in the unstructured grid and the rectilinear grid for 10M query points.

Dataset	Unstructured Search Time	Cell-tree Build Time	Total Unstructured Search Time	Rectilinear Grid Search Time	Unstr. Avg. Query Time per Point	Rect. Avg. Query Time per Point
Delta Wing	27.2s	7.3s	34.5s	9.7s	3.5×10^{-6}	9.7×10^{-7}
Fish Tank	18.1s	10.6s	28.7s	15.2s	2.9×10^{-6}	1.5×10^{-6}
ICE Train	30.9s	1.6s	32.5s	13.7s	3.3×10^{-5}	1.4×10^{-5}

approximation method outperforms naive interpolation in terms of approximation error and visualization results.

Performance of Cell Location

Many visualization algorithms and data analysis techniques require interpolation at arbitrary locations within the dataset, including determining the query location's cell. As unstructured grid cells can be arbitrary, cell location constitutes a bottleneck for visualization algorithms, despite using dedicated data structures to reduce search time.

This subsection presents a performance comparison of cell location operations between the original unstructured grid and our rectilinear grid approximation. We utilize the Cell-tree data structure [16] available in VTK [54]. The cell-tee data structure is a specialized boundary volume hierarchy tree structure for the cell location of the unstructured grid. It is arguably the state-of-the-art in cell location in terms of balancing performance and memory footprint.

We select random points in the dataset domain as query points. The unstructured grid distribution determines the random query points distribution, so more query points are selected in regions with higher point density. We then measure the time to interpolate the value at the point in the two grids. All computations use VTK's available data structures and data representation. A Cell-tree is used in the unstructured case, whereas no additional data structure is needed for interpolating in the approximation grid. You can find the performance summary for the three datasets in Table 3.2. We used the highest approximation resolution listed in Table 3.1 for the respective datasets. The cell-tree data structure is built once for each dataset in a preprocessing step for the unstructured grid case. As shown in the Table, query time using our rectilinear grid is less than that of the unstructured grid. For 1M points, taking into account the building time of the cell-tree, the Delta Wing dataset has a 3.6× speedup, the ICE Train dataset has a 2.4× speed up, and the Fish Tank dataset has a 1.9× speedup. The Delta Wing and the ICE Train datasets see a higher speedup due to their highly uneven point distribution.

Table 3.3. File storage reduction from the unstructured grid to a rectilinear grid.

Dataset	**Unstructured File size**	**Rectilinear File Size**	**Reduction**	**Resolution**	**RMS**
Delta Wing	425 MB	814 KB	534 : 1	$30 \times 31 \times 28$	5.7×10^{-2}
		2.5 MB	170 : 1	$44 \times 48 \times 39$	4.9×10^{-2}
		8.6 MB	50 : 1	$66 \times 76 \times 56$	4.2×10^{-2}
		31.5 MB	14 : 1	$108 \times 130 \times 73$	3.4×10^{-2}
Fish Tank	1.33 GB	4.7 MB	291 : 1	$57 \times 63 \times 57$	4.6×10^{-2}
		19.6 MB	69 : 1	$101 \times 120 \times 71$	3.5×10^{-2}
		44.7 MB	31 : 1	$146 \times 186 \times 72$	3.0×10^{-2}
ICE Train	148MB	1.8 MB	105 : 1	$76 \times 41 \times 19$	2.0×10^{-2}
		6.3 MB	23 : 1	$142 \times 49 \times 30$	1.4×10^{-2}
		18.0MB	8 : 1	$170 \times 66 \times 54$	8.9×10^{-3}

Storage Reduction

Storage reduction indicates storage space saved by using our approximation compared to the original input unstructured dataset. Unstructured grids need to store vertex coordinates and cell connectivity information explicitly. A large part of data reduction for rectilinear grids comes from the implicit nature of both vertex locations and cell definitions. Since a rectilinear grid is constructed as a Cartesian product in 2D/3D space, the only geometric information needed is the grid coordinates for each spatial dimension. That reduces the bulk of the storage for unstructured grids. In addition to the geometric information, we must store the values associated with each vertex. The footprint of that information in the approximation depends on the chosen resolution, which in our experiments never amounted to more data points than that of the original unstructured mesh.

We use the Visualization Toolkit (VTK) XML file format [54] for storage space comparisons. We used the same compression method in each case. Table 3.3 summarizes the file size comparison of the three datasets in the original unstructured grid and various resolution rectilinear approximation. The highest storage reduction we obtain for the Delta Wing dataset is 534$\times$ with an RMS error of 5.7×10^{-2}. For the Fish Tank dataset, we achieve 291$\times$ storage reduction with an RMS error of 4.6×10^{-2}. Similarly, for the ICE Train dataset, we achieve 105$\times$ storage reduction with an RMS error of 2.0×10^{-2}.

More generally, assuming that an unstructured grid contains N vertices and $3N$ cells, and assuming an equal resolution distribution across all 3 dimensions of the approximation grid, approximation using the same number of data points amounts to a data reduction of order $5N^{2/3}$. This memory reduction can be seen as a means of lossy compression for large scientific datasets.

3.9 Conclusion and Future Work

In this chapter, we introduced a framework for approximating unstructured data with a non-uniform rectilinear grid. We first proposed a hybrid octree refinement method to obtain an efficient rectilinear grid for approximation. We need the grid to be fine enough to capture the data features at locations where the data points are dense and vary in input data values. Refinement happens when we have dense scattered points with high variation in their functional value. The results show that with our hybrid method, we obtained a smaller approximation grid with the high resolution needed to capture the features of the original data without unnecessary refinement elsewhere. A smaller grid means a smaller system to solve, less storage cost, and faster interpolation. Additional criteria such as maximum octree depth can be used to achieve more fine-grained control of the octree refinement, thereby limiting the resolution of the octree.

Given the rectilinear grid obtained using the hybrid octree method, we aim to find the approximation values corresponding to the rectilinear grid that minimizes the L2-norm of the difference between the input point values and the interpolation value obtained from the approximation. This fitting is solved with a linear least-square system. Unfortunately, the resulting fitting system has two major problems; first, the system is often rank-deficient, causing artifacts in the global solution. Second, the system can be too large concerning RAM usage for direct solvers to handle. To overcome the first problem, we introduced smoothing/regularization to the fitting system where additional constraints are added. Applying the same smoothing weights to all parts in the approximation system can yield a low approximation quality due to over-smoothing the well-constrained part of the system and/or under-smoothing for the under-constrained part of the system. We introduced our bi-level smoothing scheme, which provides adequate regularization by setting the local regularization weight using a heuristic method based on local point distribution. Our bi-level smoothing avoids the common problems of over-smoothing and under-smoothing by setting just the right amount of smoothing that makes the system well-constrained while maintaining low approximation error. To solve the system's second problem, which is too significant and causes a shortage of RAM when using direct solvers, we use a block method to split the system into blocks with an equal number of unknowns and solve them in parallel. Each block is padded with extra cells (ghost cells) to maintain continuity between solutions.

We showed that our approximation was about 6 times faster in generating flow maps than the original unstructured data, with a reduction in the approximation file size that is a minimum of 8 times less than the original unstructured data file size. Together, those techniques with the right choice of the adaptive grid offer a means to construct high-quality approximations of large and complex simulation datasets at a fraction of their original storage cost and significantly increased performance for visualization algorithms that rely heavily on interpolation.

4. BOUNDARY-AWARE RECTILINEAR GRID: RECTILINEAR GRID WITH SOLID BOUNDARY HANDLING CAPABILITIES

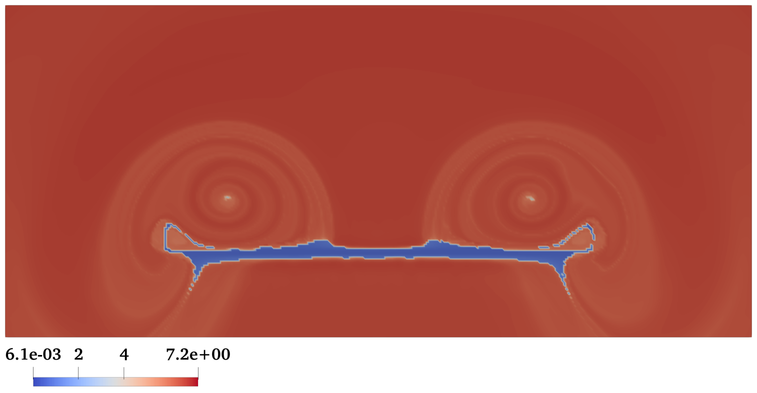
4.1 Introduction

The previous chapter presented a framework for accurately approximating unstructured datasets over rectilinear grids using linear approximation kernel. In this chapter, we expand upon the basic framework of the previous chapter by exploring the boundary handling capabilities of rectilinear grids using high-order B-spline kernels.

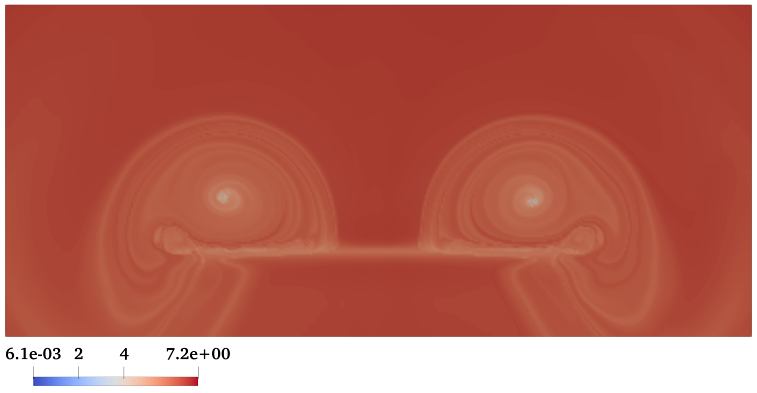
Many post-processing applications would benefit from using a higher-order approximation kernel because of the smooth approximation and efficiency in calculating high-order derivatives. These applications include edge detection [55], topological segmentation [56], curvature computation [57], ridge and valley extraction [9], and correct illumination for volume rendering [3] are all applications that use derivatives for their calculation and would benefit from a smooth approximation.

In Section 3.8, We discussed the inaccuracies of interpolation close to the boundary when visualizing the FTLE. The inaccuracies are intrinsic to the structured nature of rectilinear grids. Higher-order approximation kernels exacerbate those inaccuracies. This is demonstrated in Figure 4.1 where we compare linear kernel with cubic kernel in terms of flow map visualization used in Chapter 3. Though the linear approximation lacks smoothness and produces inaccurate results around the boundary, the cubic B-spline approximation of the flow map blends the solid boundary with the rest of the domain. This happens due to the cubic B-spline kernel spanning multiple grid cells, the thin body of the Delta Wing compared to the grid resolution, and the lack of boundary-handling capability of the rectilinear grid.

The most interesting part of studying in the simulation is where the flow patterns are formed (*i.e.*, vortices), which mostly happens close to the solid boundary. This motivates the development of *boundary-aware* rectilinear grids, which are rectilinear grids with the capability to approximate arbitrary boundaries and handle near-boundary interpolations.



(a) Approximation using linear kernel



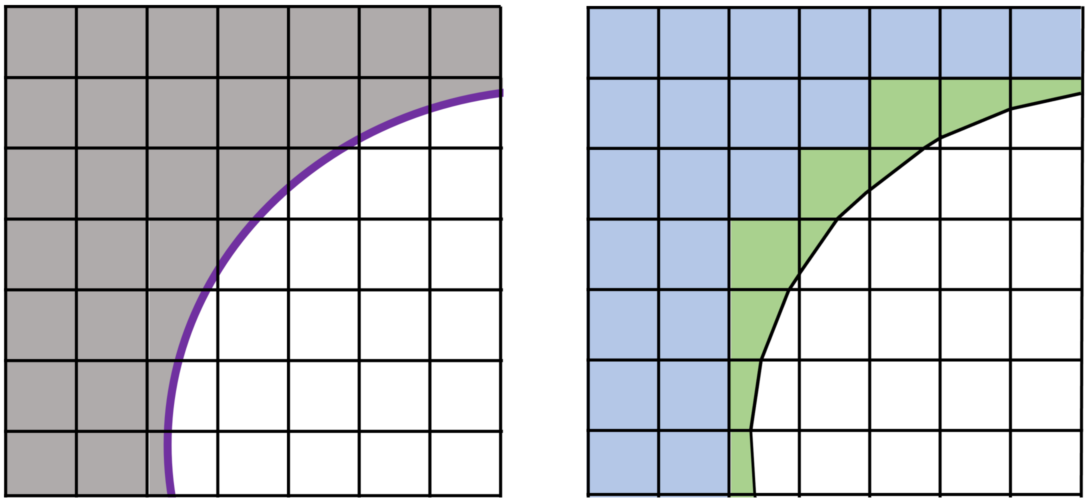
(b) Approximation using cubic B-spline kernel

Figure 4.1. Comparison between the flow map for the Delta Wing dataset approximation using kernels of different orders. The inaccuracy in the interpolation close to solid boundary is more apparent when using high-order kernels.

We first demonstrate in this chapter that the approximation method presented in the previous chapter can be extended to high-order kernels, specifically to cubic B-spline kernels. We then tackle the challenge of representing solid boundaries using rectilinear grids.

To achieve the boundary-awareness, we propose a grid-based boundary approximation. We use cut-cells where each grid cell has a boundary approximation. A 2D example of cut-cells can be seen in Figure 4.2, in which the cut-cells are shown in green. At query time, the boundary approximation in these cut-cells determines the interpolation behavior near the boundaries. The approximated grid is endowed with pre-calculated additional data arrays (*i.e.*, the intersection of the boundary with the approximated grid). The additional data array is any information necessary to approximate the solid boundary at query time. Marching cube aids in generating the boundary approximation at query time efficiently.

The evaluation shows that the proposed solution accommodates challenging boundaries while supporting high-order reconstruction kernels with a much-reduced memory footprint. As such, our data representation enjoys all the benefits of conventional rectilinear grids while addressing their fundamental geometric limitations.



(a) The rectilinear grid cut by the solid boundary (in purple), grey is for the area inside the domain.

(b) The cut-cells (in green) resulting from the solid boundary cutting the domain, blue is for pure-cells inside the domain.

Figure 4.2. The rectilinear grid is cut by the solid boundary of the cylinder and the resulting cut-cells.

4.2 Related Work

B-spline approximation has been widely used for design purposes in many computer graphics and industrial applications. A high order B-spline can be used to approximate datasets defined over unstructured grids and point clouds with a guarantee of smoothness [58]. Peterka *et al.* [59] proposed using tensor-product B-splines to represent scientific data as a means of easier post-processing and storage saving. Their work is limited to approximating structured data and does not consider unstructured data. Yeh proposed an efficient adaptive knot placement method for arbitrary order B-spline data approximation [60]. Their method estimates the high order derivative of the input data, which does not handle datasets with arbitrary boundaries well due to the numerical instability in the derivative estimation.El-Rushaidat *et al.* proposed an accurate approximation of the unstructured data using a customized regularized smoothing least-square. Still, their work was limited to linear kernels and did not handle the representation of complex boundaries [61].

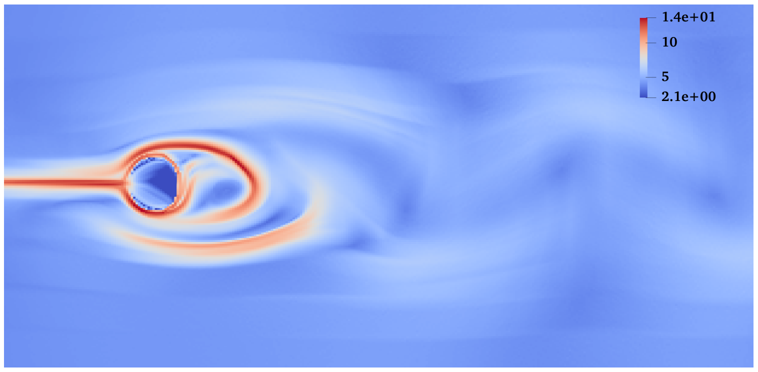
In the computational fluid dynamic community, *cut-cells* are widely utilized for representing flexible boundary representation within the framework of structured grids. A cut-cell is a cuboid cell in a Cartesian grid that intersects with the geometry of some solid boundary. The computational science community applied the concept of cut-cell for meshing [62], [63] and in simulations of viscous flows [64]–[66]. Bouchon *et al.* presented a second-order cut-cell method for these simulations [67]. Their method considers two-dimensional two-phase flows with a solid boundary embedded in the domain. The three-dimensional case was recently considered [68], whereby level sets are used to specify the geometry of the boundary, which is then approximated by a marching cubes triangulation to calculate the area and volume of the cut-cell.

Marching cubes is a computer graphics algorithm first proposed by Lorensen and Cline [69], [70] for generating a polygonal approximation of an isosurface from a three-dimensional scalar field defined over a voxel grid. The algorithm uses a look-up table to obtain a triangulation approximating the boundary, making it highly efficient. A 2D example of cut-cells can be seen in Figure 4.2, in which the cut-cells are shown in green.

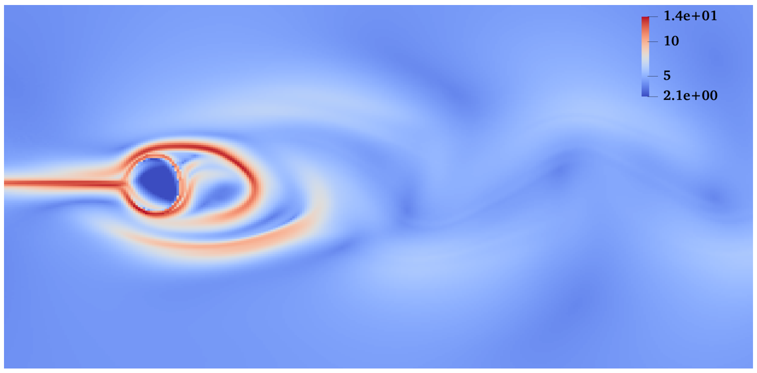
4.3 Integrating B-spline as an Approximation Kernel

In the previous chapter, we used only a linear kernel for our approximation, equivalent to using degree 1 B-splines. However, the resulting trilinear kernel suffers from a lack of smoothness. In this chapter, we extend our reconstruction kernel to the cubic B-spline. The choice of B-spline provides a highly flexible grid-based kernel due to the knot definition; the rectilinear grid defines the unique knots for the B-spline kernel.

Using a cubic B-spline as a reconstruction kernel significantly improves the smoothness of our approximation over to the linear case. This is clearly shown in Figure 4.3, which compares the finite-time Lyapunov exponent (FTLE) calculated using the flow past the cylinder dataset approximation using linear kernel vs. cubic B-spline.



(a) Approximation using linear kernel



(b) Approximation using cubic B-spline kernel

Figure 4.3. Finite-time Lyapunov exponent (FTLE) for the flow about a cylinder dataset approximation with different degrees B-spline kernels. The higher-order approximation results in smoother visualization.

4.4 Boundary-Aware Approximation

The approximation of the field values of datasets defined over unstructured grids through a rectilinear grid exhibits poor accuracy at locations close to the boundary. The limited cell structure of the rectilinear grid makes it hard to represent the boundary's geometry accurately. This is why we propose a rectilinear grid with a different cell structure that offers a good geometric approximation of the boundary surface. When designing an interpolation scheme applicable to a cut-cell, the following properties should be satisfied. First, the interpolation should yield a low approximation error. It should smoothly connect to the reconstruction scheme of the surrounding cells, ideally up to the same degree. In addition, the cut-cell reconstruction should possess other application-specific properties, such as no-slip boundary conditions, whereby the velocity value at the boundary is equal to zero. The decrease of the velocity of the flow to zero occurs in a thin layer adjoining the boundary. This is called the boundary layer, and it is characterized by the presence of considerable velocity gradients.

Within each cut-cell, boundaries are approximated using the marching cubes algorithm. Similar to the original marching cubes algorithm, which labels each grid point with respect to a given isovalue, in this implementation, we label each grid point with a binary label that indicates whether the vertex is inside or outside of the solid boundary domain.

Similar to the trade-off between using the original CFD simulation and the rectilinear approximation. The small error associated with the approximation that does not affect the flow structures and the overall flow behavior can be neglected with the performance improvement, and the storage-saving achieved. The adaptive rectilinear grid was designed using the hybrid method s high resolution close to the solid boundary. The same applies to the use of boundary approximation instead of the original geometry. The use of the grid-based boundary approximation has performance improvement and storage-saving that allows neglecting the small loss of accuracy in the geometry representation. The loss in accuracy in representing the boundary geometry will be further minimized. This grid-based boundary approximation also takes advantage of the fast access and reduced storage that comes with rectilinear grids.

The triangulation specified by the marching cube triangulation lookup table is combined with points along the edges of the grid cell to form the surface. In the marching cube algorithm, these points along the edges are where the chosen iso-value is linearly interpolated by the grid points connecting the edge. In our case, as each grid point is merely a binary label indicating whether the grid point is inside the domain or out, we acquire these vertices along the cut-cell edge by finding the intersection of each edge with the solid boundaries. In a pre-processing step, each triangle in the boundary mesh is tested against the rectilinear

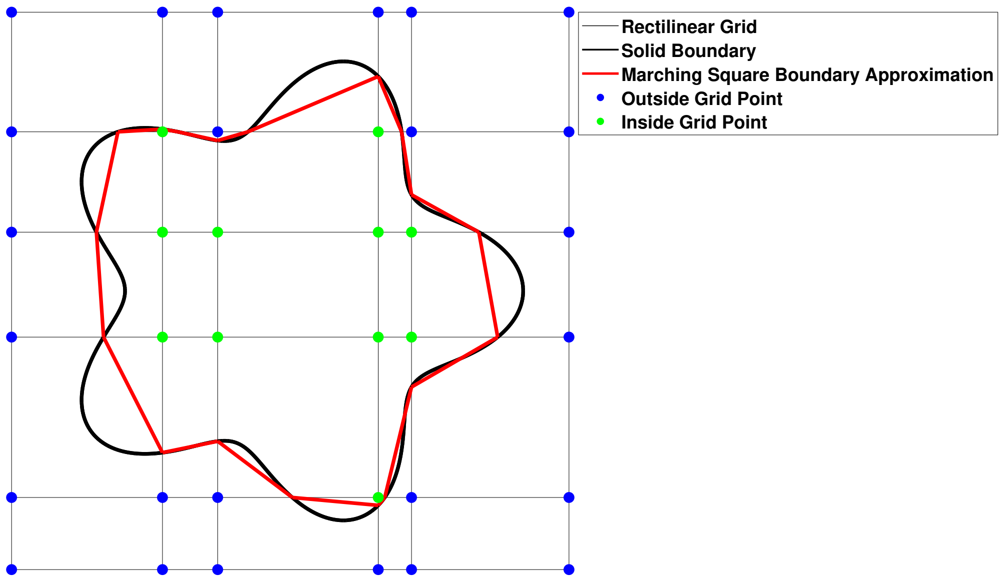


Figure 4.4. Marching Square approximation (red line) of a solid boundary (black line). Each grid point is labeled inside (green) or outside (blue) of the boundary. These labels are used to generate the marching square algorithm look-up code.

grid cells edges, we make this efficient by checking only the grids in the bounding box of the specified triangle. All the intersections per edge are then recorded in the rectilinear grid. . The edge-intersection points and the grid point labels are exported with the rectilinear grid, along with the weights of the approximation kernel to complete the boundary-aware solution.

Figure 4.4 illustrates how a 2D marching squares works within our framework. The figure shows a solid boundary (black line). Each rectilinear grid point is labeled as inside or outside of the solid boundary. The figure shows that this approach yields a piecewise linear approximation of the actual boundary geometry (in red). The accuracy of this approximation depends on the geometric complexity of the body and the available rectilinear grid resolution used for the approximation. The resolution used is coarse to demonstrate the concept. In practice, the grid would be much more refined, and approximate the solid boundary more closely. This high resolution close to the boundary is guaranteed because of the hybrid method we use to generate the adaptive grid, where we refine if both the input points density and the variation between the input points field values are high, which is the exact case close to boundary.

The original ("naive") marching cubes algorithm uses a simple lookup table that suffers from ambiguity. This ambiguity is the result of the non uniqueness of the triangulation solution for the same MC code. Thus, using this table can result in cracks and inconsistent topology for the approximated isosurface. Improvements and optimization to the original MC algorithm can resolve the ambiguity problem. One of the efficient solutions is the Chernyaev's technique [71] that expands the MC table to require extra testing for the ambiguous cases to ensure a topologically correct result. Lewiner *et al.* [72] provide the full implementation for Chernyaev's technique. The ambiguity is resolved by expanding the lookup table to include subcases for the ambiguous cases along with different triangulations and corresponding additional tests (faces to test or additional internal points) needed to decide which triangulation result in a correct topology.

Though we use the basic MC algorithm for approximating our boundary in the cut-cell and the ambiguous cases can arise, the disambiguation solutions cannot be used in our implementation since we have no scalar values to test against. In contrast to the isosurface extraction setup, the original geometry is available in our boundary-aware method. Hence, if we encounter an ambiguous case during pre-processing we can always determine the correct triangulation based on the original geometry. If the grid is sufficiently refined around the boundary, ambiguous cases should be limited. If ambiguous cases occur repeatedly, this is an indication that the grid is not refined enough and additional refinement steps are needed.

Starting from the labeled cut-cell points, we propagate the labels until all rectilinear grid points are labeled. The propagation proceeds from the labeled cut-cell to its neighbors, the neighbor cell is partially labeled, the rest of the neighbor cell points are then labeled according to those partially labeled ones. This is repeated for the neighbors of the newly labeled cells until all grid cells are labeled. Figure 4.5 shows the ICE Train with the labeling of the rectilinear grid points. Grid points colored in red are outside the domain, and blue grid points are inside the domain.

After each grid point is labeled and the intersection is calculated on each grid cell edge, there may be cases where the marching cube approximation of the boundary does not capture

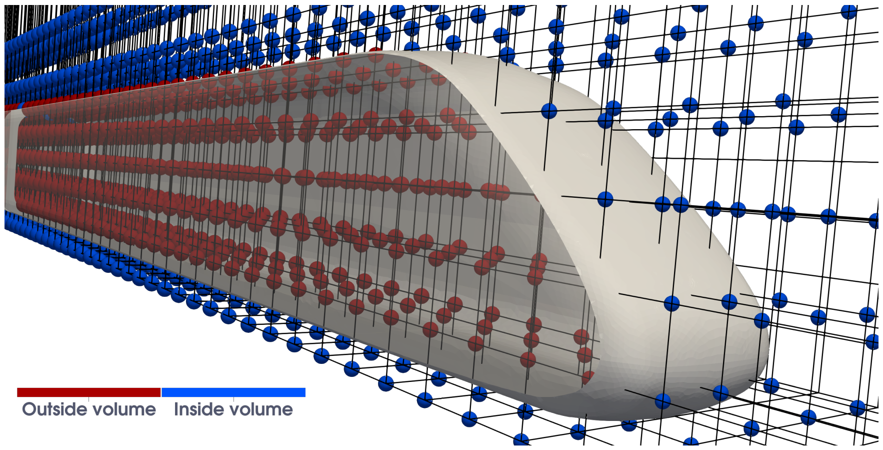
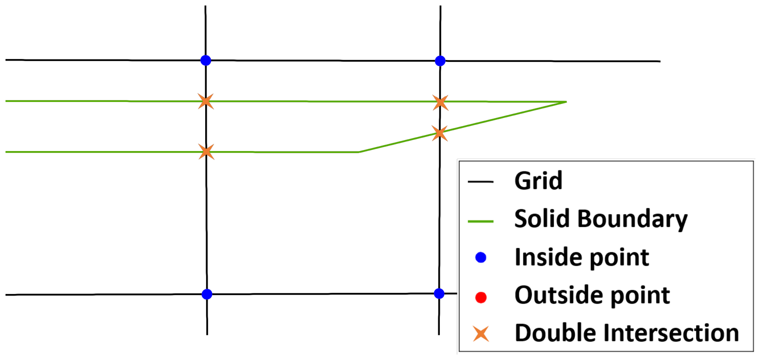


Figure 4.5. Rectilinear grid points labels for the ICE Train. Red points are outside the domain, and blue points are inside the domain. The train geometry is given for reference in grey, and the rectilinear grid wireframe in black.

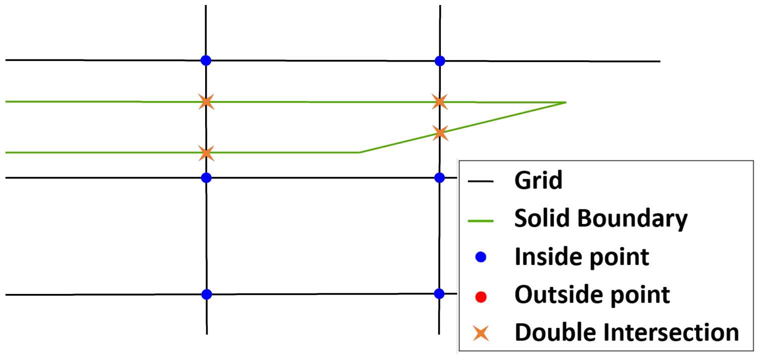
the topology of the input boundary. This can occur when the grid resolution is too coarse. We detect this by checking if a grid edge has multiple intersections. Those values are recorded in the step where we find the intersection of each edge with the solid boundaries. A simple check to the recorded value would indicate if the edge has multiple intersections. If it does, we further refine the grid by increasing the grid resolution along the dimension of the edges with multiple intersections until no edge has more than one point of intersection with the input boundary.

Figure 4.6 shows a 2D slice of the Delta Wing solid boundary and the rectilinear approximation grid. In this particular example, the double intersections at the vertical edges indicate that the grid is not fine enough to capture the boundary geometry. If the refinement is not performed, we might miss part of the boundary. Thus the Y-dimension grid resolution is refined. A simple refinement procedure is followed. We recursively split all edges with multiple intersections with boundaries into equal halves and reevaluate the newly refined edges until no more edges with multiple intersections exist. As a result, the new grid can capture flat wing geometry.

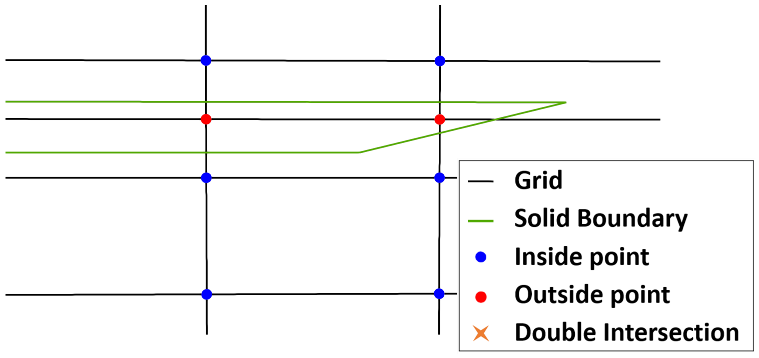
The boundary-aware rectilinear grid solution contains the approximation kernel weights, the inside/outside grid point label, and the relative intersections of the cell edges with the boundary. We should note that even though volume and boundary approximations are



(a) Before refinement



(b) After first refinement



(c) After second refinement

Figure 4.6. Zoom-in view on the rectilinear grid of Delta Wing before refinement and after two refinement steps based on the double intersection on a grid edge.

separate, we can easily save both results in one cohesive rectilinear grid with multiple data arrays, each corresponding to specific aspects of the solution.

4.5 Boundary-Aware Interpolation

This section explains the interpolation procedure in the boundary-aware rectilinear grid and how it depends on the nature of the cell that contains the query point. Given a query point, we first find the rectilinear grid cell that contains the point, which is a straightforward binary search, as previously explained in Section 2.1. At interpolation time, we determine if the enclosing grid cell is a cut-cell by checking whether the vertex labels of the grid cell vary.

The binary label is used to calculate the marching cubes look-up code into a table that stores the triangulation of the boundary within the cut-cell. After obtaining the marching cubes code, each cell can be categorized into one of three cases: entirely outside the domain, entirely inside the domain, and a cut-cell. Each query point is handled differently depending on the type of cell that includes it.

If the query point falls in a cell entirely inside the domain, the B-spline interpolation is performed normally. If the query point falls in a cell entirely outside the domain, the resulting interpolation should be a value appropriate for an out of domain location. Otherwise, if the query point falls in a cut-cell, special handling is needed.

The following should be performed when interpolating in a cut-cell; first, we construct an approximation of the solid boundary using marching cubes, an algorithm chosen for its simplicity and efficiency. Specifically, from the information exported from the boundary-aware rectilinear grid in Section 4.4, we calculate the marching lookup code from the grid cell vertex labels, retrieve the triangulation from the lookup table, and construct the triangles using the edge-intersection information also exported in Section 4.4.

For any query point that falls in a cut-cell, we first need to decide if the query point is inside or outside the domain. The test can be done using the odd-even rule [73], a test commonly used in computer graphics to determine if a point is inside or outside a polygon. We chose a cut-cell vertex with a given inside/outside label. Then we shoot a ray from

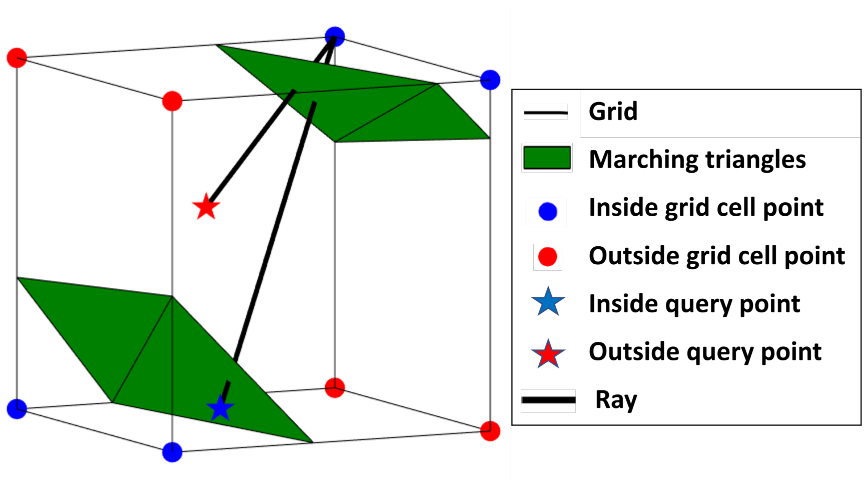


Figure 4.7. The odd-even rule is used to test whether a query point is inside/outside a cut-cell. Marching cube triangulations are green. The grid cell vertices are labeled (red for outside the domain and blue for inside the domain). Two example query points are shown. One with an odd number of intersections is labeled outside. The other query point is labeled inside with an even number of intersections.

the query point to that specified cell vertex. The number of intersections between the ray and the triangles is counted. We consider the query point to share the same in/out label with the selected cell vertex if the number is even. Otherwise, we consider the query point label to be the opposite of the selected cell vertex label. Figure 4.7 shows an example of the odd-even test in a cut-cell. The figure shows the marching cubes triangulation and two query points. A query point is considered inside if an even number of triangle intersections is found, outside if that number is odd.

The interpolation of a query point that is both in a cut-cell and inside the domain is performed in an arbitrary polygonal cell, instead of regular rectilinear grid cells (*i.e.*, rectangular cuboids). The arbitrary polygonal cell is constructed from the marching triangles and the cut-cell. this is illustrated in Figure 4.2, where the interpolation in the cut-cells (in green) is not performed in the rectangular-shaped cells.

CFD simulations, which are the main focus of this thesis, have a set of properties that an interpolation scheme should follow. First, the interpolation should be smooth inside the cut-cell, as well as a smooth transition across the cell bounds to maintain a certain level of continuity between cells. Second, the interpolation in the cut-cell should maintain the no-slip boundary condition [74], according to which, at a solid boundary, a viscous fluid will have zero velocity relative to the boundary.

A straightforward solution to interpolate in a cut-cell would be to interpolate using normal B-spline for points inside the domain, otherwise, perform no interpolation outside the domain. This solution is simple but has the drawback that there is no guarantee that the no-slip boundary condition is preserved. The solution would be either to find another interpolation scheme in the cut-cell that preserves the previously listed properties for interpolation close to the boundary or use the straightforward solution with special handling to maintain the no-slip boundary condition. We explored interpolation in the cut-cell using *Mean Value Coordinates* (MVC) [75]–[77]. Figure 4.8 shows an example interpolation using MVC. The smoothness is guaranteed inside each cell.

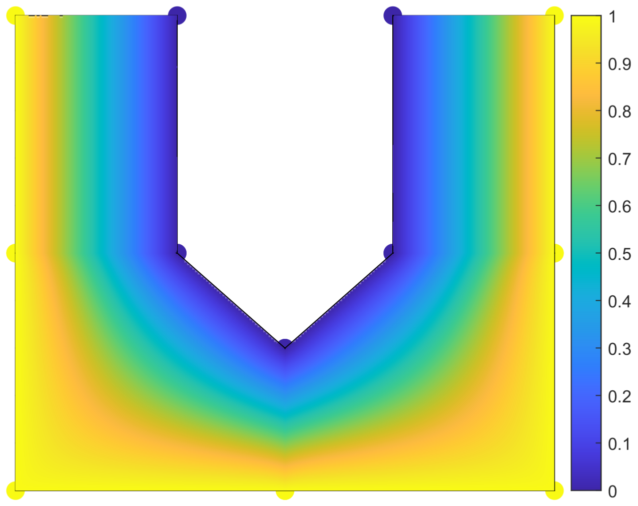
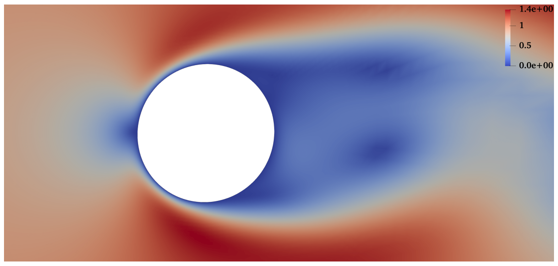
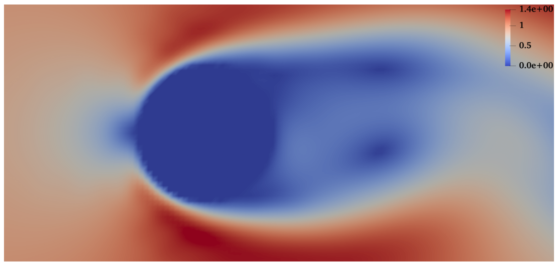


Figure 4.8. Example of MVC filter applied to the interpolating basis in 4 cut-cells. For the purpose of enforcing the no-slip boundary condition, the boundaries are set to be zero.

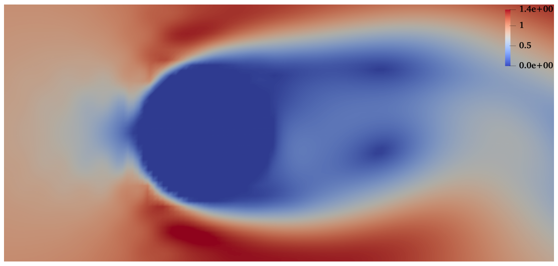
MVC is a particular instance of generalized barycentric coordinates [78] commonly used for interpolation in arbitrarily shaped polygonal cells. We considered MVC as an option for cut-cell interpolation because it provides smooth interpolation within the cut-cell, and guarantees a smooth transition to a zero interpolated value near the cut-cell boundary, this



(a) The original unstructured dataset.



(b) Reconstruction using straight forward interpolation



(c) Reconstruction using B-spline with MVC within a cut-cell.

Figure 4.9. Comparing B-spline cut-cell interpolation with and without MVC for the Flow About a Cylinder dataset.

can be seen in Figure 4.8, where the approximated boundaries resulting from the MC are set to be zero, thereby maintaining the no-slip condition along the solid boundary.

We integrate the MVC interpolation scheme inside a cut-cell by applying it as a filter over the B-spline basis in the interpolation Equation (2.7) from Section 2.4

$$\psi(\boldsymbol{x}) = C(\boldsymbol{x})M(\boldsymbol{x}) \tag{4.1}$$

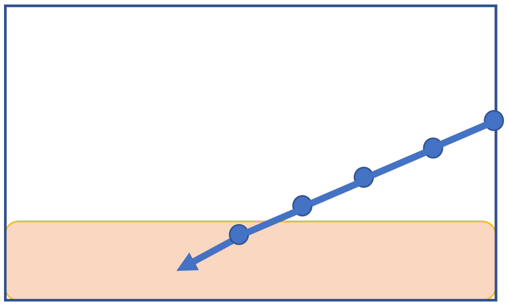
where C is the B-spline basis, and M is the MVC filter function in a cut-cell that transitions from 1 to 0 within the cut-cell, and 1 for a non-cut-cell otherwise, so as to leave the B-spline basis unchanged.

Our attempt, however, did not produce a desirable interpolation result. Figure 4.9 compares the result of using the MVC interpolation scheme versus without the added MVC filter. In Figure 4.9b, the straightforward solution with just B-splines interpolation produces a much smoother approximation compared to overlaying the interpolation kernel with MVC in Figure 4.9c. It turns out that integrating MVC in the approximation kernel causes inaccuracies at the boundary of the cell, due to the discontinuity between the cells interpolated using MVC and the neighboring cells interpolated using the B-spline kernel. In comparison, the straightforward B-spline solution provides, by construction, a much smoother and more natural interpolation result between grid cells, than the MVC solution. The cubic B-spline kernel guarantees a smooth approximation that is C^2 continuous everywhere, while the MVCs are continuous everywhere and smooth on the interior of the triangular meshes, but they are linear on the triangles of the mesh and can reproduce linear functions on the interior of the mesh.

Whilst our interpolation scheme with only B-spline provides a smooth solution, it still does not maintain the essential property of the no-slip boundary condition in CFD datasets. We, therefore, consider alternative solutions that keep the B-spline interpolation basis but mimic the no-slip condition at interpolation time, which is able to reproduce the correct flow structures close to the boundary.

As mentioned earlier in the chapter the velocity of the flow decreases until it reaches zero at the boundary. This decrease occurs in a thin layer called the boundary layer. This layer adjoining the boundary is characterized by the presence of considerable velocity gradients in it. and the velocity value for the points in the boundary layer should point the flow to move parallel to the solid boundary, which keeps propagating close to the boundary until they circulate to make the vortices.

The special handling is application-dependent to the CFD simulation data we are using in this thesis, each application can require its own special handling based on the boundary conditions of the application in hand. In approximating CFD flow data, problems occur when a streamline close to the boundary exits the domain due to small approximation inaccuracies near the boundary. To mimic the no-slip boundary behavior, if a streamline travels outside the domain, we project the interpolated velocity vector of that point onto the closest boundary such that the vector runs parallel to the boundary and does not continue moving away from the domain. Figure 4.10 illustrates the projection of the interpolated vector into the boundary.


(a) Interpolated vector exiting the domain. The Blue arrow is the interpolated vector.

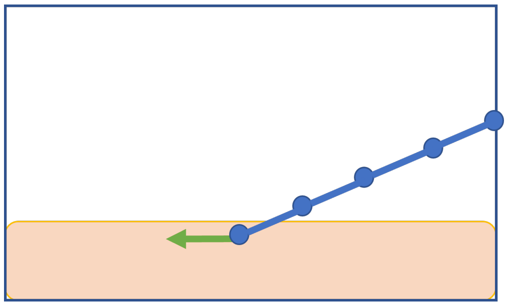

(b) Interpolated vector projected onto the boundary. The green arrow is the projected vector.

Figure 4.10. Special interpolation handling for flow visualizations.

There are two primary motivations for using grid-based boundary approximation instead of the exact geometry from the input boundary data. First, efficiency; accessing the exact boundary at query time can be expensive. Auxiliary data structures such as the celltree [16] are used to speed up queries in unstructured grids. Efficiently finding all the boundary triangles intersecting a given grid cell would require pre-processing the boundary triangulation to find which triangles intersect each cell. We would need to maintain bookkeeping of the triangle indices. This could possibly result in a large number of triangles per grid cell. At query time, we would then need to access the geometry of all the intersecting triangles. This information is necessary to decide if the point is inside or outside the volume, which will determine the correct interpolation. The total number of triangulations in the boundary approximation has an upper bound of 5 times the number of cut-cells which is dependent on the rectilinear grid resolution. The marching cubes triangulation in each cut-cell has an upper bound of 5 triangles.

Table 4.1. The percentage of the increase in rectilinear grid file size is due to the additional data arrays.

	Original unstructured grid		**Rectilinear grid**		
Dataset	**File size**	**Boundary file size**	**Naive (no-boundary) file size**	**Boundary array size**	**File size percentage increase for boundary data**
Flow about a cylinder	2.4 MB	16.8 KB	53 KB	3 KB	5%
Delta Wing	279 MB	41.9 MB	23.611 MB	6.7 KB	0.02%
ICE Train	86.9 MB	3.30 MB	21.9 MB	0.2 MB	9.1%

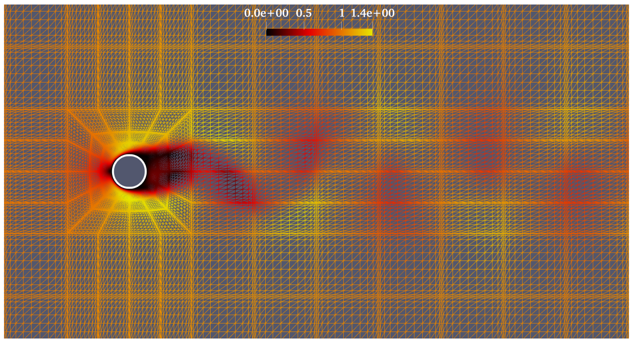
Second, storage-saving, if we use the input boundary at query time, we need to keep the solid boundary triangulation at all times, which can be expensive. In contrast, we only save extra data arrays within the same rectilinear grid file for the grid-cell-based boundary. Table 4.1 shows that the percentage of file size increase for the boundary-aware rectilinear grid compared to the naive rectilinear grid is a maximum of 9%. The files are using compressed VTR (VTK XML rectilinear grid format) file format. The ICE Train has the largest boundary mesh compared to the other datasets, which causes the file size percentage increase to be maximum in this case.

4.6 Method Evaluation

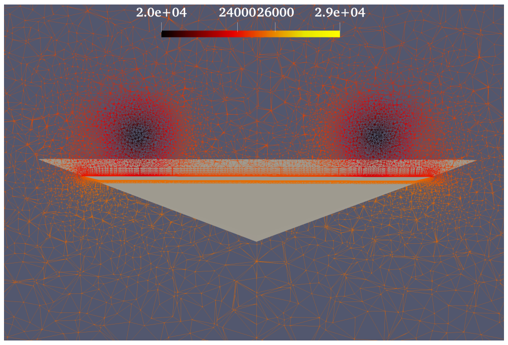
In this section, we evaluate our method in terms of approximation quality and performance.

As done in the previous chapter, we take advantage of the open-source Visualization Toolkit library [54] and its XML rectilinear grid format for our implementation. All the kernel weights, vertex labels, and edge intersection arrays are stored as data arrays in the rectilinear grid file.

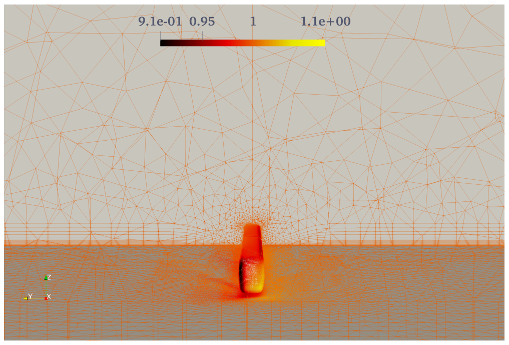
We use the same three CFD datasets for our evaluation as in previous chapters. Figure 4.11a shows the 2D viscous flow around a solid cylinder body. The solid cylinder boundary is colored white. Figure 4.11b shows the 3D Delta Wing with the wing solid boundary colored in gray. The wing boundary is considered a challenging one due to its small z-direction bounds. Figure 4.11c shows the ICE unstructured grid zoomed close to the train solid boundary.



(a) 2D flow about a cylinder. The cylinder boundary is a circle toward the left of the dataset.



(b) 3D Delta Wing. The solid boundary is a wing in the middle of the flow, colored with light gray.



(c) 3D ICE. The train body embedded in the airflow of the ICE dataset.

Figure 4.11. 2D and 3D datasets simulate flows with solid boundaries.

4.6.1 Approximation Quality Evaluation

This subsection offers a quantitative comparison between the visualization results obtained with the original unstructured grid and those achieved with our rectilinear grid approximation.

Like the previous chapter, our evaluation of the three CFD datasets focuses on the computation of the flow map, in other words, the flow-induced transport, which is the computational basis of a wide range of prevalent flow visualization algorithms.

Specifically, the flow map is computed in the original unstructured datasets and two versions of their rectilinear approximation, one without boundary handling and one with boundary handling capabilities. With proper boundary handling, no erroneous interpolation

Table 4.2. Comparison between the unstructured input files, the boundary-aware rectilinear grid files, approximation error, and generation time for the corresponding boundary-aware rectilinear file.

Dataset	Unstructured grid		Boundary-aware rectilinear grid			Approximation quality and timing			
	file size (MB)	# of points	file size (MB)	# of points	grid resolution	max error	RMS error	Approximation time (s)	Boundary process time (s)
Flow Around Cylinder	3.0	21K	0.085	2,145	55×39	6.45e-02	6.57e-03	0.27	0.079
Delta Wing	820	3.08M	18.2	704,123	119×97×61	6.51e-01	1.06e-01	374.07	5.19
ICE Train	180	1.07M	24	717,500	25×70 ×41	8.61e-01	1.08e-01	329.43	11.97

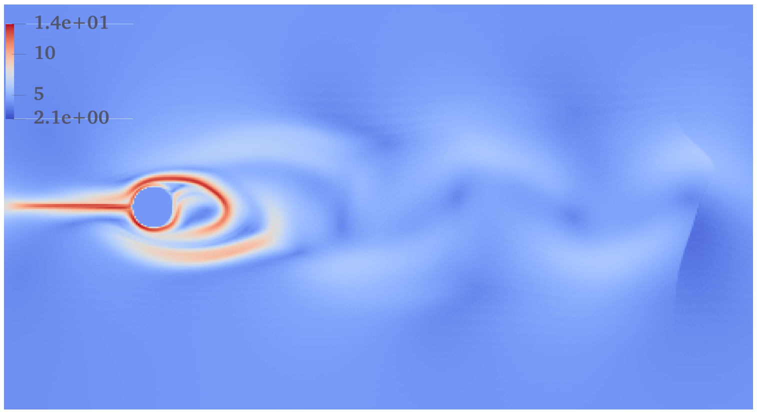
query is performed outside the domain. Hence, better flow map results should be expected close to the solid boundary using the boundary-aware rectilinear grid.

We use cubic B-spline kernels to generate the approximations used in this chapter. The high-order kernel provides a smooth approximation compared to the linear kernels. Unfortunately, the artifacts close to the boundary resulting from the rectilinear grid lacking awareness of the solid boundary exacerbate.

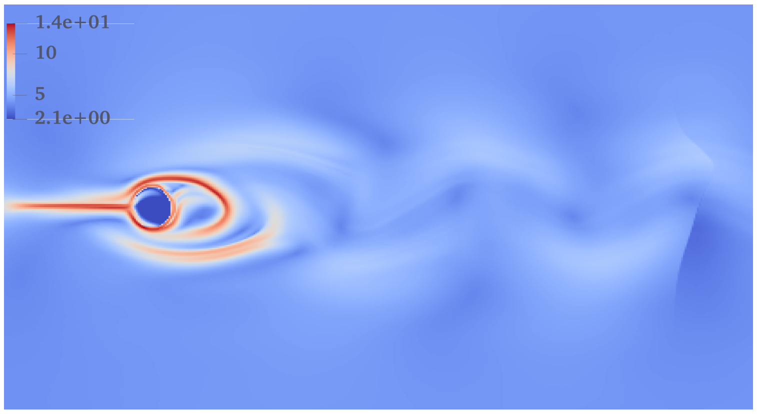
Table 4.2 presents information on the input data and our boundary-aware rectilinear grid approximation on each dataset. The Table also shows the approximation used for each tested dataset, the approximation's accuracy, the time is taken to obtain it, and the resulting file size.

To demonstrate the accuracy of our approximation, we compare visual results obtained from the FTLE of the flow map of each dataset and show the difference between the original unstructured grid, the naive approximation without boundary handling, and the result using our boundary-aware interpolation method explained in this chapter.

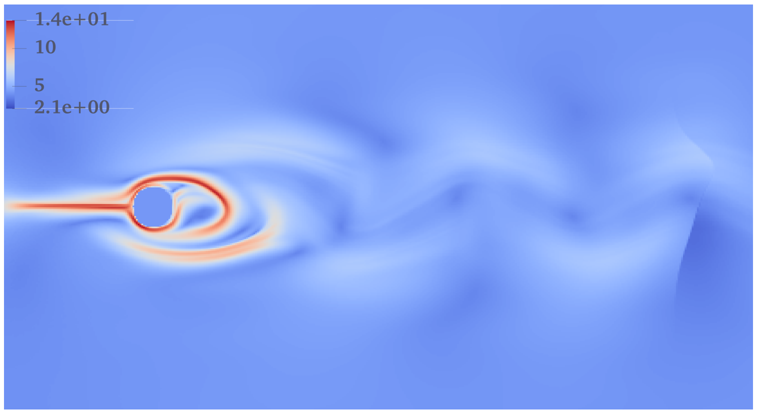
Figure 4.12 shows the FTLE computation over the flow map of the 2D flow dataset. There is a noticeable difference around the cylinder boundary between the naive approximation and the unstructured grid. The naive rectilinear grid has no information of what rectilinear grid points are inside or outside the domain and would still interpolate the outside query points, which will cause the FTLE around the solid boundary to be inaccurate. In contrast, when comparing the FTLE results of our boundary-aware approximation, the result is visually identical to the unstructured input grid. Our boundary handling approach accurately reproduced the flow behavior around the cylinder.



(a) Unstructured grid

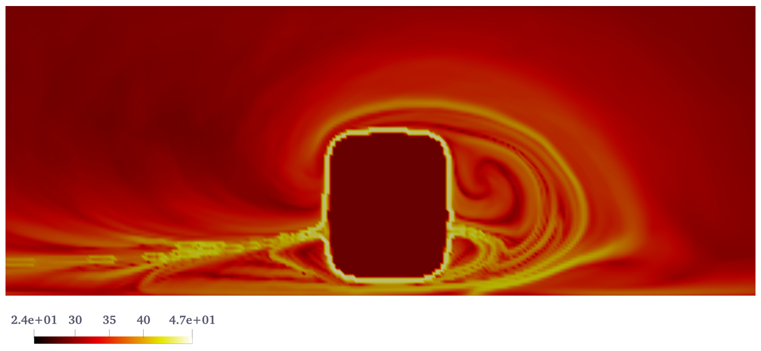


(b) Naive rectilinear grid

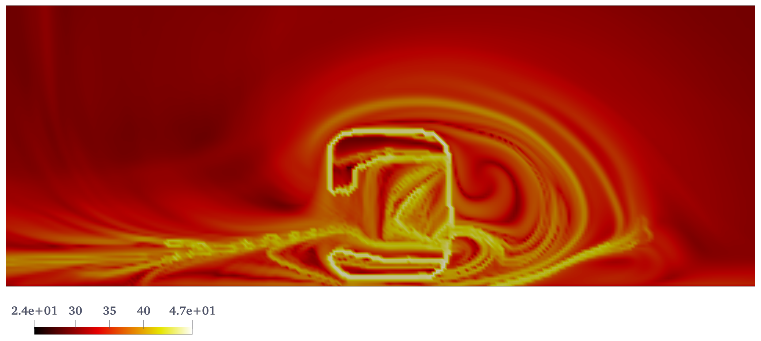


(c) Boundary-aware rectilinear grid

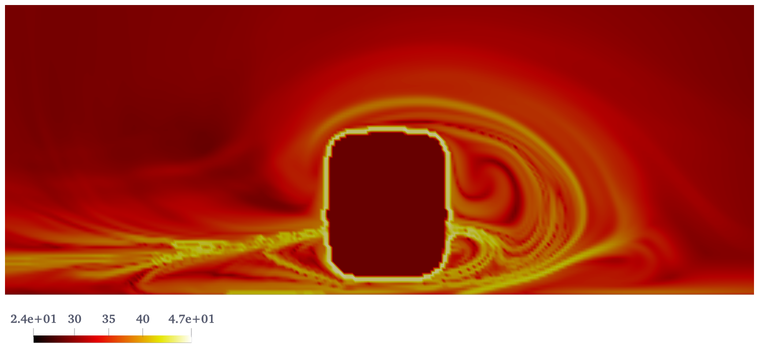
Figure 4.12. FTLE generated from the flow map run on the flow past cylinder 2D unstructured grid, the naive rectilinear grid approximation, and the boundary-aware rectilinear grid approximation. High accuracy in interpolation close to the boundary using boundary-aware rectilinear gird compared to the discrepancies in the interpolation in the naive rectilinear grid.



(a) Unstructured grid

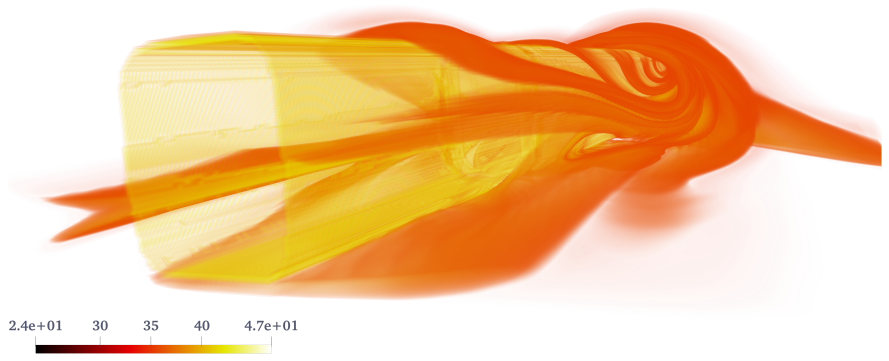


(b) Naive rectilinear grid

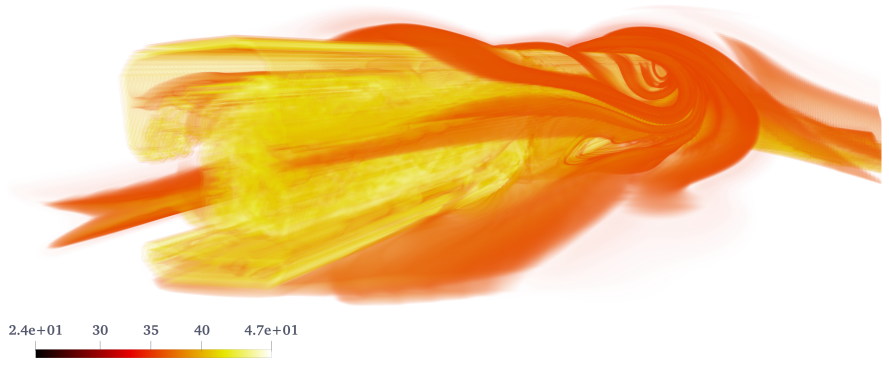


(c) Boundary-aware rectilinear grid

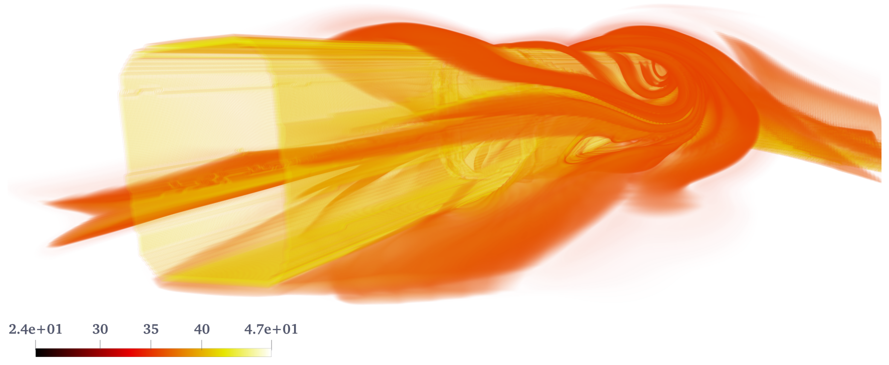
Figure 4.13. Comparison of slices of the FTLE result for the ICE Train dataset, using the original unstructured grid, the naive rectilinear approximation, and the boundary-aware rectilinear grid approximation.



(a) Unstructured grid



(b) Naive rectilinear grid



(c) boundary-aware rectilinear grid

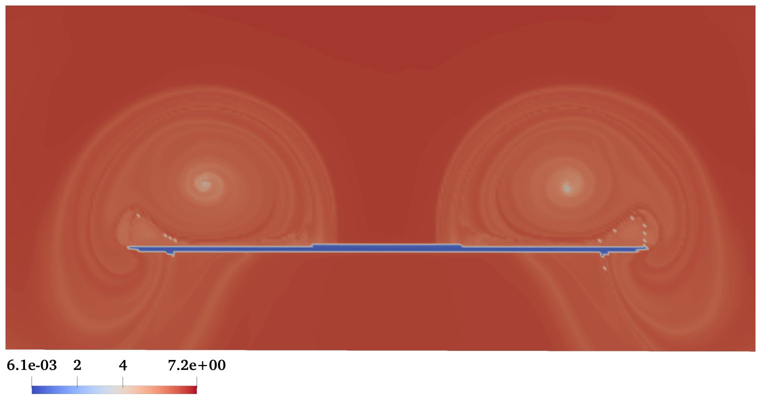
Figure 4.14. Comparing volume rendering of the flow map FTLE of the ICE Train dataset, computed over the original unstructured grid, the naive approximation, and the boundary-aware rectilinear grid approximation.

Figure 4.13 and 4.14 shows a comparison of the FTLE result obtained with the original unstructured grid input and our rectilinear approximation with and without boundary-awareness for the ICE Train dataset. Approximation without the boundary handling capability results in the nonphysical behavior of a flow crossing into the border, leaving no visual indication of the boundary. On the other hand, using our boundary-aware interpolation method, the resulting FTLE clearly outlines the boundary of the train body and closely matches the FTLE results of the input dataset around the boundary.

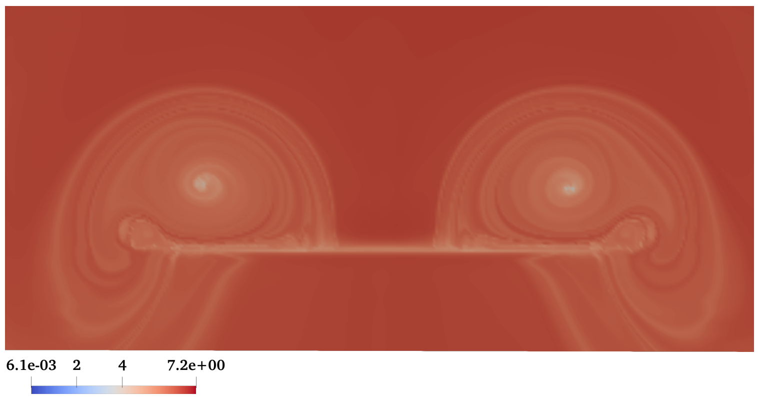
Figure 4.15 compares the flow map result of the original Delta Wing unstructured grid dataset against the approximation with and without boundary awareness. The figure shows that our boundary-aware approximation accurately captured the boundary given this dataset's challenging nature. The Delta Wing is very thin in the z-direction. Our approximation with the refinement method accurately handled the boundary and generated correct interpolations around it. Some artifacts can be seen as tiny dots around the wing-body. Those artifacts can not be avoided and are also present even in the original unstructured data flow map.

The same observation for the Delta Wing dataset can be seen in Figure 4.16 where the volume rendering of the FTLE using our boundary-aware rectilinear grid resulted in a visualization that captures the original data in all aspects. As before, the naive approximation was not able to capture the solid boundary.

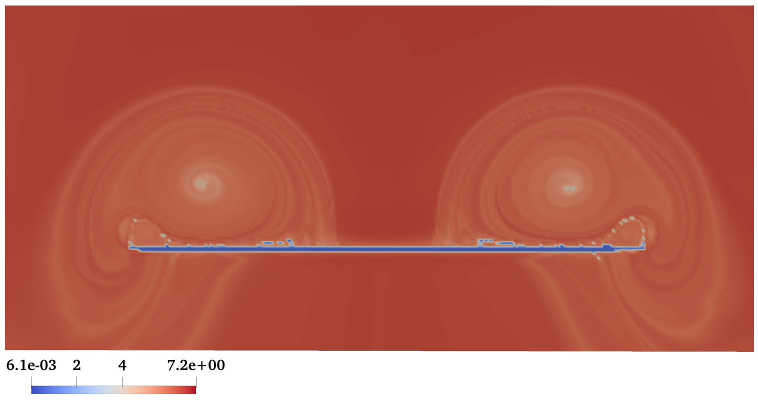
Our goal in this work is to achieve smaller file size while improving the visualization algorithms performance and the quality of the reconstruction. One could think of an alternative way to reduce the data size without the need for a least squares fit consists in extracting a decimated version of the original unstructured dataset, thereby retaining a subset of the original data points at which the value is known. To allow comparison with our approach, we decimate the original unstructured grid into a smaller grid that is equal in storage size to the rectilinear grid that we use to generate the results in this chapter. We then use this decimated data to study the flow structures. We perform this comparison on the Delta Wing dataset where we preserved all the boundary points, and the file size of the decimated data is 7% of the size of the original data. The resulting points were then re-meshed using a 3D Delaunay triangulation. We tested the performance and the accuracy of the decimated



(a) Unstructured grid

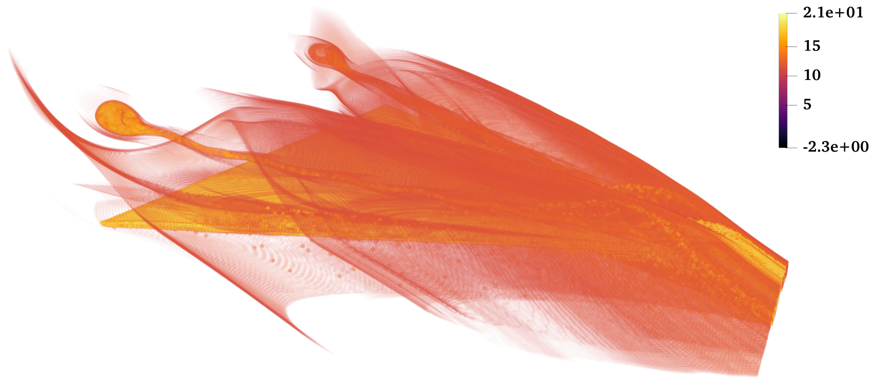


(b) Naive rectilinear grid

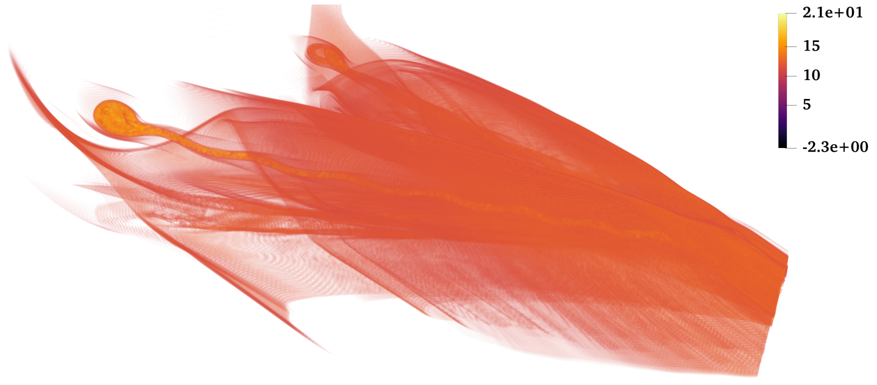


(c) Boundary-aware rectilinear grid

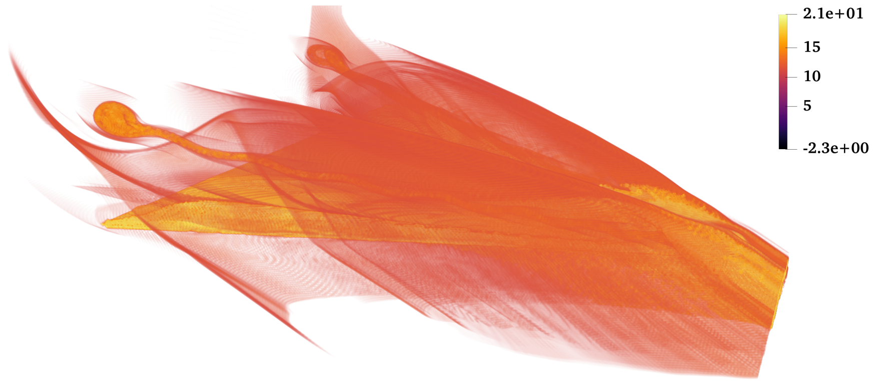
Figure 4.15. Comparison between slices of the Delta Wing flow map calculated over the unstructured grid, the naive rectilinear approximation, and the boundary-aware rectilinear approximation.



(a) Unstructured grid



(b) Naive rectilinear grid



(c) Boundary-aware rectilinear grid

Figure 4.16. Comparison of volume rendering of the FTLE generated from the flow map using Delta Wing unstructured grid, the naive rectilinear grid, and the boundary-aware rectilinear grid approximation.

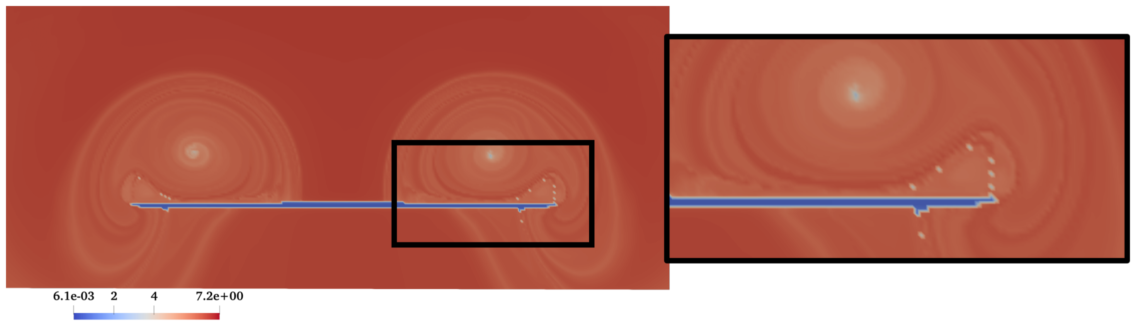
unstructured grid by generating a flow map and compare that result with the original unstructured grid and the rectilinear grid approximation. Figure 4.17 compares the slice of flow map for the original data and the decimated data. The results show that while the decimation approach is reducing the storage cost, it does so at the expense of the accuracy of the reconstruction and the efficiency of the visualization processing. Indeed, the quality of the decimated grid is poor, owing to the highly non-uniform vertices distribution; the tetrahedras have high aspect ratio and wide differences exist in local cell sizes. The bad quality of the decimated grid significantly deteriorates the cell-tree performance in this case, which leads to a poor interpolation performance that is even worse than that of the large original dataset. Running the flow map takes 3h 31m using the original unstructured data, 4h 36m using the decimated data, and 44m using the boundary-aware rectilinear grid for the same resolution. The accuracy of the flow map using the decimated data is also poor. First, the interpolation close to the wing is not accurate. This is due to the fact that the original data has an extremely high point density in the direct vicinity of the wing and decimating that grid therefore requires to remove a large number of data points in that region. In addition the Tertiary vortex on the right side of the flow map is missing as can be seen in the zoomed part of Figure 4.17.

A more carefully designed decimation procedure that considers the local spatial density of the resulting point set and the significance of a given vertex might yield better results, but that is an open research problem that is beyond the scope of this thesis.

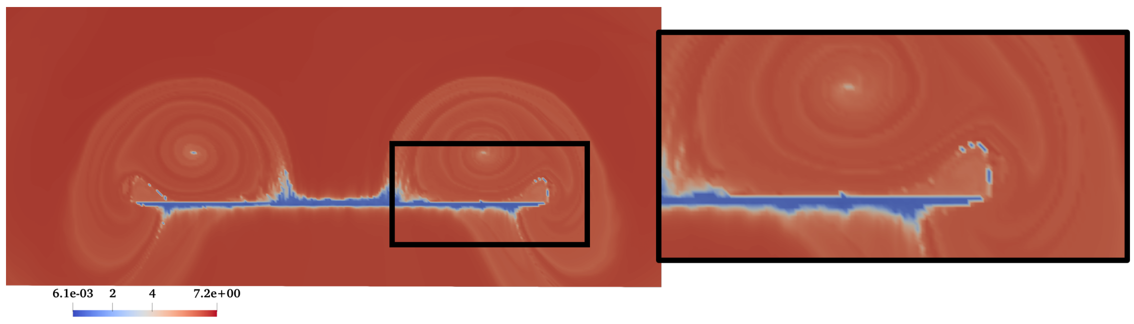
4.6.2 Timing Comparison

To test the interpolation speedup achieved using our boundary-aware rectilinear grid compared to the original unstructured dataset using the cell-tree, we interpolated one million random points in the original unstructured grid and our cubic B-spline interpolation using the different datasets, and we can find the results in Table 4.3.

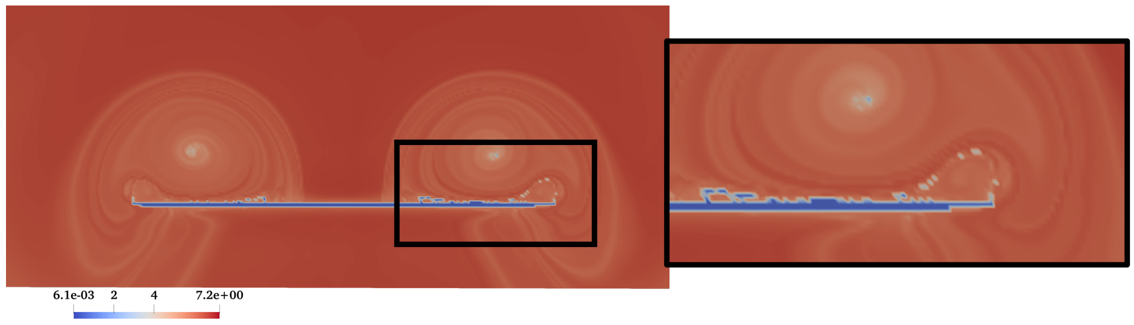
Cell-tree is a specialized boundary volume hierarchy tree structure for cell location across the unstructured grid. It is arguably the state-of-the-art in cell location in terms of balancing performance and memory footprint. We utilize the Cell-tree data structure as implemented



(a) Unstructured grid



(b) Decimated unstructured grid



(c) Rectilinear grid

Figure 4.17. Comparison between slices of the Delta Wing flow map calculated over the unstructured grid, the decimated unstructured grid , and the rectilinear grid approximation. The tertiary vortex is missing in the flow map using the decimated data.

in VTK. As previously stated and used in the thesis, we will compare the interpolation using our approximation with the interpolation in the original unstructured grid that uses a cell tree. The Cell-tree data structure is built once for each dataset pre-processing and used in all subsequent interpolation calls.

The rectilinear grid used in Table 4.3 is the same approximation used for the flow map computation.

Table 4.3. Comparison of interpolation time (in seconds).

DataSet	**Cell-tree build time**	**Unstructured Grid**	**Cubic B-spline Rectilinear Grid**	**Speed Up**
Cylinder	0.01	5.31	0.69	7.7×
Delta	16.16	8.36	1.35	18.1×
ICE	1.97	9.15	1.45	7.7×

As can be seen from Table 4.3, the B-spline rectilinear approximation for 1 million points takes a fraction of the time for the same 1 million points in the unstructured grid even with the Cell-tree speedup.

4.7 Conclusion and Future Work

We presented in this chapter a method enabling the accurate approximation of datasets defined over unstructured grids with complex solid boundaries through a simple tensor product B-spline approximation over specialized boundary-aware rectilinear grids. Our solution allows for automatic refinement of the rectilinear mesh around solid boundaries. We can obtain a piecewise linear approximation of the corresponding geometry within the simple rectilinear grids using cut-cells. As our experimentation with several CFD datasets showed, the resulting approximation makes it possible to explore the properties of large datasets through the much more efficient visualization of a significantly reduced and smoother representation with smooth derivatives. Our work utilizes VTK XML rectilinear grid without generating a new reader or writer to utilize the integration with currently available VTK applications and their existing pipeline. Several interesting avenues exist for future work. Among them, a more flexible, piecewise nonlinear boundary model could be applied in cut-cells instead of the current linear model to improve the rectilinear approximation accuracy further.

5. CONCLUSION AND FUTURE WORK

This thesis considers a framework to address the problems encountered when using an unstructured grid with highly non-uniformly distributed points and solid boundaries. These grids are commonly used in computational fluid dynamics for their flexibility and adaptivity but suffer from expensive storage and expensive post-processing visualizations. The proposed solution lies in the approximation of the input data into a rectilinear grid that can accurately represent the original unstructured data with superior performance and without the shortcomings of unstructured grids.

At the beginning of the thesis, we considered the choice of the rectilinear grid to be used for the approximation. The choice of the grid is a crucial aspect of the accuracy of the approximation. We proposed a customized quadtree/octree that aids in the adaptive subdivision of the domain based on the spatial distribution of the unstructured grid points and the local fit error in the tree nodes. This results in an adaptive grid that can accurately capture the main aspects of the simulation and is as small in size as possible.

Our evaluations show that, using the grid resulting from our customized hybrid quadtree/octree we can achieve the same high-quality approximation with a much smaller grid than a uniform grid or a grid obtained from naive quadtree/octree subdivision-based only on point distribution. A smaller grid means a smaller approximation system to solve and more efficient usage of storage space.

We continue in Chapter 3 our framework by investigating a numerical problem encountered while solving for the least-square fitting solution. The fitting least-squares system is often rank-deficient due to having insufficient constraints to solve the system. This lack of constraints is mainly due to the mismatch of the point distribution in the original data and the approximating rectilinear grid, where some grid points lack sufficient fitting constraints due to having few or no surrounding input data points within its basis support. The fitting solution of a system that is rank-deficient could complicate linear solvers or generate visual artifacts in the solution.

We adopt a regularization method where we use discrete Laplacian to add smoothness constraints. In this method, picking the regularization weight is critical. Using a high regularization weight can cause the smoothness constraint to overtake the interpolation constraint and increase the resulting approximation error. On the other hand, using a low regularization weight might not be sufficient to overcome the rank deficiency in the system. We propose a variable-smoothing approach, where we apply the right value of regularization weight per unknown, thus providing the right amount of smoothing constraint only where needed. Evaluations show that our variable-smoothing approach provides a high-quality approximation compared to the traditional approach using a singular regularization weight.

To ensure the system's scalability to large datasets, we use a block-based division of the domain to split the problem into smaller sub-problems such that larger datasets do not cause memory issues when solving the linear system. To maintain continuity between the blocks, we pad each block with overlapping ghost-cells. We showed that our block-based solution allowed for handling larger systems without a degradation in the approximation quality.

In Chapter 4, we consider the problem of lack of flexibility in representing complex geometry in rectilinear grids. The naive rectilinear grid has no notion of inside/outside the domain. This causes inaccurate interpolation in the rectilinear grid close to the boundary. The problem becomes more apparent when high-order kernels are demonstrated with cubic B-spline. To resolve the boundary problem faced, we supplement the approximation grid with a boundary approximation capability. This *boundary-aware rectilinear grid* is equipped with the information necessary to approximate a solid boundary, and at query time, handles near the boundary interpolation, thus enjoying all the benefits of conventional rectilinear grids while addressing their fundamental geometric limitations.

Though our thesis covers many aspects of the approximation problem, there remain interesting avenues for future work. Some avenues for future work can be creating a hierarchy of approximation resolutions. It can be beneficial to display the approximation result of scientific simulation data in a fine to coarse hierarchical fashion. The availability of a low-resolution approximation with acceptable accuracy can be helpful when processing using low-resource hardware or become a bootstrap for more refined resolution approximations. This could face many research challenges regarding how the hierarchical structure will adapt to the grid's nature, primarily if a rectilinear grid is used. Additional problems could arise given that the coarse grid solving system would be tall and skinny (*i.e.*, number of input points is higher than the number of unknowns), which makes the solving complex. Optimization algorithms like the Lancsoz algorithm, an algorithm used to break down the complexity of the problems, could be beneficial.

There are two types of time-dependent flow visualizations simulations; the Eulerian simulations and the Lagrangian simulations. In the Eulerian simulations, the underlying mesh remains constant while the field values at the mesh points change. Which is the type of simulations we deal with in this thesis. On the other hand, the Lagrangian simulations underlying mesh changes over the time steps. Another avenue that would be useful for large time-dependent simulations would be the creation of a streaming approach. Suppose an Eulerian time-dependent data set, having the whole time steps ready at the time of solving will improve the performance, given that the approximation matrix would be generated once and factorized once, and the right-hand side would be the aggregate of all time steps.

For Lagrangian time-dependent simulations, the streaming method would include an extra overhead of updating the approximation grid efficiently (without the construction of the octree) to adapt the edits in the data points' locations.

Other venues for future work can be figuring out ways to improve the approximation to run on large parallelized clusters/supercomputers and develop a high-performance implementation of this method on GPU/CPU.

REFERENCES

[1] J. López, A. Esteban, J. Hernández, P. Gómez, R. Zamora, C. Zanzi, and F. Faura, “A new isosurface extraction method on arbitrary grids,” *Journal of Computational Physics*, vol. 444, p. 110 579, 2021, ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110579. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999121004745.

[2] G. Dupuy, B. Jobard, S. Guillon, N. Keskes, and D. Komatitsch, “Isosurface extraction and interpretation on very large datasets in geophysics,” *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling 2008, SPM’08*, Jun. 2008. DOI: 10.1145/1364901.1364932.

[3] G. Gu and D. Kim, “Accurate and efficient gpu ray-casting algorithm for volume rendering of unstructured grid data,” *ETRI Journal*, vol. 42, no. 4, pp. 608–618, 2020. DOI: https://doi.org/10.4218/etrij.2019-0185. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.4218/etrij.2019-0185.

[4] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller, “Curvature-based transfer functions for direct volume rendering: Methods and applications,” in *Proc. IEEE Visualization*, Oct. 2003, pp. 513–520.

[5] M. Shih, Y. Zhang, and K.-L. Ma, “Advanced lighting for unstructured-grid data visualization,” in *2015 IEEE Pacific Visualization Symposium (PacificVis)*, 2015, pp. 239–246. DOI: 10.1109/PACIFICVIS.2015.7156383.

[6] N. Liu, D. Zhu, Z. Wang, Y. Wei, and M. Shi, “Progressive light volume for interactive volumetric illumination,” *Computer Animation and Virtual Worlds*, vol. 27, no. 3-4, pp. 394–404, 2016. DOI: https://doi.org/10.1002/cav.1706. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1706.

[7] R. Peikert and F. Sadlo, “Height Ridge Computation and Filtering for Visualization,” in *Proceedings of Pacific Vis 2008*, I. Fujishiro, H. Li, and K.-L. Ma, Eds., Kyoto, Japan, Mar. 2008, pp. 119–126, ISBN: 978-1-4244-1966-1.

[8] S.-K. Kim, “Extraction of ridge and valley lines from unorganized points,” *Multimedia Tools Appl.*, vol. 63, no. 1, pp. 265–279, Mar. 2013, ISSN: 1380-7501. DOI: 10.1007/s11042-012-0999-y. [Online]. Available: https://doi.org/10.1007/s11042-012-0999-y.

[9] Y. Ohtake, A. Belyaev, and H.-P. Seidel, “Ridge-valley lines on meshes via implicit surface fitting,” *ACM Trans. Graph.*, vol. 23, no. 3, pp. 609–612, Aug. 2004, ISSN: 0730-0301. DOI: 10.1145/1015706.1015768. [Online]. Available: https://doi.org/10.1145/1015706.1015768.

[10] G. Haller, D. Karrasch, and F. Kogelbauer, "Material barriers to diffusive and stochastic transport," *Proceedings of the National Academy of Sciences*, vol. 115, no. 37, pp. 9074–9079, 2018.

[11] M. V. Bilskie, D. Coggin, S. C. Hagen, and S. C. Medeiros, "Terrain-driven unstructured mesh development through semi-automatic vertical feature extraction," *Advances in Water Resources*, vol. 86, pp. 102–118, 2015, ISSN: 0309-1708. DOI: https://doi.org/10.1016/j.advwatres.2015.09.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0309170815002274.

[12] Wikipedia contributors, *Regular grid — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Regular_grid&oldid=1048438666, [Online; accessed 30-November-2021], 2021.

[13] Wikipedia contributors, *Unstructured grid — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Unstructured_grid&oldid=1024609439, [Online; accessed 30-November-2021], 2021.

[14] J. Wilhelms and A. Van Gelder, "Octrees for faster isosurface generation," *ACM Trans. Graph.*, vol. 11, no. 3, pp. 201–227, Jul. 1992, ISSN: 0730-0301. DOI: 10.1145/130881.130882. [Online]. Available: https://doi.org/10.1145/130881.130882.

[15] M. Langbein, G. Scheuermann, and X. Tricoche, "An efficient point location method for visualization in large unstructured grids.," Jan. 2003, pp. 27–35.

[16] C. Garth and K. I. Joy, "Fast, memory-efficient cell location in unstructured grids for visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1541–1550, 2010.

[17] C. Wachter and A. Keller, "Instant ray tracing: The bounding interval hierarchy," in *EGSR '06*, 2006.

[18] G. Zachmann, "Minimal hierarchical collision detection," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '02, Hong Kong, China: Association for Computing Machinery, 2002, pp. 121–128, ISBN: 1581135300. DOI: 10.1145/585740.585761. [Online]. Available: https://doi.org/10.1145/585740.585761.

[19] L. Piegl and W. Tiller, *The NURBS Book*. Berlin, Heidelberg: Springer-Verlag, 1995, ISBN: 3540550690.

[20] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2012, vol. 3.

[21] L. Evans, "Partial differential equations 1st edn (providence, ri: American mathematical society)," 1998.

[22] T. A. Davis, "Algorithm 915, suitesparseqr: Multifrontal multithreaded rank-revealing sparse qr factorization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 8, 2011.

[23] X. Tang and A. Boozer, "Finite time lyapunov exponent and advection-diffusion equation," *Physica D: Nonlinear Phenomena*, vol. 95, no. 3, pp. 283–305, 1996, ISSN: 0167-2789. DOI: https://doi.org/10.1016/0167-2789(96)00064-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0167278996000644.

[24] B. Nouanesengsy, T.-Y. Lee, K. Lu, H.-W. Shen, and T. Peterka, "Parallel particle advection and ftle computation for time-varying flow fields," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11. DOI: 10.1109/SC.2012.93.

[25] P. J. Nolan, M. Serra, and S. D. Ross, "Finite-time lyapunov exponents in the instantaneous limit and material transport," *Nonlinear Dynamics*, vol. 100, no. 4, pp. 3825–3852, Jun. 2020, ISSN: 1573-269X. DOI: 10.1007/s11071-020-05713-4. [Online]. Available: http://dx.doi.org/10.1007/s11071-020-05713-4.

[26] J. Kasten, C. Petz, I. Hotz, B. R. Noack, and H.-C. Hege, "Localized finite-time lyapunov exponent for unsteady flow analysis," in *VMV*, 2009.

[27] S. L. Guoqiao You, "Computing the finite time lyapunov exponent for flows with uncertainties.," *Journal of Computational Physics*, vol. 425, 2021.

[28] B. Jobard, G. Erlebacher, and M. Hussaini, "Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 211–222, 2002. DOI: 10.1109/TVCG.2002.1021575.

[29] Z. Ding, "Lagrangian analysis of vector and tensor fields: Algorithmic foundations and applications in medical imaging and computational fluid dynamics," PhD thesis, Purdue University, 2016.

[30] G. Haller, "Distinguished material surfaces and coherent structures in three-dimensional flows," *Physica D*, vol. 149, pp. 248–277, 2001.

[31] *Lagrangian Coherent Structures:analysis of time-dependent dynamical systems using finite-time Lyapunov exponents*, https://shaddenlab.berkeley.edu/uploads/LCS-tutorial/overview.html, Accessed: 2021-10-15.

[32] C. Garth, G.-S. Li, X. Tricoche, C. D. Hansen, and H. Hagen, *Visualization of Coherent Structures in Transient 2D Flows*, H.-C. Hege, K. Polthier, and G. Scheuermann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–13, ISBN: 978-3-540-88606-8. DOI: 10.1007/978-3-540-88606-8_1. [Online]. Available: https://doi.org/10.1007/978-3-540-88606-8_1.

[33] F. Lekien and S. Ross, "The computation of finite-time lyapunov exponents on unstructured meshes and for non-euclidean manifolds," *Chaos (Woodbury, N.Y.)*, vol. 20, p. 017 505, Mar. 2010. DOI: 10.1063/1.3278516.

[34] G. Haller, "Lagrangian coherent structures," *Annual Review of Fluid Mechanics*, vol. 47, no. 1, pp. 137–162, 2015. DOI: 10.1146/annurev-fluid-010313-141322. eprint: https://doi.org/10.1146/annurev-fluid-010313-141322. [Online]. Available: https://doi.org/10.1146/annurev-fluid-010313-141322.

[35] J. Guckenheimer and P. J. Holmes, *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. Springer Science &amp; Business Media, 2013, vol. 42.

[36] R. K. Beatson, J. B. Cherrie, and C. T. Mouat, "Fast fitting of radial basis functions: Methods based on preconditioned gmres iteration," *Advances in Computational Mathematics*, vol. 11, no. 2-3, pp. 253–270, 1999.

[37] E. Arge, M. Daehlen, and A. Tveito, "Approximation of scattered data using smooth grid functions," *Journal of computational and applied mathematics*, vol. 59, no. 2, pp. 191–205, 1995.

[38] M. Arigovindan, M. Suhling, P. Hunziker, and M. Unser, "Variational image reconstruction from arbitrarily spaced samples: A fast multiresolution spline solution," *IEEE Transactions on Image Processing*, vol. 14, no. 4, pp. 450–460, 2005.

[39] E. Vuçini, T. Möller, and M. E. Gröller, "Efficient reconstruction from non-uniform point sets," *The Visual Computer*, vol. 24, no. 7-9, pp. 555–563, 2008.

[40] E. Vuçini, T. Möller, and M. E. Gröller, "On visualization and reconstruction from non-uniform point sets using b-splines," in *Computer Graphics Forum*, Wiley Online Library, vol. 28, 2009, pp. 1007–1014.

[41] E. Vuçini and W. G. Kropatsch, "On the search of optimal reconstruction resolution," *Pattern recognition letters*, vol. 33, no. 11, pp. 1460–1467, 2012.

[42] B. Francis, S. Viswanath, and M. Arigovindan, "Scattered data approximation by regular grid weighted smoothing," *Sādhanā*, vol. 43, no. 1, p. 5, 2018.

[43] A. Dubey, A. Almgren, J. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Löffler, B. O'Shea, E. Schnetter, B. Van Straalen, and K. Weide, "A survey of high level frameworks in block-structured adaptive mesh refinement packages," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 3217–3227, Jan. 2014.

[44] S. B. Baden, D. B. Gannon, M. L. Norman, and N. P. Chrisochoides, *Structured Adaptive Mesh Refinement (Samr) Grid Methods*. Berlin, Heidelberg: Springer-Verlag, 1999, ISBN: 0387989218.

[45] B. T. N. Gunney and R. W. Anderson, "Advances in patch-based adaptive mesh refinement scalability," *Journal of Parallel and Distributed Computing*, vol. 89, pp. 64–84, Jan. 2016.

[46] M. Langbein, G. Scheuermann, and X. Tricoche, "An efficient point location method for visualization in large unstructured grids," in *Proceedings of Vision, Modeling, Visualization*, 2003.

[47] N. Andrysco and X. Tricoche., "Matrix trees," *Eurographics/ IEEE-VGTC Symposium on Visualization 2010*, vol. 29, no. 3, 2010.

[48] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs, "Data reduction techniques for simulation, visualization and data analysis," in *Computer Graphics Forum*, Wiley Online Library, vol. 37, 2018, pp. 422–447.

[49] D. Hoang, P. Klacansky, H. Bhatia, P. Bremer, P. Lindstrom, and V. Pascucci, "A study of the trade-off between reducing precision and reducing resolution for data analysis and visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, pp. 1193–1203, Jan. 2019.

[50] B. Fornberg, "Generation of finite difference formulas on arbitrarily spaced grids," *Mathematics of computation*, vol. 51, no. 184, pp. 699–706, 1988.

[51] M. Deville, P. Fischer, and E. Mund, "High-order methods for incompressible fluid flow," *Applied Mechanics Reviews*, vol. 56, Jan. 2003. DOI: 10.1115/1.1566402.

[52] T. B. Benjamin, "Theory of the vortex breakdown phenomenon," *Journal of Fluid Mechanics*, vol. 14, no. 4, pp. 593–629, 1962. DOI: 10.1017/S0022112062001482.

[53] J. Dormand and P. Prince, "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, 1980, ISSN: 0377-0427. DOI: https://doi.org/10.1016/0771-050X(80)90013-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0771050X80900133.

[54] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit–An Object-Oriented Approach To 3D Graphics*, Fourth. Kitware, Inc., 2006.

[55] G. Ma and L. Li, "Edge detection in potential fields with the normalized total horizontal derivative," *Computers & Geosciences*, vol. 41, pp. 83–87, 2012, ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2011.08.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098300411002846.

[56] K. Mahrous, J. Bennett, G. Scheuermann, B. Hamann, and K. Joy, "Topological segmentation in three-dimensional vector fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 2, pp. 198–205, 2004. DOI: 10.1109/TVCG.2004.1260771.

[57] S. Rusinkiewicz, "Estimating curvatures and their derivatives on triangle meshes," in *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.*, 2004, pp. 486–493. DOI: 10.1109/TDPVT.2004.1335277.

[58] A. A. M. Khang Jie Liew Ahmad Ramli, "B-spline surface fitting on scattered points," *Applied Mathematics & Information Sciences*, vol. 10, 2016.

[59] T. Peterka, N. G., I. Grindeanu, V. Mahadevan, R. Yeh, and X. Tricoche, "Foundations of multivariate functional approximation for scientific data," Oct. 2018, pp. 61–71. DOI: 10.1109/LDAV.2018.8739195.

[60] Y.-S. Yeh, "Efficient Knot Optimization for Accurate B-spline-based Data Approximation," Dec. 2020. DOI: 10.25394/PGS.13363571.v1. [Online]. Available: https://hammer.purdue.edu/articles/thesis/Efficient_Knot_Optimization_for_Accurate_B-spline-based_Data_Approximation/13363571.

[61] D. El-Rushaidat, R. Yeh, and X. M. Tricoche, "Accurate parallel reconstruction of unstructured datasets on rectilinear grids," *Journal of Visualization, Springer.*, pp. 1875–8975, 2021.

[62] P.G.Tucker and Z.Pan, "A cartesian cut cell method for incompressible viscous flow," *Applied Mathematical Modelling*, vol. 24, pp. 591–606, 2000.

[63] D. D.M.Ingra and and C.G.Mingham, "Developments in cartesian cut cell methods," *Mathematics and Computers in Simulation*, vol. 61, pp. 561–572, 2003.

[64] R. R. Arslanbekov, V. I. Kolobov, and A. A. Frolova, "Analysis of compressible viscous flow solvers with adaptive cartesian mesh," in *20th AIAA Computational Fluid Dynamics Conference,, Honolulu, Hawaii*, 2011.

[65] P. Tucker and Z. Pan, "A cartesian cut cell method for incompressible viscous flow," *Applied Mathematical Modelling*, vol. 24, no. 8, pp. 591–606, 2000, ISSN: 0307-904X. DOI: https://doi.org/10.1016/S0307-904X(00)00005-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0307904X00000056.

[66] B. Muralidharan and S. Menon, "A high-order adaptive cartesian cut-cell method for simulation of compressible viscous flow over immersed bodies," *Journal of Computational Physics*, vol. 321, pp. 342–368, 2016, ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2016.05.050. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999116301954.

[67] F. Bouchon, T. Dubois, and N. James, "A second-order cut-cell method for the numerical simulation of 2D flows past obstacles.," Feb. 2011, 35 pages. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00570049.

[68] T. S. Zhihua Xie, "A three-dimensional cartesian cut-cell/volume-of-fluid method for two-phase flows with moving bodies," *Journal of Computational Physics*, vol. 416, Jan. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999120303107.

[69] W.Lorensen and H.Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Applied Mathematical Modelling*, vol. 4, pp. 163–169, 1987.

[70] W.Lorensen, "Marching through the visible man," *IEEE Visualization, Proceedings of the 6th conference on Visualization*, vol. 6, pp. 368–373, 1994.

[71] E. Chernyaev, "Marching cubes 33: construction of topologically correct isosurfaces," CERN, Geneva, Tech. Rep., Nov. 1995. [Online]. Available: https://cds.cern.ch/record/292771.

[72] T. Lewiner, H. Lopes, A. Vieira, and G. Tavares, "Efficient implementation of marching cubes' cases with topological guarantees," *Journal of Graphics Tools*, vol. 8, Apr. 2012. DOI: 10.1080/10867651.2003.10487582.

[73] *Computer Graphics: Principles and Practice*, 3rd ed. Addison-Wesley, 2013, ISBN: 978-0-321-39952-6.

[74] S. Koshizuka, K. Shibata, M. Kondo, and T. Matsunaga, "Chapter 2 - fundamentals of fluid simulation by the mps method," in *Moving Particle Semi-implicit Method*, S. Koshizuka, K. Shibata, M. Kondo, and T. Matsunaga, Eds., Academic Press, 2018, pp. 25–109, ISBN: 978-0-12-812779-7. DOI: https://doi.org/10.1016/B978-0-12-812779-7.00002-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128127797000023.

[75] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, no. 1, pp. 19–27, 2003, ISSN: 0167-8396. DOI: https://doi.org/10.1016/S0167-8396(03)00002-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167839603000025.

[76] K. Hormann and M. S. Floater, "Mean value coordinates for arbitrary planar polygons," *ACM Trans. Graph.*, vol. 25, no. 4, pp. 1424–1441, Oct. 2006, ISSN: 0730-0301. DOI: 10.1145/1183287.1183295. [Online]. Available: https://doi.org/10.1145/1183287.1183295.

[77] T. Ju, S. Schaefer, and J. Warren, "Mean value coordinates for closed triangular meshes," in *ACM SIGGRAPH 2005 Papers*, ser. SIGGRAPH '05, Los Angeles, California: Association for Computing Machinery, 2005, pp. 561–566, ISBN: 9781450378253. DOI: 10.1145/1186822.1073229. [Online]. Available: https://doi.org/10.1145/1186822.1073229.

[78] M. S. Floater, "Generalized barycentric coordinates and applications *," *Acta Numerica*, vol. 24, pp. 161–214, 2015.

VITA

Dana El-rushaidat received her Ph.D. in computer science from Purdue University working under Prof. Xavier Tricoche in the Computer Graphics and Visualization Lab. She received her B.S in computer science from Yarmouk University of Irbid-Jordan in 2004 and her M.S. in computer science from Yarmouk University Irbid-Jordan in 2006. Dana's dissertation work was primarily concerned with approximation of large unstructured grids into rectilinear grids with boundary handling capabilities. She was teaching in Mu'tah University Karak-Jordan from (2007-2015), then she started her Ph.D at Purdue university. Upon graduation she will pursue her teaching work in Jordan University of Science and Technology (JUST) Irbid-Jordan.