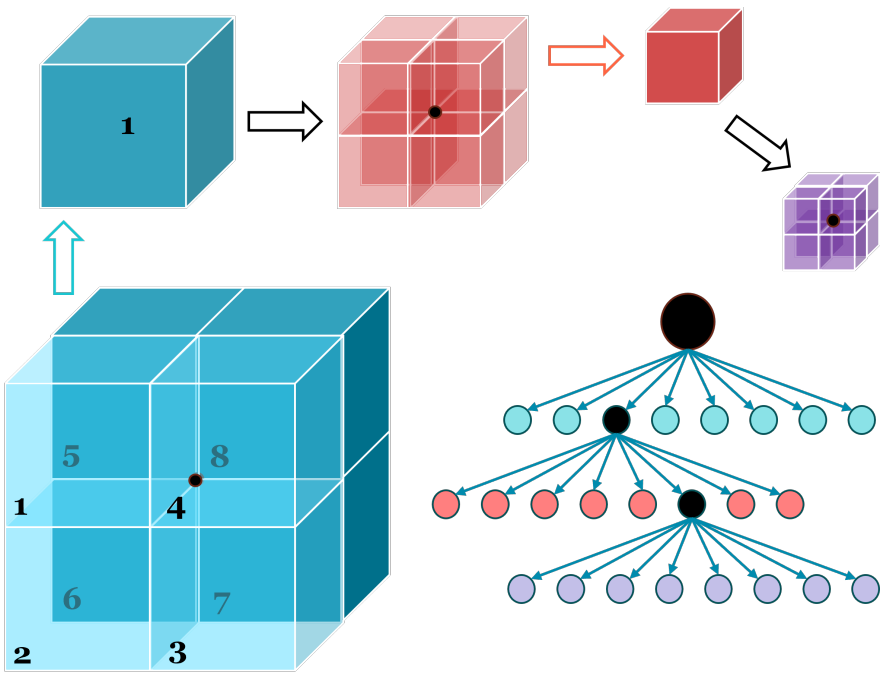
Nearest Neighbour Search Method For 3D Data Mapping of CFD Data

-Steady State Thermal Analysis

Abhishek Dhiman (abhdh)

Supervisor LiU: Raghu Chaitanya Munjulury
Supervisor Siemens Energy: Loureiro, Jordi

Examiner LiU: Petter Krus
Manager Siemens Energy: Emma Moschner



LINKÖPINGS UNIVERSITET

Linköping University
Fluid & Mechatronics Systems
Master Thesis 30 ESTC | Aeronautical Engineering
Spring 2024|LIU-IEI-TEK-A–24/04821-SE

Abstract

The advancements in computation power and the development of high-fidelity Multiphysics tools have resulted in the generation of large volumes of datasets during immense simulations. This Master's thesis work aims to investigate the effectiveness of data mapping of the simulation results from finer meshes (large datasets) onto a coarser mesh using the nearest point method to reduce the simulation time for Multiphysics analysis such as Conjugate Heat Transfer (CHT).

The data mapping process utilizes the nearest point method, which employs the nearest neighbour search algorithms. ABAQUS and subroutine files with FORmula TRANslation (FORTRAN) implementation of the nearest neighbour search method are used for the data mapping. Although various search algorithms are available in the literature, we narrowed down to the Bin (Discretise function in MATLAB) and Octree Approach, which are then compared against the Brute Force and the explicit subroutine file creation using MATLAB code. The FORTRAN implementation of these algorithms is achieved through subroutine files and is supported by the User Defined Data Structure (UDDS) to hold the data subsets and corresponding information. The four algorithms are implemented and the performance is evaluated using metrics such as Total Central Processing Unit (CPU) time, Wall Clock time, Compilation & Pre-Run time, and the average time of the subroutine execution (nearest point method). Additionally, for benchmarking, a Hollow Cylinder is utilized as Case1 and a complex geometric model of an Exhaust Valve as Case2 for validation and qualitative analysis of mapping results.

The results indicate a significant improvement in the simulation time achieved with the new methodologies compared to the existing ones. Further analysis of Case1 assisted in the identification of parameters that influence the performance. The qualitative and quantitative analysis of Heat Transfer Coefficient ($[Wm^{-2}K^{-1}]$) (HTC) and Reference Fluid Temperature $[K]$ (SINK) contours showed strong agreement with the reference fine mesh results for both verification (Case1) and validation (Case2) cases. Performance analysis of the methodologies for multi-processor analysis (parallelization using multiple cores) highlighted the strength of new methodologies developed in this thesis work.

Based on the current study, it can be concluded that the new methodologies developed for the nearest neighbour search have shown significant improvements over the existing methodologies. The new methodologies are robust and highly efficient in handling large volumes of structured and unstructured mesh datasets. Additionally, it can be concluded that the performance of the new methodologies for shared memory Message Passing Interface (MPI) based analysis surpasses the performance of the old methodologies for single and multi-core analysis (parallel execution).

Acknowledgments

I am incredibly grateful and honored to have conducted my master's thesis work at Siemens Energy AB. It has been a privilege to be associated with the Mechanical Integrity team in the Research and Development of Siemens Energy. Words cannot express my gratitude for the work culture and the environment, the amiable domain experts and intellects, and the opportunity to meet industrial experts and make new friends. I have always been surrounded by experienced professionals to discuss the technicalities of the thesis work. A very big thank you goes to all.

I am deeply indebted to my manager, Emma Moschner, and supervisor, Jordi Loureiro, at Siemens Energy AB. You are the reason for the jumpstart of the thesis from day one and the steep learning curve throughout the entire duration of the thesis work. The academic knowledge and professional ethics that I have gained from working at a global leader like Siemens Energy will be cherished.

My sincere thanks for the constant feedback from my peers, which has been the foundation for the improvements to make the thesis work robust, scalable, and dynamic. The discussions with the domain experts in the department have broadened my perspective to analyze the positive and negative scenarios and create a big-picture view of the problem statement. The fellow master thesis students have been a great support with the academic discussions, thesis progression, and feedback on the deliverables. A big thank you to all.

Eventually, the endeavor would not have been possible without my supervisor, Raghu Chaitanya Munjulury, and examiner, Petter Krus at Linköping University (LiU) for their keen observation of the thesis work. The constant motivation and inspiration from you have allowed me to be more ambitious and innovative.

Acronyms

AI Artificial Intelligence. 42, 44, 45

CAD Computer-aided design. 2, 3, 19

CFD Computational fluid dynamics. 1–6, 17, 44

CHT Conjugate Heat Transfer. 1

CPS4 Constant plane stress 4 nodes. 15

CPS8 Constant plane stress 8 nodes. 15

CPU Central Processing Unit. 1, 27, 32–36, 38–41, 43, 46

CSG Constructive Solid Geometry. 4

CSV Comma Separated Values. 19, 32, 42

D Dimensionality: No of Columns. 7, 9–11

DC3D10 10 node quadratic tetrahedron element. 37

DNS Direct Numerical Simulation. 2

DOF Degree of Freedom. 41, 43, 46, 47

FEA Finite element analysis. 1, 3

FEDES Finite Element Data Exchange System. 3

FEM Finite element method. 14, 17

FORTRAN FORmula TRANslation. 1

FVM Finite volume method. 15

GPR Gaussian Process Regression. 2

HTC Heat Transfer Coefficient ($[Wm^{-2}K^{-1}$). 1, 4, 16, 17, 19, 22, 23, 27, 29, 31, 32, 38, 40, 46

KBE Knowledge Based Engineering. 3

KD K-Dimensional. 1, 5, 7, 10–12, 20, 33, 41, 42, 44, 46

KPI Key Performance Indicator. 2

LiU Linköping University. 1, 3, 44

M1 Mesh1 with 33612 data points. 30–33

M2 Mesh2 with 98292 data points. 30–33

M3 Mesh3 with 249324 data points. 30–33

M4 Mesh4 with 421788 data points. 30–34

ML Machine Learning. 1, 2, 42, 44, 45

MPI Message Passing Interface. 1

MUTEX Mutual Exclusive Object. 20, 27

N Data Points. 7, 9, 10

NN Neural Network. 2

O Order: Big O notation. 7

PCA Principal Component Analysis. 2

PDE Partial differential equation. 2, 14

PINN Physics Informed Neural Network. 2

POD Proper Orthogonal Decomposition. 2

RAPID Robust Aircraft Parametric Interactive Design. 3

ROM Reduced Order Model. 2

SINK Reference Fluid Temperature $[K]$. 1, 4, 16, 17, 19, 22, 23, 27, 29–32, 38, 40, 46

SNAME Surface Name. 27

SVD Singular Value Decomposition. 2

TEN Tetrahedral Network. 4

UDDS User Defined Data Structure. 1, 22–27, 42, 46

UEXTERNALDB User subroutine to manage external database. 27

XML Extensible Markup Language. 3, 42

Contents

1 Introduction

The rapid development of Data Science and Machine Learning (ML) has increased the opportunities for advancements in the Computational fluid dynamics (CFD) domain and enhanced the data utilization for multi-physics simulations like CHT. The underlying coupling in multi-physics simulations must respect the mathematical formulation and account for different meshes, discretization schemes, and interpolations used in each domain to maintain the conservation of properties and the order of accuracy. A fully coupled solver is the ideal and most accurate approach but it leads to a large system of non-linear equations resulting in high computational costs [1].

Subsequently, the multi-physics coupling in space and time is linked to interpolation, and the Closest pair of points problem [2] is a well-known fundamental problem associated with computational geometry and its complexities. The existing literature for mapping Finite element analysis (FEA) data mentions four interpolation techniques: nearest point method, distance method using field of points, distance method using elements, and method using element shape function [3]. The thesis work focuses on the nearest node search to map data from a fine mesh onto a coarse mesh minimizing the Euclidean distance. Therefore, the primary scope is to map the data from fine CFD mesh onto a coarser mesh with computationally efficient [3] methods. The problem intends to find the point with the smallest distance from a given point in a metric space of points or point cloud. The data structures (e.g., K-Dimensional (KD) tree, Ball tree, Octree, slice representation) [4] are built to efficiently solve the problem and are associated with time complexity and space dimensionality (number of columns in the data file).

1.1 Problem Statement

The use of high-fidelity simulations for multi-physics problems such as CHT is constrained by computation resources, simulation time, data generation, and data handling. Achieving high solution accuracy is crucial for research and development in fields like Aerospace, Aeronautics, and Automobile Manufacturing. Siemens Energy, a global leader in energy technology, has been continuously developing gas turbines for various applications. The market demand for low emissions, sustainability, and rapid technology delivery has shifted the focus to research and data reutilization. Consequently, the concept of 3D Data Mapping, which integrates CFD and Data Science, becomes vital for effectively reusing high-resolution CFD data to predict the results for heat transfer analysis, serving as an alternate approach to complete multi-physics CHT simulations.

The industry is well aware of the limitations of single or multi-physics high-fidelity simulations, including the need for immense computation power, software capabilities, experienced domain experts, inter-dependencies between CFD and experimental results, and the significant complexity of CHT physics, particularly when combustion dynamics are involved. These factors contribute to an extended time to market for new products or upgrades to existing ones.

1.2 Literature Survey

In this section, we delve into a broader research domain to underscore the rapid integration of Machine Learning advancements with the fundamental architecture of scientific research, such as high-performance computing and CFD. In the realm of CFD, computational expenses are significant due to the non-linear Partial differential equation (PDE) governing the Navier-Stokes equations, which result from time-dependent chaotic behavior, specifically Turbulence. Conversely, the progress in ML has facilitated the development of deep learning-based Neural Network (NN) to enable algorithms to learn from data. The utilization of ML to enhance CFD methodologies, turbulence modeling, accelerated Direct Numerical Simulation (DNS), and heat transfer aspects of CFD are highlighted in [5] along with potential limitations. The discussed approaches in [5] encompass Physics Informed Neural Network (PINN) and computational chemistry. The discussion of Reduced Order Model (ROM) to identify dominant coherent structures in complex flows is associated with ML and revolves around learning from a low dimensional coordinate system using Proper Orthogonal Decomposition (POD) which is data-driven [5]. The POD is closely related to Principal Component Analysis (PCA) and Singular Value Decomposition (SVD), which are employed for the dimensionality reduction of datasets and can be generalized by deep learning or neural network NN [5]. A few challenges and the research directions in CFD and ML integration are wall-bounded flow predictions near the walls, robust models, and large energy consumption associated with large-scale simulations. Another set of ML limitations encompasses data-driven training and the requirement of large datasets for higher efficiency and prediction accuracy [5].

Next, is a systematic way to utilize the CFD data to train machine learning models and then incorporate the predicted results for optimization purposes. In the study [6], an Ejector geometry optimization is performed. the authors perform optimization of an ejector, which is a low-cost work recovery device that utilizes expansion energy to pump a secondary to high pressure without any moving parts [6]. The authors use Gaussian Process Regression (GPR) [7] as an ML approach to create a surrogate model from large sets of CFD data. GPR is a probabilistic supervised machine learning model used for regression and classification tasks, which quantifies uncertainties along with predictions. The detailed tutorial and mathematical basics can be found in [7]. The generation of a surrogate model is automated (Python v3.9) for various ejector designs, operating conditions, and model parameters for both 2D and 3D cases. The architecture of the end-to-end automated process is available in [6], where the CFD is stored in the database for Key Performance Indicator (KPI) and then analyzed by the machine learning model [6]. However, the performance and accuracy of the GPR model depend on the quality of CFD data, which is prone to numerical errors, mesh quality, discretization schemes used, and other limiting cases for GPR modeling, as explained in detail in [6].

The Aircraft Design domain extensively utilizes low and high-fidelity CFD solvers for aerodynamic analysis after the conceptual design process is almost finished and a Computer-aided design (CAD) model has been created.

The CAD model is a complex piece of geometry from a CFD perspective and requires CAD cleaning and other pre-processing before the CFD analysis of the entire aircraft can begin. At LiU, Knowledge Based Engineering (KBE) aircraft conceptual design applications are Tango using MATLAB and Robust Aircraft Parametric Interactive Design (RAPID) using CATIA have been developed. These applications allow for a full 3D CAD model of the aircraft, including detailed interior, exterior, and integration [8]. The framework is based upon a one-database approach using a central Extensible Markup Language (XML) database that connects three modules: a CAD module, an estimation, analysis, and assessment module, and a simulation and system architecture module. This setup offers seamless data sharing and parallel functionality. The application of KBE automates repetitive, rule-based, or statistical data-based processes which can be manual, semi-automatic, or automatic [9]. The automated CAD model generation using RAPID with parameterized sizing and geometric manipulation makes it flexible for integration with other architecture [6]. This leads to automated CFD simulation and aerodynamic data generation for the machine learning model [5] [6].

Integrating the conceptual design process and CFD analysis with machine learning has the potential to reduce the overall deliverable time through fully or partially automated architecture. The advancements in post-processing and visualization technologies, driven by high computation power, enable multi-domain integration with seamless data sharing for design, analysis, and visualization of results. However, achieving seamless data sharing is more complex than it initially appears. For instance, the problem addressed in this thesis work involves the nearest neighbour search [10] [11] algorithm for data mapping of CFD data, which is closely linked with multi-variant [12] [13] and scattered data interpolation [14][15].

The Finite Element Data Exchange System (FEDES) is a tool used to transfer and map FEA data between different finite element solvers and meshes. It incorporates four interpolation techniques: nearest neighbour search, distance method using the field of points, distance method using elements, and method using element shape functions [3]. Among these, the nearest neighbour search is the simplest method for interpolation from fine to coarse meshes, resulting in higher accuracy and fast calculations [3]. This method is based on the minimum distance between two points (reference and target points), where the distance is the Euclidean distance in 2D or 3D space [16]. The distance method using the field of points is preferred when mapping is performed from fine to coarse meshes and multiple element types. The distance method using elements first identifies the elements and nodes before applying the distance calculation. It calculates the average distance from all the nodes of the element. Lastly, the method using the element shape function can be used to map data from fine to coarse and vice-versa with higher accuracy. The shape of each element type in the mesh is described by its shape function and can be used to calculate values at any point within the element [3]. A detailed comparison in the literature [3] concludes that the nearest neighbour (Method 1) is computationally efficient for large datasets, while the other three methods show comparatively high accuracy but are computationally expensive.

1.3 Aim

The master thesis aims to integrate and re-utilize CFD (sink temperature and heat transfer coefficient) data to perform steady heat transfer analysis using a computer-based approach (3D Data Mapping) to reduce the computation costs of multi-physics simulations. The strength of the 3D data mapping process is highly coupled with the implementation of the solution to the closest pair of points problem, efficient use of derived data structures to handle the 3D data in FORTRAN, and the time complexity of the algorithm. The mapping process is to be performed in ABAQUS using subroutines written in the FORTRAN programming language. The primary focus is the implementation of the search algorithm with a derived data structure compatible with ABAQUS user subroutines for efficiently storing the CFD data which can be used during the analysis and data mapping. The research questions Sec. 1.4 and study limitation Sec. 1.5 are explained in detail in the respective sections.

1.4 Research Questions

The multi-physics simulations are in general bottom-up approach where individual components (uni-physics) are coupled in space and time with appropriate spatial interpolation and temporal resolution techniques [1]. This coupling is prone to introduce limitations on the stability, accuracy, and robustness and is also expensive in data transfer and conversion of data structures used [1]. The computational resource requirements are immense to perform a multi-physics simulation and achieve higher result accuracy. Therefore, this leads to immense research to build powerful and efficient solvers. The thesis explores one such approach i.e. utilizing uni-physics CFD data (HTC and SINK) to perform steady heat transfer analysis and map the results on the component or geometry to reduce the need to perform a multi-physics simulation to estimate the reduction in the computation expenses or time.

The research for better and efficient multivariate interpolation techniques to improve the computational accuracy and time complexity for the nearest neighbour search [12] in a large scattered data has been in focus from the beginning of simulations itself and is a well-studied problem [1]. The minimization of interpolation error is important to achieve higher solution accuracy had led to the development of 3D data structures like Tetrahedral Network (TEN), Constructive Solid Geometry (CSG) tree, Octree, etc [4]. Modern programming languages or newer versions at least have built-in capabilities to make use of 3D data structures and algorithms for interpolation of large datasets.

However, many powerful and industry-standard software (in the present case ABAQUS) have limited exposure to modern programming languages and still use low-level programming languages like FORTRAN to achieve faster runtime, but it is difficult for developers to utilize modern data structures and optimized algorithms directly with FORTRAN leading to Brute implementation of the code [17]. ABAQUS provides User Subroutines (written in FORTRAN) to access the extended functionalities and flexibility to perform analysis [18], thus limiting the use of 3D data structures and algorithm implementation in FORTRAN.

Therefore, creating scope for the research questions to utilize modern data structures and search algorithms in legacy programming languages to achieve faster runtime and computational efficiency is crucial. The performance of an algorithm is in general evaluated based on the time and space complexity to show a direct correlation between the number of inputs Sec. 2. Since the input CFD datasets are large, the thesis emphasizes more on the time complexity of the algorithms [19] [20]. Multiple algorithms such as Brute Force, KD tree, Octree, and Domain Slicing are selected for data partitioning, hence it is important to measure the behavior of each algorithm and the correctness of the results so that potential improvements can be made in the most appropriate approach.

The following research questions are derived from the above discussion which will be explored and the best attempts will be made to answer these research questions in this master thesis work.

Q1. Is 3D data mapping an alternate approach to overcome the challenges associated with multi-physics simulations?

Q2. Are 3D data structures compatible with nearest neighbour search algorithms to hold large volumes of data and suitable for search operation?

Q3. Is the creation of a 3D data structure correlated to the ease of implementation, data access, and nearest neighbour search time?

Q4. How does multi-processor analysis influence the simulation and are there any other influential factors associated with multi-processor?

1.5 Limitation

The thesis is limited by ABAQUS Python v2.7 (pre-processing) and FORTRAN90 (analysis subroutine development) available for ABAQUS CAE 2019. The Python version (>3.7) offers great flexibility for algorithm development and data handling with built-in libraries and dedicated modules which are unavailable in Python 2.7. The speed of the FORTRAN programming language and subroutine availability are favorable for building/implementing a search algorithm, but being old and simple promotes the Brute implementation of code and the lack of in-built libraries/modules makes it difficult to work with.

2 Theory

This section provides a brief explanation of key concepts and ABAQUS keywords to introduce the reader to the underlying principles and better understanding of the concepts presented in the report. The section encompasses the basic concepts like Euclidean distance and data sorting to advance concepts like search algorithms, interpolation, time complexity, and CFD aspects associated with this thesis.

2.1 Multivariate/Spatial Interpolation

In the mathematical sense, interpolation [21] is a method to estimate a new data point from a given set of discrete data points. The given points may represent the values of a function based on dependent and independent variables e.g. in Eq. (1), the independent variable is x, the dependent variable is y and the function $f(x)$ can be used to interpolate values of y for any value of x for a given $c = constant$. The multivariate interpolation [13] is interpolation on the function of more than one independent variable, and the spatial interpolation [13] is a special case where the independent variables are the spatial coordinates. The applications are vast for interpolation techniques e.g. geostatistics, topographic surveys, etc.

$$y = f(x) = mx + c \tag{1}$$

$y = y$ coordinate, $x = x$ coordinate, $m =$ slope of the line and $c =$ constant value of y when x is zero

A good topic to mention is the *scatter data interpolation* which constructs a function (univariate, bivariate, or multivariate) to estimate the value of the dependent variable. The multivariate interpolation is specially applied to irregular grids or point clouds as scatter data interpolation for electronic imaging systems, medical imaging, meteorological or geographical modeling, and computer-aided design [15]. There are numerous methods in the literature such as linear triangular interpolation, cubic triangular, inverse distance weighted methods, scattered data fitting, and natural neighbour interpolation which is studied by Amidror [15]. Not to confuse between scattered data interpolation and fitting, Amidror [15] mentions that interpolation is the construction of functions that pass through the data points whereas data fitting function maintains the trends of input data and may only pass near the data points.

For thesis work, spatial interpolation is utilized for 3D regular (structured mesh) and irregular (unstructured mesh) grids where the coordinates resemble a point cloud dataset. *For instance: Case 1*, the 3D data mapping is to be performed on coarse mesh ($Data_{coarse}$) for heat transfer analysis, and the CFD data ($Data_{fine}$) provided is for a very fine mesh, hence the closest point is searched in $Data_{fine}$ corresponding to each $Data_{coarse}$ point if the exact point does not exist else the search should return the exact point. There are several methods to interpolate the value for the closest point such as nearest neighbour interpolation, inverse distance weight, natural neighbour interpolation, and Kriging to name a few [13]. In this thesis, the nearest neighbour interpolation is used to solve the closest pair of points problem for computational geometry [2].

2.2 Time Complexity of Algorithm

The time complexity of an algorithm is one of the performance parameters to quantify the runtime/execution time of the algorithm as a function of the length of inputs [20]. Since there are multiple languages, ways to solve and implement the code for a single problem, dedicated techniques, etc, thus time complexity understanding becomes vital for writing efficient code and algorithms. This significantly depends on the developer's experience, following the best practices for the programming language of interest, available computational resources, etc.

The Big Order: Big O notation (O) is used to describe the time complexity of an algorithm. If Data Points (N) and Dimensionality: No of Columns (D) in the data file, then the time complexity is given by ON, where O is the order of growth [20]. Various time complexity exists to evaluate an algorithm that is briefly explained below for unit data dimensionality (D = 1):

1. **Constant time O[1]:** When the algorithm is independent of the data or input size (N) it is termed as constant time with order O[1] complexity.

2. **Linear time O[DN]:** When the algorithm runtime increases linearly with N i.e. inspecting all the values in the given dataset, it is termed as O[DN] time complexity.

3. **Logarithmic time O[log(DN)]:** When N to be inspected reduces with each step it is called O[log(DN)] time complexity. For instance, KD tree, binary tree, or binary search are good examples as these space partitioning techniques successively reduce the space or domain to a more focused subdomain thus reducing N and ensuring that operations are not performed for all the elements of the input data.

4. **Quadratic time O[DN^2]:** This is the nonlinear time complexity where the runtime increases non-linearly with N. Good examples of this are nested loops as each loop has a linear complexity but a nest combination can be represented as O[DN] * O[DN] = O[$(DN)^2$] complexity. For multiple nested loops combinations which are not considered best practices for any programming languages the time complexity becomes a polynomial time complexity O[DN*M], where M is the number of nest loops.

other important mentions are cubic, exponential, factorial, and quasi-linear time complexity which are outside the scope of the thesis but are worth mentioning.

The algorithms are mentioned in the Table. 1 with corresponding time complexity factor in the curse of dimensionality [22] as well and not to forget about the space complexity [19]. Therefore, a good balance of complexities is required for an efficient algorithm that can fulfill the intended goals.

2.3 Euclidean Distance

The closest point to a given reference point in 3D space is obtained using the Euclidean distance method which gives the length of the line segment joining the two points based on the Cartesian coordinates [16], Fig. 1. The generalized formula for distance calculation is given by Eq. 3 (3D) and Eq. 2 (2D). This generalized formula can be scaled as per the dimensions e.g. 1D, 2D, 3D, ND. Based on the application e.g. least square method in statistics, search algorithm development where time complexity is a major concern, the recommendation is to omit the last square root operation and use the squared distance [16].

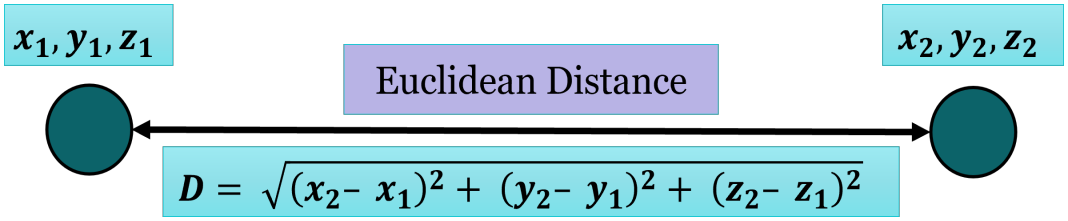


Figure 1: Euclidean distance in Cartesian coordinates

2.4 Data Sorting

Another interesting topic from the thesis point of view is data sorting which is also the requirement for a couple of nearest neighbour search algorithms explained in Sec. 2.5. The sorting operation on data e.g. 1D array or list of elements is performed by rearranging the elements according to a comparison operator which decides the new order of the elements in the array or list [23]. The order of the rearranged elements may be ascending or descending as shown in Fig. 2 for an array with 8 elements (N integers) indexed from 1 to 8. Some programming languages do the element indexing from 0 to $N - 1$ e.g. Python. There exist various algorithms to perform sorting with different time complexities and a well-optimized combination of algorithms in the modern programming languages which are not explained in this report, but interested readers are recommended to Ref. [23].

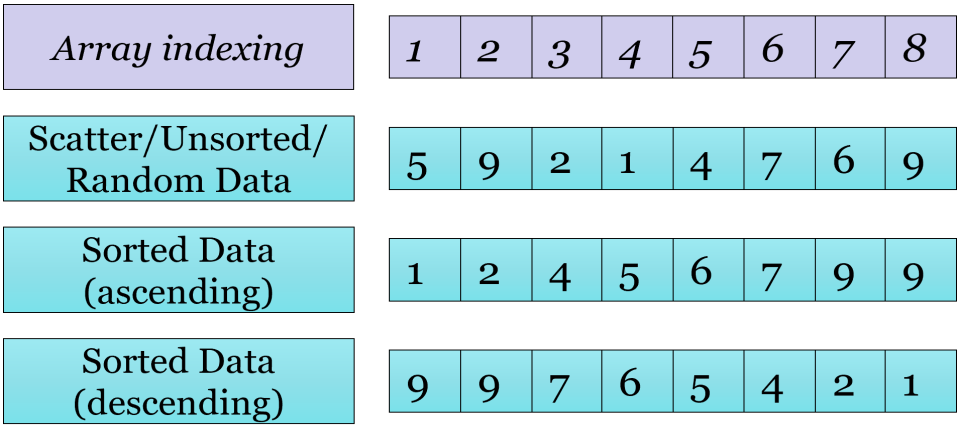


Figure 2: Array indexing and data sorting

2.5 Nearest Neighbor Search Algorithms

Some of the widely used nearest neighbour search algorithms and respective time complexity are highlighted in Table. 1. Most of these algorithms are available as built-in modules/libraries in modern programming languages e.g. Python (scikit-learn and scipy modules). The working of each of the listed algorithms is explained in a section with their advantages and disadvantages.

Table 1: Nearest Neighbor Algorithms, D , N

Algorithm	Time complexity
Brute Force	O$[DN^2]$
KD Tree	O$[DNlog(N)]$
Ball Tree	O$[DNlog(N)]$
Octree	O$[Dlog(N)]$

2.5.1 Brute Force:

This is the naive approach (linear search) to find the nearest neighbour based on the distance between all pairs of points in the dataset [11]. **Note:** The Distance = Euclidean distance [16] between two points using the Cartesian coordinates of the points. It is the length of the line segment between the points and is also known as Pythagorean distance [16]. The Eq. (2) and (3) [16] are used to calculate the Euclidean distance between two points in 2D and 3D space respectively. To speed up the algorithm runtime, the square root operation is omitted which still yields identical results [10]. Back to the Brute Force, let us understand the implementation with the same example (**Case 1**) of $Data_{coarse}$ and $Data_{fine}$ provided in Sec. 2.1. For the first point $\in Data_{coarse}$ as the query point, the algorithm evaluates the distance (2D/3D) w.r.t all the points $\in Data_{fine}$. The point with minimum / zero distance is identified as the closest neighbour and the next point $\in Data_{coarse}$ is selected as the new query point leading to the closest point in $Data_{fine}$ Fig. 3.

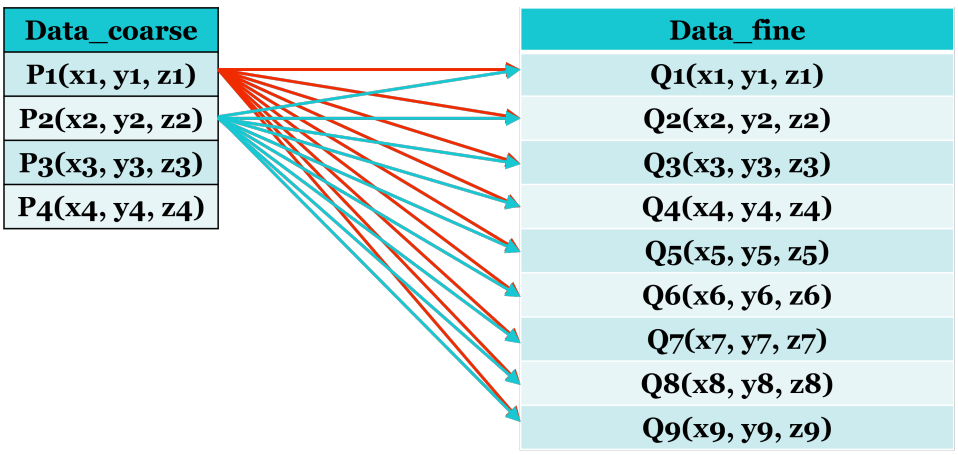


Figure 3: Brute Force implementation

$$d_{2D} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2}$$

where Cartesian coordinates of point $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$

$$d_{3D} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{3}$$

where Cartesian coordinates of point $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$

For D dimensions and N sample points, the time complexity of the Brute Force approach tends to O$[DN^2]$. The approach is efficient and suitable to small datasets (small N) values but becomes inefficient as N increases.

2.5.2 KD Tree:

It is a space partitioning approach and uses a tree data structure that recursively bisects the domain into two halves [10]. The root node of the tree structure represents the entire dataset successive levels represent the subset w.r.t the axes of partitioning. For instance, for 3D data with coordinates (x, y, z), the partition by x (Level-1 = L1) partitioning is done w.r.t x-coordinate value, then by y (Level-2 = L2) and then by z (Level-3 = L3) [24] Fig. 4. The each level of partitioning, the axes are sorted first and then get bisected at the median, thus each parent node in the tree has a left ($<$ median) and a right ($>$ median) child. The search for a query point $\in Data_{coarse}$ in the KD tree for $Data_{fine}$ starts at the root node and then transverse tree for a node coordinate value greater than or equal to that of the query point [24] i.e. is the nearest neighbour to the query point.

The tree-based data structures were invented to mitigate the computational inefficiencies of the linear Brute force search by reducing the distance calculations. The construction of the KD tree is fast due to partitioning along the data axes and gives the O$[DNlog(N)]$ time complexity which is a significant improvement over Brute force [11]. However, the algorithm slows down for high dimensionality data (D $>$ 20) referred to as the *curse of dimensionality* [22]. The dimensionality (D= 3) is fixed for the thesis work as the scope is limited to 3D space and regular/irregular computational geometry.

Points	X	Y	Z
P1	0	1	8
P2	3	4	2
P3	5	7	2
P4	1	2	9
P5	7	4	7
P6	8	6	5
P7	4	7	1
P8	1	3	3
P9	5	7	1

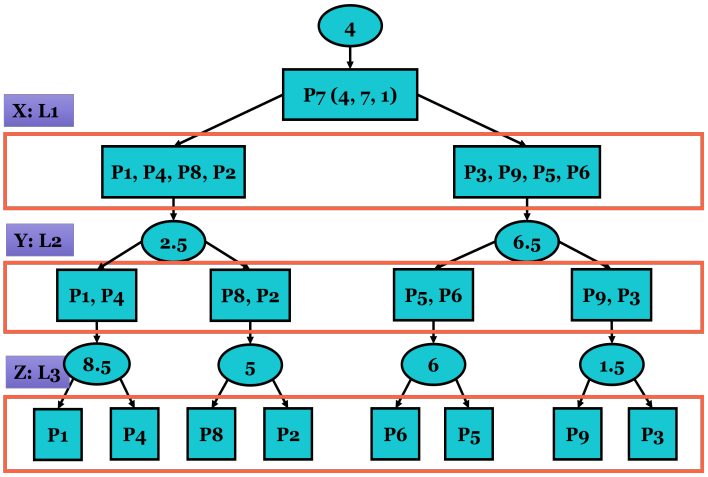


Figure 4: KD tree structure for 3D coordinates (scatter data)

2.5.3 Ball Tree:

The Ball tree structure is similar to the KD tree and is designed to address the inefficiencies of the KD tree in higher dimensions (D>20). The difference between the two is that the Ball tree partitions the data in a series of nested hyper-spheres whereas the KD tree utilizes Cartesian axes [11] Fig. 5. This makes the Ball tree structure highly efficient for high dimensions and structured data but also increases the tree construction more expensive than the KD tree.

The recursive division of data in the Ball tree structure is defined around a centroid (C) as diamond markers Fig. 5 and radius (r). Construction of the ball tree starts with a random point (say (0, 0)), and the farthest point is found (P6). Then, from P6 the farthest point is found i.e. P1, and a circle (red) is constructed which represents the entire domain. The centroid (red diamond) of the data points is then placed inside this circle as the new reference point and the process repeats recursively [25], Fig 5. The nearest neighbour search is narrowed down by using triangle inequality [26] which states that the sum of lengths of any two sides of a triangle must be greater than or equal to the length of the remaining side, Eq. (4). The spherical geometry of the Ball tree aids in fast estimation of lower and upper bounds for distance calculation leading to higher efficiency for higher dimensions.

$$z \leq x + y, \quad |x + y| \leq |x| + |y| \tag{4}$$

where x, y, and z are the three sides of the triangle. For Euclidean geometry, this inequality theorem is specifically about the distance, thus z is replaced by the vector sum of x and y.

Points	X	Y
P1	0	1
P2	3	4
P3	5	7
P4	1	2
P5	7	4
P6	8	6
P7	4	7
P8	1	3
P9	5	7

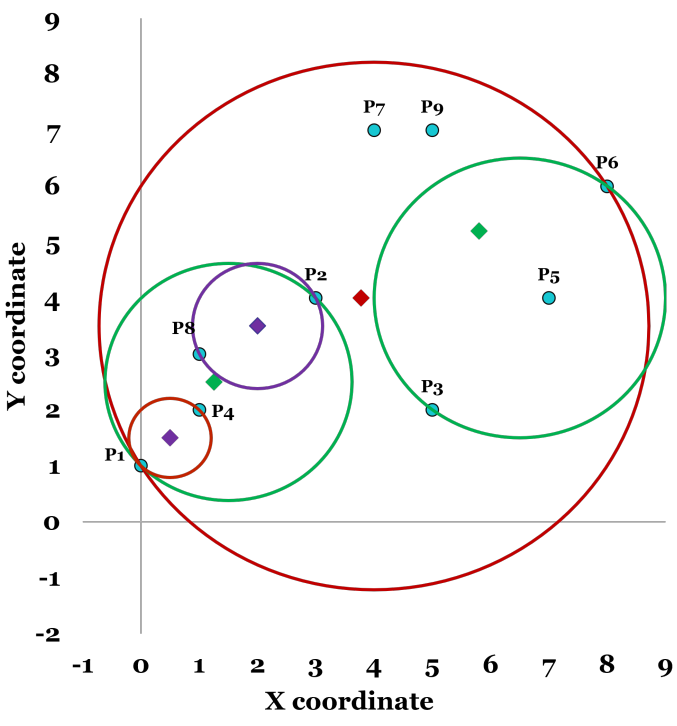


Figure 5: Ball Tree structure for the 2D spatial data

2.5.4 Octree:

The Octree is a tree data structure with superior control of 3D data representation and resolution over KD and Ball Tree. Donald Meagher pioneered the Octree at Rensselaer Polytechnic Institute in 1980 [27]. The data is stored in a hierarchical tree structure where the nodes represent disjoint cubes of exponentially decreasing size [28]. Each parent node has 8 sub-regions / children/octants described in a single primitive shape i.e. *cube* [29]. The Octree format requires spatial sorting only once and newer objects are created using the boolean operations (union, intersection, and difference) [28]. The main disadvantage of the Octree stated by Meagher [28] is the large memory requirement based on the 3D object complexity and more importantly the resolution required. The search time complexity of the Octree is close to O[$log(N)$] [30] which makes it very efficient for problems like the nearest neighbour search.

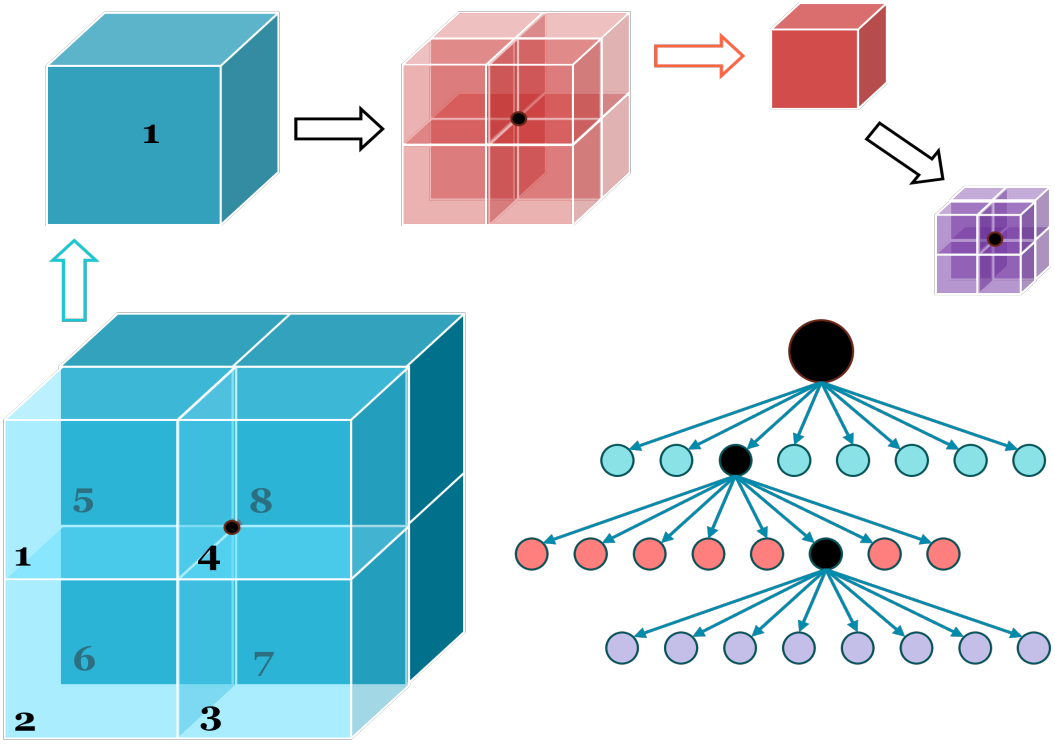


Figure 6: Multi-level Octree structure representation for a cube

2.5.5 Slice Representation Approach:

Another dynamic and expendable approach is the slice representation by [4], developed to handle large datasets in 3D space from a data management perspective and for high computation efficiency [4]. The algorithm decomposes the 3D spatial object in sequential slices along one of the axes (e.g. z) and the union of all the slices constructs the entire object Fig. 7. The advantage from the database perspective is the efficient memory utilization by loading only a few slices to perform query, manipulation, and other operations [4].

Along the axis of interest, the slices are made such that each slice plane contains at least one vertex of the 3D objects implying that the unique values of the required axis are used to create the slice planes. Compared to tree structure approaches where the storage required to store the tree structure is a concern for large volume/datasets and external storage, the slice approach can be used with a database to sequential arrange the 3D data and load the fragment of it when needed promoting fast querying over the data. The basic idea of slice representation is to decompose the object or data into non-overlapping pieces along one of the axes, namely the z-axis in Fig. 7. For simplicity, a few z-slice planes are shown in Fig. 7. The approach has 4 basic slice units for points, lines, surfaces, and volumes, and is based on the linear approximation of the 3D object.

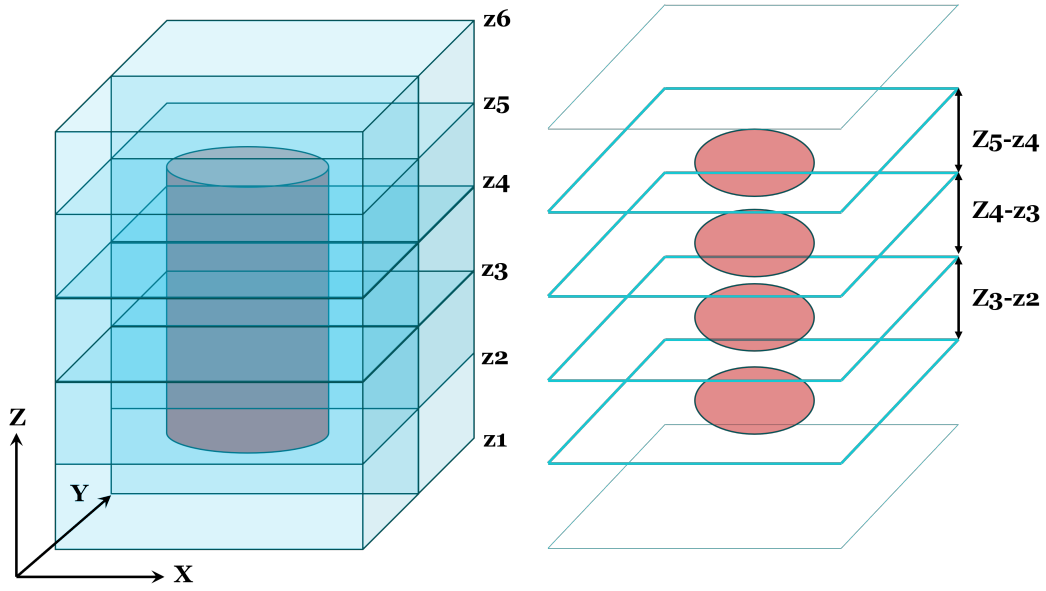


Figure 7: Slice representation of a cube

2.5.6 Bin Approach

Data discretization or categorization is another approach to converting the continuous data into discrete counterparts [31]. The principle is well utilized for data analysis and many modern programming languages provide this functionality e.g. MATLAB (discretize [32]) and Python (Pandas library [33]). The processing is referred to as binning or bucketing and aids in categorizing data in meaningful discrete intervals which can be used for further analysis. Let's take the MATLAB function, which can be used to create a specified number of Bins for a given data (X). The function returns the indices of discretized data intervals (Y) of uniform width and the Bin edges (E) [32]. For 3D spatial data, if the Bin approach is applied, the 3D space or geometry can be divided into required blocks/Bins and the nearest neighbour search can be performed Bin-wise. The Bins are created along the Z-axis in Fig. 8 but can be created along any axis of user choice. The approach is similar to the slice representation Sec. 2.5.5 but instead of taking slice planes entire interval is considered for the search.

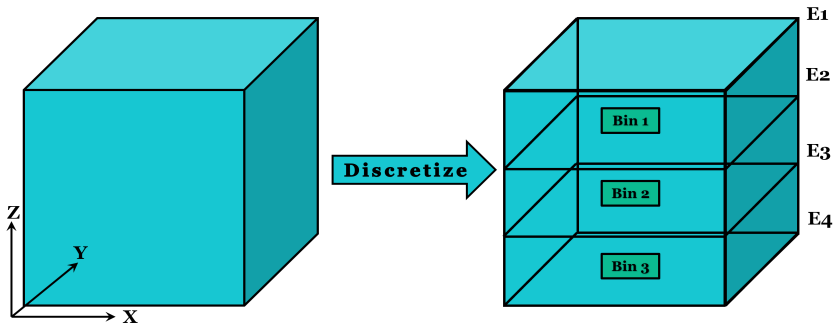


Figure 8: Discretised data representation with 3 Bins and 4 Edges

2.6 ABAQUS

The robust and powerful ABAQUS with FEM scheme makes it versatile and allows users to access the extended functionalities by using ABAQUS keywords and subroutines. The subroutines can be utilized to implement the nearest neighbour search algorithms presented in Sec. 2.5.

2.6.1 Finite Element Method (FEM)

Engineering simulation software has been used to solve real-world problems to understand physical phenomena and technological improvements better. The governing equations which are the fundamental units of the simulation model dependent on spatial and temporal resolution in the form of PDE [34]. Solving this PDE to obtain exact solutions analytically very difficult, thus approximations are used using various discretization schemes and one such scheme is FEM.

Much popular software like ABAQUS and COMSOL multi-physics use the FEM scheme as it offers great freedom to discretize the space (meshing the domain) and basis functions [34]. Another advantage of FEM is the decades of research resulting in a well-developed theory that aids in the quantification of the associated errors. The range of elements is vast in FEM for 1D, 2D (triangular, rectangular), and 3D (prismatic, tetrahedral, hexahedral, and pyramidal) applications which can also be linear (first-order) or quadratic (second order) which provide great result accuracy with the modern computation power and resources. The industry standard software has been well-tailored and utilizes certain discretization schemes for different sets of engineering problems which are well-documented in the literature.

2.6.2 User Subroutines

ABAQUS provides extended access to the functionality in the form of user subroutines which is not accessible through ABAQUS main features and can be used for flexible analysis [35] [18]. The programming language functions and subroutines are commonly used for better readability, reusability, and modularization of the code. Both are very similar to each other but yet have some key dissimilarities which are important to know.

The function and subroutine both are reusable blocks of code, but a function returns a value that can be assigned to a variable whereas a subroutine does not return a value but changes the value of original variables [35]. Another key difference is in the scope and visibility, the subroutine works within the scope of the calling function which allows the subroutine to access the variables of the calling code, but the variables defined inside are not accessible outside its scope. On the other hand, variables defined in a function have local scope, explicitly defined global variables can be accessed by the function, and calling code variables are not accessible [35]. The user sub-routines in ABAQUS are generally written in FORTRAN, C, or C++ programming languages and must included in a model. The subroutines can make use of the utility routines that are readily available in ABAQUS [18]. The requirements of subroutines written in FORTRAN and C/C++ are different to some extent in terms of must include, naming convention, etc which are explained in the ABAQUS user guide [18].

2.6.3 Full and Reduced Integration

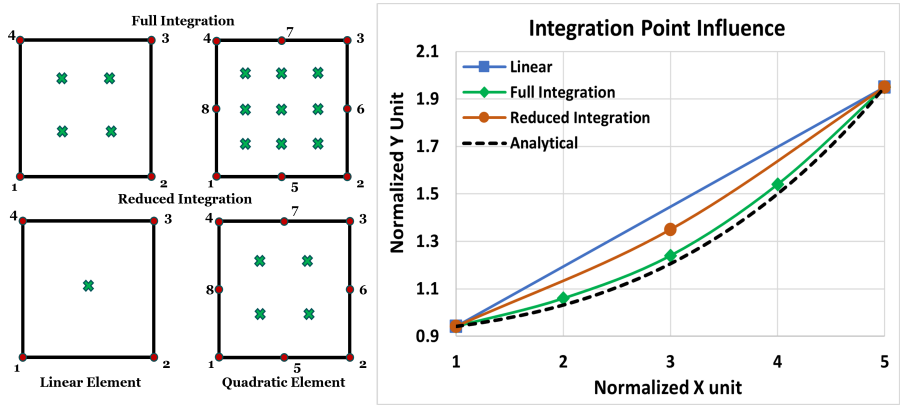


Figure 9: Full integration of linear (CPS4) and quadratic (CPS8) elements in 2D

The concept of integration points is frequently used in ABAQUS and supports full and reduced integration. The concept originates from the Gauss Integration Scheme which is a very efficient method to perform numerical integration over regular domains [36]. For a polynomial function of appropriate degree, the scheme can produce exact results. The technique increases the robustness of the FVM method.

Let us understand Full integration first, it refers to the number of Gauss points (integration points) required to exactly integrate the polynomial terms in the stiffness matrix of an element of regular shape. Fully integrated linear element (2D) uses two integration points in each direction i.e. 2 X 2, whereas for 3D it is 2 X 2 X 2. Similarly, regular quadratic elements use 3 X 3 points in 2D and 3 X 3 X 3 in 3D [18] Fig. 9. Next is the Reduced integration, it can only be used by quadrilateral and hexahedral elements. Compared to Full integration, a Reduced integration uses one lesser integration point in each direction. This is done to reduce the size of the stiffness matrix and balance the over-stiffness introduced thereby reducing the computation cost with the reduced accuracy trade-off [36] Fig. 9. The full and reduced integration techniques have their respective advantages and disadvantages for certain problems and thus require consideration before use.

2.6.4 ABAQUS Command: *SFILM

The *SFILM ABAQUS keyword is used to define heat transfer or film coefficient (HTC) and sink temperature (T_{fluid}) over the surface for a fully coupled thermal stress, heat transfer, and coupled thermal electrical analysis [18], Eq. 5, Newton's law of cooling, Sec. 2.7. For non-uniform film coefficient and sink temperature, the values must be defined in the user subroutine (FILM [18]). During the analysis, the *SFILM keyword mentioned in the ABAQUS input file directs to the FILM subroutine in the subroutine file. Under the FILM subroutine, the user can write the FORTRAN code to assign the value for HTC and SINK. The Keyword *SFILM and subroutine FILM are surface-based, the convection is applied to edges in 2D and faces in 3D [18]. The FILM subroutine snippet is readily available in ABAQUS documentation and hence not discussed in the report [18].

$$q = HTC * (T_{surf} - T_{fluid}), \tag{5}$$

where, q = surface heat flux due to convection, HTC = heat transfer coefficient, T_{surf} = temperature on the surface and T_{fluid} = SINK = reference sink or fluid temperature [18].

2.6.5 ABAQUS Command: *FIELD

Next, is the *FIELD keyword used to specify pre-defined field variable values and the keyword must be specified in the ABAQUS input file similar to *SFILM. When specified, the *FIELD directs to the UFIELD subroutine for user-defined values of the field variables during the analysis and will be called for each node. Unlike the *SFILM and FILM combination, the field variables values are prescribed at nodes of the model [18]. The user subroutine interface for UFIELD is also readily available in the ABAQUS documentation, hence now discussed in the report. The results from both *SFILM and *FIELD keywords are discussed later in the report where the salient features are highlighted.

2.7 CFD Data: SINK and HTC

The HTC is defined by Newton's law of cooling, Eq. 5, [37]. The states that "The rate of heat loss of the body is directly proportional to the difference in the temperatures between the body and its environment", Eq. 5. In the master thesis work, this heat transfer is due to the fluid movement from one place to another, and thus called Convection or convective heat transfer [38].

The temperature difference between the surface wall and the fluid in Eq. 5 is given as $(T_{surf} - T_{fluid})$ and works for both scenarios where the wall is hotter or colder than the fluid. The T_{fluid} is important for the calculation of the HTC and its determination depends on the thermal boundary layer [39] formed due to the moving fluid leading to variation in the temperature distribution near the wall or surface, Fig. 10. The focus in this thesis is the internal flow and thus the T_{fluid} is the mass flow averaged value of this temperature profile, but in the case of an external flow, this temperature is the free stream temperature [40].

However, from the CFD perspective, the geometries used for CFD analysis can be very complex leading to a mix of internal and external flow. In many CFD solvers, there are two methods to define the T_{fluid}, The first is the temperature at the centroid of the nearest cell to the wall which varies based on the turbulence model used, wall functions, wall treatment, etc and the second is the user input [40].

The T_{fluid} calculation highly depends on the near-wall mesh resolution as the centroid of the nearest cell depends on the first layer thick or first element height from the wall. To mitigate this problem a general method is employed to calculate the T_{fluid} as a function of y^+, Eq. 6 [41], instead of giving a single value as the user input for complex simulations e.g. combustion chamber analysis. The strength of this method comes from the high-resolution meshes created for CFD where $0 < y^+ < 5$ is recommended for capturing the flow physics near the wall (depending upon the turbulence model) leading to an accurate T^+ value. The T^+ value is then used to calculate the htc which is directly proportional to the T^+ value, Eq. 7 [41]. Once the $y+$ based HTC and SINK data are extracted from the CFD solvers for a high-resolution mesh it can be utilized to perform the heat transfer analysis on the same geometry with the FEM approach on a coarse mesh with integration point.

$$T^+ = f(y^+); \quad T^+ \propto \frac{T_{surf} - T_{fluid}}{q} \tag{6}$$

$$htc = \frac{q}{T_{surf} - T_{fluid}} \rightarrow \quad htc \propto \frac{1}{T^+} \tag{7}$$

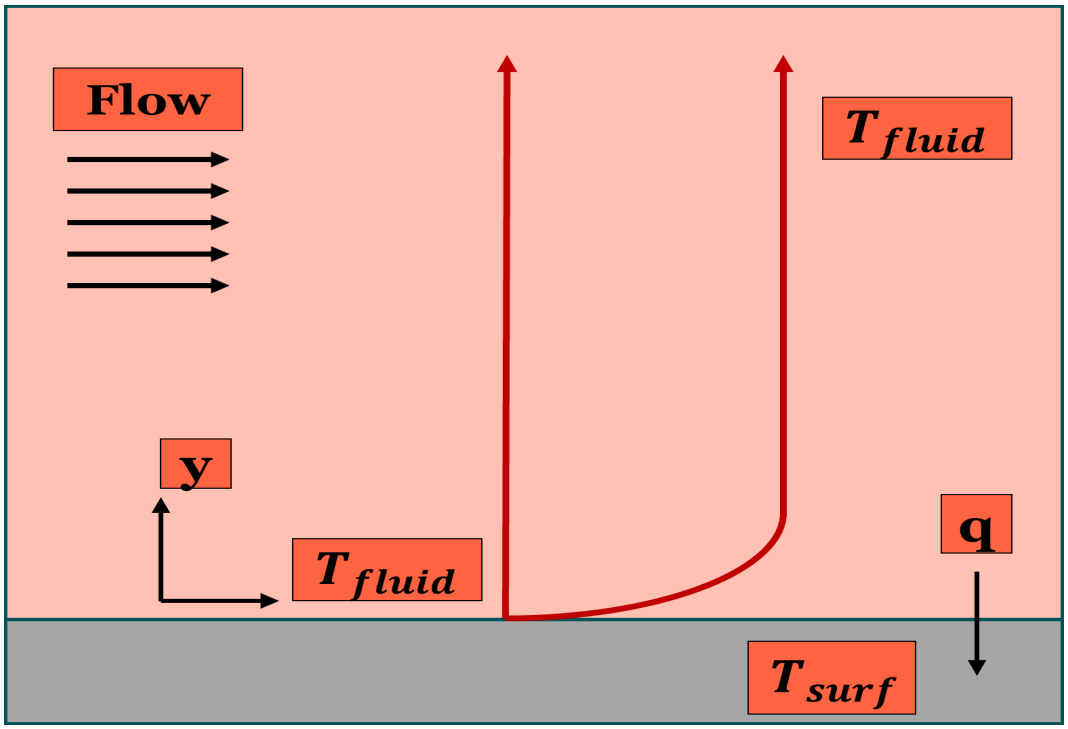


Figure 10: Velocity and thermal boundary layer

3 Methodology

The mapping is performed using the closest neighbour search method which can be implemented in several ways as discussed above. There were two methodologies implemented for the data mapping work in FORTRAN fixed form [35], namely the Brute Approach (Sec. 3.2.1) and MATLAB subroutine file Approach (Sec. 3.2.2). Each of these approaches has its merits and demerits which are discussed in the subsequent section. To embrace the merits and overcome the demerits of the implemented methodologies a few more approaches are explored for efficient mapping and reduced time complexity. The improved methodologies, namely Bin (Sec. 3.3.1) and L1_Octree (Sec. 3.3.2) Approaches are developed keeping in mind the functionalities offered by the FORTRAN90 programming language and ABAQUS for analysis. The limitations are briefly explained in Sec. 1.5 and Sec. 1.4, but are described in detail in this section. The four approaches are tested on Case1 i.e. hollow cylinder to perform the methodology benchmarking, and Case2 i.e. exhaust valve is considered as the validation of the methodologies.

3.1 Workflow and Boundary Conditions

Every analysis or simulation has a high-level workflow i.e. generic and corresponds to the software and the methodology followed. The boundary conditions of the problem statement are identified and a method is selected to create the model. The verification and validation of this model are performed to establish the credibility of the results. All these processes are described in the following section.

3.1.1 Generic Workflow Model

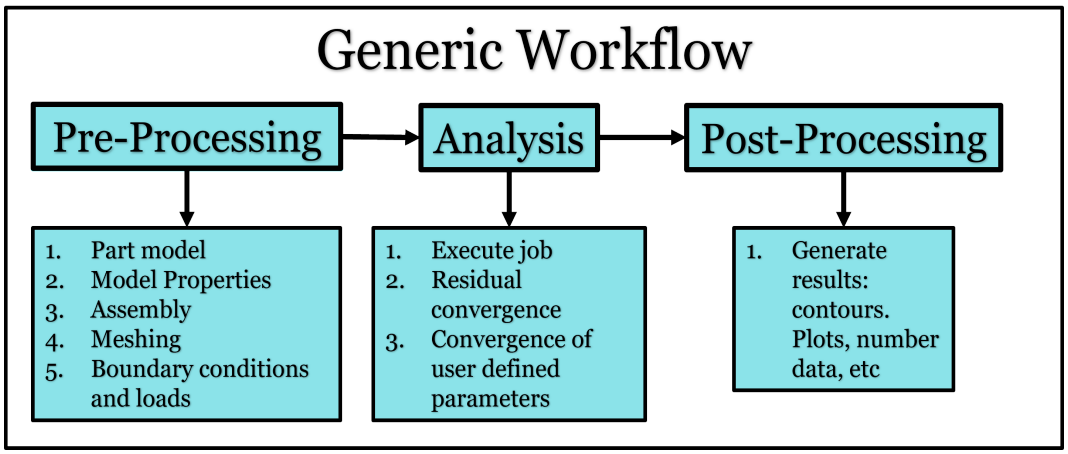


Figure 11: Generic workflow for computation analysis

The computational analysis steps are well established in the academic domain and followed across various fields. Some of the steps are optional and depend on the problem statement hence can be skipped. A simplified generic workflow is shown in Fig. 11 consisting of pre-processing, analysis, and post-processing [42]. Once the problem statement and a planning report are created, the first step is to perform the pre-processing. It consists of CAD model creation, CAD cleaning, assembly of models, meshing process (geometric discretization[42]), and defining physics (setting boundary conditions and applying loads) Fig. 11. The next step is to start the computation or solution of the model. In this step, residual convergence is monitored along with some other user-defined parameters if required which varies with the problem statement as well. Eventually, the post-processing of the solution is performed to generate meaningful contours, plots, streamlines, iso-surfaces, etc.

3.1.2 ABAQUS Analysis Steps

The Analysis step in Fig. 11 is performed after pre-processing. A simplified working of the analysis step is shown in Fig. 12. The central station of the entire workflow is the ABAQUS which takes two files namely the input file and the subroutine file. The input file contains the geometric data, boundary conditions, and load details (with ABAQUS keywords). The working of the relevant keywords is provided in Sec. 2.6.1, 2.6.2. The subroutine file contains the corresponding subroutines for the keywords used in the input file. During the analysis, ABAQUS applies loads and solves the problem for each node and integration point. The coordinates of these points are passed to the corresponding subroutines where the *sink* and *htc* values are initialized for that point. The values for *sink* and *htc* are from the closest point found in the Comma Separated Values (CSV) data file using various approaches e.g. Brute, Bin, L1_Octree, etc Fig. 12. This process repeats for all the points given by ABAQUS to the subroutine and post-completion all the points have the HTC and SINK values and the job is completed. Next is the post-processing step, where meaningful contours, plots, and other relevant charts are extracted to convey the results and present the findings from the analysis. For the present case, the results are fetched to evaluate how well the search approach has performed in finding the closest points from the CSV data file to the point required by ABAQUS and in terms of execution taken by the search approach.

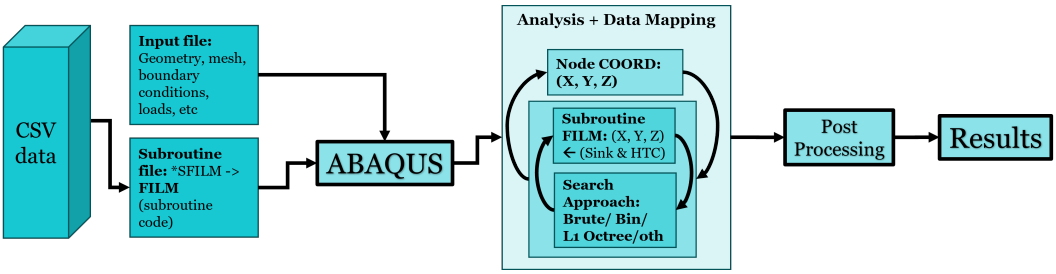


Figure 12: ABAQUS analysis flow chart

3.2 Previous Methodology

There are two existing methodologies i.e. Brute Force and MATLAB subroutine file approach following the generic flow explained in Sec. 3.1.2. These two methodologies share common theoretical concepts and yet very different as Brute Force reads the data file whereas the MATLAB approach uses data hard coded subroutine file. The merits and demerits are explained below.

3.2.1 Brute Force Approach

The approach is based on the Brute force search which is explained in detail in Sec. 2.5.1 for the nearest neighbour/closest point (Q) to a given point (P). The methodology is easy to implement as it reads the input data file (CSV, TEXT) and an ABAQUS input file which describes the model geometry, mesh, loads, solver settings, etc. These two files are fed to ABAQUS which internally calls the user-defined subroutines for each integration point in the model geometry to assign the film coefficient and sink temperature values. The data file is read only once and stored in a matrix (FORTRAN "SAVE" attribute [35]) by the subroutine program under a Mutual Exclusive Object (MUTEX) [18] [43] which is available in FORTRAN programming language.

The data reading and storing process is the same for all the subsequent approaches discussed in this section. For each integration point (P), the user-defined subroutines search all the points (Q) in the data matrix to find the closest point (smallest Euclidean distance [16]) and return the film coefficient and sink temp values of that point. The flow diagram for Brute force implementation can be seen in Fig. 13.

The Brute force approach is the simplest and easiest to implement and has several merits. It can handle small and large volumes of datasets and is compatible with both structured and unstructured mesh data (spatial data) due to the search of all the points (Q).

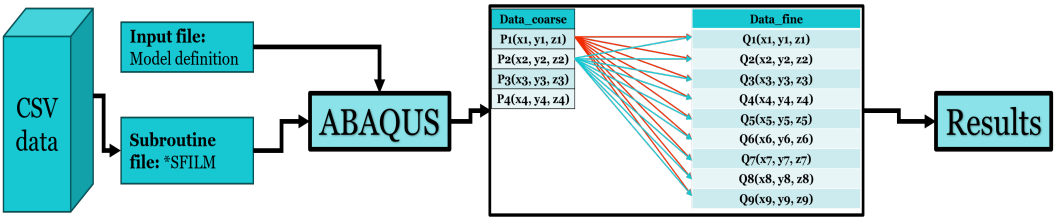


Figure 13: Brute force approach flow chart

The data fed to the Brute force does not require pre-search cleanup and data sorting as required by space partitioning approaches like the KD tree Sec. 2.5.2. However, the demerits of Brute are significant too and the associated linear time complexity [20] [19] becomes immense with large datasets. There are recommended improvements that can reduce the computation like omitting the square root operation from the Euclidean distance calculation [16] and if the exact point is found or some pre-defined threshold is achieved then the search can be stopped and the new integration point can be taken up.

3.2.2 MATLAB Subroutine File Approach

The subroutine file using MATLAB is based on the principle of Bin approach Sec. 2.5.6. In this approach the data file is read using a MATLAB script, the data sorting and cleanup are performed using MATLAB functions, and the Bin creating is performed on the sorted data along one of the axes e.g. Z-axis. The FORTRAN subroutines and the Bin data are then hard coded in FORTRAN fixed form [35] and the subroutine file is created with (.f) extension.

This resulted in a subroutine file containing the subroutines and the entire data hard-coded into separate Bins and for each integration point the corresponding Bin is searched using the Brute Force approach Fig. 13. The ABAQUS input file and the MATLAB-created subroutine files are then used for analysis giving reduced execution time compared to the Brute force as the search domain for each integration point gets reduced to a single Bin. The approach worked well with small to medium unstructured datasets i.e. random scatter points. However, the demerits of this approach are significant and may lead to execution failure due to FORTRAN limitations. The approach worked well for unstructured data but failed for medium to large volumes of structured datasets as the hard-coded file with data led to an increased number of continuation lines crossing the FORTRAN compiler limit [35]. The structured data has uniformity along the axes and thus the higher possibility of the same X, Y, or Z coordinate values being repeated. A FORTRAN compiler allows more than 39 continuation lines and the Intel FORTRAN compiler allows up to 511 lines by default (used for present thesis work) [35]. Another issue observed with the MATLAB approach is the compilation time of the hard-coded subroutine file, ranging from a few minutes to a few hours depending upon the dataset volume, but once compiled the execution time was lower than the Brute force. However, compilation time and un-compatibility with the large structured datasets cannot be neglected. Consequently, the requirement for robust and scalable methodologies becomes vital with reduced time complexity and compatibility with large datasets.

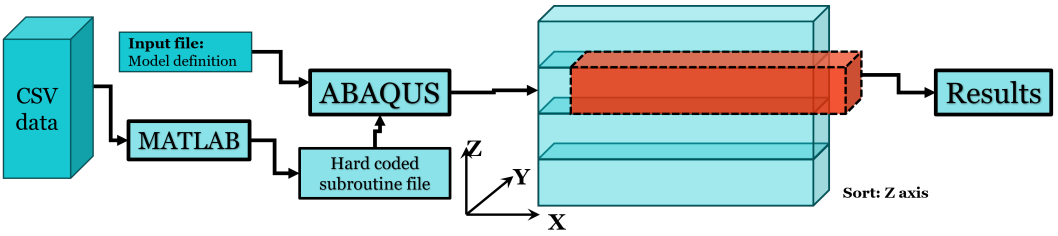


Figure 14: MATLAB approach flow chart

3.3 Improved Methodologies

The two approaches that are presented above, namely the Brute force Sec. 3.2.1 and MATLAB subroutine file approach Sec. 3.2.2 perform well but are also well associated with the demerits of time complexity, handling of large datasets, hard-coded data in the subroutine file leading to FORTRAN and compilation error in some cases. Therefore, to rectify the above-mentioned issues two new approaches are implemented and discussed below, namely Bin and L1_Octree approach.

3.3.1 Bin Approach

Similar to the MATLAB approach, the Bin approach is implemented in FORTRAN which works on the principle of data categorization and the "Discretize" function available in MATLAB [32]. The Bin approach is a combination of Discretize and Brute force which divides the data into a user-defined number of Bins and along one of the axes (Z axis in this case) Fig. 15.

At the implementation level, a UDDS to store the data with the "SAVE" attribute of FORTRAN, hence a one-time activity. The UDDS capability was introduced in FORTRAN90 [35] and allows users to create derived data types. In this thesis work, UDDS is created to store the data e.g. into Bins, and is indexed with a Bin number along with details of the Bin edges. The UDDS created for the Bin approach is utilized on the subsequent approaches with little modification to better data storage and access. This makes the identification of a Bin easier.

When ABAQUS asks for integration point coordinates for the values of HTC and SINK, the code first identifies the corresponding Bin based on the edges, and then Brute force is applied to the data points in that Bin to find the nearest neighbour. This allows a dedicated search and reduces the execution time. The distribution of data into Bins does not require any special treatment of data like sorting, thus maintaining the implementation ease.

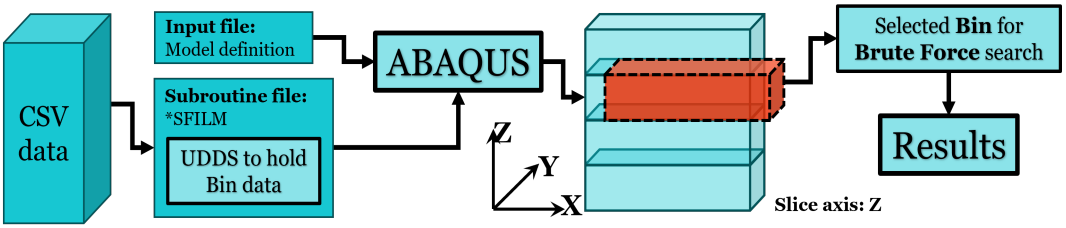


Figure 15: Discretization or Bin approach flow chart

3.3.2 L1_Octree Approach

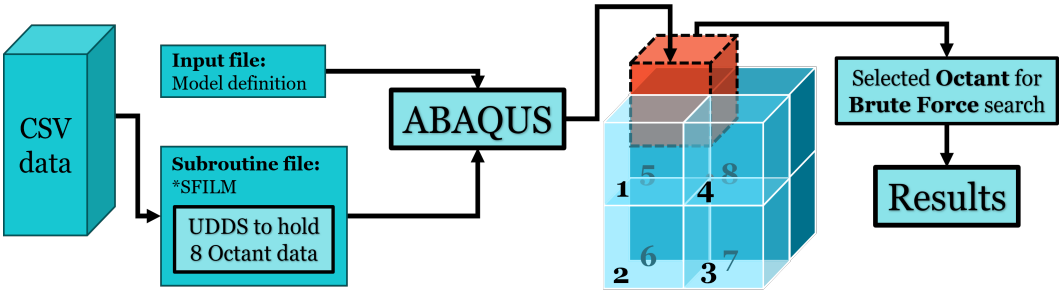


Figure 16: L1_Octree approach flow chart

Another approach implemented is the Level-1 Octree (L1_Octree) which utilizes a similar UDDS as the Bin approach to store the data. However, instead of discretizing data into Bins, octants are created in Fig. 16 Sec. 2.5.4. Here L1 implies that the entire data is divided into 8 octants only and each octant is not further divided into child octants. The theory of an Octree creation is provided in Sec. 2.5.4.

Similar to the Bin approach, no special treatment of data is required like sorting the data which makes the implementation easy. When ABAQUS asks for the HTC and SINK values for an integration point an octant is identified whose minimum and maximum dimensions could encompass the integration point and this octant is then fed to the Brute force search for the nearest neighbour data point. Once the nearest neighbour is located, the HTC and SINK values are assigned to the integration point. The Octree division is a bottom-up approach, unlike the top-down Bin approach.

Additionally, the octant centroid information is also stored which serves two purposes i.e. to identify the octant (small distance between centroid and integration point) and the octant dimension limits. These two conditions can be used to select an octant for Brute force, in this thesis only the octant dimensional limit is used. The octant centroid information can further be used to sub-divide the octants into 8 child octants and store the data in a tree form Sec. 2.5.4. Improvements in L1_Octree approach using subsequent child octant creation for multiple levels and handling the edges cases (point closer to partition face) are explained in Sec. 3.4.

3.4 UDDS Architecture and Data Storage

The UDDS is a powerful tool for data and corresponding information storage. In FORTRAN, these UDDS are called derived data types, a collection of in-built data types and other UDDS as well [35]. In this thesis work, the UDDS are utilized to create the tree structure to hold the input file data, better index the tree nodes, increase the search capability of the data subsets stored at each node, etc. There are two UDDS created for this purpose e.g. for the L1_Octree approach, one UDDS holds the information and data for 8 child octants along with a copy of the parent data at each octant, Fig. 17. The second UDDS is like a 1D array (Parent array) which acts like a linked list between parent and child octants and indexing to transverse the tree levels, Fig. 17.

Thus, identifying the octant where the closest point can be found is performed using the 1D array *index* value $\in \{1, 9\}$ and the *Oct_id* $\in \{1, 8\}$. If the number of data points in the selected octant is lesser than a threshold value (user-defined) then the parent data of that octant (copy stored at octant level) is searched with the Brute Force approach for the nearest neighbour of the given point.

The architecture helps in storing large volumes of data with additional information that can be utilized to perform pre-checks on octants, indexing, parent-child relation, index, etc. Multiple UDDS can be created for multiple data files and the switching between these UDDS can be controlled using a user-defined identifier for each. Therefore, aiding in a narrowed-down search with octants and sub-octants partitioning.

Two levels of partitioning is shown in Fig. 17, Level-1 = 1 (8^0) oct_obj and Level-2 = 8 (8^1) oct_obj. A Level-3 partition will lead to 64 (8^2) octants. For the master thesis work the partitioning is set to Level-1 resulting in 8 child octants which are stored in 1D parent UDDS for easy accessibility without using FORTRAN pointers. For Level-1 partitioning i.e. 8 octants the number of edges of one octant with its neighbouring octants (3) gives rise to 12 common edges (faces) and drastically increases for Level-2 and Level-3 partitioning. In this manner, the code implementation complexity is reduced for the users and the debugging is the code for any adverse cases is easier.

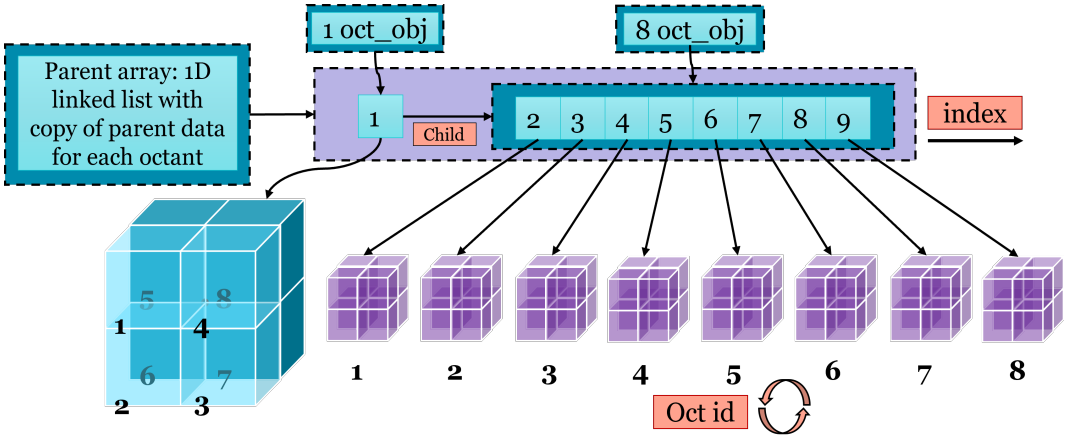


Figure 17: UDDS and data storage architecture for L1_Octree approach

Similar to the L1_Octree approach, the Bin approach also utilizes two UDDS. One to hold the Bins and the data, and the second parent UDDS to hold the Bin UDDS. This way it allows for multi-level or axis data partitioning. However, similar to the L1_Octree approach the Bin approach is also set for Level-1 partitioning along the most scattered axis, and the number of Bins are user-defined. The Bin storage example in Fig. 18 shows that 4 Bins are stored in the parent UDDS with the corresponding data and relevant information for each Bin.

An important note for the Bin approach is the number of Bin edges if the number of Bins is N and the number of Bin edges common between two Bins in total is $N+1$ along a single partition axis.

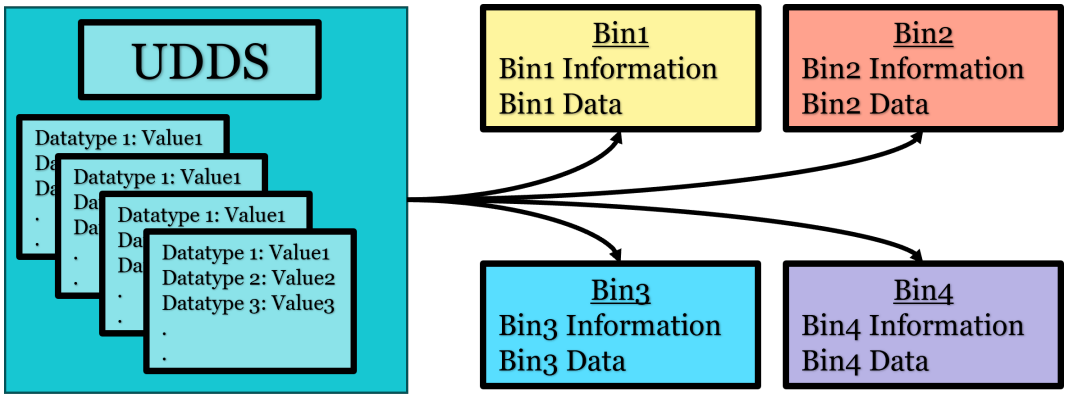


Figure 18: UDDS and data storage architecture for Bin Approach

Handling Edge Cases: As the partition level increases the number of partition edges increases drastically which needs to be handled. The example of 3 Bins and 4 edges where the *coordinates* (black) finds the closest point in Bin 1 is shown in Fig. 19, but if the neighbour (Bin 2) is checked the closest points will be found this situation is called the Edge cases. To handle this situation a simple technique is used i.e. to store the data of the common edge of two Bins in both Bins but the *coordinates* search is limited by the Bin edges (blue). The extent of this overlap domain is controlled using a *Fraction* parameter, thus the Bin holds extra data along with Bin width ΔBin, Fig. 19. This *Fraction* extends to both left and right sides for the internal Bins and left or right for the boundary Bins and in all directions for L1_Octree.

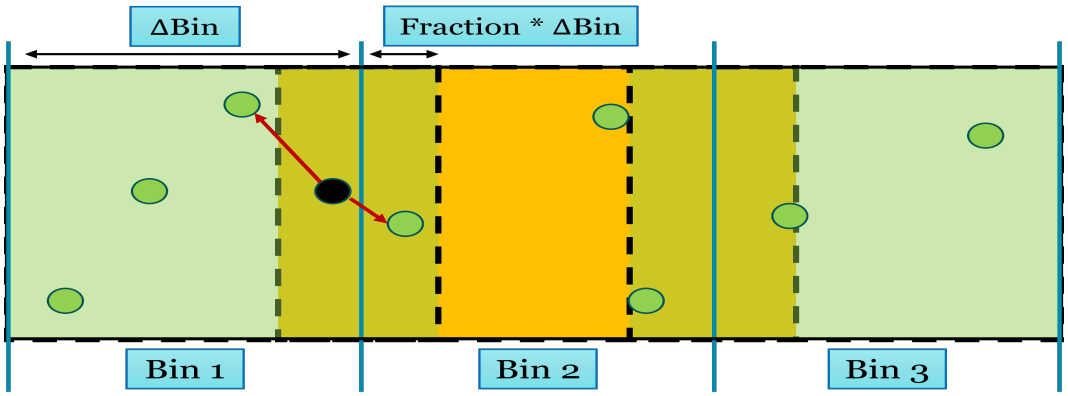


Figure 19: Edges case handling for L1_Octree and Bin Approach

3.5 Detailed Algorithm Flow chart

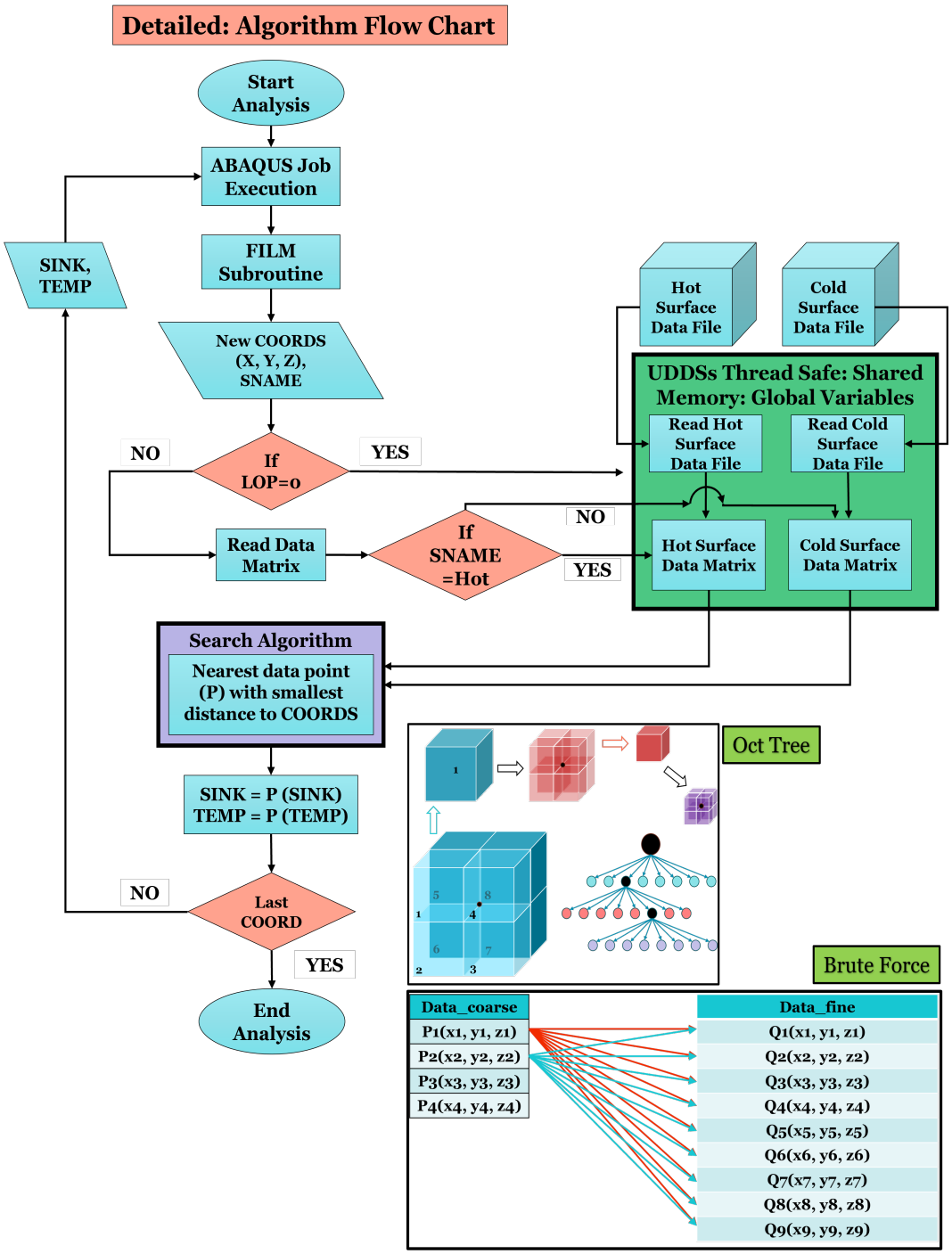


Figure 20: New Algorithm Flow chart

A detailed flow chart of the algorithm combination (Brute + Octree) developed during the thesis work with the introduction of UDDS (green block) is presented in Fig. 20. The start of the analysis is marked by the job submission i.e. the input and the subroutines files are provided for the ABAQUS analysis.

The *SFILM keyword in the input file calls the FILM subroutine from the subroutine file for the mesh node or integration point coordinate (*coordinates*) along with the Surface Name (SNAME) to which that *coordinates* belongs and where the boundary conditions are applied. A total of two files are used, one with Hot and the other with cold surface data (.csv) for the corresponding two surfaces (Hot and cold surface), Fig. 20,. The data files are read when User subroutine to manage external database (UEXTERNALDB) subroutine is called [18] with LOP = 0 meaning at the time of initialization only i.e. the data files are read only once in the entire analysis and the data is stored in the two UDDS. This UDDS information is stored using the *save* attribute for the variables.

The call to UEXTERNALDB subroutine is made under the MUTEX thread locks [43] which also gets executed once during the entire analysis. This combination creates a thread-safe shared memory for the UDDS. The UDDS and the associated variables are declared and instantiated under FORTRAN MODULE for global use and updation. Therefore, reading data files, storing data into UDDS, and all the global variables are instantiated at the start of the analysis and performed only once. Subsequent use of these shared memory UDDS (read-only) and global variables (read-only) is performed by the Search Algorithm (Brute + Octree) based on the SNAME which serves as a switch between hot and cold surface and corresponding UDDS data. Once the UDDS is selected, the search for the *index* and *oct_id* Sec. 3.4 is performed for an octant potentially holding the closest neighbour point to the *coordinates* point.

Post, octant selection for the octant data i.e. the subset of the entire data is fed to the Brute Force search to find the nearest neighbour to *coordinates*. The HTC and SINK values corresponding to the nearest neighbour are sent to the ABAQUS for analysis and the entire process repeats for the new *coordinates* until the nearest neighbour to all the *coordinates* points is searched and then the execution stops Fig. 20. Similar to the octant search is the Bin search before the Brute Force where the number of Bins before an important parameter is provided by the user. The Oct Tree approach option can be changed to the Bin approach, Fig. 20 which is preferred for a geometry with one major axis.

3.6 Case1 Bench-marking: Hollow Cylinder

The verification of the two improved implementations is performed on the hollow cylinder with two holes case Sec. 3.6.1 and Fig. 22. The verification of the approaches is evaluated based on the job (analysis) execution time, contours of remapped data, and the plots of HTC and SINK values to provide visual means for better understanding. The Wall Clock and CPU time are selected along with the Compilation and Pre-Run of the FORTRAN code because the FORTRAN code is compiled first and then executed.

Various meshes are created for the case geometry with HTC and SINK values generated using Eq. 8 and 9 respectively. These equations are based on the coordinates of the integration point only and do not comply with the physics. These equations only aim to make HTC and SINK values solely dependent on the coordinates and obtain an Analytical solution reference for post-processing of the results.

$$sink = \sqrt{|coordinate(Z)|} + 30 \tag{8}$$

$$htc = \sqrt{(|coordinate(X)| + |coordinate(Y)|)} + 100 \tag{9}$$

where *coordinates* is the coordinate of the integration point asked by ABAQUS during the analysis.

For a basic understanding of the user, a brief theory of the difference between compilation and execution of a FORTRAN code is provided. The compilation process converts the user-written FORTRAN code into computer-readable form, thus every line of code is checked to result in an object or executable file for the code [44] (a good source for FORTRAN basics). Certain syntax-related issues or lines of code incompatible with FORTRAN are identified by the compiler like the continuation line error discussed in Sec. 3.2.2. Once the compilation is completed, the FORTRAN code is executed and execution errors are identified like the provided file directory is invalid, etc.

3.6.1 Case1: Boundary conditions

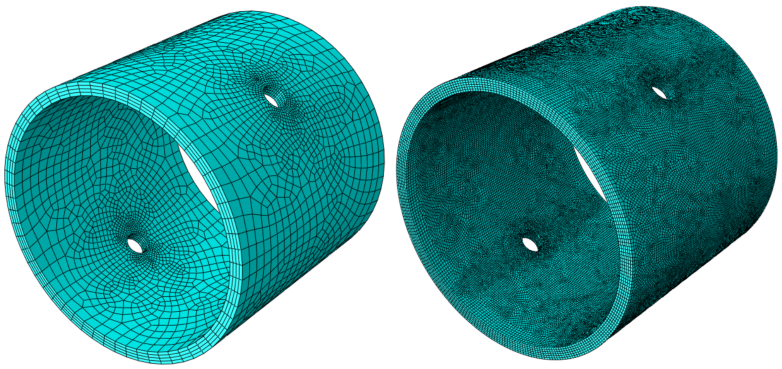


Figure 21: Coarse (left) and fine (right) mesh: Case1 Hollow cylinder

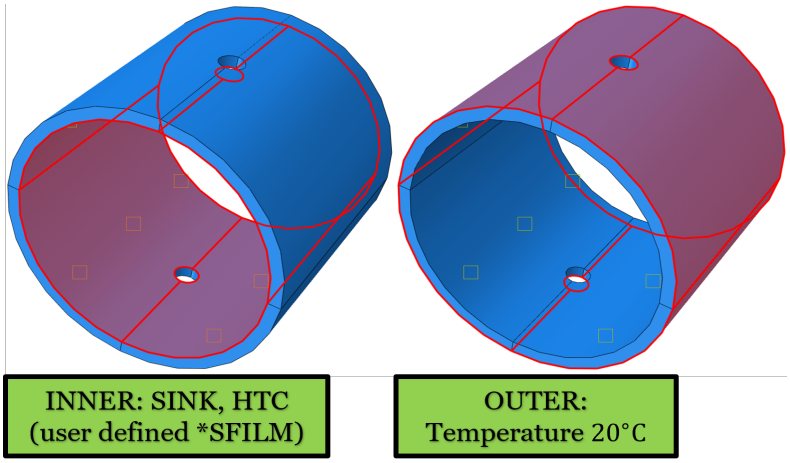


Figure 22: Boundary conditions for Case1: Hollow cylinder

As a benchmark case, a hollow cylinder with two holes is selected so that the mesh refinements can be applied near the holes resulting in high node density Fig. 21. In the present thesis, the data of a finer mesh is mapped onto the mesh of a coarse mesh, thus for a visual representation both are shown in Fig. 21.

Post mesh creation, the boundary conditions and loads (thermal loads in the present case) are applied at the required regions. For the entire geometric model the *Conductivity, *Density, *Elastic, *Expansion, *Specific Heat are defined in the ABAQUS input file. The presence of **(*)** implies ABAQUS keywords [18]. The ABAQUS model uses two types of analysis steps 1: Initial step ($20°C$ at all nodes) and 2: Analysis Step. The initial step is used to apply boundary conditions, predefined fields, and interactions required at the start of analysis [18].

Whereas, multiple analysis steps can be added depending on the type of analysis (steady or transient state) [18]. In the present case, only one analysis step is included to achieve a steady-state analysis. The boundary conditions are applied on the inner and outer surfaces of the model at the initial step but for the analysis step the HTC and SINK are user-defined and applied using *SFILM ABAQUS keyword [18] Fig. 22.

4 Results

The four methodologies presented in Sec. 3 are implemented and evaluated qualitatively and quantitatively using various contours, plots, and histograms. The obtained results for each of these methodologies are compared against each other to highlight the quality of mapping and time complexity for Case1 i.e. hollow cylinder and Case2 i.e. exhaust value. The performance influencing factors like data sorting, level of partitioning (No. of Bins), and multi-processor execution are also evaluated.

4.1 Case1: Hollow Cylinder Mapping

The benchmark case is presented below to evaluate the quality of data mapping onto a coarse mesh mesh for half of the path (red). The coarse mesh is shown in the top-left of the and the SINK values for meshes $\in$ {Mesh1 with 33612 data points (M1), Mesh2 with 98292 data points (M2), Mesh3 with 249324 data points (M3), Mesh4 with 421788 data points (M4)} are remapped using various approaches against the Analytical solution, Fig. 23. The Brute force trends are also the reference since it is the base approach. **Note:** The results for the MATLAB subroutine file approach are not shown for an important reason mentioned in the following paragraphs.

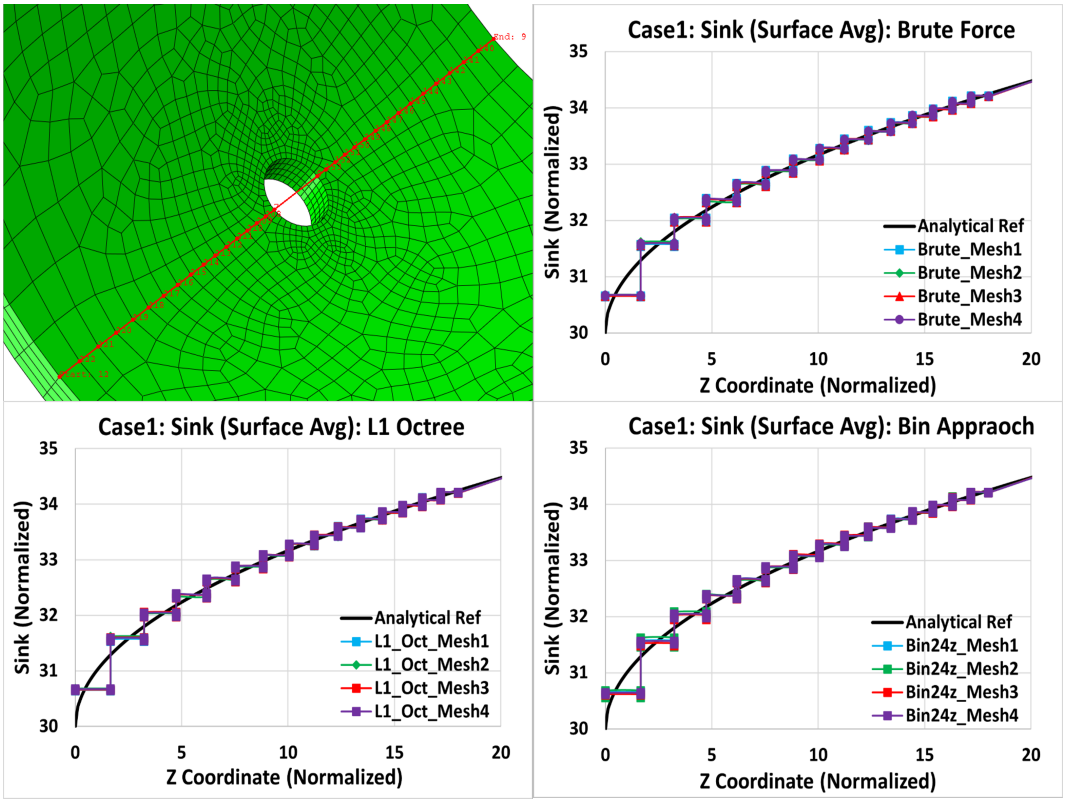


Figure 23: SINK temperature remapping comparison with Analytical data

The remapped data plots are in stair-step form because the SINK value is the element face average (*SFILM) Sec. 2.6.4 as shown in Fig. 23. The average is taken over the element face nodes and integration points. The Brute, L1_Octree, and Bin approach trends follow the Analytical solution well.

The trends are overlapping for all the meshes $\in$ {M1, M2, M3, M4} and negligible discrepancies are observed for the Bin approach (bottom-right) Fig. 23 for 24 Bins. For L1_Octree (bottom-left) the trends show negligible discrepancies (almost the same as Brute). The stair-stepping has higher jumps for Z $<$ 5 and convergences well for 5 $<=$ Z. For visual understanding Fig. 24 shows the SINK contour when M4 data is mapped onto M1 mesh using various search approaches. The contour resolution of the M4 mesh is better than the M1 mesh as the number of elements in M4 is significantly higher (bottom-left). The Base M1 is the reference contour in Fig. 24. The M4 SINK values are remapped using the Brute, Bin, and L1_Octree approach and the contour shows negligible discrepancies and captures the resolution on the surface. The SINK contour Fig. 24 and plot Fig.23 results can be seen in conjunction to understand the quality of data mapping. **Note:** No legend for contours in Fig. 24 25 as those are random values based on Eq. 8 9.

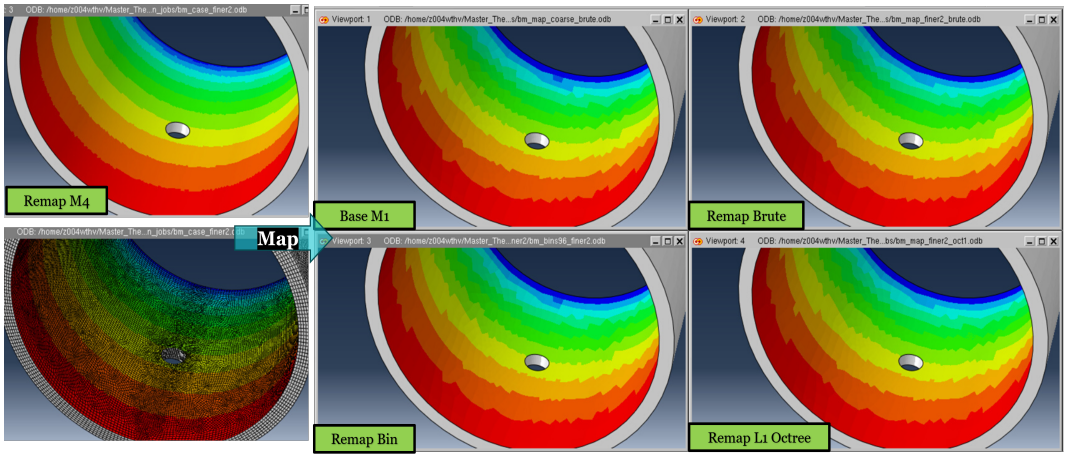


Figure 24: Remapped normalized SINK data contour onto M1

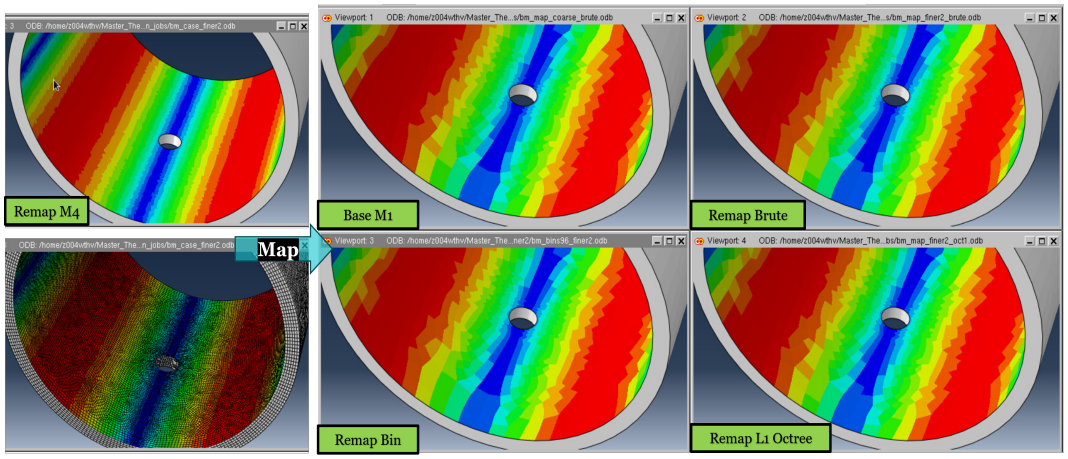
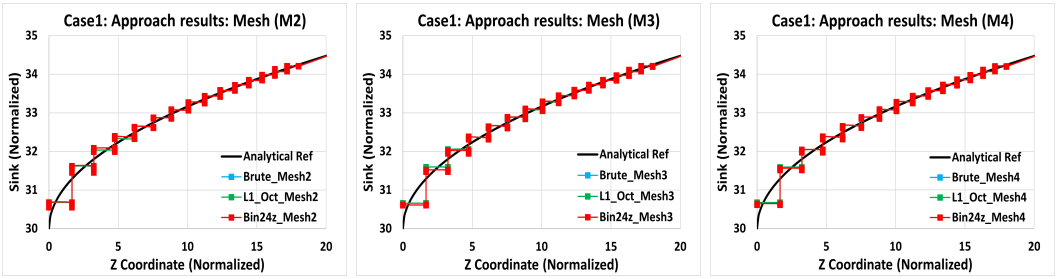


Figure 25: Remapped normalized HTC data contour onto M1

Similar to the SINK contour in Fig. 24, the M4 HTC values are also remapped using the Brute, Bin, and L1_Octree approach, and the contour shows negligible discrepancies and captures the resolution on the surface Fig. 25. As the HTC values are the function of X and Y coordinates Eq. 9, the contour variation is profound and well captured by the mapping process.

The comparison of the four approaches w.r.t the meshes for mapping the SINK values is shown in Fig. 26. For all the meshes the stair-stepping plots are in good agreement and almost overlapping. However, some minute discrepancies are observed for the Bin approach compared to Brute and L1_Octree (almost overlapping). The plots are also in great agreement with the Analytical solution for all three meshes. The stair-stepping has higher jumps for Z coordinate < 5 and convergences well for higher Z coordinate values. This is similar to what is observed in Fig. 23.

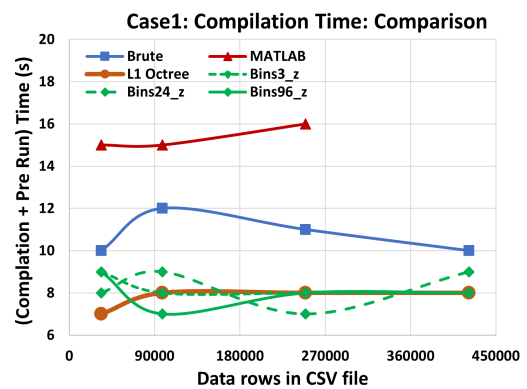


((a)) M2 remapping ((b)) M3 remapping ((c)) M4 remapping

Figure 26: Comparison of remapped SINK by different approaches on meshes

The approaches are evaluated on the Wallclock time, Total CPU time, and the Compilation time taken by the analysis, Fig. 27. The definition and significance of each of these time variants are provided in Fig. 27(b) for the reader's understanding. Let's highlight the MATLAB approach trend for which the curve truncates as the approach fails to generate the subroutine file due to FORTRAN errors hence the approach is not considered for further analysis, Fig. 27. The results of Compilation and Pre-Run (linking of input and subroutine file) time are compared, Fig. 27(a). The MATLAB approach shows an increasing trend whereas the Brute approach is stabilizing towards the larger dataset. The L1_Octree and Bin approach compilation time is well stabilized and reduced by 20% - 50% w.r.t Brute and MATLAB approach respectively.

Similar trends for all the approaches on the Case1 bench-marking are displayed for the CPU time Fig. 27(c) and Wall Clock time Fig. 27(d). The L1_Octree approach (8 Octants only) shows significant improvement in CPU time with a reduction of 44.85% - 80.68%, from lowest to highest data rows in the CSV file respectively, Fig. 27(c). For the Bin Approach, the reduction in CPU time improves with an increment in the number of Bins. The improvement for 3 Bins w.r.t to Brute is 20.15% - 49.29% which increases to 40.44% - 65.29% for 24 Bins and so on. However, the difference between 24 and 96 Bins is small. The MATLAB approach showed a tremendous reduction of 62.94% - 92.40% in CPU time w.r.t Brute Force for the first three meshes (M1, M2, M3) but failed for M4. Similar trends and % reductions are observed in the Wall Clock time compared to CPU time with negligible differences, Fig. 27(d).



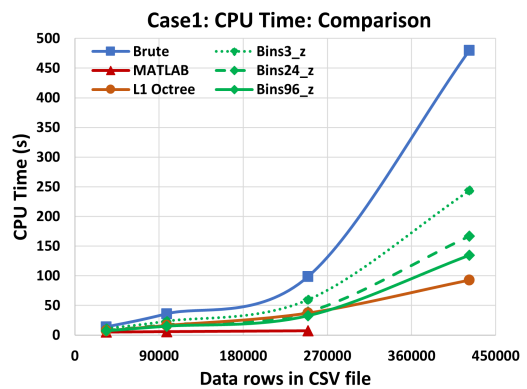
((a)) Compilation and Pre-Run time comparison

Wall Clock Time: The time elapsed between the start and completion of the simulation. Includes computation and resource wait time. Influenced by the available memory, system load, concurrent processing by the machines, etc. Holistic description of computation time and system delays.

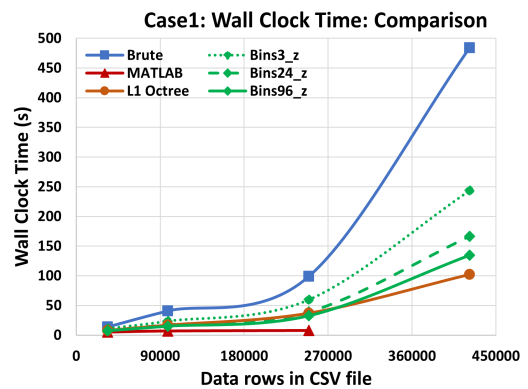
Total CPU Time: Sum of User CPU and System CPU time. Amount of time spend on user specific tasks or code and relevant to evaluate efficiency of the simulation code is User CPU time. The time spent in kernel mode i.e. service system calls is System CPU time. It includes memory management, system task, input-output operations. Gives an overview of system load and resource utilization. Influenced by parallel processing, system loads, hardware specification, etc

Compilation Time: The conversion of high-level code (FORTRAN) into low level or assembly level instructions.

((b)) Types of Time variants



((c)) CPU time comparison



((d)) Wall Clock time comparison

Figure 27: CPU and Wall Clock time evaluation of approaches on single core

The average time taken for each search implementation to find the nearest neighbour to the reference point is highlighted in Fig. 28. For all four meshes, $\in$ {M1, M2, M3, M4} the Brute Force approach takes the highest amount of time followed by the Bin3_z (Bins along the z-axis). As the No. of Bins increase from 3 to 96 the average time reduces significantly highlighting the influence of the number of partitions. The average time for L1_Octree is comparable to Bin24_z to Bin48_z for all meshes, Fig. 28. The MATLAB approach with hard-coded data values show a reduction in average time w.r.t L1_Octree as the mesh size increases, but the MATLAB approach had failed for M4 as explained earlier, thus results are not available for M4, Fig .28

Another set of important results is shown in Fig. 29(a), which is only obtained for the Bin approach for various numbers of Bins. Some algorithms like KD Tree mentioned in Sec. 2.5.2 required data sorting Sec. 2.4 along the axis of partition, thus becoming an important influence parameter for performance. The Wall Clock time for the Bin approach which internally sorts the data along the Z axis results in an unsteady trend and no improvement in Wall Clock time even for small datasets, Fig. 29(a). Whereas, the Bin approach implemented with no internal sorting as mentioned above shows great improvement and stabilized trends. Next is the influence of Bins on the Compilation and Pre-Run time which shows the mean trend of approx 8 seconds with small deviations of all the meshes used for Case1.

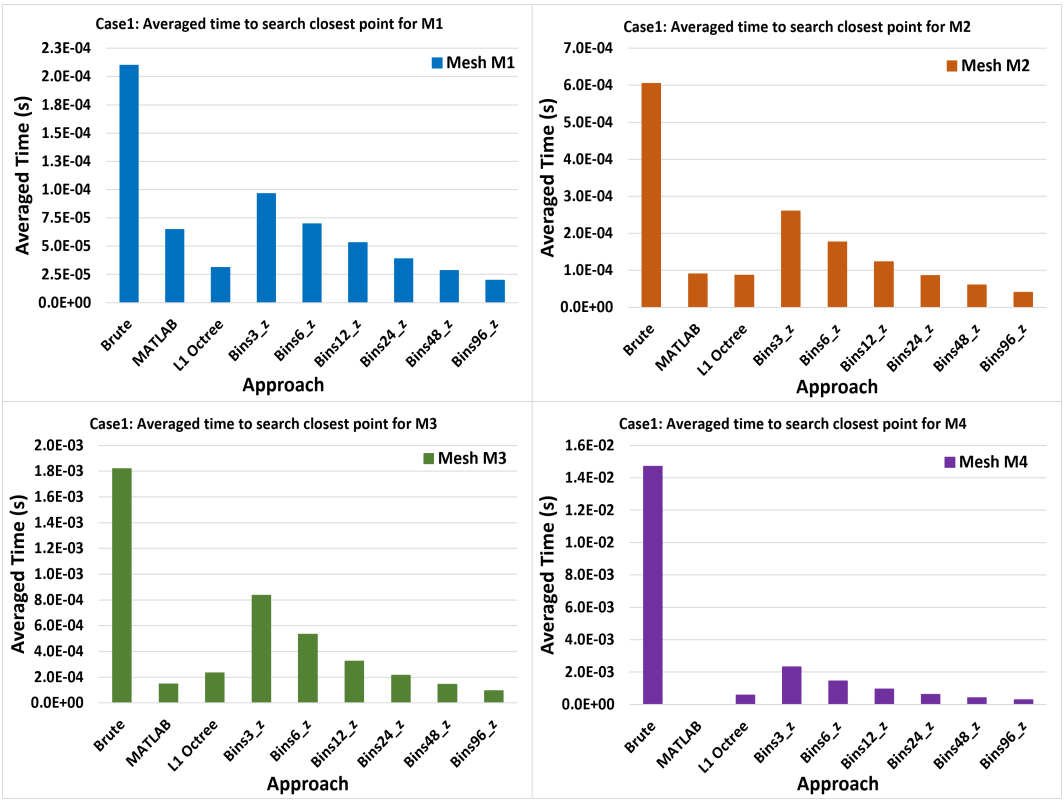
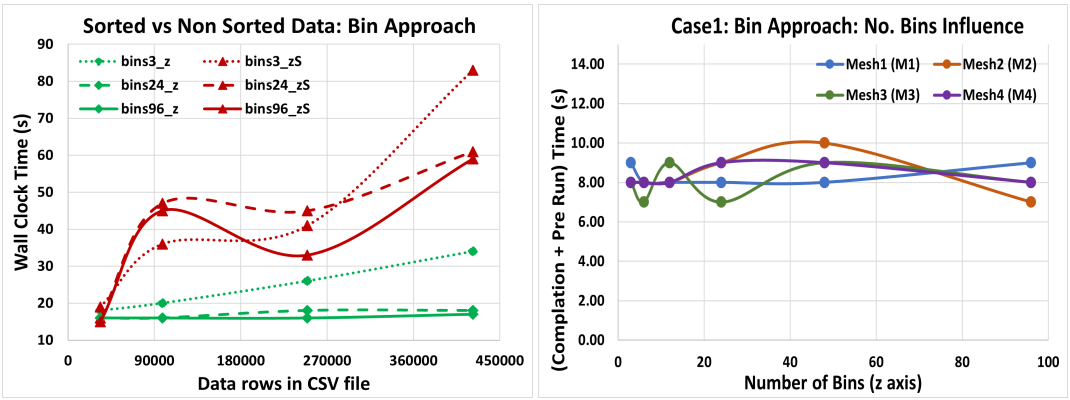


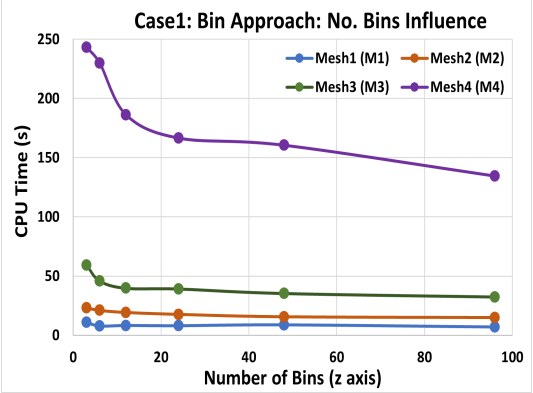
Figure 28: Case1: Average time (subroutine call) to search the closest point on a single core for various mesh sizes associated with small case



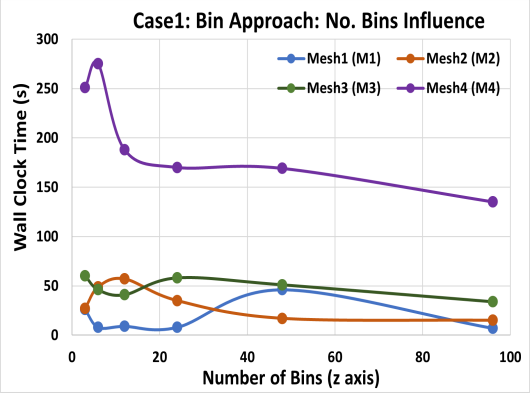
((a)) Data sorting influence on Wall Clock time ((b)) No. of Bin influence on Compilation time

Figure 29: Case1: Evaluation of Bin approach w.r.t data sorting and compilation time

The influence of No. of Bins on the CPU and Wall clock time for all the meshes is shown in Fig. 30. The minimum of 3 and maximum of 96 Bins are evaluated and show similar trends for both CPU and Wall Clock time. The CPU and Wall Clock time reduces with increments in Bin number and stabilizes for higher Bins. For mesh M4, the CPU and Wall Clock time offset is significant compared to other meshes and with the increasing number of Bins, the maximum reduction observed for CPU time is 44.71 % and Wall Clock time is 50.99 %.



((a)) No. of Bin influence on CPU time

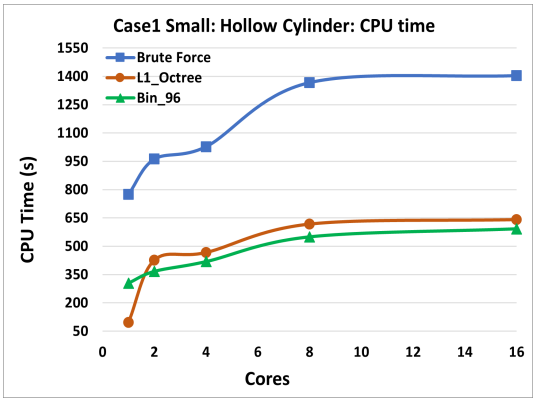


((b)) No. of Bin influence on Wallclock time

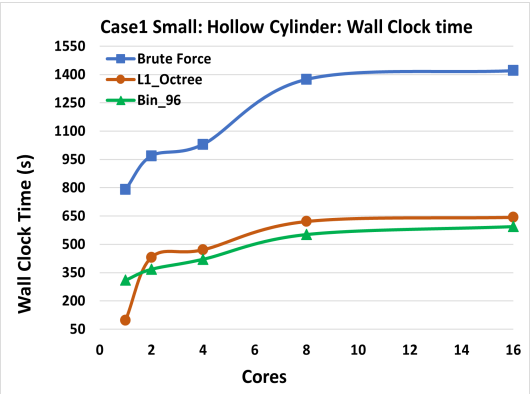
Figure 30: Performance evaluation of Bin approach w.r.t CPU and Wall Clock time

4.2 Case1: Multi-Core Execution

The Brute Force, L1_Octree, and the Bin approaches are implemented with thread-safe shared memory explained in Sec. 3.5 that can be shared across multiple threads or cores for parallelization of the analysis to faster simulation. The results in Fig. 31 are presented to show the influence of the multi-core analysis for a small case i.e. 11204 mesh elements. Both the CPU and the Wall Clock time increases with the increment of several cores which is contrary to the purpose of multi-core simulation (discussed in the following paragraph). For all three approaches, the CPU and Wall Clock time converge for cores $>= 8$, and for cores < 8, the increase in time is significant. The L1_Octree approach reduces the CPU time by 87.5% and the Wall Clock time by 87.6%, whereas the Bin approach reduces the CPU time by 60.72% and the Wall Clock time by 60.81% for 1 core, Fig. 31. The % reduction is almost constant for all the cores up to 16 cores w.r.t Brute Force approach. The L1_Octree and Bin (96 Bins) approach show a negligible difference, but for 1 core the time taken by L1_Octree is significantly smaller than the Bin approach i.e. 68.18%, for the rest of the cores the offset in the curves is constant.

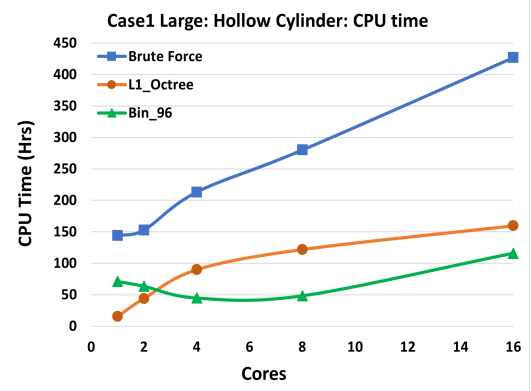


((a)) No. cores influence on CPU time

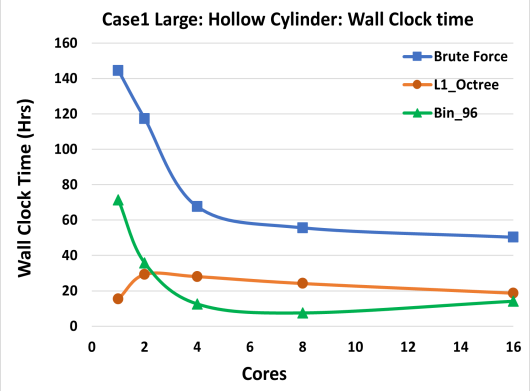


((b)) No. of cores influence on WallClock time

Figure 31: Benchmark: Influence of No. of Cores for a small case with 11204 elements



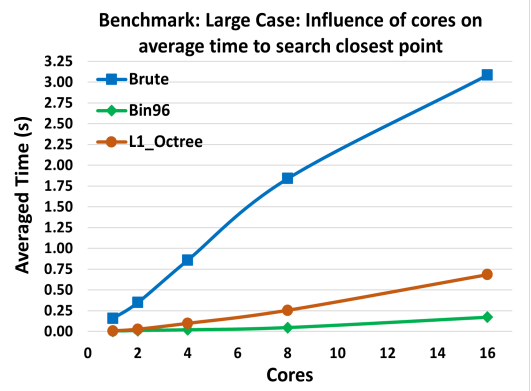
((a)) No. cores influence on CPU time



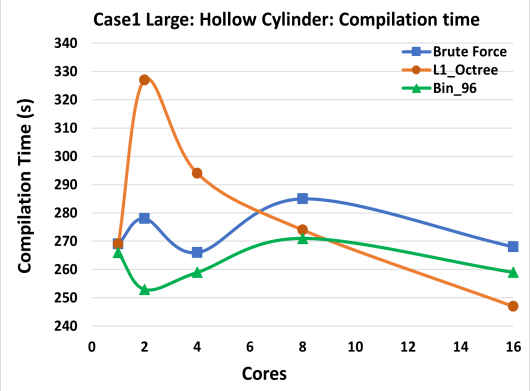
((b)) No. of cores influence on Wallclock time

Figure 32: Influence of No. of Cores for a large case with 2138819 elements

The performance is evaluated for a larger case with all 2138819 quadratic elements, Fig. 32 which is similar to Fig. 31. The CPU time in Fig. 32(a) shows that Bin_96 approach outperforms the Brute and L1_Octree approach for Cores >= 2 and there onwards follows a constant offset w.r.t to L1_Octree approach. However, for a single core CPU time, the L1_Octree approach performs significantly better than the Bin_96 and Brute Force approaches. The Wall Clock time in Fig. 32(b) shows good convergence with increases in the number of cores, but the with single core L1_Octree outperforms the Bin_96 and Brute Force approach with a great margin. The clear offsets between the three approaches for both CPU and Wall Clock time can be seen, especially for the Wall Clock time where the offset trends are constant and smooth for Cores >= 4, Fig. 32(b).



((a)) Average time (subroutine call)



((b)) Multi-core influence on compilation time

Figure 33: Benchmark: Influence of No. of Cores on large case with 2138819 elements

Again, similar to Fig. 28, the results for the average time to find the closest point for the large case w.r.t to No. of Cores are in Fig.33(a). The average time increases with the cores for all three approaches and Bin_96 outshines the Brute and L1_Octree. The Brute approach takes 17.94 times and L1_Octree takes 3.98 times more average time compared to the Bin_96 approach for searching the closest point for 16 cores. The magnitude order of compilation time does not show significant variation among the three approaches Fig. 33(b).

4.3 Case2: Exhaust Value Mapping

The exhaust manifold geometry and boundary conditions, Fig. 34 and the discretized geometry with unstructured mesh with 317418 quadratic tetrahedron elements, Fig. 35. The Element type is 10 node quadratic tetrahedron element (DC3D10) in ABAQUS that is suited for diffusive heat transfer or mass diffusion.

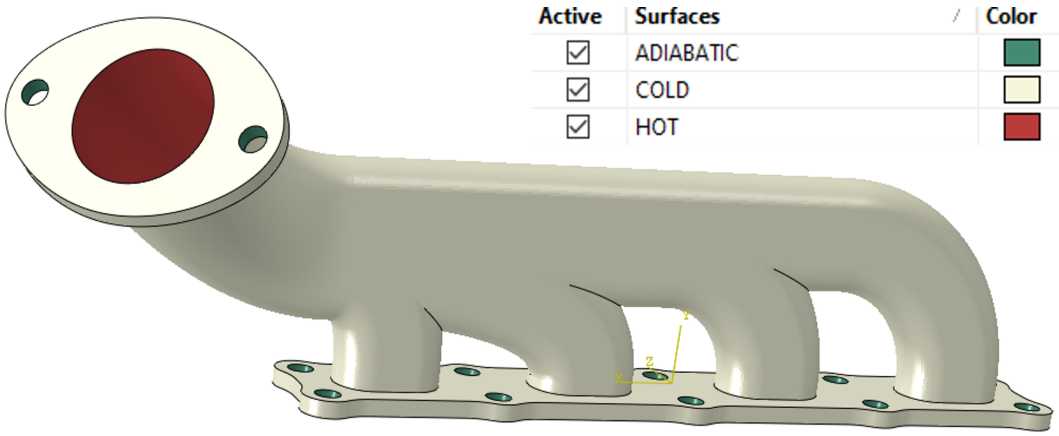


Figure 34: Validation case 1: Exhaust valve: Geometry and Boundary conditions

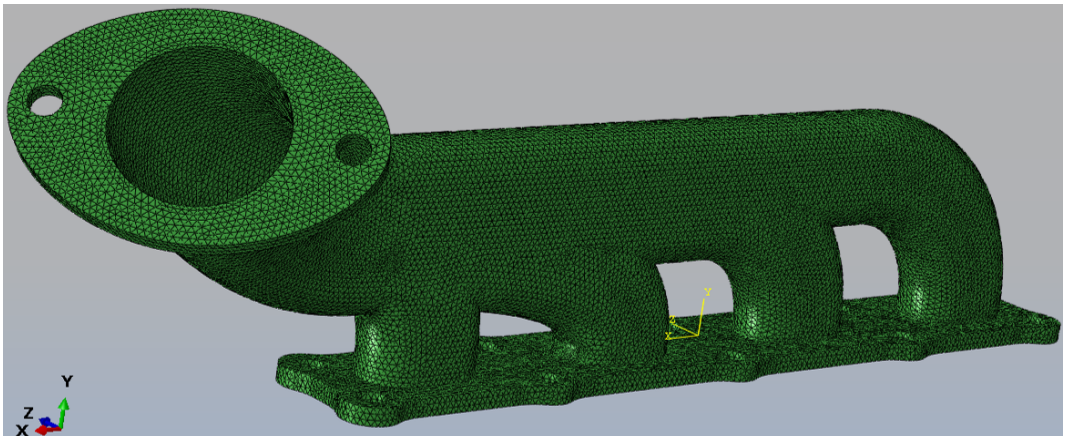


Figure 35: Exhaust valve: Mesh with 317418 DC3D10 quadratic elements

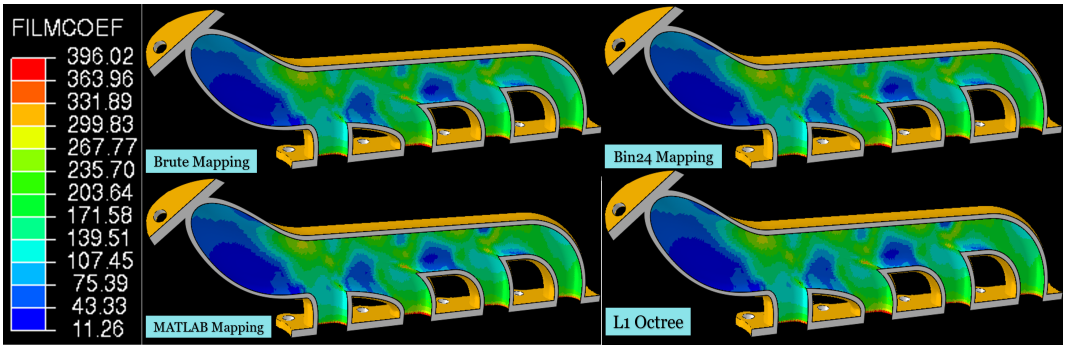


Figure 36: Exhaust valve: Mapping comparison for HTC $(W * K/m^2)$ contour

The thermal properties like thermal conductivity, elasticity, expansion factor, and specific heat are material specific and temperature dependent, hence provided for $20°C <$ temperature $< 1000°C$ in the ABAQUS input file. The SINK (K) and HTC $(W * K/m^2)$ values are assigned using *SFILM Sec. 2.6.4 on the HOT and COLD surfaces using Brute Force, MATLAB subroutine file approach, Bin (24 Bins), and L1_Octree approaches Fig. 36.

The post-mapping HTC contours are shown in Fig. 36 where Brute and MATLAB approaches serve as the reference or base as they were the implemented and working approaches. The HTC mapping using Bin and L1_Octree approach is in good agreement with the base approach and well captures the HTC transitions. The high HTC concentration near the pipe interface and inlet pipe edges is also captured along with small regions of lowest HTC values Fig. 36. This implies that the Bin and L1_Octree approaches work well to find the closest node from the data files during the ABAQUS analysis and the corresponding SINK and HTC values are assigned.

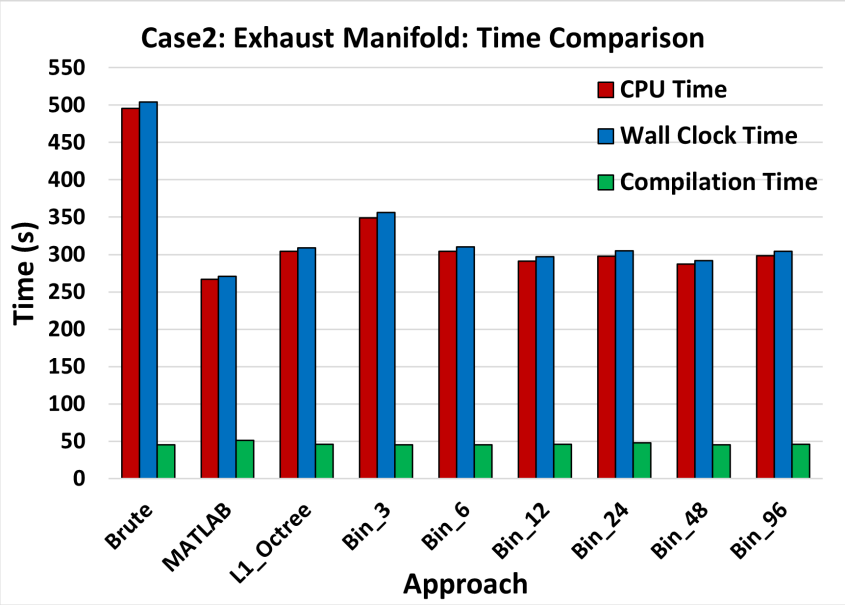


Figure 37: Exhaust valve: Wall Clock and CPU time (s) comparison

Table 2: Time improvement (%) of approaches w.r.t Brute Force, (-) = Increase

Time	Brute	MATLAB	L1_Octree	Bin_3	Bin_24	Bin_96
CPU	0	46.17	38.69	29.67	39.90	39.85
Wall Clock	0	46.23	38.69	29.37	39.48	39.68
Compilation	0	-13.33	-2.22	0	-6.67	-2.22

Next is the CPU, Wall Clock, and Compilation time taken by each approach for the complete analysis Fig. 37. The Brute approach being the simplest takes the highest CPU and Wall Clock time. The % reduction in CPU and Wall Clock time is quantified in Table. 2 which shows that for the MATLAB approach the Wall Clock time is reduced by 46.23%, CPU time by 46.17% w.r.t Brute approach Table. 2, but the Compilation time is increased by 13.33% begin the highest.

The stabilization in Wall Clock, CPU, and Compilation time for the Bin approach corresponding to the No. of Bins is also captured for Case2 (Fig. 37) which is also seen for the Case1 Fig. 29 Fig.27. The time reduction for the Bin approach compared to the Brute approach for CPU time is in the range of 29.67% - 39.90% and the same for the Wall Clock time, but a slight increase in Compilation time is also present i.e. 2.22% - 6.67%, Table. 2. Similarly, the Wall Clock and CPU time reduction by L1_Octree is 38.69% and increment of 2.22% in Compilation time w.r.t Brute approach. The result trends of the Exhaust Valve are comparable to the verification case in Sec. 3.6.

5 Discussion

The results obtained in Sec. 4 have highlighted the observations of the four methodologies employed for both Case1 i.e. hollow cylinder and Case2 i.e. exhaust valve. In this section, attempts are made to discuss the reasoning for these observations and find correlations with the theory and the immense existing literature.

5.1 Case1: Hollow Cylinder Mapping

The quality of data mapping for normalized SINK values for various meshes using different approaches is in great agreement w.r.t the Analytical solution Fig. 23 to Fig. 26. This is due to the utilization of integration points (Full or Reduced) [36] and the *SFILM keyword [18] to apply the HTC and SINK values of the surface in ABAQUS. An improved surface averaged value of HTC and SINK is obtained by using the integration points on the element face resulting in a better estimation of the entire surface Fig. 24 Fig. 25. This is the reason for the close agreement of the mapped value of SINK w.r.t Analytical solution in Fig. 23 and Fig. 26. Also, the stair-steps plots are due to surface averaging meaning the entire element face has one value of SINK which is the average of all nodal and integration points on that element face (horizontal part of the stair-step).

The CPU and Wall Clock time plots in Fig. 27 signifies the strength of spatial partitioning [4] for the nearest neighbour search [2] [11] [10] for a given integration point. The spatial discretization of data using the L1_Octree and Bin Approach into Octants and Bins respectively improves the CPU and Wall Clock time compared to the Brute Force, Fig. 27. The creation of smaller subdomains (Octant or Bin) by these approaches is fed to the Brute Force approach to find the closest point. Therefore, reducing the time from linear trend (Brute Force) towards close to logarithmic (L1_Octree and Bin approach) [19] [20], also explained in Sec. 2.2.

The discussion of CPU time becomes important in Fig. 27(c) as it is the actual time the processor spends on a specific task (here subroutine code). Therefore, this time is generally smaller than the Wall Clock time in Fig. 27(d) for a single processor/core as the extra time is spent by the CPU to acquire the resource like input-output instruction, file reading, and creation, logging, etc [45]. The quantification of CPU time becomes vital for multi-processor simulation where more than one processor is used to execute the same job to reduce the Wall Clock time, which internally increases the CPU time depending upon the number of processors (cores) used.

Next are two specific analyses performed for the Bin approach i.e. the data sorting operation Fig. 29 and the influence of the number of Bins on the CPU and Wall Clock time Fig. 30. The increment in the number of Bins reduces the analysis time significantly and stabilizes well for Bins > 24 for the Benchmark case i.e. a positive influence Fig. 30 because more Bins lead to the smaller subdomain and thus fewer points for the Brute Force to search. The edge cases discussed in Sec. 19 are handled to increase the credibility of the nearest neighbour search for only the Level-1 partitioning i.e. only 8 octants for L1_Octree and Bins along the most scattered axis only for the Bin approach.

The influence of data sorting is negative on the analysis time Fig. 29, leading to unstable trends as the dataset volume increases. Further analysis showed that the time taken for sorting the dataset was a significant percentage of the total time as FORTRAN90 does not have an optimized intrinsic sort function [35]. Based on the benchmarking case and the obtained results, the KD tree algorithm is ruled out as it requires finding the median value and performing the space partitioning [24] [11]. Consequently, the L1_Octree and Bin approaches are implemented for the validation cases.

5.2 Case1: Multi-Core Execution

The computation cost of the ABAQUS/Standard linear analysis grows with the problem size i.e. the number of nodes and elements influences the No. of Degree of Freedom (DOF) leading to increased iteration time. For smaller problems, Fig. 31 under 100000 DOF will not show any decrease in execution time [46]. The reason for this is the efficient management of element computation [46]. In the multi-core analysis (parallelization) the input file pre-processor and initialization of the code are performed on a single core only. Hence for a smaller problem the overhead of parallelization, communication establishment between threads and processors, division of elements on each core, etc are a few concerning factors for the execution time [47], Fig. 31. This overhead of parallelization on CPU time is established in Fig. 31(a) and Fig. 32(a) 33(a) for small and large case respectively [46]. The reason for the time increment with the increasing number of cores is the cache coherence and resource contention [48] which is the extra cost associated with the multi-processor analysis. This happens because each processor creates a copy of the shared data in its local cache which gets retrieved and modified locally giving rise to inconsistency with the original shared data. A considerable overhead is added by the cache coherence mechanism to keep the original shared data up-to-date from all the processors such that the most updated shared data can be accessed by each processor for read, write, and update operations [48]. The processors use MESIF (Modified, Exclusive, Shared, Invalid, and Forwarding) cache protocols to maintain this cache coherence [48].

5.3 Case2: Exhaust Valve Mapping

An Exhaust manifold is a component used in the Automotive industry that collects hot exhaust gases from multiple cylinders into one pipe Fig. 34. The manifold is exposed to extreme heating and subsequent cooling causing repeated expansion and contraction over its lifetime. It performs several functions like cooling down hot toxic exhaust gases, moving the combustion system gases away from the car, and reducing the noise associated with the combustion system and high-speed exhaust [49]. The surfaces are defined in Fig. 34, the inner surface is exposed to hot exhaust gases (HOT), the outer surface is exposed to ambient air (COLD), and the rest all surfaces like the bolt points (ADIABATIC) are highlighted in green.

The results from the parallelization of the benchmark case Sec. 4.2 offer the advantage of limiting the size of the validation case to a smaller size. The validation case in Fig. 35 has 317418 elements $<$ 2 million, which will not show significant performance improvements with parallelization [46] and avoid the overhead of parallelization [47]. Therefore, the results are presented in Fig. 37 and Table. 2 are for single-core analysis.

5.4 Research Questions Exploration

Now is the time to make a sincere attempt to answer the research questions that are presented in Sec. 1.4. The literature survey, theoretical concepts, methodology development, and results provide a firm base for the following answers.

Q1. Is 3D data mapping an alternate approach to overcome the challenges associated with multi-physics simulations?

Ans. Yes, 3D data mapping is a feasible approach to overcome multi-physics computational expenses and data utilization. The approach is dynamic e.g. in this thesis only the nearest neighbour search method is explored, but other methods are yet to be explored for strength and weakness analysis. The 3D mapping approach provides more control to the user over the data input and the accuracy of the results obtained. The scalability of the mapping approach is well-studied and documented in the existing literature for both academic and industrial benefits. It also serves as a good medium between mechanical, computer science, and integration with efficient ML and AI models and databases.

Q2. Are 3D data structures compatible with nearest neighbour search algorithms to hold large volumes of data and suitable for search operation?

Ans. Yes, immense research to create new data structures, file types, and memory management has paved the way for the KD tree, Quadtree, Octree, Slice Representations, etc. The development of faster search algorithms and data storage in tree or graph form are highly correlated. In this thesis work, the strength of FORTRAN90-derived data structures is explored which offer immense potential to hold large volumes of data. The created UDDS were flexible in holding information, indexing, and subsets of data. However, were limited to the use of intrinsic variables, and very limited use of pointer variables (allocatable data storage) was done to hold the data. However, if combined with pointer variables correctly and full potential utilization would make the derived data structures more powerful and robust. The use of UDDS for the development of search algorithms has provided fruitful results and if used with modern newer versions of FORTRAN and modern file types may also allow users to export UDDS data into external files e.g. TEXT, CSV, JSON, XML, etc.

Q3. Is the creation of a 3D data structure correlated to the ease of implementation, data access, and nearest neighbour search time?

Ans. Yes, as explained in **Q2**, the pointer variables are used in the creation of UDDS only as allocatable variables to hold large amounts of data leading to ease of implementation and data access. Implementation complexity is influenced by the problem statement which may include complex concepts and adversely affect the time complexity and performance of the algorithm.

However, unlike most programming languages where the pointer variable stores the memory address of the object, in FORTRAN it holds more information about the object like type, rank, memory address, and data. The pointer variables must be allocated with the space based on the data it is going to hold and in the same way, the de-allocation can be performed to clear the same space.

Q4. How does multi-processor analysis influence the simulation and are there any other influential factors associated with multi-processor?

Ans. The multi-processor analysis influences the CPU and Wall Clock time of the analysis, but many factors determine the extent of the negative and positive influence. The first highlight is that the Compilation + Pre-Run is now affected much as it is performed by a single core even for a multi-processor analysis. The size of the case i.e the number of elements, element type, DOF, etc make a case small (DOF <= 2 million) or big (DOF > 2 million). The thesis work also highlighted the ABAQUS performance data recommendations that for a small case single-core analysis is efficient as it eliminates the communication setup between the multiple processors to reduce messaging passing and CPU overlays. If multi-processor analysis is performed on a small case then both the CPU and Wall Clock time show significant performance downgrade. Contrary, for a big case, elements are shared between the multiple processors which increases the total CPU time but significantly reduces the Wall Clock time which is also presented in the thesis work.

6 Future Scope

The Master thesis is well restricted by ABAQUS and its use of programming language i.e. FORTRAN90 for analysis and subroutine creation. The subroutines can also be written in C/C++ but require the binding with FORTRAN which includes additional complexity to the code. The compatibility of ABAQUS with modern languages like Python or MATLAB for analysis would ease the subroutine creation and the use of the functionality and well-optimized libraries associated with these languages. However, multiple software or tools can be explored for the same purpose to achieve flexibility and ease of implementation.

Next, is the KD tree implementation in FORTRAN readily available in the literature that can be explored for the nearest neighbour search. The integration of this module requires minor changes in the developed methodology and can be explored for performance improvements and quality of 3D data mapping. However, the available KD tree implementation in FORTRAN is highly coupled with pointer variables, pointer associated and de-association, and a few more concepts that are not often used in modern programming languages and thus in-depth knowledge for debugging and code failures. Indeed a powerful code snippet must be handled with care and keeping in mind the scalability aspects, integration without utilities, maintaining the code readability and ease of use and implementation, and avoiding unnecessary complexities e.g. pointer variables.

6.1 Academic Implementation & Integration

The relevance of the academic future scopes is oriented toward mechanical and computer science engineering programs as the advancements in ML and AI commence the integration with CFD to achieve results in lesser time [5]. In many universities e.g. LiU, there are numerous courses and programs available at Basic, advanced, and applied engineering levels that utilize significant computation resources for simulation, algorithm development, research, multi-utility software/tool development [9] which generate immense data and the concept of 3D data mapping can be utilized. This data can be used as an integration medium between the courses at LiU, especially offered in the MSc Aeronautical program. For instance, the data from advanced CFD courses where the simulations are performed on the entire aircraft can be used for aircraft conceptual design courses which eventually leads to manufacturing of the aircraft in one of the courses. This will lead to additional courses well updated with the ongoing developments at academic and industrial levels.

6.2 Industrial Applications

The concept of 3D data mapping is well studied and associated with multi-variant interpolation techniques [6], [12], [14], [15] and nearest neighbour search algorithms [10]. The research involved in this concept enhances the data re-utilization of various domains and promotes the development of efficient 3D data structures [4] and storage databases.

This leads to enhanced create, read, update, and delete (CRUD) operation on this stored data, creating a strong foundation for ML and AI development for data-relevant domains, e.g., mechanical, aeronautical, complex geometry creation and simplification, etc. The data generated at industrial scales e.g. Siemens Energy for multi-physics simulations, experimentation, and regular repair and maintenance is fruitful and holds great potential for integration with ML and AI. Thus reducing the computation resources required by the industries, the deliverable time to market, etc. This will also promote a focus on technological innovation, sustainability, multi-domain scale-up for an industry, etc.

7 Conclusions

The thesis worked aimed to develop new methodologies for the nearest neighbour search method with better performance w.r.t the two existing methodologies. The development of the methodologies was based on the space partitioning search algorithms and constraints by the analysis tool and associated dependencies. The two new methodologies (L1_Octree and Bin approach) were compared against the existing methodologies (Brute Force and MATLAB subroutine file approach) for various performance parameters that include the quality of 3D data mapping with SINK and HTC contours, Compilation, CPU and Wall Clock time quantification, influence parameters like data sorting, average time to find nearest point and level of discretization, and analysis on multi-processors computer cluster network. Based on the thesis work following conclusions are made.

1. Large volumes of datasets can be stored in discretized form using UDDS available in FORTRAN with better indexing, and forward-backward traceability.

2. The square root operation (distance calculation) and data sorting adversely influence the time complexity of the nearest neighbour search thus Octree and discretization-based algorithms are recommended over the KD tree and Ball tree.

3. The L1_Octree and Bin (Bins >= 24) approaches showed a reduction of 44.85% - 80.86% in CPU and Wall Clock time w.r.t to Brute Force with the quality of data mapping and gradient captures same as Brute Force for the Case1.

4. The MATLAB approach showed 62.94% - 92.40% reduction in CPU and Wall Clock time w.r.t to Brute Force, but fails for large structured meshes or less scattered data for Case1.

5. The L1_Octree and Bin approaches showed a reduction of 10% - 41.67% in compilation time w.r.t Brute Force, whereas the MATLAB approach showed an increment of 25% - 50% for the Case1.

6. The average time to find the nearest neighbour point is highest for the Brute Force approach and lowest for the Bin approach with maximum number of Bins used studied i.e. 96.

7. The combination of UDDS and Global Module (FORTRAN) to create a thread safe shared memory results in the efficient parallelization for both MPI or Thread based analysis on a computer cluster network.

8. For a small size Case1 (DOF $<$ 2 million), the parallelization (multi-core analysis) downgrades the performance of Brute, Bin, and L1_Octree approaches. However, the L1_Octree outperforms the Brute and Bin approaches on a single core.

9. For a large size Case1 (DOF $>$ 2 million), the parallelization (multi-core analysis) significantly improves the performance of Brute, Bin, and L1_Octree approaches. The Bin approach outperforms the Brute and L1_Octree approaches for cores $>=$ 4.

10. The MATLAB approach for the validation case (small case with highly unstructured mesh), showed the highest reduction in CPU and Wall Clock time, but also the highest increment of 13.33% in the compilation time w.r.t Brute Force.

11. The L1_Octree approach is recommended for single small cases for single core, and the Bin approach with user-defined number of Bins is recommended for larger cases for multi-processor analysis.

References

[1] Keyes DE, McInnes LC, Woodward C, Gropp W, Myra E, Pernice M, et al. Multiphysics simulations: Challenges and opportunities. The International Journal of High Performance Computing Applications. 2013;27(1):4-83.

[2] Shamos MI, Hoey D. Closest-point problems. In: 16th Annual Symposium on Foundations of Computer Science (sfcs 1975); 1975. p. 151-62.

[3] Afazov SM, Becker AA, Hyde TH. Development of a Finite Element Data Exchange System for chain simulation of manufacturing processes. Advances in Engineering Software. 2012;47(1):104-13.

[4] Chen T, Schneider M. Data structures and intersection algorithms for 3D spatial data types. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. Association for Computing Machinery; 2009. p. 148â157.

[5] Vinuesa R, Brunton SL. Enhancing computational fluid dynamics with machine learning. Nature Computational Science. 2022;2(6):358-66.

[6] Ringstad KE, Banasiak K, Ervik Å, Hafner A. Machine learning and CFD for mapping and optimization of CO2 ejectors. Applied Thermal Engineering. 2021;199:117604.

[7] Wang J. An intuitive tutorial to Gaussian processes regression. Computing in Science & Engineering. 2023.

[8] Munjulury RC. Knowledge-based integrated aircraft design: An applied approach from design to concept demonstration. vol. 1853. Linköping University Electronic Press; 2017.

[9] Munjulury RC, Staack I, Berry P, Krus P. A knowledge-based integrated aircraft conceptual design framework. CEAS Aeronautical Journal. 2016;7:95-105.

[10] Vaidya PM. An O (n log n) algorithm for the all-nearest-neighbors problem. Discrete & Computational Geometry. 1989;4:101-15.

[11] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825-30.

[12] Renka RJ. Multivariate interpolation of large sets of scattered data. ACM Trans Math Softw. 1988 jun;14(2):139â148.

[13] Bajaj CL. Multi-dimensional Hermite Interpolation and Approximation for Modelling and Visualization. In: ICCG; 1993. p. 335-48.

[14] Franke R. Scattered Data Interpolation: Tests of Some Method. Mathematics of Computation. 1982;38(157):181-200.

[15] Amidror I. Scattered data interpolation methods for electronic imaging systems: a survey. Journal of electronic imaging. 2002;11(2):157-76.

[16] Liberti L, Lavor C. Euclidean distance geometry. vol. 3. Springer; 2017.

[17] Sebesta RW. Concepts of programming languages. Pearson Education India; 2016.

[18] Hibbitt, Karlsson, Sorensen. ABAQUS/Standard: User's Manual. vol. 1. Hibbitt, Karlsson & Sorensen; 1997.

[19] Ben-Amram AM, Jones ND. Computational complexity via programming languages: constant factors do matter. Acta Informatica. 2000;37:83-120.

[20] Kasai T, Adachi A. A characterization of time complexity by simple loop programs. Journal of Computer and System Sciences. 1980;20(1):1-17.

[21] Davis PJ. Interpolation and approximation. Courier Corporation; 1975.

[22] Trunk GV. A Problem of Dimensionality: A Simple Example. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1979;PAMI-1(3):306-7.

[23] Mehlhorn K. Data structures and algorithms 1: Sorting and searching. vol. 1. Springer Science & Business Media; 2013.

[24] Beineke LW, Pippert RE. Properties and characterizations of k-trees. Mathematika. 1971;18(1):141-51.

[25] Dolatshah M, Hadian A, Minaei-Bidgoli B. Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces. arXiv preprint arXiv:151100628. 2015.

[26] Khamsi MA, Kirk WA. An introduction to metric spaces and fixed point theory. John Wiley & Sons; 2011.

[27] Elseberg J, Magnenat S, Siegwart R, Nüchter A. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. Journal of Software Engineering for Robotics. 2012;3(1):2-12.

[28] Meagher D. Geometric modeling using octree encoding. Computer Graphics and Image Processing. 1982;19(2):129-47.

[29] Ho SLS. Solid modelling using linear octree representation. University of British Columbia; 1985.

[30] Yamaguchi K, Kunii T, Fujimura K, Toriya H. Octree-related data structures and algorithms. IEEE Computer Graphics and Applications. 1984;4(01):53-9.

[31] Kotsiantis S, Kanellopoulos D. Discretization techniques: A recent survey. GESTS International Transactions on Computer Science and Engineering. 2006;32(1):47-58.

[32] Inc TM. MATLAB version: 24.1 (R2024a). Natick, Massachusetts, United States: The MathWorks Inc.; 2024.

[33] Pandas cut() Function: Binning and Grouping Data (With Examples);. Accessed 10/03/2024. https://machinelearningtutorials.org/pandas-cut-function-binning-and-grouping-data-with-examples/.

[34] Zienkiewicz OC, Taylor RL, Zhu JZ. The finite element method: its basis and fundamentals. Elsevier; 2005.

[35] Adams JC, Brainerd WS, Martin JT, Smith BT, Wagener JL. fortran 90 Handbook. vol. 32. McGraw-Hill New York; 1992.

[36] Adeeb S. Introduction to solid mechanics and finite element analysis using Mathematica. Kendall Hunt; 2011.

[37] Maruyama S, Moriya S. Newton's Law of Cooling: Follow up and exploration. International Journal of Heat and Mass Transfer. 2021;164:120544.

[38] Som S. Introduction to heat transfer. PHI Learning Pvt. Ltd.; 2008.

[39] Sundén B. Introduction to heat transfer. Wit Press; 2012.

[40] Mangani L, Cerutti M, Maritano M, Spel M. Conjugate heat transfer analysis of NASA C3X film cooled vane with an object-oriented CFD code. In: Turbo Expo: Power for Land, Sea, and Air. vol. 43994; 2010. p. 1805-14.

[41] Fluent A, et al. Ansys fluent theory guide. Ansys Inc, USA. 2011;15317:724-46.

[42] Charania S, Soni V, Patel K, Desai J, Chauhan V. Evaluation of Vertical Kaplan Turbine using CFD; 2012. .

[43] Attiya H, Welch J. Distributed computing: fundamentals, simulations, and advanced topics. vol. 19. John Wiley & Sons; 2004.

[44] Hiranandani S, Kennedy K, Tseng CW. Evaluating compiler optimizations for Fortran D. Journal of Parallel and Distributed Computing. 1994;21(1):27-45.

[45] Kato S, Ishikawa Y, Rajkumar R. CPU scheduling and memory management for interactive real-time applications. Real-Time Systems. 2011;47:454-88.

[46] Abaqus Performance Data;. Accessed 05/05/2024. https://www.3ds.com/support/hardware-and-software/simulia-system-information/abaqus-69/performance-data/.

[47] Obbink-Huizer C. 7 Tips to Help Abaqus Run Faster;. Accessed 05/05/2024. https://simulation-blog.technia.com/simulation/7-tips-to-help-abaqus-run-faster.

[48] Thomadakis ME. The architecture of the Nehalem processor and Nehalem-EP SMP platforms. Resource. 2011;3(2):30-2.

[49] Xu SL, Shi Y, Li SC. Heat Transfer and Thermal Load Analysis of Exhaust Manifold. In: Vibration, Structural Engineering and Measurement II. vol. 226 of Applied Mechanics and Materials. Trans Tech Publications Ltd; 2012. p. 2240-4.