June 2018

Efficient Adjacency Queries and Dynamic Refinement for Meshfree Methods with Applications to Explicit Fracture Modeling

James Olliff
University of South Florida, jjolliff@mail.usf.edu

Efficient Adjacency Queries and Dynamic Refinement for Meshfree Methods with Applications to Explicit Fracture Modeling

by

James Olliff

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Civil and Environmental Engineering
College of Engineering
University of South Florida

Major Professor: Daniel C. Simkins Jr., Ph.D.
Andrés Tejada-Martinez, Ph.D.
Stuart Wilkinson, Ph.D.
Nathan Crane, Ph.D.
Seung-Yeop Lee, Ph.D.

Date of Approval:
May 31, 2018

Keywords: laminated composites, nearest neighbor, particle placement

DEDICATION

To my loving mother and late father, who instilled in me the work ethic and drive necessary to complete this journey.

ACKNOWLEDGMENTS

Someone once told me I was not "gifted". I am a believer in the words of G.K. Nielson, that successful people are not gifted; they just work hard, then succeed on purpose. The following people have helped me achieve success beyond what I thought was possible. I would like to express my appreciation to my advisor, Dr. Dan Simkins, for allowing me to explore a variety of research topics, guiding me along the way, and teaching me more than I could every give him credit for. I would also like to express my appreciation for Dr. Stuart Wilkinson, for providing me with many opportunities outside my primary research that helped me develop invaluable skills and knowledge beyond what can be taught in a classroom. I would also like to thank my committee members: Dr. Andrés Tejada-Martinez, Dr. Nathan Crane, Dr. Seung-Yeop Lee, and Dr. Craig Lusk for their insightful comments and challenging questions pushing me to a better understanding. I would also like to thank the CEE staff, especially Barbara Johnson for generous T.A. appointments when I needed them. I would also like to thank Dr. Brad Alford, who brought me into this research group, and upon graduation, has helped me embark on what is sure to be a rewarding career. Last but not least, I would like to thank my family and friends for the endless encouragement and support. I would like to especially thank my loving wife and best friend, Dr. Bailee Olliff. Words cannot express how grateful I am to call you my wife. You are also likely one of the only people to read every word of this dissertation, even if at times it was under duress.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Meshfree methods provide a more practical approach to solving problems involving large deformation and modeling fracture compared to the Finite Element Method (FEM). However meshfree methods are more computationally intensive compared to FEM, which can limit their practicality in engineering. Meshfree methods also lack a clear boundary definition, restricting available visualization techniques. Determining particle locations and attributes such that a consistent approximation is ensured can be challenging in meshfree methods, especially when employing h-refinement. The primary objective of this work is to address the limitations associated with computational efficiency, meshfree domain discretization, and h-refinement, including both placement of particles as well as determination of particle attributes. To demonstrate the efficacy of these algorithms, a model predicting the failure of laminated composite structures using a meshfree method will be presented.

CHAPTER 1: INTRODUCTION

1.1 Motivation

Engineering design requires determination of geometric configuration, material properties, and knowledgeable selection of other parameters. Traditionally, this design process consisted of a trial-and-error approach with the design engineer establishing the relevant properties based on approximations and experience. The initial design would then be constructed into a physical prototype to be used in the testing phase, which determines if the initial configuration is suitable. This process could be repeated many times until a satisfactory solution is found. Depending upon the product, materials, and testing procedures, design could be costly. In order to minimize the cost and inefficiency of constructing physical prototypes for testing, virtual prototypes are desirable. Programs used for engineering analysis are commonly referred to as computer-aided engineering (CAE) software. CAE packages are used in conjunction with computer-aided design (CAD) software. Together they allow an engineer to create multiple configurations and subject them to various conditions without the need for physical prototypes.

A well-established technique for conducting simulations is known as the Finite Element Method (FEM). While FEM has achieved remarkable success, there are limitations that reduce the efficacy of FEM for certain classes of problems. One of the limitations is the need to construct and maintain a quality mesh, which has been stated to consume more

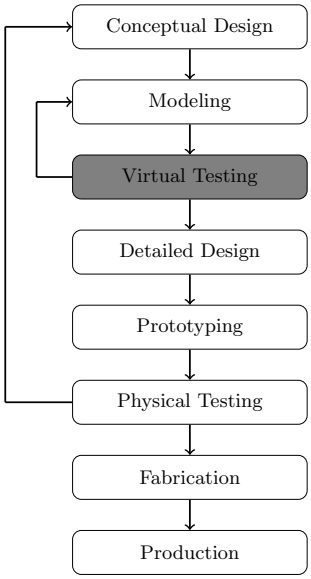


Figure 1.1: Engineering design flowchart

than 70 percent of the total computation time [31]. For certain problems, the construction of a suitable mesh requires human intervention in the form of tedious removal of distorted elements. In addition, a high level of refinement is often necessary to capture finer details of a simulation.

Meshfree methods provide an alternative to mesh-based approximation techniques. In contrast to FEM, where a function space is constructed on elements of a mesh, meshfree methods construct a function space on a set of particles. Eliminating the need for a disjoint polygonal or polyhedral decomposition makes meshfree methods well suited for handling problems where generating and maintaining a mesh is difficult, such as the simulation of large deformations and modeling arbitrary crack propagation. Despite its utility, meshfree methods are more computationally intensive then FEM, which can limit their practicality in engineering. Meshfree methods also lack a clear boundary definition, restricting available visualization techniques. Determining particle locations and attributes such that a consistent

approximation is ensured can be challenging in meshfree methods, especially when employing h-refinement.

1.2 Objective

The primary objective of this work is the development of algorithms to address several limitations of meshfree methods which hinder their practical applicability. To demonstrate the efficacy of these algorithms, a model predicting the failure of laminated composite structures using a meshfree method will be presented.

The tasks involved in achieving these objectives include:

- Development of efficient data structures and algorithms for handling inefficiencies associated with computing meshfree interpolants
- Development of algorithms for meshfree domain discretization and h-refinement, including both placement of particles as well as determination of particle attributes
- Demonstration of the novel algorithms developed in this work to model fracture in laminated composites

1.3 Overview

The formulation and fundamental challenges with meshfree methods will be reviewed, with attention to challenges that limit the practical applicability of these methods. The work will present solutions to efficient computation of meshfree interpolants, specifically as it pertains to adjacency queries. The work will also present a method for determination of suitable particle distributions in a truly meshfree approach. After presenting these solutions,

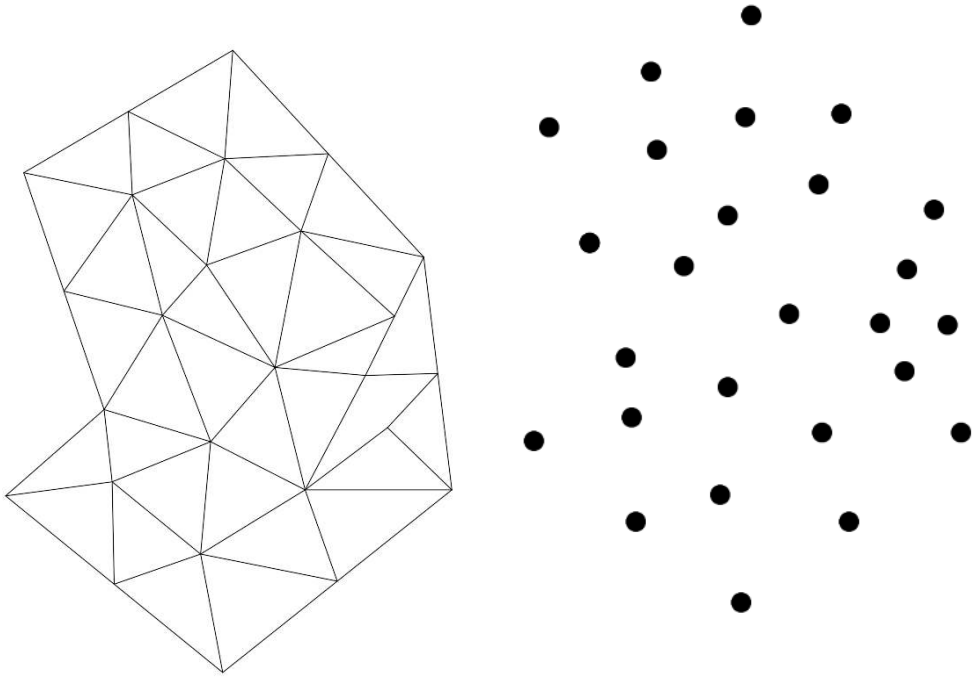
they will be used in a Galerkin formulation for approximating the governing partial differential equations (PDEs) of solid mechanics. In addition, a methodology for modeling explicit fracture in laminated composites will be presented in order to demonstrate the utility of the newly developed algorithms.

CHAPTER 2: MESHFREE METHODS

This chapter will introduce the numerical approximation scheme that is used in this dissertation. As the approximation method is less well-known than its more established counterpart FEM, the chapter will begin with a brief history of its development. Next the formulation will be developed along with the requirements and limitations associated with the numerical method will be discussed. Previously developed techniques to overcome these deficiencies will be presented where applicable and outstanding issues will be emphasized.

2.1 Historical Development

For performing engineering analysis and the closely related topic of geometric modeling, meshfree methods provide an alternative to mesh-based approximation techniques. In contrast to FEM, where a function space is constructed on elements of a mesh, meshfree methods construct their function space on a set of particles. Typically, the shape functions associated with a particle are compactly supported. This difference is illustrated in Figure 2.1. Obviating the need for a disjoint polygonal or polyhedral decomposition makes meshfree methods ideal for handling problems where generating and maintaining an analysis-suitable mesh is difficult. One scenario that makes maintaining a mesh difficult is when large deformations are present. Contrary to FEM, meshfree methods are not as susceptible to the challenges of large deformation and have been used in a wide range of simulations.



(a) Finite Element Domain (b) Meshfree Domain

Figure 2.1: Comparison of FEM and meshfree discretizations

Smooth Particle Hydrodynamics was the first type of meshfree method developed in 1977 for solving problems in astrophysics [27, 47]. Since then numerous meshfree methods have been developed, including the well-known Reproducing Kernel Particle Method (RKPM) [44], Element Free Galerkin (EFG) method [8], Meshfree Local Petrov Galerkin (MLPG) [2], HP-Clouds [21], and the Finite Point Method (FPM) [50]. A thorough review of meshfree methods is given in [24, 39, 40, 43]. Within the context of modeling discontinuities in solids, the most prominent meshfree methods used are the Reproducing Kernel Particle Method (RKPM), and the Element Free Galerkin (EFG) method which uses the Moving Least Squares (MLS) method for construction of the EFG approximation functions.

2.2 Reproducing Kernel Method

A class of operators that reproduce themselves through integration over a domain are known as reproducing kernels and are of the form:

$$u(x) = \int_{-\infty}^{\infty} \delta(x-s)\, u(s)\, \mathrm{d}s, \tag{2.1}$$

where $\delta(x)$ is the Dirac delta function. The reproducing kernel method is a meshfree method that uses this class of operators to construct an approximation of a function.

While Equation 2.1 provides an exact representation of $u(x)$, it is not feasible in a practical application given the finite nature of the discretization. Therefore an approximation of the Dirac delta function is made as:

$$\delta(x-s) \approx \frac{1}{\rho}\phi\left(\frac{x-s}{\rho}\right) = \phi_\rho(x-s). \tag{2.2}$$

Here ϕ is a smooth compactly supported function commonly known as the kernel or window function, and ρ is defined as the smoothing length which characterizes the size of the support domain associated with a particle. Numerous choices exist for the window function [3]. A popular choice that will be used in this study is the conical function defined as:

$$\phi(x) = \begin{cases} (1-x^2)^q & |x| \leq 1 \\ 0 & |x| > 1 \end{cases} \tag{2.3}$$

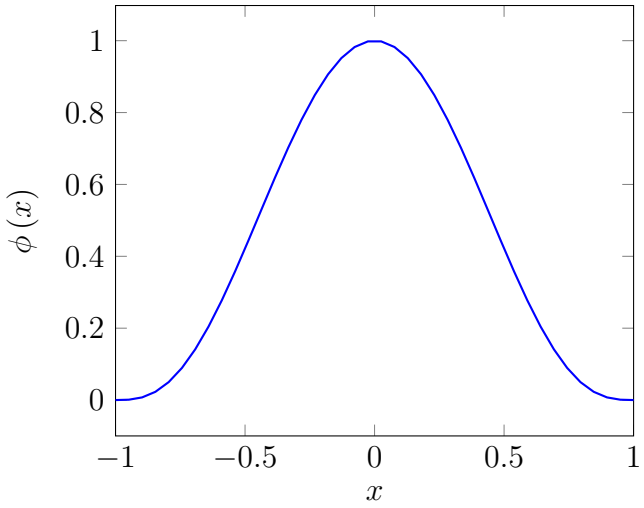


Figure 2.2: Window function

where q is the order of the conical function, for this study $q = 3$. A graphical depiction of the conical window function is given in Figure 2.2. Substitution of the the delta approximation from Equation 2.2 into Equation 2.1 leads to the SPH approximation,

$$u^{\rho}(x) = \int_{\Omega} \phi_{\rho}(x - s)\, u(s)\, \mathrm{d}s. \tag{2.4}$$

Equation 2.4 will exactly reproduce the continuous function u when the kernel is constructed to satisfy certain requirements. However, a well-known issue with the SPH approximation is the lack of completeness when Equation 2.4 is discretized using a quadrature formula. An approximation is n order complete if capable of exactly reproducing a polynomial of degree n. In the context of this work, completeness, consistency, and reproducibility are analogous and all refer to the previous definition. Correcting this issue has been the topic of numerous works and has led to a variety of numerical methods referred to as corrective SPH. An early approach to restoring consistency to the discrete SPH approximation was proposed by Liu *et al.* [45], yielding the Reproducing Kernel Particle Method (RKPM).

Here a correction term is incorporated into the kernel approximation leading to the following approximation:

$$u^{\rho}(x) = \int_{\Omega} C(x; x-s)\, \phi_{\rho}(x-s)\, u(s)\, \mathrm{d}s, \tag{2.5}$$

where $C(x; x-s)$ is the correction function and is represented by a polynomial of order N, that is

$$C(x; x-s) = \mathbf{P}^{T}(x-s)\, \mathbf{b}(x). \tag{2.6}$$

Here $\mathbf{P}$ is a vector of monomial basis functions and $\mathbf{b}$ is a vector containing the set of unknown functions to be determined in order to ensure consistency. Consistency of the approximation can be evaluated by considering the Taylor series expansion of u evaluated at s about x given as:

$$u(s) = \sum_{i=0}^{\infty} \frac{(-1)^{i}}{i!} u^{(i)}(x)\, (x-s)^{i}. \tag{2.7}$$

This can be expressed in vector form as:

$$u(s) = \mathbf{P}^{T}(x-s)\, \mathbf{D}(x), \tag{2.8}$$

with $\mathbf{P}$ representing a vector of monomial basis up to order N and $\mathbf{D}$ containing coefficient terms, given as:

$$\mathbf{P}^{T}(x-s) = \begin{bmatrix} 1 & x-s & \dots & (x-s)^{N} \end{bmatrix}, \tag{2.9}$$

and

$$\mathbf{D}^{T}(x) = \begin{bmatrix} 1 & \frac{-1}{1!}\frac{du}{ds}\big|_{\mathbf{x}} & \dots & \frac{(-1)^{N}}{N!}\frac{d^{N}u}{ds^{N}}\big|_{x} \end{bmatrix}. \tag{2.10}$$

Substituting Equation 2.8 into Equation 2.5 yields

$$u^{\rho}(x) = \int_{\Omega} \mathbf{P}^{T}(x-s)\,\mathbf{b}(x)\,\phi_{\rho}(x-s)\,\mathbf{P}^{T}(x-s)\,\mathbf{D}(x)\,\mathrm{d}s. \tag{2.11}$$

For consistency Equation 2.11 must satisfy

$$\mathbf{P}^{T}(0)\,\mathbf{D}(x) = \int_{\Omega} \mathbf{P}^{T}(x-s)\,\mathbf{b}(x)\,\phi_{\rho}(x-s)\,\mathbf{P}^{T}(x-s)\,\mathbf{D}(x)\,\mathrm{d}s \tag{2.12}$$

which is obtained by replacing the right-hand side of Equation 2.11 with the Taylor series in Equation 2.8 evaluated at $s = x$. Equation 2.12 must hold for all x and u is an arbitrary function. Therefore, Equation 2.12 can be stated as:

$$\left\{\int_{\Omega} \mathbf{P}(x-s)\,\mathbf{P}^{T}(x-s)\,\phi_{\rho}(x-s)\,\mathrm{d}s\right\}\mathbf{b}(x) = \mathbf{P}(0) \tag{2.13}$$

or more succinctly as a linear system of equations,

$$\mathbf{M}(x)\,\mathbf{b}(x) = \mathbf{P}(0)\,. \tag{2.14}$$

Here $\mathbf{M}$ is commonly known as the moment matrix and is constructed from the integral term in Equation 2.13. Solving for the unknown correction terms, $\mathbf{b}$, and combining Equations 2.6 and 2.5 we arrive at the corrected kernel approximation given as:

$$u^{\rho}(x) = \left\{\int_{\Omega} \mathbf{P}^{T}(x-s)\,\phi_{\rho}(x-s)\,u(s)\,\mathrm{d}s\right\}\ \mathbf{M}^{-1}(x)\,\mathbf{P}(0)\,. \tag{2.15}$$

The generalization of the one-dimensional formulation to a higher dimension requires the modification of the one-dimensional window function. This modification can be done by taking the cartesian products of the one-dimensional window function:

$$\phi_\rho^d = \prod_{i=1}^{d} \phi_{\rho_i} (x_i - s_i) . \tag{2.16}$$

Alternatively, a radial support can be used:

$$\phi_\rho^d = \frac{1}{\rho^d} \phi \left(\frac{\|\mathbf{x} - \mathbf{s}\|}{\rho} \right) . \tag{2.17}$$

In Equation 2.16 the one-dimensional dilation parameter has also been modified from a scalar to a vector, allowing for an anisotropic support domain. In Equation 2.17, the dilation parameter remains a scalar resulting in an isotropic support domain. Further discussion on the dilation parameter and support domain will be given in §4.2.2. The multi-dimensional approximation is:

$$u^\rho (\mathbf{x}) = \int_\Omega \mathbf{P}^T (\mathbf{x} - \mathbf{s}) \, \phi_\rho^d (\mathbf{x} - \mathbf{s}) \, u (\mathbf{s}) \, ds \; \mathbf{M}^{-1} (\mathbf{x}) \, \mathbf{P} (0) . \tag{2.18}$$

Here the scalar arguments have been replaced with vectors of size corresponding to the spatial dimension.

2.3 Reproducing Kernel Particle Method

The continuous integral in Equation 2.15 needs to be evaluated numerically, which leads to the Reproducing Kernel Particle Method (RKPM). Let the domain be discretized by a

set of nodes $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{N_p}\}$. Using the nodal locations as quadrature locations the kernel approximation can be restated as:

$$u^\rho(\mathbf{x}) \sum_{I=1}^{N_p} C(\mathbf{x}; \mathbf{x} - \mathbf{s}) \phi_\rho(\mathbf{x} - \mathbf{x}_I) u_I \Delta V_I, \tag{2.19}$$

where u_I is the function value at $\mathbf{x}_I$ and ΔV_I is the portion of the domain associated with node I. The integration of the moment matrix in Equation 2.13 must also be discretized. In [46] it is noted that the same integration technique employed for Equation 2.19 should be used when discretizing the integration of the moment matrix. This leads to the discrete moment matrix given as:

$$\mathbf{M} = \sum_{I}^{N_p} \mathbf{P}(\mathbf{x} - \mathbf{x}_I) \mathbf{P}^T(\mathbf{x} - \mathbf{x}_I) \phi_\rho(\mathbf{x} - \mathbf{x}_I) \Delta V_I. \tag{2.20}$$

Generally the integration weight will be combined with the corrected kernel to form the RKPM shape functions. This allows the RKPM approximation to be stated as:

$$u^\rho(\mathbf{x}) = \sum_{I=1}^{N_p} \Psi_I(\mathbf{x}) u_I \tag{2.21}$$

where $\Psi_I(\mathbf{x})$ are the RKPM approximation functions,

$$\Psi_I(\mathbf{x}) = \mathbf{P}^T(\mathbf{x} - \mathbf{x}_I) \phi_\rho^d(\mathbf{x} - \mathbf{x}_I) \mathbf{M}^{-1}(\mathbf{x}) \mathbf{P}(0) \Delta V_I, \tag{2.22}$$

and u_I is the value to be approximated at node I. The summation is taken over the set of particles that have non-zero shape functions at $\mathbf{x}$.

When approximating the solution to a PDE the shape functions as well as their derivatives are required. Before proceeding with the computation of the derivatives several comments on the notation need to be made. To simplify the the following formulation the indicial notation form of the approximation will be used and can be stated as:

$$\Psi_I(\mathbf{x}) = P_i(\mathbf{x})\,\phi_\rho^d(\mathbf{x}-\mathbf{x}_I)\,M_{ij}(\mathbf{x})^{-1}\,P_j(0)\,\Delta V_I. \tag{2.23}$$

Differentiation will be denoted using comma subscript notation. For example the gradient of some scalar function f is expressed as:

$$\boldsymbol{\nabla} f(\mathbf{x}) = \frac{\partial f}{\partial x_i} = f_{,i}, \tag{2.24}$$

with the last form being adopted here. For detailed overview of indicial notation see [55]. Proceeding with differentiation of Equation 2.23 results in the following:

$$\Psi_{I,i} = C_{,i}\phi_\rho\Delta V_I + C\phi_{\rho,i}\Delta V_I. \tag{2.25}$$

Differentiation of the correction term leads to:

$$C_{,i} = P_{k,i}M_{kj}^{-1}P_j(0) + P_k M_{kj,i}^{-1}P_j(0). \tag{2.26}$$

The derivative of the inverse of the moment matrix can be computed by the formula:

$$M_{kj,i}^{-1} = -M_{kl}^{-1}M_{lm,i}M_{mj}^{-1}, \tag{2.27}$$

and the derivative of the moment matrix by:

$$M_{kj,i} = \sum_I P_{k,i} P_j \phi_\rho \Delta V_I + P_k P_{j,i} \phi_\rho \Delta V_I + P_k P_j \phi_{\rho,i} \Delta V_I. \tag{2.28}$$

2.3.1 Discretization Requirements

While no mesh is necessary in the meshfree discretization, the distribution of particles must satisfy certain requirements in order to guarantee a valid simulation. A distribution satisfying these requirements is referred to as an admissible particle distribution [46]. Before proceeding with the requirements several definitions will be introduced.

Definition 2.3.1. The *support domain* is the region associated with a particle in which it has a non-zero shape function. The support domain is parameterized by the dilation parameter ρ. Using set notation $\Omega_I^\rho = \{\mathbf{y} | \|\mathbf{y} - \mathbf{x}\| \leq \rho\}$

Definition 2.3.2. The *influence domain* is the region associated with any point in the domain that is composed of the particles contributing at the point.

Every point in the domain is associated with an influence domain. The influence domain in general is not of a simple geometry. However, a simple geometric approximation of the region can be made with the minimum size of the influence domain associated with a point being approximated by largest support domain of a particle contributing at the point the influence domain is associated with. Since the particles are in the domain, they also associate with an influence domain. The influence domain of a particle is not equivalent to the support domain. To elaborate, if the surrounding particles have larger support domains, then the domain of influence at $\mathbf{x}_I$ will be larger than its compact support. Therefore, the

compact support is a subset of the influence domain. With these concepts established, the requirements for an admissible particle discretization are as follows.

1. The union of all the support domains cover the domain.

2. The ratio of the largest support domain to the smallest support domain is finite.

3. For any point in the domain there should be N_c particles contributing, where N_c is in the interval $N_L \leq N_c \leq N_U$. Where N_L is the order of polynomial from Equation 2.9 and N_U is an upper bound that is necessary for the accuracy of the approximation.

4. Particles should be capable of forming a non degenerate simplex.

The first requirement is illustrated in Figure 2.3. The domain Ω is depicted as a rectangle and the particles' support domains are the circles. From the figure it is clear that the domain

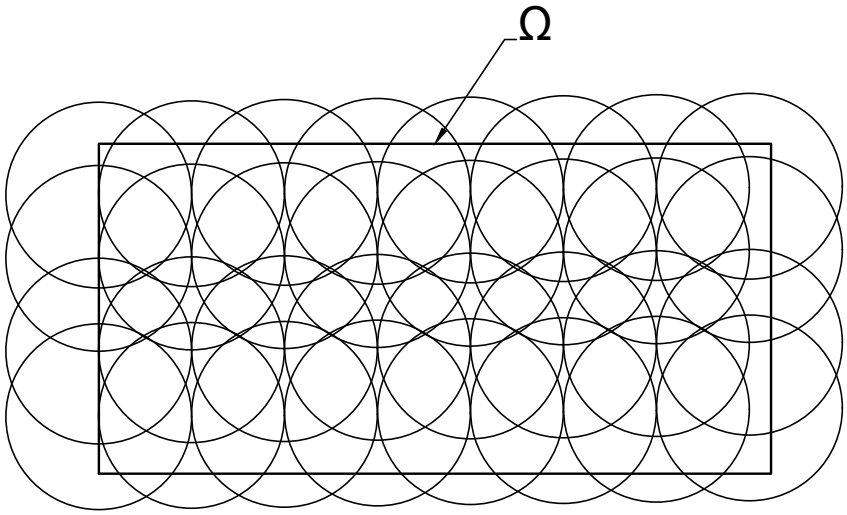


Figure 2.3: Domain coverage of particles' supports

being discretized is a subset of the union of the support domains, that is

$$\Omega \subset \bigcup_{I=1}^{N_p} \Omega_I^\rho. \tag{2.29}$$

In order for the approximation to be valid for any point in the domain, the first requirement is necessary, but is not sufficient. The second requirement not only places an upper bounds on the support of a particle, but also precludes the characteristic size of a particle from being zero. The third requirement stems from the construction of the discrete moment matrix in Equation 2.20. As the construction is performed by rank-one updates, a necessary condition for the moment matrix to be invertible, is the existence of N_L non-zero contributing terms, where N_L is the number of terms in the polynomial field from Equation 2.6. The third requirement alone does not ensure the moment matrix is invertible. This situation will arise when the particles within the influence domain of a point are oriented such that the corresponding moment matrix does not span the space of the polynomial field. In a two-dimensional problem with a linear polynomial, the second requirement specifies that a minimum of three particles must contribute at each sampling point. The fourth requirement dictates that these three particles must not be collinear, they must form a triangle with a non-zero area. Therefore, the third and fourth requirements can be summarized as the condition that the moment matrix must be invertible at any point in the domain.

While these conditions should hold for any point in the domain, the discrete nature of the problem makes it more practical to require they hold at the particles. Approaches for determining admissible particle distributions will be discussed in Chapter 4.

2.4 Challenges of Meshfree Methods

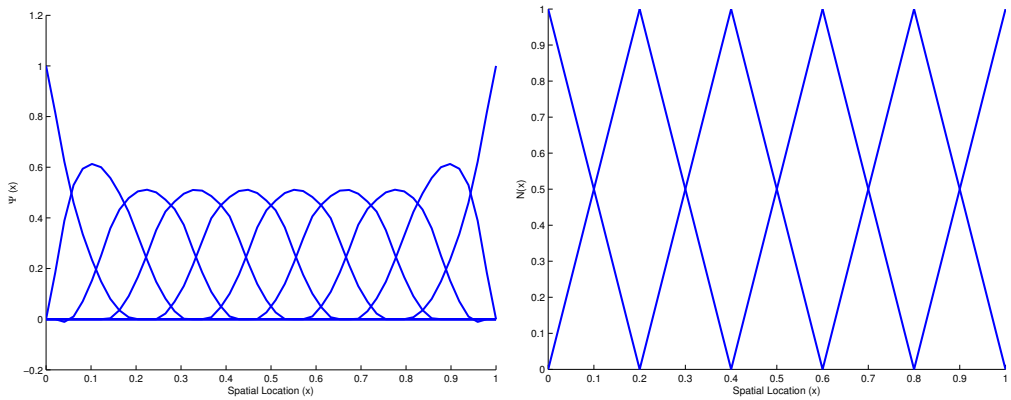
While the form of the approximation appears identical to the FEM function approximation, the meshfree approximation functions have several key differences.

2.4.1 Application of Boundary Conditions

The imposition of essential boundary conditions in meshfree Galerkin methods is problematic in general because the meshfree shape functions are not interpolating functions, that is they do not satisfy the Kronecker-δ property [40] i.e.,

$$\delta_{ij} \neq \Psi_i \left(\mathbf{x}_j \right) . \tag{2.30}$$

Hence, the nodal coefficients in Equation 2.21 do not correlate to the nodal displacements. Thus, the prescribed values cannot be directly set in the global displacement vector in Equation 5.28, which is the typical approach used in FEM where the shape functions are interpolating. A plot of the one-dimensional meshfree shape functions is shown in Figure 2.4a plotted alongside the one-dimensional linear FEM shape functions, Figure 2.4b, which do satisfy the Kronecker-δ property. An excellent overview of the various approaches to



(a) Meshfree shape functions (b) Finite element shape functions

Figure 2.4: Comparison of FEM and meshfree shape functions

enforce essential boundary conditions is given in [22] and will be briefly summarized here.

Two general approaches have been taken to correctly enforce essential (Dirichlet) boundary conditions in meshfree Galerkin methods. The first approach is based on a modified weak form, which accounts for the lack of the approximation functions to meet the Kronecker-δ property. Methods such as the Lagrange Multiplier method [8], Nitche's method [30], and the Penalty method [69] are based on the use of a modified weak form. The second approach is based on the modification of the meshfree shape functions. A boundary kernel was proposed that would allow the meshfree shape functions to be constructed with the Kronecker-δ property. A transformation method that expresses the meshfree shape functions as a linear combination of unknowns described in [40]. A method coupling meshfree and finite element methods was proposed in [9, 32, 62]. A general approach to coupling the meshfree and finite element shape functions coined the continuous blending method was developed by Huerta *et al.* [32].

The meshfree approximation functions also differ from FEM when imposing natural (Neumann) boundary conditions. In FEM, only vertices located on the surface need to be considered when integrating over the traction boundaries. The meshfree procedure has to not only account for boundary particles, but also interior particles as they may have contributions at the boundary.

2.4.2 Numerical Integration

In order to evaluate the integrals arising from a meshfree Galerkin approximation, a numerical integration scheme is necessary. The purpose of a numerical integration scheme

is to replace a continuous integral with a discrete sum, i.e.,

$$\int_{\Omega} f(x)\, dx \approx \sum_{I}^{N_q} f(x_I^*)\, w_I \tag{2.31}$$

where the sum is taken over N_q integration points located at x_I^* with weight w_I. A variety of numerical integration techniques exist, with the difference being in the number of quadrature points, locations, and weights. In the context of FEM, a commonly used method is Gaussian quadrature. Gaussian quadrature is developed such that one-dimensional polynomials of degree $2N_q - 1$ are integrated exactly with an integration rule of order N_q. While this works well for the polynomial shape functions of FEM, the meshfree shape functions are non-polynomial, requiring special techniques for conducting integration. In addition to the complex form, the overlapping particle supports add further complexity to the integration. If not properly addressed, a poor integration scheme can result in deterioration of accuracy of nodal results and loss of convergence. Two general approaches are used when numerically integrating the weak form with the function spaces constructed of meshfree shape functions. The first is based on the construction of a background mesh [8]. In this approach, the domain is subdivided into integration domains over which Gaussian quadrature is employed. The problem of background meshes and cells is that an integration error arises from the misalignment of the supports and the integration domains; this misalignment error can be higher than the error associated with the non-polynomial form of the shape functions [19]. Consequently, a higher order quadrature rule, which would reduce the error associated with the non-polynomial form, might not lead to better results due to the error associated with the misalignment of integration cells [19]. To reduce the error associated with the misalignment,

a method to construct integration cells that align with the particle supports is given in [19]. This allows for the reduction of integration error with the addition of more integration cells. Approaches that use the support of the shape functions as integration cells have also been proposed [2, 18]. These approaches adopt special integration techniques depending on the geometrical configuration used for the particles' support domain. This method addresses the issues associated with the background mesh but at an increased computational expense, as integration points are evaluated numerous times. The second approach, referred to as nodal integration, does not use integration points; instead, the integrals are evaluated at nodal locations. This approach leads to a more efficient integration scheme compared to the background mesh approach but can suffer from numerical instability [4, 40]. This instability is the result of under-integration and the vanishing of the shape function derivatives at the nodes. Beginning with the development of the Galerkin meshfree methods, an accurate nodal integration scheme has been an area of active research with numerous techniques resulting. An early technique stabilizes the nodal integration by adding a stabilization term in the form of the residual of the equilibrium equations to the nodally-integrated potential energy functional [4]. This approach comes at the cost of a higher order basis in the shape function construction due to the second-order derivatives in the stabilization term, thus increasing the size of the support domain and negatively impacting the overall computational performance of the method.

2.4.3 Visualization

Another drawback of meshfree methods is the lack of a well defined description of the boundary. This creates significant difficulty in providing an accurate graphical depiction of

the meshfree simulation results. A common approach to visualizing meshfree geometries is done by constructing a mesh, which is then rendered by standard surface-based rendering techniques such as rasterization, ray tracing, or radiosity. The mesh needed for visualization does not need to meet the stringent requirements of the mesh needed for analysis. However, the generation of this mesh may still not be a trivial task. Figure 2.5 depicts a domain that has been meshed in order to provide a continuous graphical representation. In order

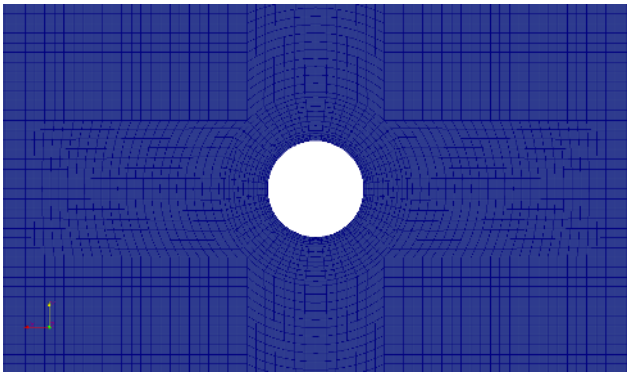


Figure 2.5: Visualization of a meshfree domain that has been meshed

to graphically depict the analysis results on the mesh, the data must be mapped from the meshfree particles to the mesh vertices leading to the accruement of computational time and memory. In addition to the computational expense associated with the interpolation, the resulting image could portray incorrect information. The mesh-based approaches suffer from their inability to capture small features such as cracks. Consider the case where a discontinuity is modeled in the meshfree domain. Upon mesh generation, if this discontinuity is not captured and enforced properly, the resulting mesh will display data that is continuous across the discontinuity as shown in Figure 2.6. In order to capture the discontinuity, the mesh must conform to the fracture. In two dimensions the description and meshing may be feasible, the same cannot be said for three dimensions. Another approach commonly

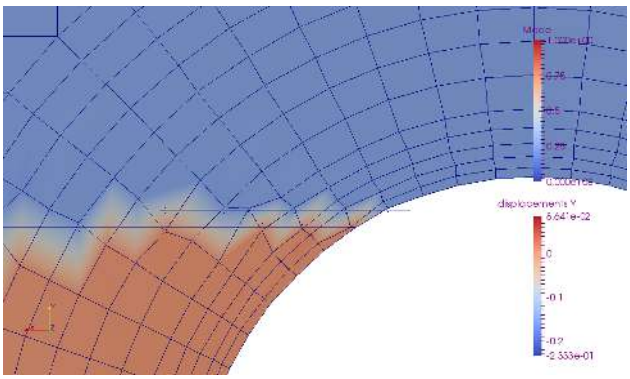


Figure 2.6: Interpolation of field values across discontinuity

used to render meshfree analysis results is visualizing the particles themselves as points or small spheres. This method does not require the extra computational burden associated with creating a mesh, but the resulting image does not provide a detailed representation of the boundary as illustrated by Figure 2.7. Visualization methods that do not operate on

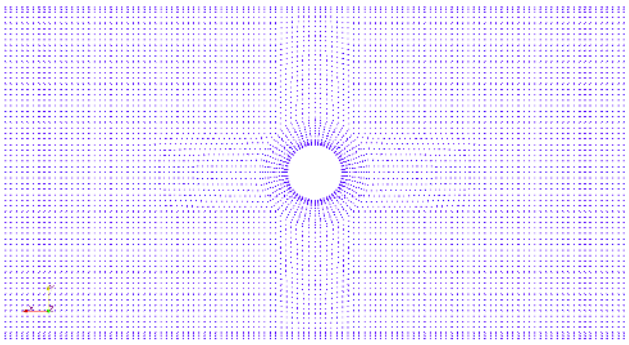


Figure 2.7: Visualization of meshfree domain as discrete point set

geometric primitives, such as volumetric ray casting, are also used for rendering meshfree geometries. The general process involves casting an eye ray through each pixel of the image into the volume to be visualized, where the ray is sampled at a specified step size. At each sample position, trilinear interpolation is used to determine the scalar value from the grid of precomputed data values. This scalar value is then mapped to a color and opacity via a transfer function. While volumetric ray casting does have promising results for rendering

meshfree geometries, complications with selecting transfer functions and the cloudy nature of the resulting images need to be addressed. A challenge that the previously described visualization techniques do not address is the ability to treat the meshfree domain as a general geometric primitive. In the case where a combination of meshfree methods and element based methods are used, a transformation must be performed to generate a data type that can be rendered. In order to visualize a geometry, the geometric data must be in the same form, either volumetric or represented as a standard geometric type such as triangles.

2.4.4 Computational Efficiency

While meshfree methods offer some advantages over FEM, they also possess several disadvantages, including increased computational time and, depending on implementation, memory overhead. This cost is attributed to the increased complexity of both shape function evaluation and determination of connectivity. There are three types of connectivity information required in geometric modeling and analysis as enumerated below, and discussed in detail in §3.

Type 1: *Field point - particle*, all the particles that contain a field point y in their support.

Type 2: *Particle - field point*, all the field points within the support of a given particle.

Type 3: *Particle - particle*, all particles whose support overlap a given particle's support.

The first type of connectivity arises in evaluating quadrature points and problems involving contact between bodies. The second type comes about when one has quantities at field points and wishes to project those values to the particle locations, e.g. stress or strain values. The

third type is useful for determining the sparsity pattern and entries in the stiffness matrix of a Galerkin solution to a partial differential equation (PDE). The various connectivities are a type of graph and can be represented through an *adjacency matrix* [15].

In FEM, a common approach is to use the isoparametric concept where the shape functions are formulated in a parent or natural coordinate system and then the geometric domain values are computed by applying an affine transformation. The description of the mesh can be used as a graph to provide the first two types of connectivity. Since Gauss quadrature is an efficient method for integrating FEM shape functions, they are normally defined in the parent domain, eliminating the need to find the reverse transformation from geometric to parent coordinates and evaluating Type 1 connectivity directly. If one uses conforming shape functions, Type 3 connectivity can also be quickly determined from the mesh. In the case of stiffness matrix assembly using the element viewpoint, explicit Type 3 information is not required. Since the parent element is fixed, the parent shape functions are fixed and simple to compute. In contrast, meshfree shape functions are constructed for the current point of interest based on the collection of surrounding particles that have a non-zero contribution at the point. This collection of contributing particles varies with the spatial position of the point to be evaluated and there is no predefined adjacency information, making the construction of meshfree shape functions more involved compared to FEM. The computation of the meshfree shape functions also require the computation of a correction term at each point of interest in the domain. Computation of the correction term requires the inversion of the moment matrix. While the moment matrix is not particularly large, the number of inversions required for a practical problem may be significant. To alleviate the computational cost of computing the inverse, a method to avoid the direct computation is given in [6]. Research

to reduce the cost of meshfree analysis by reducing the number of necessary shape function evaluations by nodal integration has been published in [17] and [16].

To summarize, the increased cost of meshfree analysis over FEM is due to increased shape function evaluation cost, increased number of shape function evaluations in quadrature when using a background mesh, and evaluation of the various connectivities. This work will only focus on the latter.

CHAPTER 3: EFFICIENT SEARCH SCHEMES FOR PARTICLE METHODS

[1] The first objective of this chapter is to provide a thorough review of the adjacency information that is commonly required in geometric modeling and analysis when employing a meshfree method, accomplished in §3.1. The second objective is to describe the various approaches that can be used for determining the adjacency information. Since the reader may not be familiar with data structure concepts from Computer Science, §3.2 is provided as background for a new data structure for meshfree searches in §3.2.2.2. The final objective is to demonstrate the effectiveness of the various approaches under a variety of circumstances, in order to aid those using meshfree methods in selecting an appropriate search scheme for their problem, presented in §3.3. While this work is focused on applications involving meshfree methods, the primary algorithm discussed is essentially a nearest neighbor search and therefore the data structures presented can be abstracted to any set of compact non-disjoint or disjoint objects. The data structures are capable of operations such as ray-object intersections, which occur in computer graphics, but are also of interest for meshfree methods when enforcing visibility conditions.

[1]This chapter was published in the journal Computational Mechanics. James Olliff, Brad Alford, and Daniel C. Simkins. Efficient searching in meshfree methods. *Computational Mechanics*, April 2018. https://doi.org/10.1007/s00466-018-1574-9. Permission is included in Appendix A.

3.1 Adjacency Information

The finite set of particle locations discretizing the domain is denoted as $\mathbb{X}$, which is assumed to be indexed. The finite set of field points where shape functions will be computed will be denoted as $\mathbb{Y}$, which is also assumed to be indexed. The term *evaluation point* will refer to points in $\mathbb{Y}$, to distinguish them from the particle locations. These two sets are not necessarily disjoint, as one may evaluate shape functions at particle locations. Given these two sets, there are three connectivities which commonly arise when employing a meshfree method. Any given type of connectivity is represented as a set of sets, indexed by either a particle index, or an evaluation point index. Each entry I in the connectivity is a set of indices of objects adjacent to I defined by the type of adjacency.

Type 1: Given an evaluation point, $\mathbf{y}_J \in \mathbb{Y}$ define the index set, Θ^J, containing the indices of those particles that have a non-zero shape function at $\mathbf{y}_J$, which can be stated as

$$\Theta^J = \{I | \mathbf{x}_I \in \mathbb{X} \wedge \Psi_I(\mathbf{y}_J) \neq 0\}. \tag{3.1}$$

Evaluation point - particle adjacency, denoted Θ, is defined to be the set whose elements are the Θ^J.

Type 2: Given a particle, $\mathbf{x}_I \in \mathbb{X}$, define the index set, Λ^I, containing the indices of those evaluation points that result in a non-zero shape function value with respect to particle $\mathbf{x}_I$, or more formally

$$\Lambda^I = \{J | \mathbf{y}_J \in \mathbb{Y} \wedge \Psi_I(\mathbf{y}_J) \neq 0\}. \tag{3.2}$$

Particle - evaluation point adjacency, Λ, is the set whose elements are the Λ^I. Figure 3.1 shows a graphical depiction of Equation 3.2. The circles and squares represent points in $\mathbb{Y}$. The particle of interest $\mathbf{x}_I$ is shown as a triangle with its corresponding support domain represented by the circle. The index set of evaluation points that result in a non-zero shape function value with respect to particle $\mathbf{x}_I$ are denoted by the squares.

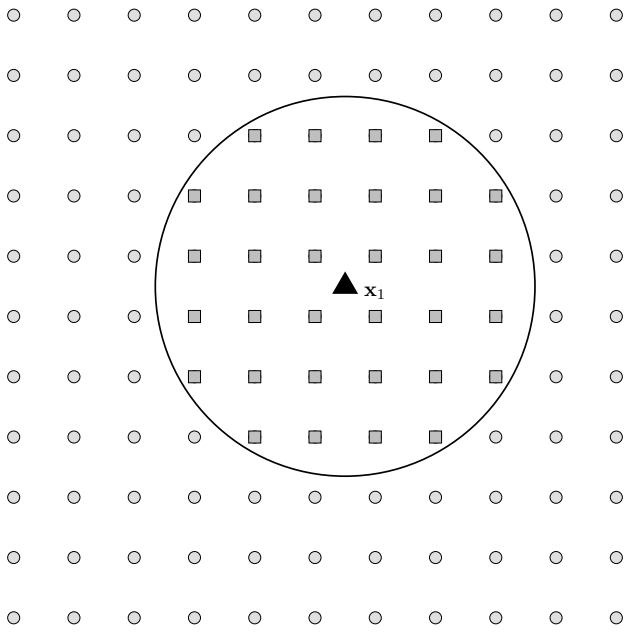


Figure 3.1: Graphical depiction of Λ^I, an element in Type 2, particle - evaluation point adjacency.

Type 3: Given a particle, $\mathbf{x}_I$, determine the index set of particles, Π^I whose support domain overlaps with the support of $\mathbf{x}_I$, specifically

$$\Pi^I = \{K | \mathbf{x}_K \in \mathbb{X} \wedge supp\left(\mathbf{x}_I\right) \cap supp\left(\mathbf{x}_K\right) \neq \emptyset\}. \tag{3.3}$$

The particle-particle adjacency is given as the set Π whose elements are the Π^I.

The prescription in Equation 3.3 is applicable in a continuous setting. In the solution of PDEs employing a numerical integration technique, the discretization will result in particle to particle adjacencies that do not occur due to the discrete nature of the integration. This situation can be seen in Figure 3.2. Here the lightly shaded symbols represent the integration grid and the particles are denoted by dark triangles and labeled $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$. According to Equation 3.3 all three particles are neighbors, but since particles $\mathbf{x}_1$ and $\mathbf{x}_2$ do not share an evaluation point, their respective discrete adjacency is broken. The lightly shaded rectangles and diamonds are the evaluation points in the overlapping supports yielding $\mathbf{x}_1$ and $\mathbf{x}_3$ as neighbors and $\mathbf{x}_2$ and $\mathbf{x}_3$ as neighbors, respectively. Therefore, Equation 3.3 can be restated

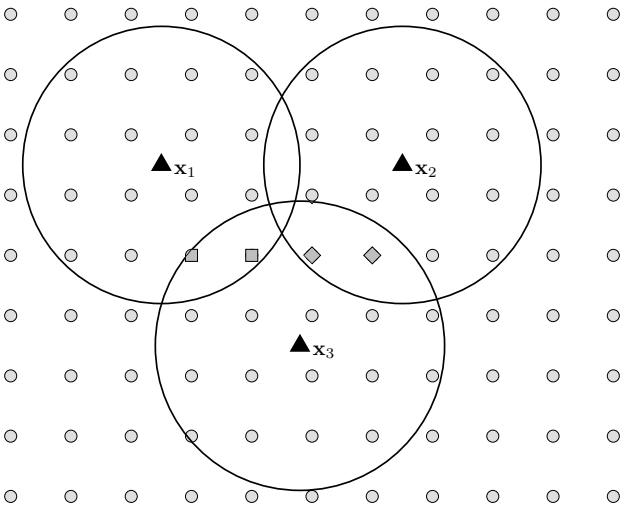


Figure 3.2: Graphical depiction of Type 3, particle-particle adjacency

to define the discrete particle-particle adjacency as

$$\Pi^I = \left\{ K | \mathbf{x}_K \in \mathbb{X} \wedge \Lambda^I \cap \Lambda^K \neq \emptyset \right\}. \tag{3.4}$$

3.1.1 Relation Between Type 1 and Type 2 Adjacency

The Type 1 and Type 2 adjacencies are not mutually independent. Given the evaluation point to particle adjacency information, Θ, the particle to evaluation point information, Λ^I, for particle I can be stated as

$$\Lambda^I = \left\{ J | \mathbf{y}_J \in \mathbb{Y} \wedge I \in \Theta^J \right\}. \tag{3.5}$$

Similarly, if the particle to evaluation point information, Λ, is known the evaluation point to particle information, Θ^J, for evaluation point J is

$$\Theta^J = \left\{ I | \mathbf{x}_I \in \mathbb{X} \wedge J \in \Lambda^I \right\}. \tag{3.6}$$

Note that since Equation 3.4 does not contain any information on evaluation points, it cannot be used by itself to construct Type 1 or Type 2 adjacency, and alone is insufficient for meshfree applications involving evaluation points. In a formulation in which exact integration over analytic geometric domains were used, this might not be the case. Given the insufficiency of Type 3 adjacency and its construction through Equation 3.4, it will not be further discussed.

The dual nature of Type 1 and Type 2 adjacency means that either adjacency can be used to fulfill any role in which either type is required. However, some applications are more naturally addressed by one type than the other. It should be noted that it is possible to construct all three types of adjacency simultaneously. However, in a multi-threaded computation, this will lead to race conditions impeding efficient shared-memory parallel

(SMP) performance. This work will focus on solutions that both work naturally for their purpose and scale well in multi-threaded implementations.

3.1.2 Example Uses of Adjacency

In a finite-dimensional Galerkin solution of PDEs, the weak form of the governing equations needs to be integrated. This results in the formation of a system of algebraic equations; this procedure is known as assembly. In the context of solid mechanics, the coefficient matrix is known as the stiffness matrix, whose entries depict the interaction between the degrees of freedom. Generally speaking, there are two approaches that can be used to perform assembly, the evaluation point-wise or the particle-wise approach.

In the evaluation point-wise method, all contributions to the various matrix elements from the single evaluation point are computed and accumulated. Successively considering each evaluation point results in a complete stiffness matrix. This approach requires Type 1 adjacency, which is the information contained in Θ. The algorithm for the evaluation point-wise assembly is given in Algorithm 1.

Algorithm 1: Evaluation point wise assembly of stiffness matrix

Input: $\mathbb{Y}$ & $\mathbb{X}$
Output: $\mathbf{K}^G$

```
Initialize K^G ;
for y_I ∈ 𝕐 do
    Determine Θ^I;
    Evaluate shape functions and derivatives at y_I;
    for x_J ∈ Θ^I do
        for x_K ∈ Θ^I && K ≥ J do
            K^G_JK += B^T_J D B_K
```

In this assembly procedure each evaluation point affects multiple degrees of freedom, which means multiple entries in the global matrix will be affected by the same evaluation point. This presents a race condition when parallelizing over the evaluation points, as multiple evaluation points will need to write to the same location in memory.

Alternatively, the assembly procedure can be done on groups of evaluation points, as is often done in FEM. The group of evaluation points used are those which affect the same degrees of freedom. In finite elements this is simply the Gauss points contained within an element with the nodes being those of the element. Instead of adding the contributions of each evaluation point into the global matrix, a local or element matrix is created, which is then added to the global matrix. Note this approach also has a race condition on an SMP.

The second assembly procedure is based on the particle viewpoint. In this approach the particle to particle, Π, and particle to evaluation point, Λ, adjacencies are needed. The procedure is to iterate over the particles and for each particle, $\mathbf{x}_I$, determine the particle to particle adjacency Π^I. Then for each particle in Π^I determine the evaluation points that are common between each particle pair and compute the entry in the global matrix. The algorithm for the particle-wise assembly is given in Algorithm 2.

Algorithm 2: Particle wise assembly of stiffness matrix

Input: $\mathbb{Y}$ & $\mathbb{X}$
Output: $\mathbf{K}^G$
Initialize $\mathbf{K}^G$;
for $\boldsymbol{x}_I \in \mathbb{X}$ **do**
    Determine Λ^I;
    for $\boldsymbol{y}_J \in \Lambda^I$ *with corresponding* Θ^J **do**
        for $x_K \in \Theta^J$ && $K \geq I$ **do**
            $\mathbf{K}^G_{IK} += \mathbf{B}_I^T \mathbf{D} \mathbf{B}_K$;

This approach allows for straightforward parallelization over the particles. However, if the shape functions are not pre-computed and stored, a large number of redundant calculations will occur, since the same evaluation point will be in multiple particles' support domain. To avoid this, the evaluation point to particle adjacency information Θ can be pre-computed and stored along with shape function values. Depending on the size of the problem this memory overhead could result in a large payoff in run time since the shape function values are needed not only during assembly, but often during post-processing.

The matrix assembly example assumes that all of the quadrature points and particles are known, and either Type 1 or Type 2 adjacency can easily be used. On the other hand, one may have a problem in which either the evaluation points or the particles are known only at run-time. Such applications would be contact problems, where additional quadrature points along boundaries in contact are determined during the course of the simulation. Another example would be dynamic refinement by particle insertion. Depending on how adjacency is stored and constructed, one type may be easier than the other.

A naive approach to determining the adjacency information contained in Θ or Λ would involve checking all the particles against all the evaluation points. This would result in an $\mathcal{O}(N_e * N_p)$ algorithm, and the particle to particle interactions would have complexity $\mathcal{O}\left(\frac{N_p(N_p+1)}{2}\right)$. These naive approaches are very inefficient as they have not exploited the fact that particles are compactly supported. In order to reduce this cost, a data structure that excludes most of the objects from the search can be utilized. Data structures of this type are of fundamental importance in many fields of computer science, such as computer graphics. The construction of this type of data structure requires either the space in which the objects reside to be partitioned or the set of objects themselves to be partitioned.

3.2 Data Structures

A data structure is simply a method for organizing, storing, and retrieving data. All data structures inherently balance a number of attributes: memory overhead, search times, access, retrieval, and dynamic size. Depending upon the application, one of these attributes may take precedence over the others, and data structures focused on optimizing that attribute can be developed. In the context of meshfree methods, there are two options for data to be stored. One can store evaluation points, or one can store particles. The former have zero extent, but the latter carry volume information in the form of the size of their support. Both Type 1 and Type 2 queries must account for the support size of the particles. Note that particle extent can be accounted for either in the construction of the data structure, when storing particles, or during a search. Therefore, three different viewpoints can be taken when developing a data structure for conducting adjacency queries in meshfree methods, described in the enumeration below.

Case 1: Use particles for construction and search per evaluation point.

Case 1a: account for particle support size during data structure construction

Case 1b: account for particle support size during search.

Case 2: Use the evaluation points for construction and search per particle.

The key to efficient searching of a data structure is to reduce the number of potentially expensive detailed evaluations of the search condition. In meshfree methods, the search condition typically entails determining whether a given point is within the support of a particle. Given that evaluation points and particles are distributed in space, a natural

approach to reduce the number of detailed support checks is to organize the data so that points and particles that are far from each other are excluded from detailed checks. The following sections will describe the two basic approaches to partitioning the data, spatial partitioning and object partitioning. The data structures and associated algorithms will be presented in an abstract manor. The purpose of this is not to obfuscate the implementation details, but to provide the reader with a generic algorithm that could be applicable to problems outside the context of meshfree methods. In addition, the spatial partitioning data structures can be used to determine both Type 1 and Type 2 adjacency with only minor changes to the algorithm. For brevity these subtleties have been abstracted, allowing for a single algorithm associated with each data structure for both searching and construction. The specifics of applying these algorithms for answering the adjacency questions that arise in meshfree methods will be given in §3.2.1.4.

3.2.1 Spatial Partitioning

Spatial partitioning data structures decompose the space in which the objects reside into disjoint regions and distribute the objects into the resulting partitions. A drawback to this type of structure arises for objects that have a non-zero spatial extent, or size. Often this results in an object overlapping multiple partitions, leading to multiple references to the same object. Pursuing the goal of using the data structure to reduce the candidate sets of objects to be checked, a relatively fine subdivision results, relative to the support size of the particles, and thus many particles have multiple references in nearby subdivisions. For a large collection of objects these references may result in a sizable memory footprint. In the context of meshfree methods, the particle supports are required to overlap, thereby guaranteeing that

each spatial subdivision intersects multiple particle supports. To circumvent the memory issue while still using a spatial partitioning data structure two options exist. The first is to construct the data structure on the particles, but not account for their supports. The second is to construct the data structure on the evaluation points, which have zero volume. Three spatial partitioning structures will be discussed: grids, kd-trees, and the well-known octree/quadtrees. Further discussion on spatial partitioning data structures can be found in [11, 65].

3.2.1.1 Regular Grids

The grid spatial subdivision was proposed as an acceleration method for generating images using ray tracing by Fujimoto and Iwata in 1985 [26]. The concept of a grid is to subdivide an axis aligned space into equal sized rectilinear regions or cells. Each cell stores references to the objects that overlap it or are contained within it. While the grid data structure was originally designed to accelerate ray-object intersections, it can also be used during query operations occurring in meshfree methods, discussed in the following two sections.

3.2.1.1.1 Construction

The algorithm for constructing a grid is given in Algorithm 3. The construction routine operates on the a set of geometric objects, $\mathbb{Y}$, with bounding box $\mathbf{B}$. The result of the construction is an array of grid cells $\mathbf{C}$, where each cell contains a set of references to those objects associated with the cell. The first task of the construction routine, Line 1, is to determine the resolution of the grid. The resolution of the grid can be a prescribed value by the user, but determining an optimal setting for this value is non-trivial and some

choices can result in a subdivision that is either too coarse, causing poor performance, or an overly refined grid that results in a large memory footprint. To address this issue we use the approach to determine the resolution such that the total number of cells in the grid is linearly proportional to the number of objects in the grid and the cells are approximately cuboidal in three-dimensions or square in two-dimensions [63]. Using these conditions the resolution of the grid can be computed as shown by Equation 3.7.

$$n_i = \left\lceil \Delta y_i \left(\frac{\gamma N_p}{\prod_j^d \Delta y_j} \right)^{\frac{1}{d}} \right\rceil \qquad i = 1, \ldots, d \tag{3.7}$$

Here n_i is the number of cells along direction i, Δy_i is the ith extent of the grid, N_p is the total number of objects in the grid, d is the spatial dimension, and γ is a proportionality constant that linearly relates the number of objects to the total number of grid cells, i.e., $N_{cells} = \gamma N_p$. Choosing an appropriate value of γ requires experimentation and will vary from problem to problem. The authors have found values ranging from one to ten to be a good choice for most problems. Choosing a larger value will result in a more refined grid, which could reduce the number of objects residing in a single cell. Given the resolution, Line 2 allocates an array of empty cells corresponding to the resolution. Determining the indices of a cell containing a specific coordinate is a linear interpolation problem along each spatial dimension as shown in Equation 3.8.

$$\frac{i_j - i_j^{min}}{i_j^{max} - i_j^{min}} = \frac{y_j - y_j^{min}}{y_j^{max} - y_j^{min}} \qquad j = 1, \ldots, d \tag{3.8}$$

The upper and lower cell indices along the jth direction are defined by i_j^{max} and i_j^{min} respectively. The upper and lower limits of the grid are defined as y_j^{max} and y_j^{min} and y_j representing the jth component of the known coordinate. Assuming a zero based indexing scheme for the cell numbering and solving for the unknown cell number i_j leads to the following expression,

$$i_j = \left\lfloor \left(\frac{y_j - y_j^{min}}{\Delta y_j} \right) n_j \right\rfloor \qquad j = 1, \ldots, d \tag{3.9}$$

Equation 3.9 does suffer from one pitfall in practical applications, which is the scenario where y_j is greater than or equal y_j^{max}. This will result in $i_j = n_j$, but the cells are numbered such that $i_j \in [0, n_j - 1]$. To handle this situation a simple routine to clamp the values to the interval $[0, n_j - 1]$ is used. With the ability to compute the indices of a cell associated with a single coordinate the cell numbers pertaining to those cells intersecting the geometric object can be determined by considering an axis aligned bounding box (AABB) encompassing the domain of $\mathbf{y}$, which provides two spatial locations that define the range of cells the particle intersects. The cells in this range are updated with a reference to $\mathbf{y}$.

Algorithm 3: ConstructGrid($\mathbb{Y}, \mathbf{B}$)

Input: Set $\mathbb{Y}$ of geometric objects and Bounding Box $\mathbf{B}$
Output: Array of grid cells $\mathbf{C}$

```
[n_x, n_y, n_z] ← Determine grid resolution ;
C ← Create n_x x n_y x n_z array of grid cells ;
for y ∈ 𝕐 do
    I ← Compute indices into C of grid cells intersecting supp(y) ;
    for i ∈ I do
        Insert y into C[i] ;
return C ;
```

3.2.1.1.2 Searching

The regular grid search algorithm, given in Algorithm 4, seeks to determine those objects that intersect the search domain, **S**, defined here by a location and radial vector as shown in Figure 3.3. The algorithm begins with an intersection test against the search domain and

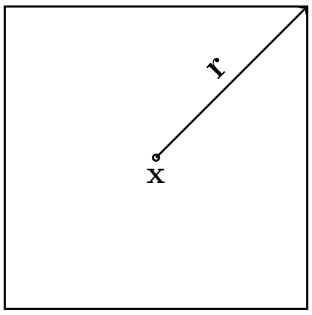


Figure 3.3: Search domain

the bounding box of the grid. If no intersection occurs the search is terminated. Given an intersection does occur the cells that have a non-zero intersection with the search domain are determined and the objects residing in these cells are then queried to determine those that intersect the search domain.

Algorithm 4: SearchGrid(**C**,**S**)

Input: Array of grid cells **C** and search region **S**
Output: Set of objects Θ intersecting **S**

```
1 I ← Compute indices into C of grid cells intersecting S ;
2 for i ∈ I do
3 |   for x ∈ C[i] do
4 |   |   if S ∩ x then
5 |   |   |   Insert x in Θ ;
6 return Θ ;
```

3.2.1.2 Kd-Tree

The kd-tree [10] is a popular variant of the more general Binary Space Partitioning (BSP) tree [25]. The kd-tree adaptively decomposes a space into disjoint rectilinear partitions. This differs from the uniform grid decomposition of the space in that it adapts to irregularly distributed objects. A kd-tree restricts the splitting plane to be orthogonal to one of the coordinate axes. This restriction allows for efficient construction with the sacrifice of how the space is divided.

3.2.1.2.1 Construction

The general procedure for constructing a kd-tree is given in Algorithm 5. The input to the construction routine is a set of geometric objects, $\mathbb{Y}$, with associated bounding box $\mathbf{B}$. The output of the construction algorithm is a kd-tree τ. The routine begins by checking if the termination criteria has been met. Several options exist for the termination criteria. The first possibility uses a preset limit on the number of objects that a single kd-tree node is allowed to contain, commonly referred to as the bin size. Another possibility for the termination condition is restricting the height of the tree. In addition, the volume of the half space that a tree node represents could be restricted, this termination condition is useful when the geometric objects the tree is constructed upon are of a non-zero volume. Once the termination condition is satisfied the splitting of the tree node is stopped and a single leaf node is constructed and updated with references to the objects that are contained within the node. If the termination criteria is not met the algorithm proceeds to Line 4 where a

splitting plane, $\mathcal{P}$, is determined. Several options for determining a split plane are shown below.

- *Median splitting* The splitting dimension is the dimension with the greatest variation. The splitting location is taken as the median of the coordinates along this dimension.
- *Cyclic splitting* Similar to the median splitting rules except the splitting axis is chosen in a cyclic manner. That is the root will start with the split axis being along the x-axis, the next level of the tree will then use the y-axis as the split directions and the next level using the z-axis.
- *Midpoint splitting* Similar to the median splitting rule, the split direction is determined by the the greatest variation in the bounding box of the tree node being split. The split location is then taken as the mid-point of this side.

Given $\mathcal{P}$, the left and right half spaces can be defined. Line 7 computes the intersection of $\mathbb{Y}$ with $\mathcal{H}^L$ resulting in the subset $\mathbb{Y}^L = \{y_j \in \mathcal{H}^L\}$. The $ConstructKDTree$ function is then called recursively at Line 8 with $\mathbb{Y}^L$ as the input, resulting in the left kd-tree node τ^L. A similar procedure is conducted for the right half space. Line 11 creates a single tree node and with references to the left and right child nodes.

3.2.1.2.2 Searching

The kd-tree search algorithm is given in Algorithm 6. The inputs to the search routine are the constructed kd-tree, τ, from Algorithm 5 that is rooted at tree node $\mathbf{r}$ and a search domain. The result of Algorithm 6 is the set of objects, $\Theta \in \tau$, that have a non-zero intersection with the search domain $\mathbf{S}$. The search begins by checking if the root of the input kd-tree is a leaf node. If this condition is true the algorithm queries each item associated with

Algorithm 5: ConstructKDTree($\mathbb{Y}, \mathbf{B}$)

Input: Set $\mathbb{Y}$ of geometric objects and Bounding Box $\mathbf{B}$
Output: A kd-tree τ

```
1  if termination criteria is met then
2      τ ← KDNode(𝕐) ;
3  else
4      Choose splitting plane 𝒫 ;
5      ℋ^L ← left half space with upper bound 𝒫 ;
6      ℋ^R ← right half space with lower bound 𝒫 ;
7      𝕐^L ← 𝕐 ∩ ℋ^L ≠ ∅ ;
8      τ^L ← ConstructKDTree(𝕐^L, B^L) ;
9      𝕐^R ← 𝕐 ∩ ℋ^R ≠ ∅ ;
10     τ^R ← ConstructKDTree(𝕐^R, B^R) ;
11     τ ← KDTree(τ^L, τ^R) ;
12 return τ ;
```

the given leaf to determine those that intersect $\mathbf{S}$ appending them to the set Θ. Provided the root of the input kd-tree is not a leaf node the algorithm proceeds to Line 6, where the split plane associated with the given node is retrieved. The split plane, $\mathcal{P}$ is then used to define the left half space in Line 7. An intersection test between the search region, and the left half space $\mathcal{H}^L$ is performed. Since the search region is capable of intersecting both left and right half spaces each must be tested independently of the other. If either intersection occurs the respective child node is used as the input to a recursive call to Algorithm 6. The resulting sets from Lines 11 and 14 are combined as shown by the union operation at Line 15.

3.2.1.3 Octree

An octree is a three-dimensional space partioning data structure that uses three mutually orthogonal planes to recursively decompose the space into eight axis-aligned boxes, or octants, at each step. The quad-tree is the two-dimensional analog to the octree, where

Algorithm 6: SearchKDTree(τ,**S**)

Input: Kd-tree τ with root **r** and search region **S**
Output: Set of objects Θ intersecting **S**

```
1  if r is a leaf node then
2      for x ∈ r do
3          if S ∩ x then
4              Insert x in Θ ;
5  else
6      P ← Retrieve split plane or r ;
7      H^L ← left half space with upper bound P ;
8      H^R ← right half space with lower bound P ;
9      if S ∩ H^L then
10         τ^L ← left child of r ;
11         Θ^L ← SearchKDTree(τ^L,S) ;
12     if S ∩ H^R then
13         τ^R ← right child of r ;
14         Θ^R ← SearchKDTree(τ^R,S) ;
15     Θ ← Θ^L ∪ Θ^R ;
16 return Θ ;
```

instead of the eight axis-aligned boxes used by the octree, the quad-tree recursively partitions the space into four rectilinear regions or quadrants. An octree differs from a kd-tree in several ways. The kd-tree is a binary partitioning data structure in that at each level the space is decomposed into two sub-regions, where as the octree decomposes the space into eight sub-regions at each level. In general, an expression for the size of a kd-tree node is not possible due to the varying nature of the split location and direction. The extents of an octree node can be expressed as

$$\Delta x_i^s = \frac{\Delta x_i^0}{2^s} \quad i = 1, \ldots, d. \tag{3.10}$$

Here Δx_i^s represents the extents of the octree node at level s of the tree with Δx^0 representing the bounds of the root node. In Equation 3.10 the depth value, s, is an identifier on the left side of the equals sign, but is the power when appearing on the right hand side.

3.2.1.3.1 Construction

The general procedure for constructing an octree is given in Algorithm 7. The inputs to the construction routine are the set of geometric objects, $\mathbb{Y}$, and the bounding box, $\mathbf{B}$, encompassing $\mathbb{Y}$. The output of the construction algorithm is an octree τ. The routine begins by checking if the termination criteria has been met. Several options exist for the termination criteria. Possible termination conditions include:

- if the number of objects within a leaf falls below a prescribed value
- if the number of subdivisions exceeds a prescribed value
- if the extents of the leaf reaches a minimum size, which is another form of the max subdivision criteria as the size of a tree node can be determined at each level of the tree provided the root node's extents using Equation 3.10.

Once the termination condition is satisfied a leaf node is constructed and updated with references to the objects that intersect it. If the termination criteria is not met the algorithm proceeds to Line 5 where three splitting planes, $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, which partition the current tree node into eight equal sized octants. The partitioning is performed around the current tree node's centroid. The next step involves partitioning the set of objects $\mathbb{Y}$ into eight subsets where the ith subset contains those objects that intersect child i of the current node. Following this partitioning the construction routine is recursively called as shown in Line

9. Once the recursion routine returns the constructed sub-tree is added to the current tree node.

Algorithm 7: ConstructOctree($\mathbb{Y}$,$\mathbf{B}$)

Input: Set $\mathbb{Y}$ of geometric objects and bounding box $\mathbf{B}$
Output: An Octree τ

```
1  if termination criteria is met then
2      Create octree with one leaf node τ ;
3      Store data from 𝕐 in τ ;
4  else
5      Compute planes 𝒫1, 𝒫2, 𝒫3 orthogonal to the coordinate axes ;
6      Decompose B into eight octants,{b^0, b^1, ..., b^7}, using splitting planes ;
7      for i ← 0 to 7 do
8          𝕐^i ← 𝕐 ∩ b^i ≠ ∅ ;
9          τ^i ← ConstructOctree(𝕐^i,b^i) ;
10         τ ← AddSubTree(τ, τ^i) ;
11 return τ ;
```

3.2.1.3.2 Searching

The octree search algorithm is given in Algorithm 8. The inputs to the search routine are the constructed octree, τ, from Algorithm 7 that is rooted at tree node $\mathbf{r}$ and the search region. The result of Algorithm 8 is the set of objects, Θ, in τ, that intersect the search domain. The search begins by checking if the root of the input octree is a leaf node. If this condition is true the algorithm queries each object associated with the given leaf to determine those objects that intersect $\mathbf{S}$, appending those that intersect to the set Θ. Provided the root of the input octree is not a leaf node the algorithm proceeds to Line 6, where the child nodes that the search region intersects are determined. This determination is based on the three planes used to partition the current node $\mathbf{r}$ into octants. This child node is then retrieved and used as first argument to a recursive call to Algorithm 8.

Algorithm 8: SearchOctree(τ,**S**)

Input: Octree τ with root **r** and search region **S**
Output: Set of objects Θ intersecting **S**

```
1  if r is a leaf node then
2      for x ∈ r do
3          if S ∩ x then
4              Insert x in Θ ;
5  else
6      I ← determine indices of child nodes that intersect S ;
7      for i ∈ I do
8          Θ^i ← SearchOctree(τ^i,S) ;
9          Θ ← Θ ∪ Θ^i ;
10 return Θ ;
```

3.2.1.4 Meshfree Details

The above algorithms can be used to answer the adjacency questions arising in meshfree methods. Each of the three data structures described can be used to determine either Type 1 or Type 2 adjacencies.

Using a spatial partitioning data structure for determining evaluation point to particle, Type 1, adjacency information the Case 1a or Case 1b viewpoint is taken with the space containing the particles being partitioned. For Case 1a the space encompassing the particle locations and their supports are used and only the particle locations for Case 1b are used. The concept of an intersection between the particles and a cell or node is utilized in each construction algorithm. For the Case 1a this intersection accounts for the support of the particle, where in Case 1b this intersection only considers the spatial position of the particle. Therefore, the intersection used in the Case 1b construction results in a single intersection,

i.e., a particle can associate with at most one cell or tree node. This differs from Case 1a where a particle support can intersect multiple cells or tree nodes.

The search is then conducted per evaluation point resulting in those particles that contribute at the given evaluation point. Each of the search algorithms requires a search domain. The search region for the Case 1a is the evaluation point location or a region with zero extent. The search method in Case 1b requires a search domain be assigned to the evaluation point. In general the particles are associated with a support size, and the evaluation point is not. A method to choose a bounds for the search must be applied. A solution that is guaranteed to work is to choose the supremum of the particle supports. However, this approach could result in a large number of particles needing to query the evaluation point if particle distribution is graded.

When determining the particle to evaluation point, Type 2, adjacency information, the spatial partitioning data structures decompose the space containing the evaluation points and search per particle as described by Case 2. The only aspects that differentiate this case from Case 1 are the input arguments and the results. Here the construction is done on the evaluation point locations instead of on the particle locations. The search for Case 2 is done per particle with the search region being the particle's support domain.

3.2.2 Object Partitioning

Object partitioning structures recursively subdivide the collection of primitives into disjoint sets. Therefore, object hierarchies do not suffer from the additional memory requirement associated with multiple references to the same object, since the object is at most referenced once. Data structures of this type have received little attention in the context

of meshfree methods. This could be due to the fact that querying a single point is not a common operation; instead, all of the points of interest are evaluated at once, making the spatial partitioning data structures constructed of evaluation points feasible. However, a spatial partitioning data structure is not feasible if all the evaluation points are not known *a priori*. This case may arise, for example, in problems involving contact, where the points of contact are part of the problem to be solved [41]. Another disadvantage of determining adjacency information on a per particle basis is the race conditions that are present, which requires special attention to allow for parallelization. The race condition potentially arises when storing the shape functions from multiple particles at the same evaluation point. A final disadvantage to using a Case 2 structure for shape function computations occurs when one prefers to compute shape functions on the fly, rather than pre-compute and store them. In the former case, one would have to evaluate the set Θ^J from Equation 3.6. This could be very costly. Alternatively, a spatial partitioning data structure following the Case 1a or Case 1b mantra can be used as described in the previous sections. While these structures do address the challenges associated with the Case 2 variants, they do present their own challenges.

3.2.2.1 Bounding Volume Hierarchy

A well-known object partitioning data structure is the Bounding Volume Hierarchy (BVH). As object partitioning data structures are designed to partition objects with non-zero volume, they naturally address the adjacency query corresponding with Case 1a. Bounding Volume Hierarchies, introduced in [53], partition the set of objects into a hierarchy of

non-disjoint sets. References to the objects are stored in the leaves and each node stores a bounding box of the primitives in the nodes beneath it.

3.2.2.1.1 Construction

The construction of a BVH is performed on the particles accounting for their support domains. The BVH construction shown in Algorithm 9 resembles the construction of the kd-tree in Algorithm 5. Comparison of the two algorithms shows the only apparent difference being the computation of the bounding-box at Line 1 of Algorithm 9. However, another significant difference exists that is masked by the abstract intersection between the particles in $\mathbb{X}$ and the half spaces. These intersections for the Case 1a kd-tree were between the support of the particles and the half spaces, but the intersections shown in Algorithm 9 are between the coordinates of the particles and the half spaces. Since the classification is based on a spatial position, not on a volume, there are no repeated references as in the Case 1a spatial partitioning structures. Once the set of particles have been placed in their respective tree node the newly created leaves must recompute their bounding boxes accounting for the support of the particle. In contrast to spatial partitioning data structures, where the splitting plane defines one of the extents of a tree node's bounding box, the extents of the bounding box for tree nodes can span across this plane overlapping one another.

3.2.2.1.2 Searching

As the BVH is constructed of particles, the goal of the search algorithm is to query the BVH with an evaluation point and determine Type 1 adjacency. The search process shown in Algorithm 10 begins with a point inside test between the root's bounding box

Algorithm 9: ConstructBVH($\mathbb{X}$)

Input: Set of particles $\mathbb{X}$
Output: A BVH τ

```
B ← ComputeBoundingBoxSupports(X) ;
if Termination criteria is met then
    τ ← BVH(B, X) ;
else
    Choose splitting plane P ;
    H^L ← left half space with upper bound P ;
    H^R ← right half space with lower bound P ;
    X^L ← X ∩ H^L ≠ ∅ ;
    τ^L ← ConstructBVH(X^L) ;
    X^R ← X ∩ H^R ≠ ∅ ;
    τ^R ← ConstructBVH(X^R) ;
    τ ← BVH(B, τ^L, τ^R) ;
return τ ;
```

and the evaluation point. If the point is not inside the root's bounds, then the search is terminated. Provided the evaluation point is inside the root's bounding box, the search proceeds by determining if the root is a leaf node. If the node is a leaf, the algorithm iterates through the particles associated with the node and performs a containment query between the evaluation point and each particle. If the root has children, then each child node of **r** is retrieved and a point inside test is used to determine if the search should proceed with a recursive search using that node. Once each child node has been processed, the resulting sets are combined and returned. While the BVH search routine shown in Algorithm 10 follows a similar procedure to that of the kd-tree search routine previously discussed, several key differences exist. The first is the determination of whether a child node should be processed. This selection is done using a splitting plane in the kd-tree. This differs from the BVH, which conducts a point inside bounding box test. Another difference is the combining of results from each subtree search. For the BVH these results are guaranteed to be disjoint

allowing for a simple structure to be used for combining these results. However, for the Case 1a kd-tree variant, the results are not necessarily disjoint requiring a more sophisticated data structure or algorithm for combining the results.

Algorithm 10: SearchBVH(τ,$\mathbf{y}$)

Input: BVH τ rooted at $\mathbf{r}$, Evaluation point $\mathbf{y}$
Output: Set of particles Θ with $\mathbf{y}$ in their support

```
1  if r is a leaf node then
2      for x ∈ r do
3          if y ∈ supp(x) then
4              Insert x in Θ ;
5  else
6      B^L ← Retrieve left child's bounding box ;
7      B^R ← Retrieve right child's bounding box ;
8      if y ∩ B^L then
9          τ^L ← left child of r ;
10         Θ^L ← SearchBVH(τ^L,y) ;
11     if y ∩ B^R then
12         τ^R ← right child of r ;
13         Θ^R ← SearchBVH(τ^R,y) ;
14     Θ ← Θ^L ∪ Θ^R ;
15 return Θ
```

3.2.2.1.3 Remarks

Unlike the spatial partitioning data structures, the BVH partitions the set of particles and treats them as individual bounding boxes. Therefore, the BVH falls under the Case 1a description and is best suited for determining Type 1 adjacency. If the BVH were to be constructed of objects with no extents, it would be identical to the kd-tree.

3.2.2.2 A New Support Tree Structure

A new data structure is now introduced that can be classified as an object partitioning structure with a tree-type hierarchy. This data structure will be composed of a root node, collection of internal nodes, and leaf nodes. The root node and internal nodes are identical in that they are composed of three child nodes, a split plane, and a bounding box. The leaf nodes are similar, but instead of a split plane and child nodes, they contain references to the data that is contained within them.

3.2.2.2.1 Construction

The construction uses a divide-and-conquer algorithm, also referred to as top-down, that groups the objects according to a splitting heuristic. The algorithm begins with all the particles at the top level or root as illustrated in Figure 3.4. The bounding box of the root node is taken as the minimum box that contains the particles and their supports. Construction proceeds with the determination of the location and direction to be used for the splitting of the objects. The current approach employs a mid-point splitting heuristic, where the split direction is determined by computing the largest deviation along the current node's bounding box. The location is then taken to be the mid-point along this direction. Considering the domain coverage in Figure 3.5a and the greatest deviation to be along the horizontal or x-direction, the split plane is shown as the blue vertical line.

With the split plane determined, the particles are classified as either strictly left, intersecting the split plane, or strictly right. This classification is done by determining whether the given particle's support bounding box overlaps the split plane. If the bounding box does

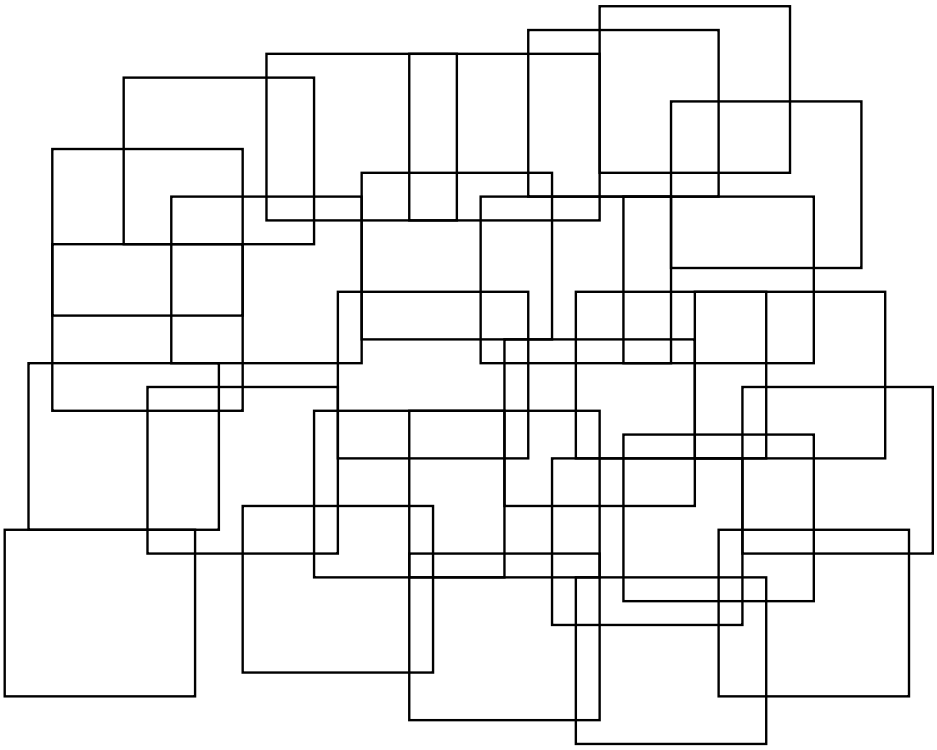


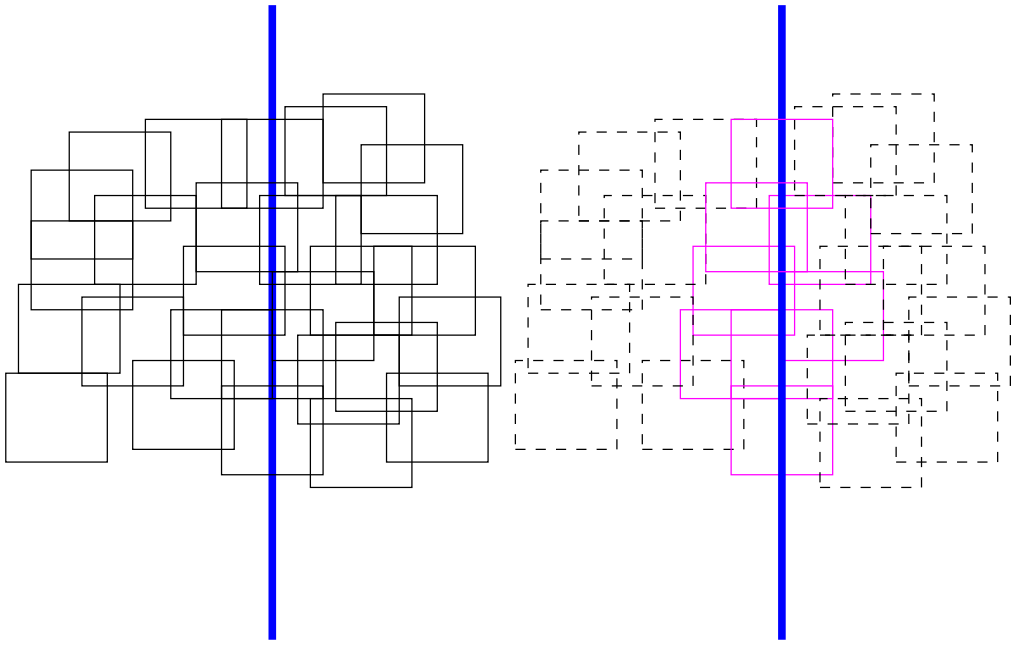
Figure 3.4: Domain coverage

overlap, the particle is classified as belonging to child one; if the bounding box does not overlap, then the particle is classified as belonging to child zero or two. This depends on which side of the split plane the support bounding box lies, with child zero corresponding to the left of the split plane and child two corresponding to the right. In Figure 3.5b the magenta boxes represent the supports of the particles that belong to child one, those particles with dashed lines that are left of the blue line belong to child zero, and the remaining particles belong to child two. As the objects are partitioned and distributed to the appropriate child nodes, the bounding box for each new child node is updated to enclose the particles and their supports. This procedure is repeated for each child node until a termination criteria is met. The termination criteria used in this work occurs when all the particles' supports overlap the splitting plane. Pseudo code for the construction is given in Algorithm 11.

Algorithm 11: ConstructSupportTree($\mathbb{X}$)

Input: Set of particles $\mathbb{X}$
Output: A Support Tree τ

```
Choose splitting plane 𝒫 ;
ℋ^L ← left half space with upper bound 𝒫 ;
ℋ^R ← right half space with lower bound 𝒫 ;
for x ∈ 𝕏 do
    if supp(x) ∩ ℋ^L ≠ ∅ && supp(x) ∩ ℋ^R == ∅ then
        Insert x into 𝕏^L
    else if supp(x) ∩ ℋ^L ≠ ∅ && supp(x) ∩ ℋ^R ≠ ∅ then
        Insert x into 𝕏^C
    else
        Insert x into 𝕏^R
if Cardinality(𝕏^L) == 0 && Cardinality(𝕏^R) == 0 then
    Create Support Tree with center leaf node τ ;
    Store data from 𝕏 into τ ;
else
    if Cardinality(𝕏^L) ≠ 0 then
        τ^L ← ConstructSupportTree(𝕏^L) ;
        τ ← AddSubTree(τ^L) ;
    if Cardinality(𝕏^C) ≠ 0 then
        τ^C ← ConstructSupportTree(𝕏^C) ;
        τ ← AddSubTree(τ^C) ;
    if Cardinality(𝕏^R) ≠ 0 then
        τ^R ← ConstructSupportTree(𝕏^R) ;
        τ ← AddSubTree(τ^R) ;
return τ ;
```



(a) Splitting plane location for root node (b) Partitioning of root into child leaves

Figure 3.5: Construction of data

3.2.2.2.2 Searching

Given a spatial location, the objective of the search algorithm is to determine the particles that contribute at the given point. The search begins with a containment check between the root's bounding box and the provided point. If the containment inquiry returns false then the search is terminated. If the point is within the root's bounding box, then a check is done to determine if the root is a leaf node. If the node is a leaf node, the algorithm iterates through the particles associated with the node and performs a containment query between the point of interest and each particle. If the root has children nodes, then the search point is checked against the root's split plane to determine the side on which it lies and the child node corresponding to that side is tested for existence along with child node one. Each of

these nodes if they exist is processed following the same procedure used on the root of the tree.

Algorithm 12: SearchSupportTree(τ,$\mathbf{y}$)

Input: SupportTree τ rooted at $\mathbf{r}$, Evaluation point $\mathbf{y}$
Output: Set of particles Θ with $\mathbf{y}$ in their support

```
1  if r is a leaf node then
2      for x ∈ r do
3          if y ∈ supp(x) then
4              Insert x in Θ ;
5  else
6      B^L ← Retrieve left child's bounding box ;
7      B^C ← Retrieve center child's bounding box ;
8      B^R ← Retrieve right child's bounding box ;
9      if y ∩ B^L then
10         τ^L ← left child of r ;
11         Θ^L ← SearchSupportTree(τ^L,y) ;
12     if y ∩ B^C then
13         τ^C ← center child of r ;
14         Θ^C ← SearchSupportTree(τ^C,y) ;
15     if y ∩ B^R then
16         τ^R ← right child of r ;
17         Θ^R ← SearchSupportTree(τ^R,y) ;
18     Θ ← Θ^L ∪ Θ^C ∪ Θ^R ;
19 return Θ
```

3.2.2.2.3 Dynamic Insertion

In order for the data structure to be applicable for analysis that employs refinement based on insertion of new particles, i.e. h-refinement, a dynamic insertion algorithm is necessary. To add a particle to an existing tree, the algorithm follows a similar procedure to that of the construction algorithm. The particle must first traverse down the tree using the particle's support bounding box and the current node's split plane to determine which child node to

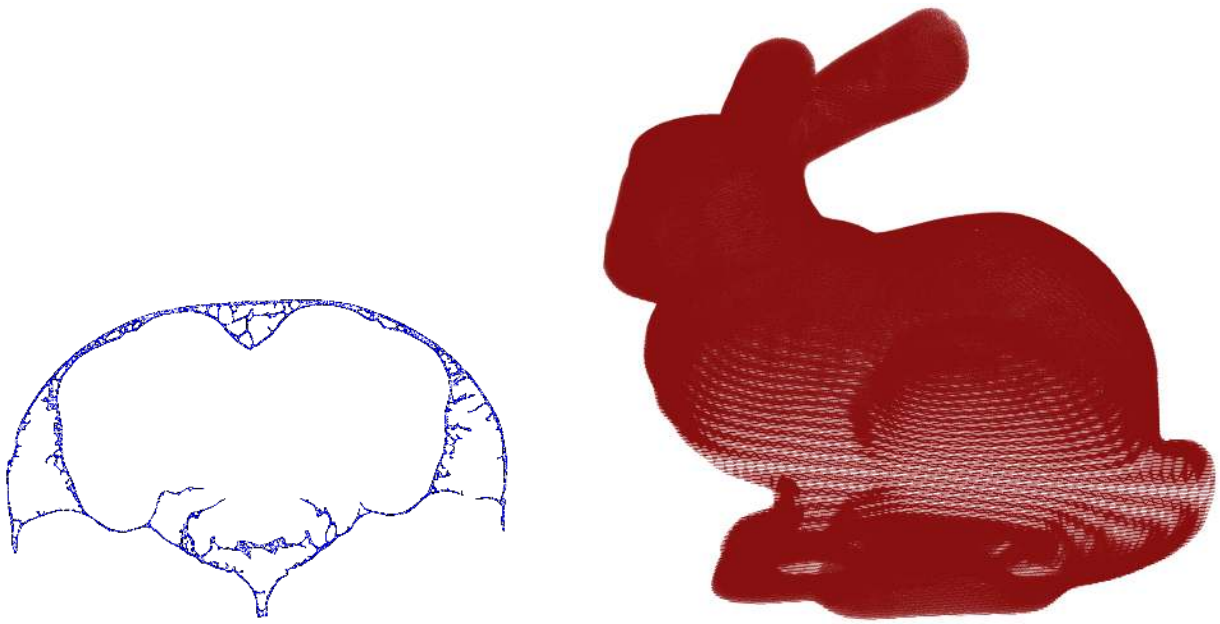
proceed to next. For each interior node traversed, the bounding box must be updated to enclose the support of the particle. Upon reaching a leaf, a reference to the particle is added to the existing list of references and the bounding box for the leaf is updated. The leaf is then processed to test whether a split needs to occur following the procedure described in the construction algorithm. To avoid splitting prematurely, a minimum split requirement can be enforced.

3.3 Performance Comparison

In this section, the performance of the previously discussed data structures will be presented. A complete comparison of all the methods under a wide range of the various user-supplied parameters or algorithm choices is too cumbersome. Rather, similar choices for each of the methods are selected so that a fair comparison can be made. A study of how memory cost, search times, and construction times scale with the number of particles or evaluation points is presented. For any given data set, any of the methods may be significantly improved by optimizing the selection of parameters.

Two meshfree domains are used to measure the construction time, search time, and memory footprint for each method on both structured and unstructured arrangements of particles. All of the data structures were implemented in C++ and run on an Intel Xeon CPU E5-2680 v2 @ 2.80GHz with 64GB of RAM. The unstructured data is two-dimensional and comes from the micro-CT scan of a Scarlet Macaw skull, shown in Figure 3.6a, with discretizations ranging from 31,139 to 157,317 particles. Figure 3.7 shows a close up view of the particles and their distribution. Clearly, this particle distribution is graded and shows large regions with no particles, and small regions with varying particle density. It should

provide a good test for both spatial and object partitioning data structures and their ability to handle a wide variation in particle distribution and spacing. The structured data example is a particle distribution of the Stanford Bunny[2], with discretizations ranging from 196,017 to 3,793,349 particles as shown in Figure 3.6b. It is worth recapping that the approaches



(a) Scarlet Macaw (b) Stanford Bunny

Figure 3.6: Particle distribution for example problems

of Case 1a and Case 1b seek to solve the same problem, namely determining which particles have a non-zero contribution at a given evaluation point i.e., Type 1 connectivity. Case 2 is different, in that it is best suited for determining what evaluation points lie in the support of a given particle i.e., Type 2 connectivity. Recall that all the construction routines required some user defined parameters, which can have significant impacts on the performance of the data structures. In addition to these parameters, the actual implementation of these data structures can have drastic effects on performance. In all of the implementations used, an

[2]Available at https://graphics.stanford.edu/data/3Dscanrep/

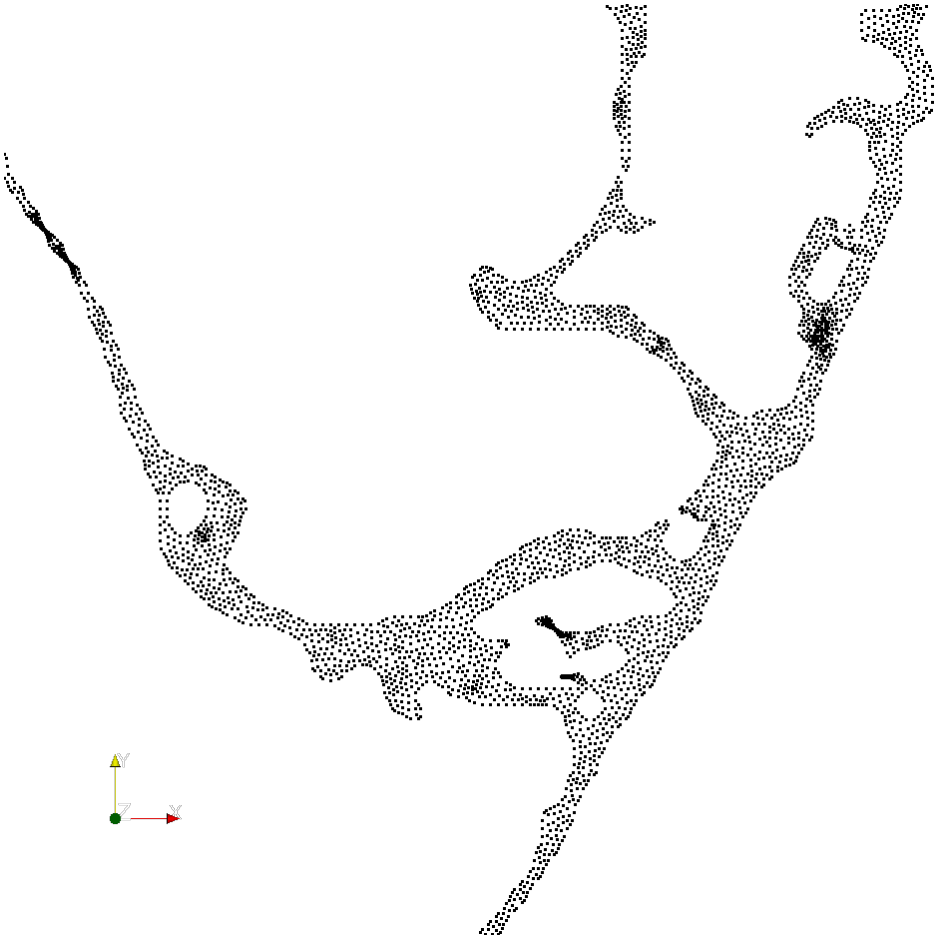


Figure 3.7: Close-up view of the particle distribution of Scarlet Macaw skull

effort to program the best version of each method was made. The grid data structure required a method to determine the grid resolution. The numerical studies conducted determined the resolution with Equation 3.7. The build factor γ, used to determine the grid resolution, was set to one, resulting in the total number of grid cells being approximately equal to the number of particles. While using a larger value for γ would result in a more refined grid and potentially better search times, this would also result in a larger memory footprint. A smaller value of γ could be used to yield slower search times, but the memory costs would be reduced.

The kd-tree requires a splitting rule and termination criteria be defined. The kd-trees in this study used a mid-point splitting rule with the split axis corresponding to the dimension with the greatest deviation of a tree node's bounding box. The termination criteria was satisfied when the number of objects associated within a tree node fell below a user defined threshold or if the volume of the node's bounding box became less than the volume of the largest support size. While it is true that the first termination criteria will eventually be satisfied, the Case 1a kd-tree requires objects that overlap the splitting plane be associated with each child node. This requires numerous references to the same object resulting in a large memory footprint as will be discussed in the next section. The Case 1b and Case 2 data structures do not suffer from this ailment and a bin size of one was used for both kd-tree variants. As with the kd-tree, the BVH requires a splitting rule and termination criteria. The same splitting rule and similar termination conditions used for the kd-tree were employed for the BVH in this study. Differing from the Case 1a kd-tree, the BVH partitions the objects and therefore does not require multiple references to the same object. Therefore, the BVH cannot create tree nodes of a size that is significantly smaller than particles' support sizes, reducing the potential growth in memory. No restriction on the volume of a tree node was necessary. Several bin sizes were tested and from these trials a bin size of one was chosen, as it provided the best performance in regards to searching at a higher memory cost, but this cost was not considered substantial.

Depending on the splitting rule used, the proposed data structure may not need a termination criteria. By using a mid-point splitting rule, the termination criteria is defined once all the objects overlap the splitting plane. This removes an element of user interaction and the potential for a poorly chosen value that could result in an ill-performing data structure.

However, the use of a different splitting rule may require a different termination criteria. The implementation used in this work employed a mid-point splitting rule and therefore did not require a user defined termination criteria.

3.3.1 Memory Cost

The memory usage for those data structures constructed based on Case 2 are independent of the number of particles, as they are constructed of evaluation points. The memory usage for the tree based data structures is dependent on the termination criteria; for the BVH, Kd-tree, and Support tree, the splitting method is also a factor. The memory for the Case 1a and Case 1b data structures are plotted on the same graph for each example in Figures 3.8a and 3.8b, respectively. From the figures, it is clear that among the Case 1a implementations, the grid, depicted by the green line, exhibits the worst memory usage. However, for the Case 1b approaches, the grid appears to have the lowest memory footprint, which is due to the fact that the particle support size is not taken into account. Considering only the Case 1a data structures, the proposed Support Tree data structure shown here by the grey line, has the lowest memory impact for the second example, as seen in Figure 3.8b. It is nearly identical to the kd-tree and octree for the first example with all three contending for the lowest memory impact. The grid data structure has the lowest memory footprint for Case 2 searches as shown in Figures 3.9a and 3.9b.

3.3.2 Construction Cost

Here, the time required to construct the various data structures is analyzed. A direct comparison of the two cases is not feasible, as the data structures associated with Case 1a

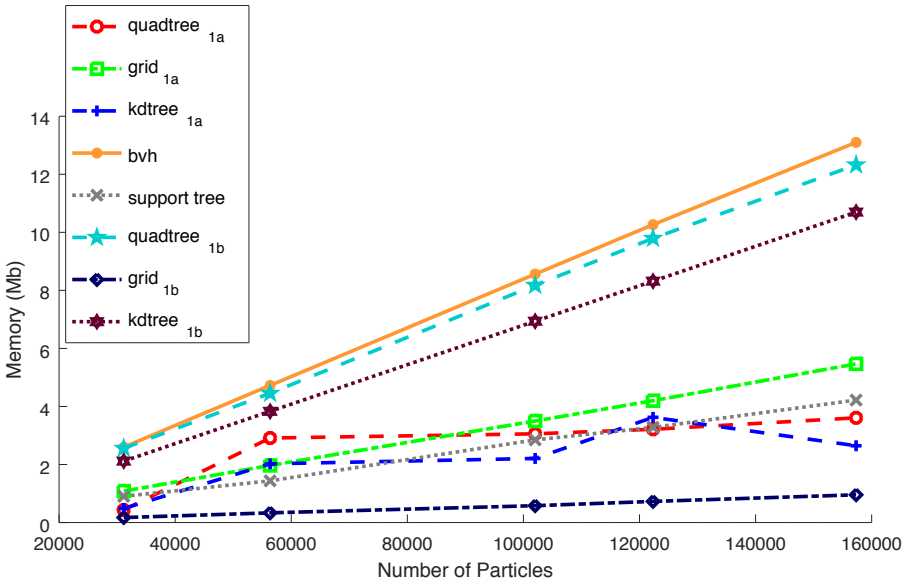
and Case 1b are constructed on the set of particles, where as those data structures associated with Case 2 are constructed on the set of evaluation points. The construction times for Case 1b approaches are significantly lower than Case 1a approaches, since accounting for particle support size is not necessary. For a mechanics problem using a meshfree approximation method, the construction routines will generally occur once at the beginning of the simulation and could be conducted as a pre-processing step, obviating the time from the main processing step entirely. However, if a refinement method is being used or evolving topologies are of interest, the ability to quickly reconstruct the data structures could become important. The time required to construct the data structures is represented graphically in Figures 3.10 and 3.11.

3.3.3 Search Cost

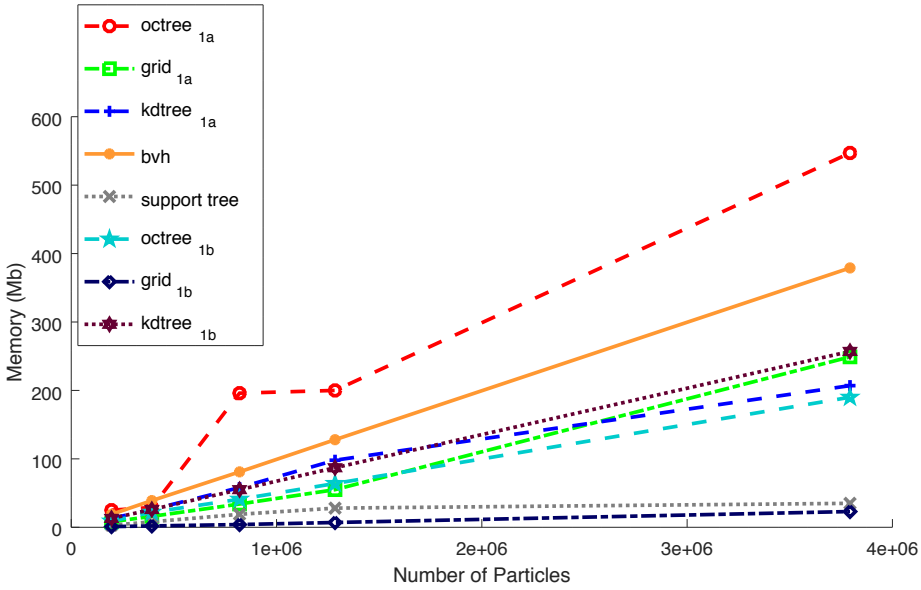
In this section, the time it takes to search the various structures for computing the connectivities used in meshfree methods is reported. Note that construction costs are one-time costs for a given problem, whereas the search times are accumulated over many calls. In general, many searches will be performed over the course of a computation. In some cases, for example constructing a stiffness matrix, the number of searches is known *a priori*. In other cases, for example contact, the number of searches is not known. Figures 3.12-3.15 show the time required to search the data structures for the given number of particles and evaluation points. For both examples, the most efficient data structure is the regular grid constructed on the particles accounting for their support size.

3.3.4 Guide to Choosing a Search Structure

As previously mentioned, one can fine tune any of these methods to improve search times or reduce memory cost, so it is not possible to recommend a single 'best' data structure. As demonstrated, when determining Type 1 adjacency, in general, a grid structure or the support-tree structure will be ideal. The grid, in one of the Case 1 scenarios, will substantially sacrifice memory for speed, or speed for memory. While it is not usually optimal in either speed or memory, the Support Tree does perform well in regards to the two examples presented. If a single structure is needed to perform well in most situations without user interaction, the support tree may be the best choice. Furthermore, the Support Tree scales well with the number of particles stored and with the number of evaluation points searched. If either speed or memory are paramount, a properly tuned grid will likely be the best choice. From the numerical results, it is clear that when tasked with determining Type 2 adjacency, the Case 2 grid is the best choice.

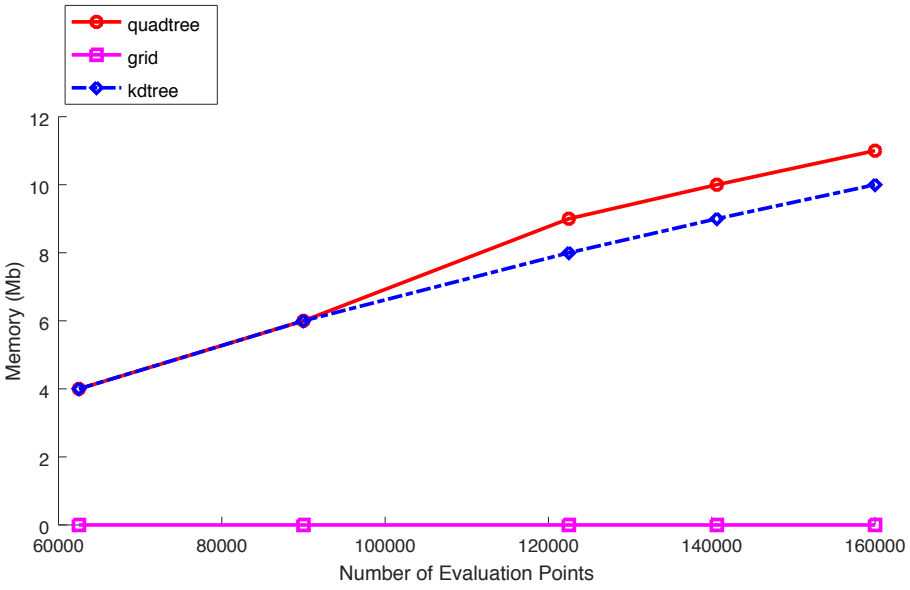


(a) Memory vs number of particles for Scarlet Macaw.

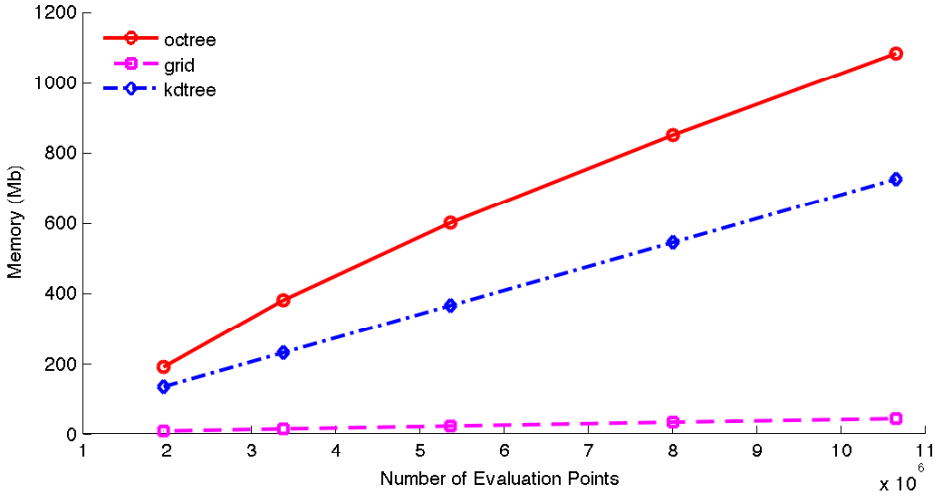


(b) Memory vs number of particles for Stanford Bunny.

Figure 3.8: Memory requirements corresponding to Case 1 data structures.

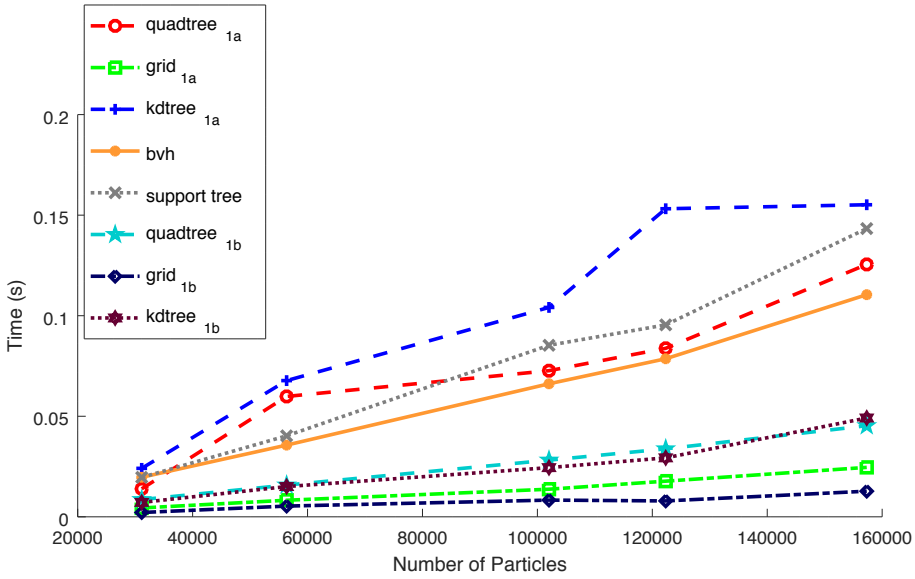


(a) Memory vs number of evaluation points for Scarlet Macaw.

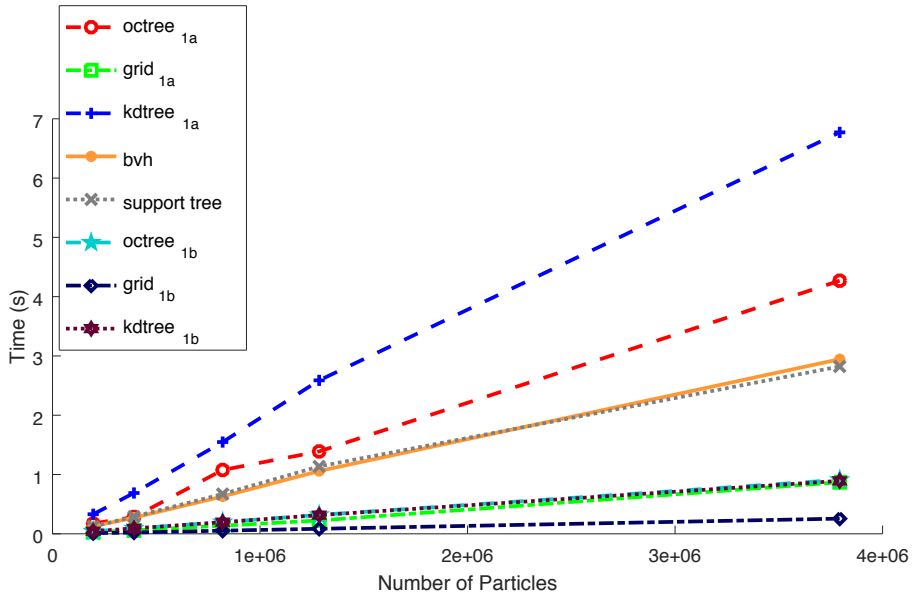


(b) Memory vs number of evaluation points for Stanford Bunny.

Figure 3.9: Memory requirements corresponding to Case 2 data structures for Ex. 1 and 2

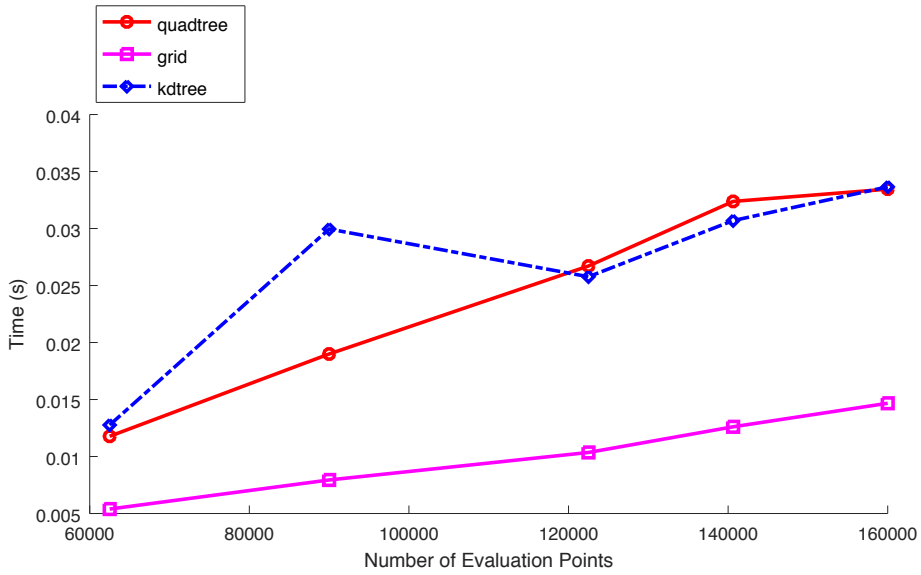


(a) Construction time vs number of particles for Scarlet Macaw.

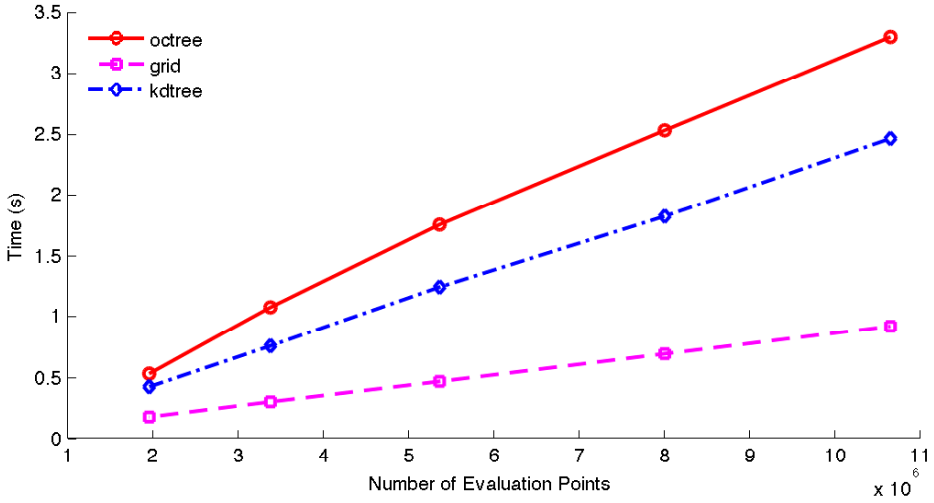


(b) Construction time vs number of particles for Stanford Bunny.

Figure 3.10: Construction time corresponding to Case 1 data structures.

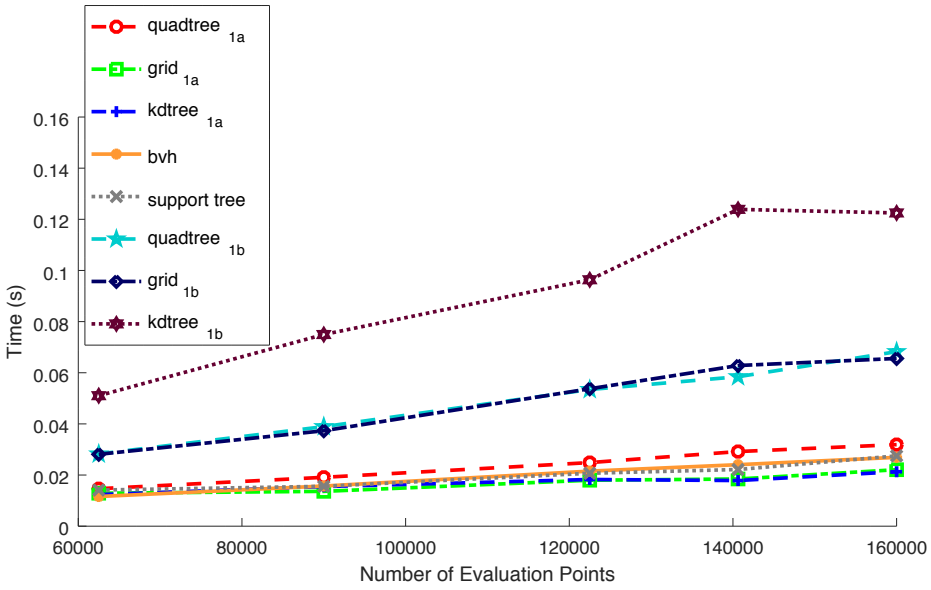


(a) Construction time vs number of evaluation points for Scarlet Macaw.

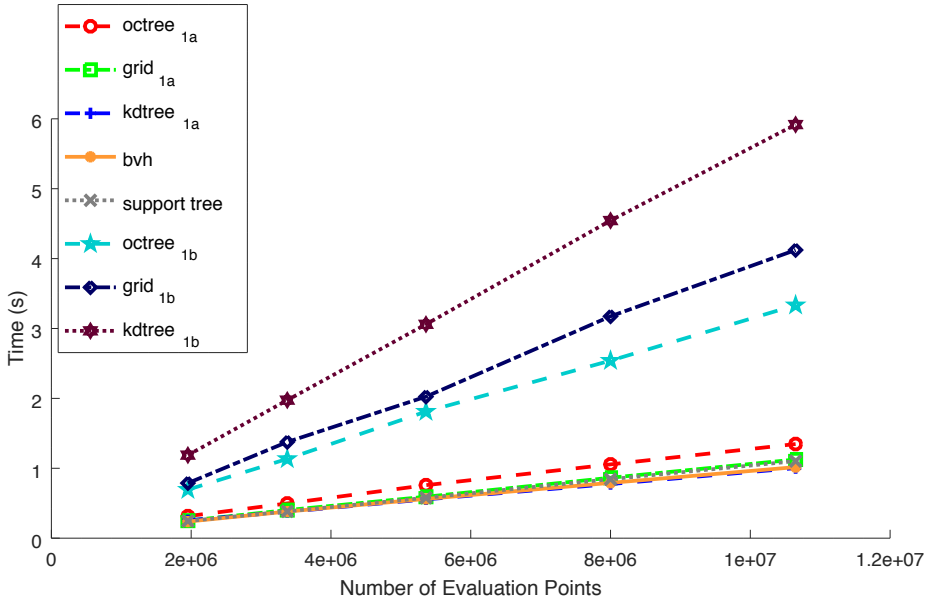


(b) Construction time vs number of evaluation points for Stanford Bunny.

Figure 3.11: Construction time corresponding to Case 2 data structures.

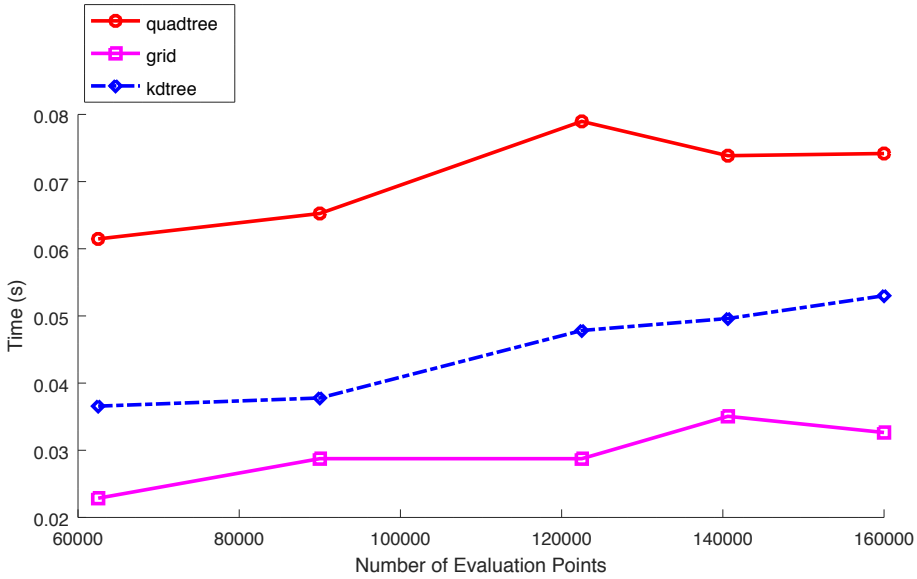


(a) Scarlet Macaw

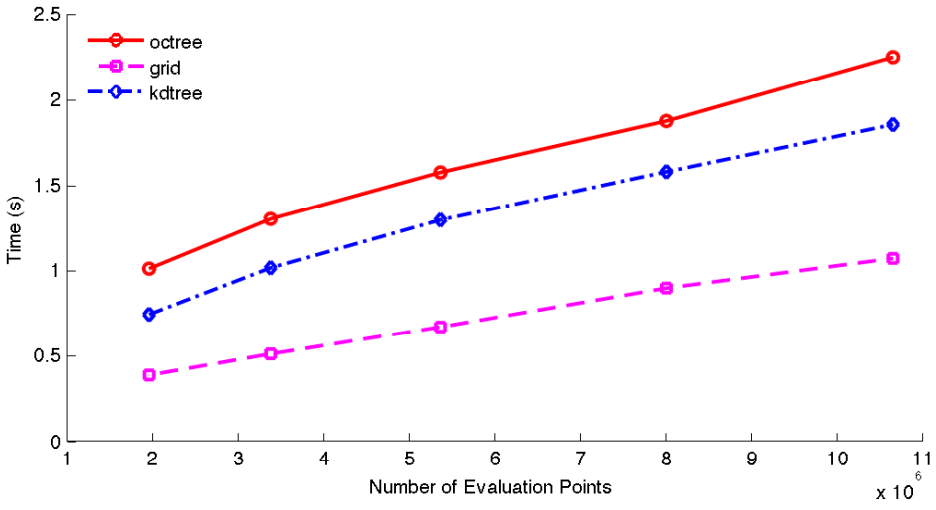


(b) Stanford Bunny

Figure 3.12: Search time vs number of evaluation points for Case 1 data structures for the Scarlet Macaw and Stanford Bunny discretized with 122,334 and 1,284,920 particles respectively.

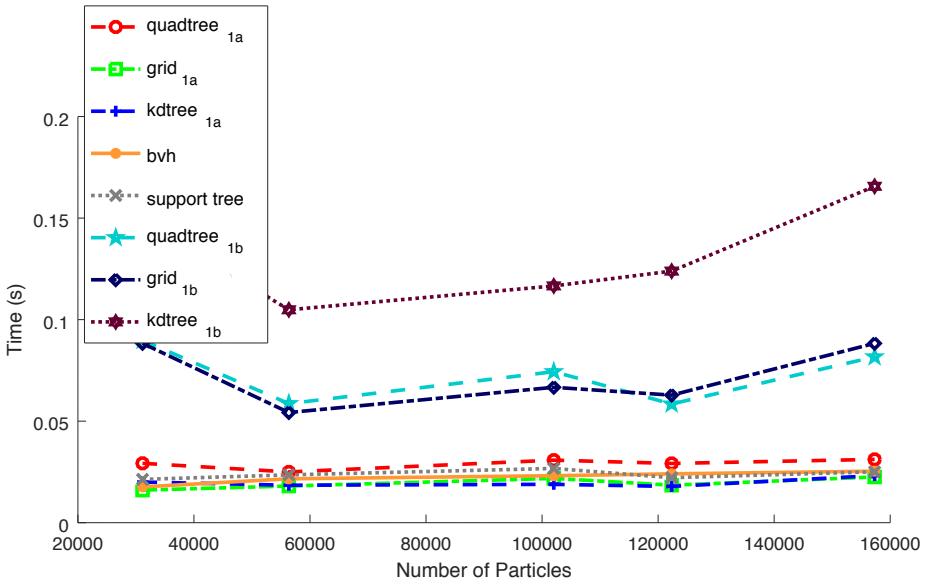


(a) Scarlet Macaw

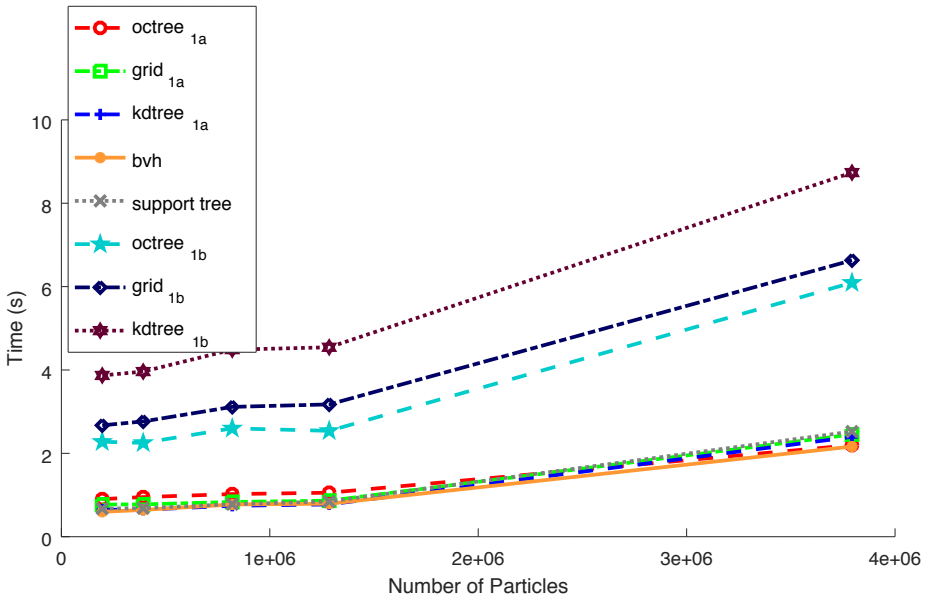


(b) Stanford Bunny

Figure 3.13: Search time vs number of evaluation points for Case 2 structures the Scarlet Macaw and Stanford Bunny discretized with 122,334 and 1,284,920 particles respectively.

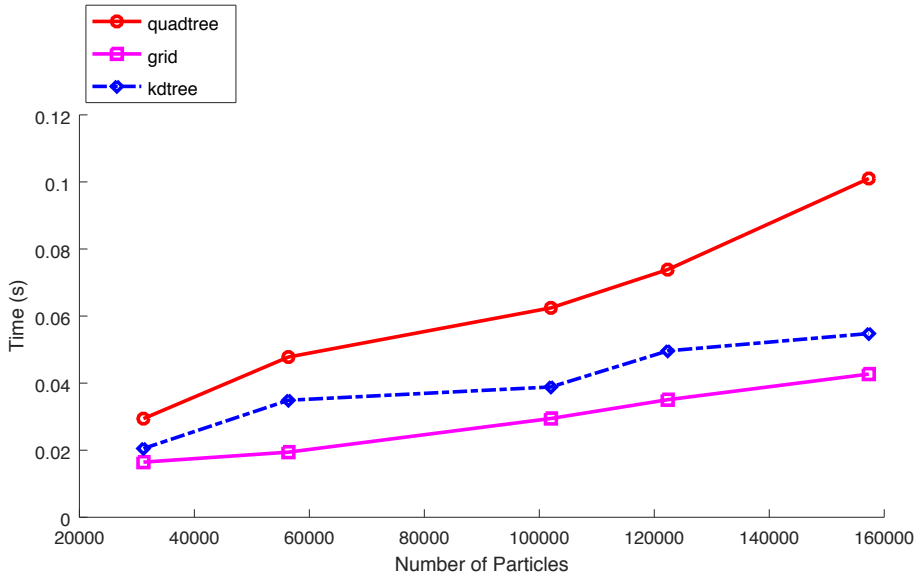


(a) Scarlet Macaw

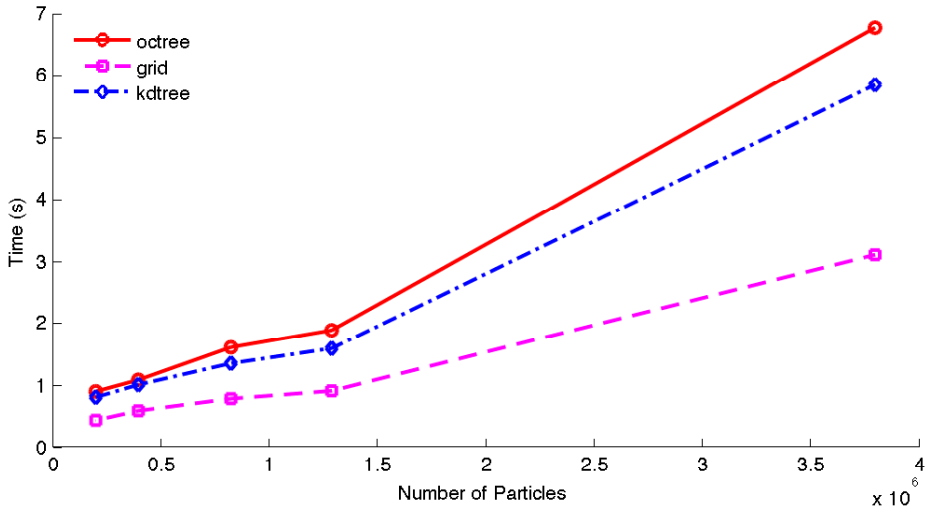


(b) Stanford Bunny

Figure 3.14: Search time vs number of particles for Case 1 structures for both examples using 140,625 evaluation points for the Scarlet Macaw and 8,000,000 evaluation points for the Stanford Bunny.



(a) Scarlet Macaw



(b) Stanford Bunny

Figure 3.15: Search time vs number of particles for Case 2 structures for both examples using 140,625 evaluation points for the Scarlet Macaw and 8,000,000 evaluation points for the Stanford Bunny.

CHAPTER 4: DOMAIN DISCRETIZATION FOR PARTICLE METHODS

Meshfree methods and the formulation of the Reproducing Kernel Particle Method have been introduced. While meshfree methods do not require the computation of a mesh, they do require the domain be discretized by particles. The transformation of the continuous domain into a discrete set of particles is analogous with the mesh generation step of FEM.

Triangularization or tetrahedralization of the domain is a common approach for determining the spatial positions of particles, support domains, and nodal volumes [43]. The concept of using a mesh to determine the meshfree discretization may seem to violate one of the key benefits of using meshfree methods. To some degree this is true, but aspects that plague finite elements, such as the low order differentiability, can be overcome with meshfree methods with the particle discretization constructed from a low order finite element mesh.

This chapter will introduce the methodology used to construct a meshfree discretization of the domain in such a way that the quality of the meshfree approximation is not compromised. Approaches for determining the attributes associated with a particle, such as the nodal volume and support domain, will be discussed.

4.1 Particle Placement

The concept of placing nodes in a domain is not limited to meshfree methods and may also be necessary for tessellation. The nodes can be considered the lowest level geometric

entity, the construction of higher level entities, such as edges and faces, is impossible without knowledge of the nodes. Similarly the particles used in meshfree methods cannot be constructed without knowledge of their spatial positions.

Determining the optimal point distribution is the fundamental problem for spatial discretization and many approaches have been considered. This work will ultimately employ a physics-based approach for point distribution, and shares similarities with several existing techniques that are used for meshing. One approach for distributing points in some finite domain is to model the geometric problem as a particle simulation, in which the motion of the particles is dictated by physical laws or some stochastic set of rules. This approach was originally developed for automating triangular mesh generation and is commonly referred to as Bubble Meshing [57]. In [68], an approach for node placement by molecular dynamics is proposed. This approach is similar to that of Shimada *et al.*, with the main difference being in the form of the interaction between particles. Another approach by Persson *et al.* models the discretization as a structural analysis problem, in which the mesh is modeled as a truss, with the truss joints being the vertices of the mesh [49]. The vertices are moved to what is considered the optimal location by an incremental procedure, which involves an edge determination process followed by force equilibrium. The system is then updated by the displacements and the process is repeated. A similar approach to that of [57] is given in [67], but instead of modeling the motion of the particles using Newtonian mechanics, the authors minimize the potential energy of the system using a Monte Carlo simulation.

While the aforementioned approaches were originally conceived for mesh generation, the actual tessellation is performed as a post processing step using methods such as those proposed by [12, 37, 64], which operate on an existing point distribution to generate a triangu-

lation. Meshfree discretization techniques can be adapted from those used for meshing by omitting the tessellation step. Alternatively, several methods have been developed that focus directly on meshfree discretization. An octree based approach is given in [34]. The Biting Method [42], originally developed for mesh generation, is adapted for generating meshfree particle distributions [66].

4.1.1 Physics Based Approach

The physics-based approach to point distribution models a collection of points as a set of interacting particles. Differing from the advancing front and Delaunay insertion algorithms, which insert points in the domain incrementally, the physics-based approach determines an initial placement for all the nodes first and then optimizes their positions. A compromise between computational effort used in the initial placement of the nodes and the relocation of the nodes has to be made. The closer the initial configuration is to the optimal configuration, the less time spent in the simulation step. However, if obtaining the initial configuration is computationally expensive in and of itself, then little is gained by the physics-based approach.

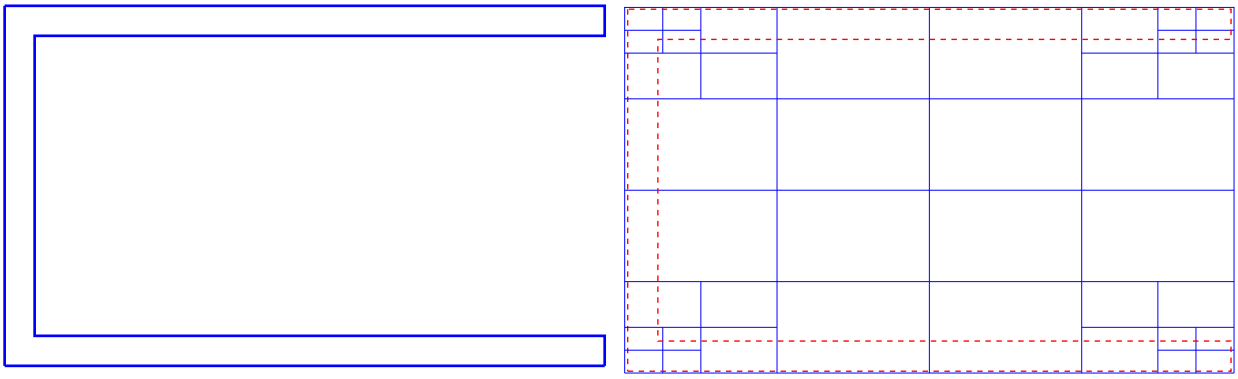
4.1.1.1 Initial Particle Distribution

An approach to generating the initial point distribution is through spatial partitioning data structures. Alternatively, uniform discretization of the bounding box and rejection sampling can be used to generate initial node distribution. In order to generate an analysis-suitable particle distribution, a description of the geometry is needed. For this work, it will be assumed this description is in the form of a PSLG (2D) or PLC (3D). Given this collection of primitives denoting the boundary, an initial guess at the particle configuration

is needed. There are several attributes of the initial configuration needed to ensure suitable quality of the resulting particle distribution after the simulation procedure. First, the initial set of particles should cover the entire region. This is especially important for regions of the domain with finer details. If these regions of finer detail are not sufficiently sampled, the simulation may take an unreasonable amount of time to move particles there, or may not place particles in these areas at all. Second, the initial particle configuration should only locate points within the boundary to be discretized. This eliminates the need to incorporate a removal step in the actual simulation. Third, the spacing field should be determined such that no regions are vacant of particles.

The first two attributes can be achieved by employing a quad-tree in 2D or octree in 3D for the initial point placement. Alternative to the versions discussed in §3.2.1.3, the version used here is not for accelerating search queries. Instead it is used as a type of pseudo-mesh, identifying sharp features which require a more refined discretization and preventing unnecessarily fine discretizations in regions where sharp features are not present. Given an octree, the leaf nodes are used to generate the particles. The manner in which the particles are generated in each leaf node can vary, but a simple approach is to locate a single particle at the leaf node's center.

Recall from the discussion in §3.2.1.3, the octree requires a termination condition. The versions of the octree used for adjacency queries utilized heuristics based on the tree node size, tree depth, or bin size. Here these stopping conditions are inadequate, as small features may not be resolved. Consider the boundary given in Figure 4.1a, with eight vertices and edges. If a bin-size termination criterion based on the vertices of the boundary is used, the resulting quad-tree leaves contain large amounts of void space as seen in Figure 4.1b.



(a) Boundary (b) Quadtree using bin size termination condition

Figure 4.1: Two-dimensional domain to be discretized

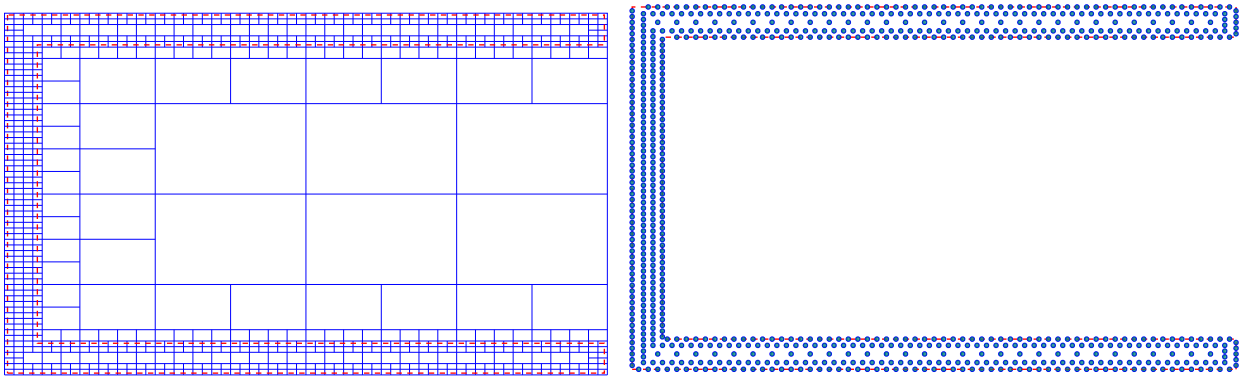
Using the center of these leaf nodes as the particle locations results in an inadequate particle distribution. Therefore, a new termination criterion is needed to allow finer details of the domain to be resolved, which given in Algorithm 13. The algorithm takes in a single tree node τ and the region to be discretized Γ. The algorithm returns a boolean that is true if the tree node should be subdivided and false if the tree node should be marked as a leaf. The first conditional checks if the tree node is the root of the tree, and if so, returns true. This check is necessary, as the bounding-box of the root will encompass the domain resulting in no intersection between the edges of the root's bounding box and the domain. If the tree node is not the root node, the algorithm proceeds to the second conditional. Here an intersection between each edge of the quadrilateral (2D) or hexahedron (3D) representing the tree node and the boundary is performed. If an intersection does occur, the tree node is to be subdivided with the algorithm returning true. If no intersection occurs, the tree node is either completely inside the domain and will be used in the generation of particles or the tree node is completely outside the domain and will not be used.

Algorithm 13: TerminationCondition(τ,**S**)

Input: Tree node τ and region Γ
Output: Boolean stating if τ should be subdivided

```
if τ is root then
    return true ;
if Intersection(τ,Γ) then
    return true ;
return false ;
```

Figure 4.2a shows the quadtree constructed on the boundary (red dashed line) in Figure 4.1a using Algorithm 13.



(a) Quadtree termination condition in Algorithm 13 (b) Initial point distribution from quadtree

Figure 4.2: Quadtree discretization of boundary and initial point distribution

4.1.1.2 Particle Motion

After the initial node distribution is determined, the next step is to move the nodes to their optimal locations. The optimal location can be considered as one where the particles are well-distributed and cover the entire domain. In the physics-based approach, this corresponds to a state of mechanical equilibrium, which is achieved when the resultant force on each particle is zero. For a conservative system, equilibrium can be classified as a state where the potential energy is minimum. Therefore, the optimal locations are given by the following

constrained optimization problem:

$$\arg\min_{\mathbf{x}\in\Omega} U\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\right). \tag{4.1}$$

The assumption is made that the region to be discretized is known. If the region is the result of a CAD system, the boundary defining the limits of the domain is generally given as a piecewise linear complex or nurbs surface.

The force-based approach for seeking equilibrium is the fundamental concept behind bubble meshing. Pairwise forces are generally used and are constructed to be repulsive when two particles are closer than some equilibrium spacing and attractive when the distance exceeds the equilibrium spacing. In molecular dynamics these forces are derived from an energy potential. A popular potential is the (12-6) Lennard-Jones [38] given in Equation 4.2 with graphical depiction in Figure 4.3.

$$V(r) = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) \tag{4.2}$$

The parameters σ and ε in Equation 4.2 are user-defined parameters and depicted in Figure 4.3. The separation distance is computed as the Euclidean norm of the vector from particle I to particle J,

$$\mathbf{r}_{IJ} = \mathbf{x}_J - \mathbf{x}_I. \tag{4.3}$$

The total energy of a system composed of N particles can be computed as

$$U = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} V\left(r_{ij}\right). \tag{4.4}$$

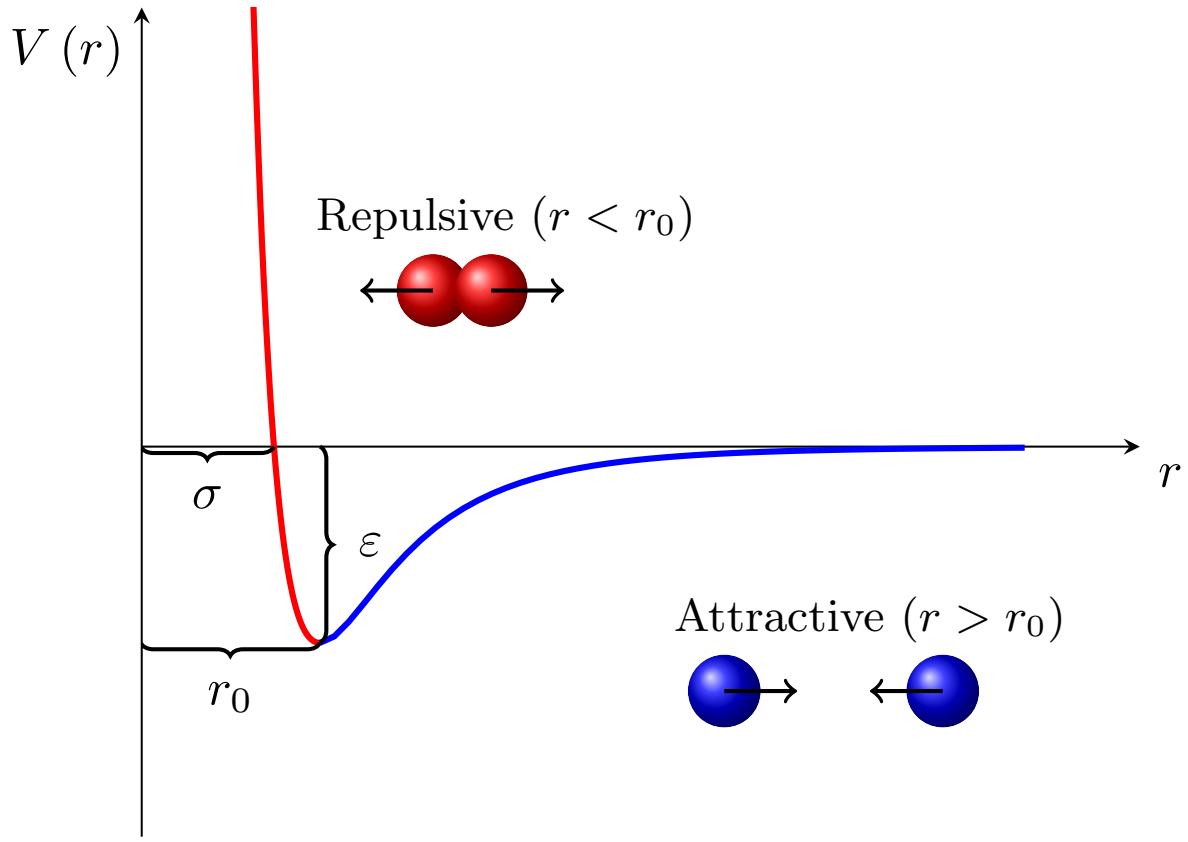


Figure 4.3: Lennard-Jones 12-6 potential schematic

From the conservation of energy and the work-energy theorem, the force on particle I due to particle J can be determined as the negative gradient of the potential:

$$\mathbf{F}_{IJ} = -\boldsymbol{\nabla} V_{IJ} = -24\varepsilon \left(2 \left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right) \frac{\mathbf{r}_{IJ}}{r}. \tag{4.5}$$

Simple in theory, this approach proves challenging in practice. Since mechanical equilibrium is not a state of zero motion, the particles can still have velocity, and determining the equilibrium configuration numerically is challenging. Particles might move past equilibrium without taking on the position such that force equilibrium is satisfied; the particles oscillate about their equilibrium position at some frequency. To mitigate the challenge of oscillatory

behavior, a velocity-dependent drag force is added, which dissipates energy from the system, allowing the particle to settle into an equilibrium state.

The force-based approach described above resembles that of a gradient descent type optimization method. In gradient descent optimization, the next probable solution is found by moving along the negative gradient of the potential, which is the definition of the force. Both of these approaches can be classified as deterministic approaches to the minimization problem. During simulation, these deterministic approaches move all particles in a system at once, which makes it challenging to keep particles inside the boundary. In addition, it can be difficult to determine suitable parameters such that a stable calculation is ensured. An alternative to the deterministic approach for locating an equilibrium configuration is to minimize the potential energy of the system using the concept of stochastic optimization. Stochastic optimization refers to a collection of numerical minimization techniques that include some form of randomness. The manifestation of the randomness varies, but in general refers to randomness in the objective function, search direction, or in both. Numerous stochastic optimization techniques have been proposed, but this work will utilize the method referred to as Simulated Annealing.

4.1.1.2.1 Simulated Annealing

Annealing is a metallurgical process where a material is heated above its melting point followed by a gradual cooling, allowing the atomic structure to rearrange itself into a crystalline structure. In simulated annealing, this crystalline structure is analogous to the minimum of some cost function. Simulated annealing is a variant of the Metropolis Monte Carlo method (MMC), where a cooling schedule is added to control the temperature. Simulated

annealing is a descent algorithm where lower energy states are preferred. Differing from deterministic methods, such as steepest descent, simulated annealing uses random moves to explore the solution space. A change that results in a lower energy is always accepted. This alone could easily result in the algorithm becoming trapped at some local minima. To escape these local minima, higher energy states are accepted based on some acceptance probability. From statistical thermodynamics, the probability of a system being in state A with energy U_A at temperature T is given by the Boltzmann distribution:

$$P_A = \frac{1}{Z} e^{\frac{-U_A}{kT}}, \tag{4.6}$$

with k being the Boltzmann constant and Z being the canonical partition function defined as

$$Z = \sum_I e^{\frac{-U_I}{kT}}. \tag{4.7}$$

Here the sum is taken over all the available states a system could occupy. To see how this is useful in the simulated annealing algorithm, consider a particle at some location $\mathbf{x}_A$ with energy U_A. Now consider the same particle system except the particle previously at $\mathbf{x}_A$ is now at a different location $\mathbf{x}_B$ with energy U_B. The general idea is that configurations at lower energy states are always accepted and transitions to higher energy states are conditionally accepted dependent on the probability of occurrence. These conditions are given as:

$$P_{A\to B} = \begin{cases} \text{true} & \text{if } P_B \geq P_A, \\ \text{maybe} & \text{otherwise.} \end{cases} \tag{4.8}$$

This can be recast as a ratio of the probabilities of state A and B given as:

$$P_{A\to B} = \frac{P_B}{P_A} = e^{\frac{(U_A - U_B)}{kT}}. \tag{4.9}$$

If the ratio in Equation 4.9 is greater than one, then state B is always preferred. If the probability of state B is lower than the probability of state A, the transition is conditionally accepted. For a state of lower energy A to transition to a state of higher energy B, the following condition needs to be satisfied:

$$P_{A\to B} > R, \qquad R \in (0, 1)\,, \tag{4.10}$$

where R is a random number. The probability is dependent not only on the change in energy between two states, but also the temperature. A high temperature results in a greater likelihood of accepting transitions from lower states of energy to higher ones. The starting value for the temperature should be chosen such that each particle has a probability of moving. As the simulation proceeds, the temperature should gradually decrease. The way in which the temperature is controlled is referred to as an annealing schedule. The simulated annealing algorithm is given in Algorithm 14. The simulated annealing algorithm provides a straightforward approach for determining particle locations but does result in a computationally expensive approach. This expense comes from the evaluation of the objective function at the current state and perturbed state as shown in Lines 1 and 3 of Algorithm 14. A change in state involves moving an individual particle. The objective function evaluation at any given particle requires the summation of all the interaction potentials between the given particle

Algorithm 14: SimulatedAnnealing($\mathbb{A}$)

Input: Initial configuration $\mathbb{A}$
Output: Final configuration $\mathbb{B}$

$U_A \leftarrow$ Evaluate objective function of current state;
$\mathbb{B} \leftarrow$ Apply random perturbation to current state ;
$U_B \leftarrow$ Evaluate objective function of candidate state ;
if $U_A < U_B$ **then**
    $\mathbb{A} \leftarrow \mathbb{B}$;
else if $P(U_A, U_B, T) > random[0, 1)$ **then**
    $\mathbb{A} \leftarrow \mathbb{B}$;
Reduce temperature T ;
return $\mathbb{B}$;

and all other particles in the domain. The particle is then perturbed and the evaluation of the objective function performed, again requiring the interaction between the perturbed particle and all other particles in the domain to be computed. For a given particle I in a domain with N particles, the change in potential from states A and B is computed as:

$$\Delta U_{AB}^{I} = \sum_{i=1, i \neq I}^{N} V\left(\bar{r}_{Ii}\right) - \sum_{i=1, i \neq I}^{N} V\left(r_{Ii}\right). \tag{4.11}$$

The first summation evaluates the potential with respect to the modified location of particle I, therefore the linearity of the summation operator cannot be exploited. This $2N$ operation is performed for each particle at every step of the annealing simulation, resulting in a potentially expensive operation. To reduce this cost, several optimizations can be performed. The weak attractive potential generated by particles at large distances from one another can be exploited. This can be done by only evaluating the contributions to the potential energy at a given particle at nearby particles since the contribution from distant particles can be assumed to be negligible. This requires the determination of all the particles within some

specified radius of a given point. This type of search is easily handled by any of the spatial partitioning data structures discussed in §3.2.1. Those data structures were assumed to be static with regard to particle motion, and for the simulated annealing, the particles need to be moved. While each one of the structures can be used to handle dynamically evolving particle positions, the grid data structure will be used here. The construction of the grid follows the same procedure described in §3.2.1.1 with the particles treated as points with zero extents. The evaluation of the potential energy at a given particle can then be computed by searching the grid given the spatial location of the particle and cutoff radius. The result of the search will be the set of particles to be used for evaluating the potential energy. This procedure is outlined in Algorithm 15.

Algorithm 15: EvaluatePotentialGrid($\mathbf{C}$,P,r_c)

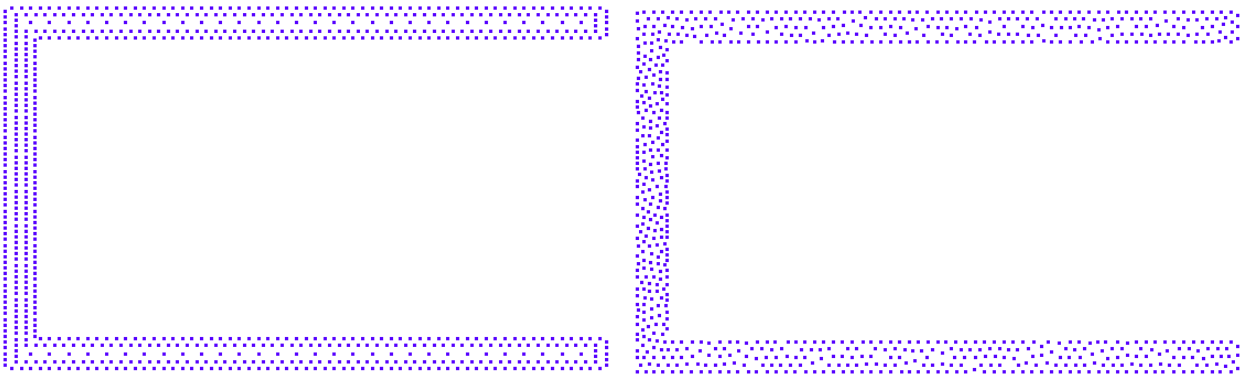
Input: Grid C, Particle P, Cutoff radius r_c
Output: Potential energy V

```
1 I ← Compute indices into C of grid cells intersecting P with search radius r_c ;
2 V ← 0 ;
3 for i ∈ I do
4 |   for x ∈ C[i] do
5 |   |   V += EvaluatePotential(P,x) ;
6 return V ;
```

Figure 4.4 illustrates the simulated annealing process applied to the bracket from Figure 4.1a.

4.2 Computation of Particle Attributes

Provided the spatial locations of the meshfree nodes, the next step is to determine two attributes associated with each node, the nodal volume and support domain.



(a) Initial distribution (b) Final point distribution quadtree

Figure 4.4: Initial and final point distributions resulting from simulated annealing process

4.2.1 Particle Volumes

In general the nodal weights should be chosen such that the following is true:

$$\sum_I \Delta V_I = meas(\Omega). \tag{4.12}$$

Given a uniform discretization, the computation of the particle volumes such that Equation 4.12 holds is straightforward. Consider a 1D domain $[0, L]$ that is uniformly discretized by N_p particles. The spacing between particles is determined by,

$$\Delta x = \frac{L}{(N_p - 1)}. \tag{4.13}$$

The associated volume for each interior particle is Δx with the two boundary particles having a nodal volume of $\frac{1}{2}\Delta x$. This can be easily expanded to higher dimensions, but is only applicable for uniform discretizations of rectilinear domains. A study on the effects of particle volumes on the approximation functions is carried out in [1] with particle volumes

set to unity, random values, and exact values. Here the authors find that the consistency of the approximation is obtained for all three cases. The correction function is guaranteed to enforce the reproducing requirements and any error in the nodal volumes will be absorbed by the correction function coefficients. However, the use of random values is discouraged as the accuracy of the numerical method does show deterioration. The computation of nodal volumes is also addressed in [24], where the authors note the nodal volume appears in the integration of the kernel as well as in the integration of the moment matrix. Since the moment matrix is later inverted, any constant value for the integration weight will result in the same approximation functions. If the integration weight is set to one, the Moving Least Squares (MLS) approximation functions are recovered [46].

4.2.2 Support Domain

The support domain of a particle dictates the region of the domain where that particle contributes. The support domain is centered at a particle and generally has symmetries with respect to each direction. The shape of the support is generally taken to be a simple geometric object such as a sphere or rectilinear region, however this is not a requirement. The concept of a support domain is not exclusive to meshfree methods. In FEM, each node also has a support and is the union of the elements that the node belongs to. The support of a node in FEM is generally an n-sided polygon (2D) or an n-faceted polyhedron (3D).

4.2.2.1 Geometrical Representations

Recall Definition 2.3.1 for the support domain,

$$\Omega_I^\rho = \{\mathbf{y} | \|\mathbf{y} - \mathbf{x}\| \leq \rho\}. \tag{4.14}$$

The definition for the norm $\|\cdot\|$ and ρ are dependent on the geometrical configuration. For spherical support domains the Euclidean norm is used, with the Euclidean norm defined as:

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}. \tag{4.15}$$

If the support is rectilinear, then the infinity norm is used, with the infinity norm defined as:

$$\|\mathbf{x}\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}. \tag{4.16}$$

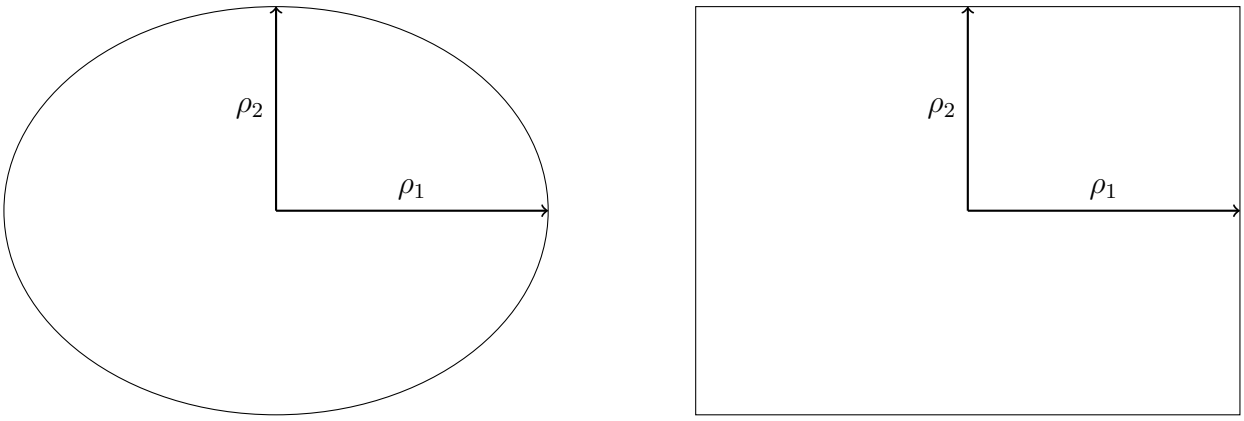
A modified version of Equation 4.14 that allows for the characteristic length, ρ, to vary in each direction is given as:

$$\Omega_I^\rho = \{\mathbf{r} | \|\mathbf{r}\| \leq 1\}. \tag{4.17}$$

Here $\mathbf{r}$ is defined as:

$$r_i = \frac{y_i - x_i}{\rho_i} \quad i = 1, \ldots, d. \tag{4.18}$$

This allows for the extension of the definition of the support domain to include ellipsoidal domains when the Euclidean norm is used and ρ is a vector containing the lengths along the principal minor and major axes as shown in Figure 4.5a. Cartesian support domains



(a) Ellipse support domain (b) Rectilinear support domain

Figure 4.5: Characteristic length description for ellipsoidal and rectilinear support domains

with varying lengths in each dimension can be obtained by employing the infinity norm and modifying ρ as shown in Figure 4.5b.

4.2.2.2 Construction

Once a geometrical representation for the support domain is chosen, the characteristic lengths for each particle's support needs to be determined such that the requirements for an admissible particle distribution are met. These requirements were introduced in §2.3.1.

Several approaches for determining the support size have been developed. An approach that relies on the triangulation of the domain is presented in [43], where each particle is taken as a vertex of a mesh. The support domain of a particle is then based on a heuristic using the average area of those elements connected to the vertex. In an alternative approach, particles are not directly associated with a support domain, but the integration points are. Here the selection of the nodes contributing at a point are determined by what are referred to as T-Schemes [43]. Again this technique relies on the tessellation of the domain. In [29, 31], an approach to determine the support size is proposed where additional points

are introduced to ensure the union of the supports completely covers the domain. These points are acquired from a coarse mesh and only serve to ensure the coverage requirement is met; they are not introduced as additional degrees of freedom and have no associated support domain themselves. In [20], a probabilistic algorithm for determining the size of the support domains is proposed. This approach is similar to [29, 31], but instead of using the vertices of a mesh, the domain is randomly sampled. Both of these approaches seek to find the characteristic lengths for each particle's support domain, with the difference being how additional points are introduced. The introduction of these additional points is warranted by the requirements for an admissible particle to hold for any point in the domain, not only at the particles themselves. This work will utilize a similar approach with the general procedure given in Algorithm 16. The inputs are the set of particles used to discretize the domain and an additional set of sample points.

The consequences of increasing the support sizes have not been discussed. If the support of a particle were to encompass the entire domain, the first three requirements for an admissible particle distribution are trivially satisfied. Assuming the particles are distributed in a manner such that the fourth requirement is satisfied, the requirements for a suitable approximation have been met. However, over sizing the support domains has ramifications. The sparsity pattern of the stiffness matrix is a direct consequence of the particles' support sizes. As the supports are expanded, the number of particles interacting increases. The increased number of interactions causes an increase in memory, as the number of non-zero entries in the stiffness matrix is increased. This also has a negative impact on the computational time for assembling and solving the sparse linear system. Excessively large supports result in the loss of the local character of the approximation, as will be demonstrated next.

Algorithm 16: DetermineSupportSizes($\mathbf{X}$,$\mathbf{Y}$)

Input: Lower bound on number of particles contributing N_L, Particles $\mathbf{X}$, sample points $\mathbf{Y}$

Output: Support domain of each particle in $\mathbf{X}$

```
1  X ← initialize support domains to zero size ;
2  Z ← X ∪ Y ;
3  τ ← Partition(X) ;
4  r ← initialize search radius ;
5  neighbors ← initialize empty neighbor list ;
6  for z ∈ Z do
7      neighbors ← Search(τ,z,r) ;
8      while card(neigbors) < N_L do
9          r ← increase search radius ;
10         neighbors ← neighbors ∪ Search(τ, z, r)) ;
11     neighbors ← sort by distance to z ;
12     c ← 0 ;
13     while x ∈ neighbors && c < N_L do
14         if IsVisble(x, z) then
15             IncreaseSupport(x, z) ;
16             c ← c + 1 ;
```

4.2.2.3 Numerical Implications

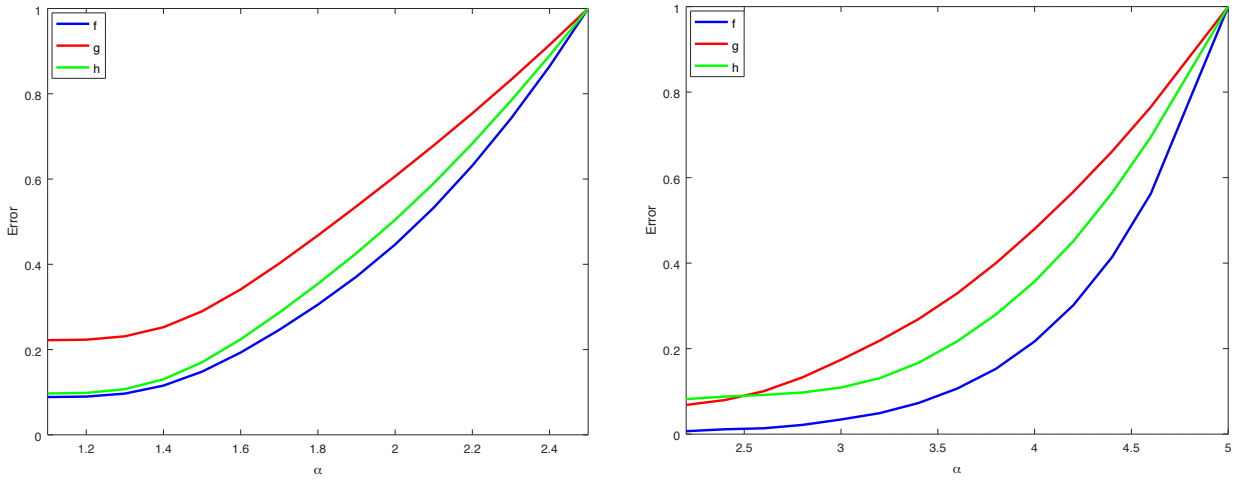
This section will study the impact of the support size on the accuracy of the approximation. In the first part of this section, the implications of the increased support will be demonstrated with three one-dimensional curve fitting functions. For the one-dimensional curve fitting, consider the following set of functions:

$$\begin{aligned} f(x) &= \tfrac{1}{1000}x^3, \\ g(x) &= e^{-x}, \\ h(x) &= \tfrac{x\sin(x)}{1+x^2}, \end{aligned} \tag{4.19}$$

to be approximated by ten equally spaced nodes in the interval $[0, 10]$. The support domain of each particle is the same and is a scaled multiple of the particle spacing determined by $\rho = \alpha\Delta x$, where α is the scale factor and Δx is computed using Equation 4.13. The approximation is performed using 100 evaluation points equally distributed over the domain. The approximation will be constructed as detailed in §2 with both linear and quadratic polynomial fields used. A total of 15 scale factors are used with the jth scale factor for the linear polynomial computed as:

$$\alpha_j = (0.1j + 1.0) \qquad j = 1, 2, \ldots, 15. \tag{4.20}$$

When using the quadratic polynomial field for the reconstruction, the scale values are increased by a factor of two. This is necessary to ensure enough particles cover every evaluation point such that the moment matrix is invertible. The error in the approximation for the jth



(a) Linear polynomial field

(b) Quadratic polynomial field

Figure 4.6: Error vs scale factor

scale factor is computed as:

$$e_j = \sum_{i}^{100} \left(f_i - \hat{f}_i^j \right). \tag{4.21}$$

Here f_i represents the exact value of the function at the ith evaluation point, and $\hat{f}_i^j$ represents the approximated value at the ith evaluation point using the jth scale factor. In the graphical representation of the errors given in Figure 4.6, the error for each function is scaled by the reciprocal of the max error associated with that function. This is done to normalize the error values to be on the same order of magnitude. Figure 4.6a depicts the normalized error in the function reconstruction with linearly consistent shape functions. Figure 4.6b shows the error in the approximation for quadratically consistent shape functions. In both cases it is clear that increasing the size of the support domain can have negative impacts on the quality of the approximation. Based on these results, the support domain should be chosen to be as small as possible, while still satisfying the requirements for an admissible particle distribution when using the meshfree approximation functions for function reconstruction.

4.3 Discretization Refinement

The primary focus of the previous discussion was on domain discretization. However, the same concepts apply when refining an existing discretization. This type of refinement is typically referred to as h-refinement, where h refers to the characteristic length of the discretization. The concept of h-refinement is a technique used to improve the accuracy of a discrete approximation by increasing the resolution of the discretization. Within the context of FEM, this is done by sub-dividing elements. This generally requires the incorporation of simplexes into the mesh, thereby reducing a quadrilateral element to a triangular element or a hexahedron to a tetrahedron, as examples. Due to the lower order interpolation capabilities of the simplex elements, this reduction in element type could result in loss of accuracy. Contrary to FEM, meshfree methods have no concept of element types, making them well-suited for adaptive refinement.

4.3.1 Particle Placement

Assuming some indicator exists for determining a region where refinement should occur, the next step is the insertion of new particles. Given the number of particles and region to refine, the particles are inserted using a simple rejection sampling method, followed by execution of the simulated annealing algorithm previously discussed. The particles used in the simulated annealing algorithm require an associated radius and constraint value. An ideal radius associated with each particle in the simulated annealing algorithm is difficult to obtain. From the previous discussion on using the simulated annealing algorithm in the domain discretization process, the particles' radii were determined based on the size of the tree

node used to generate the particle. In the context of local refinement, no tree is necessary. However, a simple method based on the number of particles that can fit within the refinement region may be used. The constraint value is dependent on the necessity of particles retaining their original positions. Nodes located along the boundary are likely candidates to be constrained. Another set of particles that might be constrained are those particles not within the refinement region. This allows those newly inserted particles and the existing particles within the refinement region to move, but restricts movement of other particles in the domain. The reason for including the particles from outside the refinement region is to allow their contributions to the unconstrained particles' potentials, which restricts an unconstrained particle from becoming too close to another particle in the domain. The use of constraints also allows for a more expedient simulation, as the potential at the constrained particles does not need to be calculated, only their contribution to the unconstrained particles' potentials.

4.3.2 Adaptive Integration

In FEM, the additional integration points are obtained from the newly added elements. In meshfree methods, the newly added nodes do not come with additional integration points. This presents a challenge when performing refinement in meshfree methods, as the increased number of nodes discretizing the domain might result in larger integration errors. This integration error could offset the increased accuracy coming from the resolution of the discretization. This work will use a grid approach similar to [8]. The general idea is to use a non-conforming mesh, which facilitates easy construction, then use Gauss Quadrature on each element. This can be done by using the regular grid concept presented in §3.2.1.1.

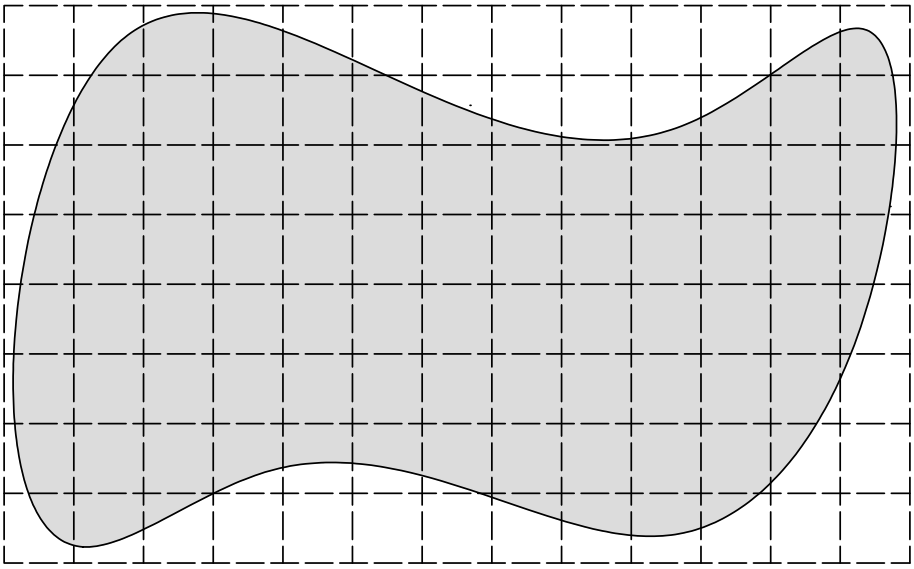


Figure 4.7: Non-conforming integration grid

Figure 4.7 illustrates the concept for the integration over some arbitrary domain. The construction routine requires the bounds of the domain, grid resolution, and the quadrature rule. In [8], the grid is composed of N_c x N_c, where the number of cells, N_c, is computed as

$$N_c = \sqrt{N_n}, \tag{4.22}$$

with N_n being the number of nodes discretizing the domain. Each cell is then used to generate an N_g x N_g set of quadrature points with the the number of quadrature points determined by the following expression:

$$N_g = \sqrt{m} + 2. \tag{4.23}$$

Here m is the number of particles residing within a cell. These particles within a cell are determined based strictly on their spatial location and do not account for their support

domain. Therefore, those cells that do not have a particle may still be within the domain and necessary for integration, requiring the addition of the constant of two in Equation 4.23. Deviating from the work in [8], this work will use a grid with resolution N_{cx} x N_{cy}, where the resolution in each spatial dimension is computed from Equation 3.7. This ensures that the grid cells are approximately square (2D) or cuboidal (3D) and eliminates issues with mapping of integration weights, which occur for cells with high aspect ratios. These high aspect ratios may result if Equation 4.22 is used. The quadrature rule to use within each cell is based on the number of particles within the cell, but determining whether a particle is within a cell is based upon whether the particle's support overlaps the cell. The resulting grid is identical to the Case 1 grid data structure described in §3.2.1.1. In contrast to increasing the quadrature rule as in [8] with Equation 4.23, the quadrature rule is kept constant and the cell is subdivided. The number of subdivisions is based upon the number of particles and is computed as $N_{sub} = \sqrt{m}$, with m being the number of particles intersecting the grid cell. Increasing the subdivision allows for problems with integration points not aligning with supports to be addressed with a relatively coarse initial grid and low order quadrature rule. This approach allows for a straightforward integration scheme when additional particles are added to the domain, or the particle locations are modified during the simulation. For the case of additional particles being added to the domain, the particles simply need to be added to the integration grid, and the integration points within those cells intersected by the particles can be recomputed if necessary.

4.3.3 Numerical Examples

To study the efficacy of the particle placement algorithm with regard to domain refinement for boundary value problems, two solid mechanics problems will be used. For both examples the RKPM approximation functions are constructed with linear consistency. The approximation functions are used as a basis for the Galerkin weak form discussed in §5. As previously discussed, the meshfree shape functions do not possess the kronecker delta property, therefore the imposition of essential boundary conditions is done using a modified weak form approach. The integration of the weak form is performed using the grid method previously discussed.

The measure of error used is the L_2 norm of the displacement error given as

$$\|\mathbf{u}^h - \mathbf{u}_{exact}\| = \left(\int_\Omega \left(\mathbf{u}^h - \mathbf{u}_{exact}\right)^T \left(\mathbf{u}^h - \mathbf{u}_{exact}\right) d\Omega \right)^{1/2}. \tag{4.24}$$

Here $\mathbf{u}^h$ is the numerical results and $\mathbf{u}_{exact}$ is the analytic solution. The integration is done using the same background grid and quadrature as used for the weak form.

4.3.3.1 Cantilever Beam

The first example is the cantilever beam as shown in Figure 4.8. This example is commonly used in numerical studies of integration techniques [4] and boundary condition enforcement for meshfree methods. The goal here, however, is to investigate the effectiveness of the domain refinement and support domain determination discussed earlier in this chapter.

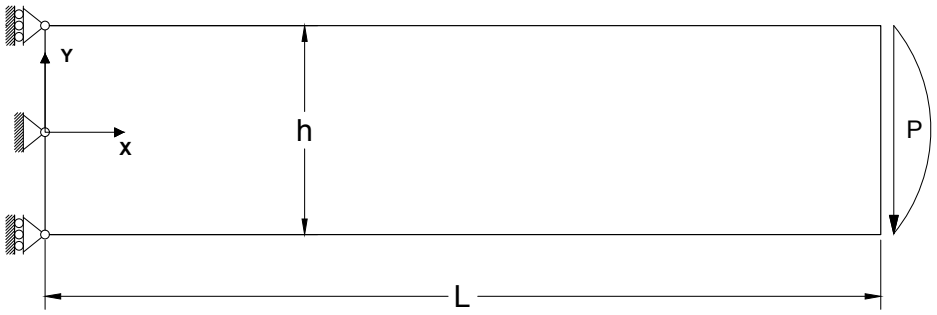


Figure 4.8: Timoshenko Beam

The exact solution can be found in [61] and for plane stress is

$$\begin{aligned} u_x &= \frac{-Py}{6EI}\left(\left(6L-3x\right)x+\left(2+\nu\right)\left(y^2-\frac{1}{4}h^2\right)\right), \\ u_y &= \frac{P}{6EI}\left(3\nu y^2\left(L-x\right)+1/4\left(4+5\nu\right)h^2x+\left(3L-x\right)x^2\right), \end{aligned} \tag{4.25}$$

and

$$\begin{aligned} \sigma_{xx} &= \frac{P}{I}\left(L-x\right)y, \\ \sigma_{yy} &= 0, \\ \sigma_{xy} &= \frac{P}{2I}\left(\frac{h^2}{4}-y^2\right), \end{aligned} \tag{4.26}$$

where E and ν represent the elastic modulus and Poisson's ratio, respectively. The second moment of area is denoted by I and for a unit width rectangular cross section is computed via

$$I = \frac{h^3}{12}, \tag{4.27}$$

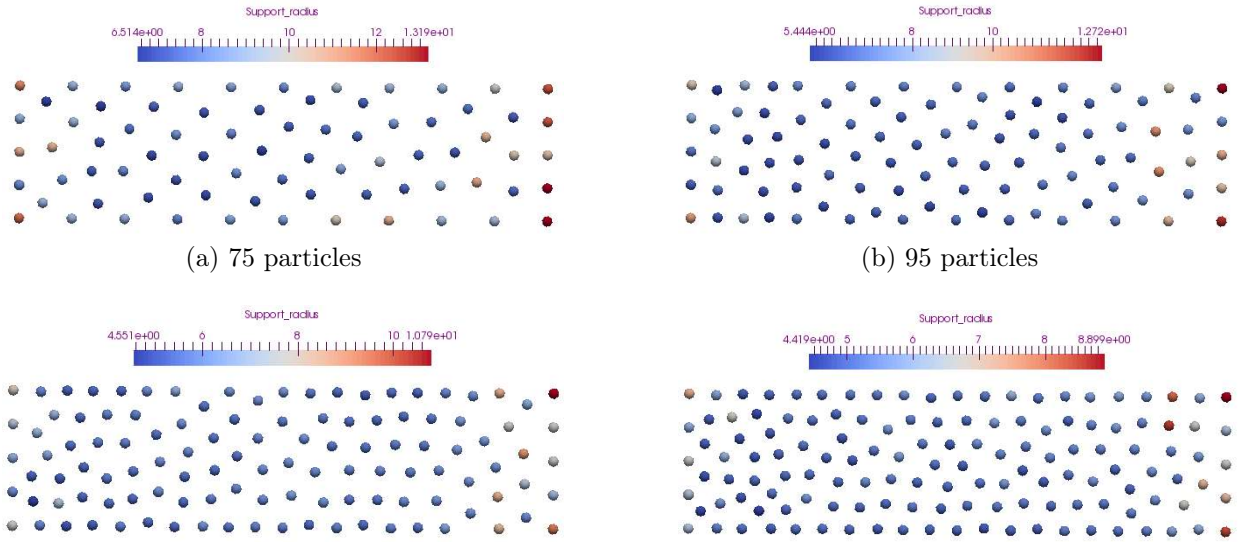
where h denotes the height of the beam as shown in Figure 4.8. The length of the beam is denoted by L and is subjected to a parabolic traction at $x = L$ using the exact solution from Equation 4.26 with the magnitude of the force represented as P. The essential boundary

Table 4.1: Cantilever beam parameters

E (Pa)	ν	L (m)	h (m)	P (N)
3 x 10^6	0.3	48	12	1000

conditions are applied along the boundary $x = 0$ using the exact solution from Equation 4.25. The parameters used for this study are given in Table 4.1.

Four discretizations were considered and are shown in Figure 4.9, with the particles' support domain shape being rectilinear. The particles are colored according to the norm of their support radius. The determination of the support domains was done using Algorithm 16. For the linear basis, the lower bound on the number of particles contributing at a point is three.



(a) 75 particles

(b) 95 particles

(c) 115 particles

(d) 135 particles

Figure 4.9: Particle configurations for beam example

Figure 4.10 depicts the error computed using Equation 4.24 as a function of the discretizations characteristic length. The characteristic length is taken as the largest support

radius magnitude i.e.,

$$h = \max(\|\boldsymbol{\rho}_I\|_2). \tag{4.28}$$

The slope of the line is 2.5, which is below the result obtained in [4] for the same model, but greater than the optimal rate of two. This deviation can be attributed to different techniques used to enforce boundary conditions, integration, and particle distribution. The authors in [4] employed the Lagrange Multipliers technique for enforcing boundary conditions, where this work used Nitsche's method. A two point quadrature rule in conjunction with the hierarchical grid technique described earlier was used here. Beissel *et al.*, used a regular grid technique with higher order quadrature rule within each cell. The most notable difference between the two approaches is in the particle distribution. Here the particle distribution is quasi-uniform compared to the perfectly uniform distribution applied in [4]. The simplicity of the geometry makes generating a uniform discretization straightforward. The goal of this work was to investigate the viability of the stochastic approach for h-refinement, which in general will not result in perfectly uniform discretizations, but is applicable to geometries where perfectly uniform configurations are unobtainable, at least by the way of an automatic procedure.

4.3.3.2 Plate with Hole

The second example is the classical infinite plate with a circular hole as shown in Figure 4.11a. Utilizing symmetry, only the upper right corner of the plate is modeled as shown in Figure 4.11b.

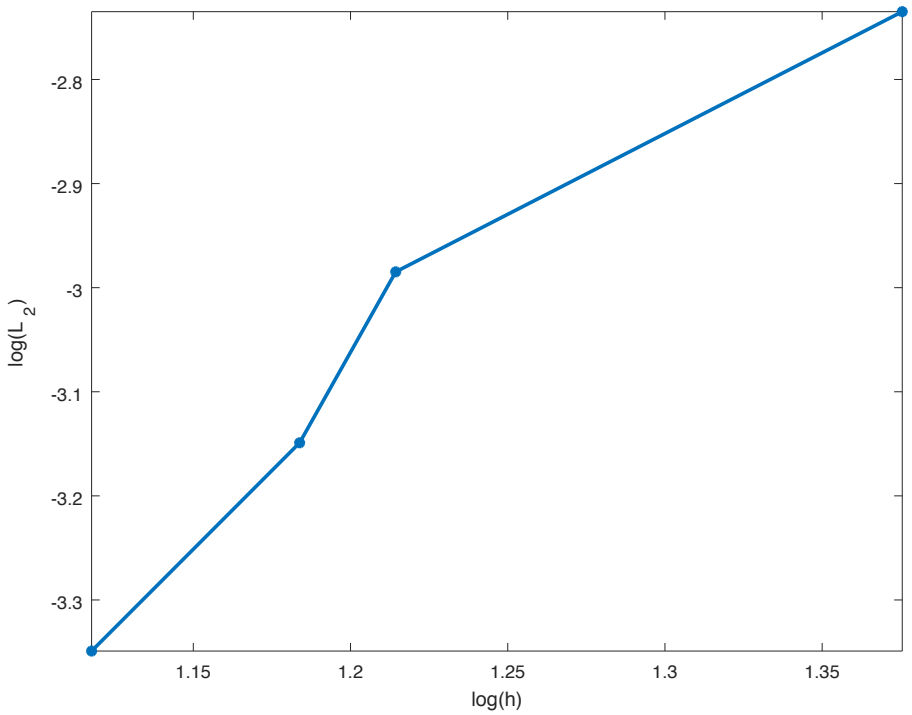


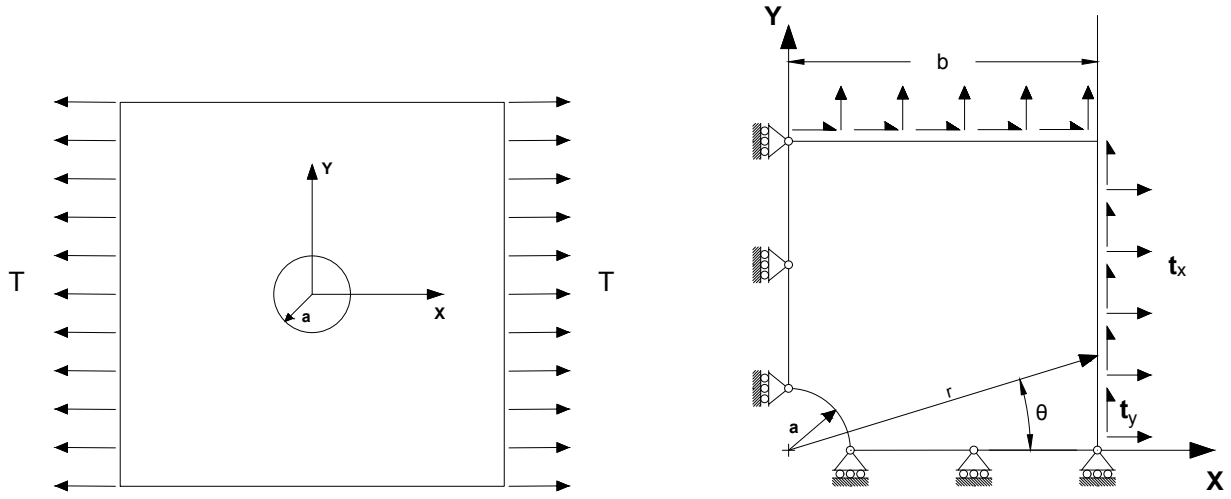
Figure 4.10: L_2 displacement error norm for cantilever beam

The exact solution for the stress is

$$\begin{aligned}
\sigma_{xx} &= T - T\frac{a^2}{r^2}\left(\frac{3}{2}\cos(2\theta) + \cos(4\theta)\right) + T\frac{3}{2}\left(\frac{a}{r}\right)^4 \cos(4\theta), \\
\sigma_{yy} &= -T\frac{a^2}{r^2}\left(\frac{1}{2}\cos(2\theta) - \cos(4\theta)\right) - T\frac{3}{2}\left(\frac{a}{r}\right)^4 \cos(4\theta), \\
\sigma_{xy} &= -T\frac{a^2}{r^2}\left(\frac{1}{2}\sin(2\theta) + \sin(4\theta)\right) + T\frac{3}{2}\left(\frac{a}{r}\right)^4 \sin(4\theta),
\end{aligned} \tag{4.29}$$

where r is the radial distance as measured from the center of the hole and θ is the angle measured with respect to the positive x-axis in the counterclockwise direction. The exact solution for the displacement field is given as

$$\begin{aligned}
u_x &= \frac{Ta}{8\mu}\left(\frac{r}{a}(\kappa+1)\cos\theta + 2\frac{a}{r}\left((\kappa+1)\cos\theta + \cos 3\theta\right) - 2\frac{a^3}{r^3}\cos 3\theta\right), \\
u_y &= \frac{Ta}{8\mu}\left(\frac{r}{a}(\kappa-3)\sin\theta + 2\frac{a}{r}\left((1-\kappa)\sin\theta + \sin 3\theta\right) - 2\frac{a^3}{r^3}\sin 3\theta\right).
\end{aligned} \tag{4.30}$$



(a) Plate with hole (b) Quarter plate model

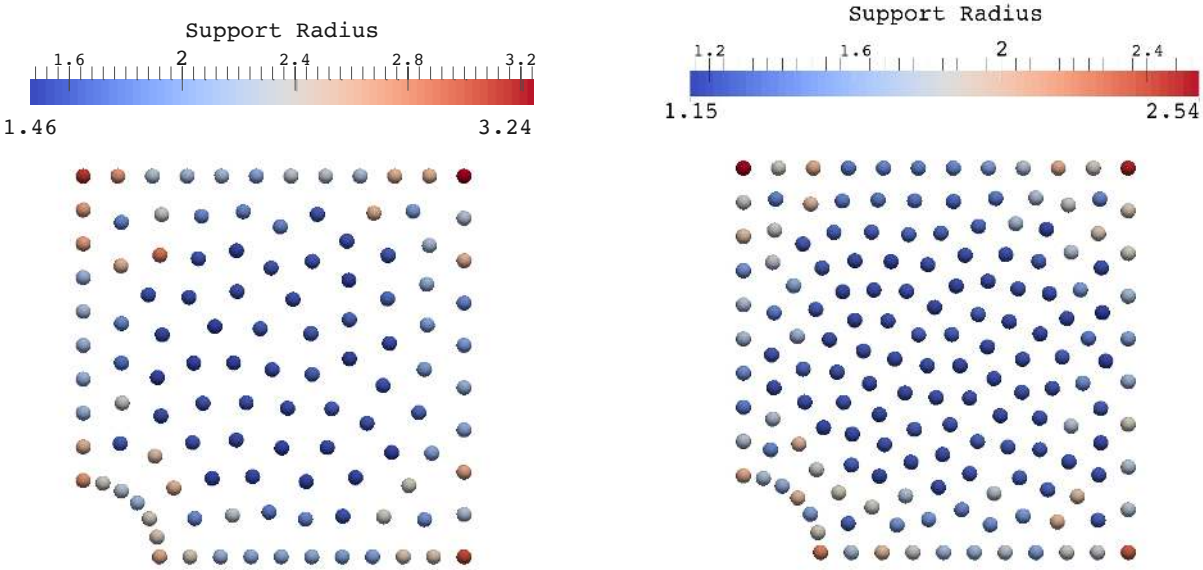
Figure 4.11: Plate with hole

Table 4.2: Plate with hole parameters

E (Pa)	ν	b (m)	a (m)	T (N)
3 x 10^6	0.3	5	1	1000

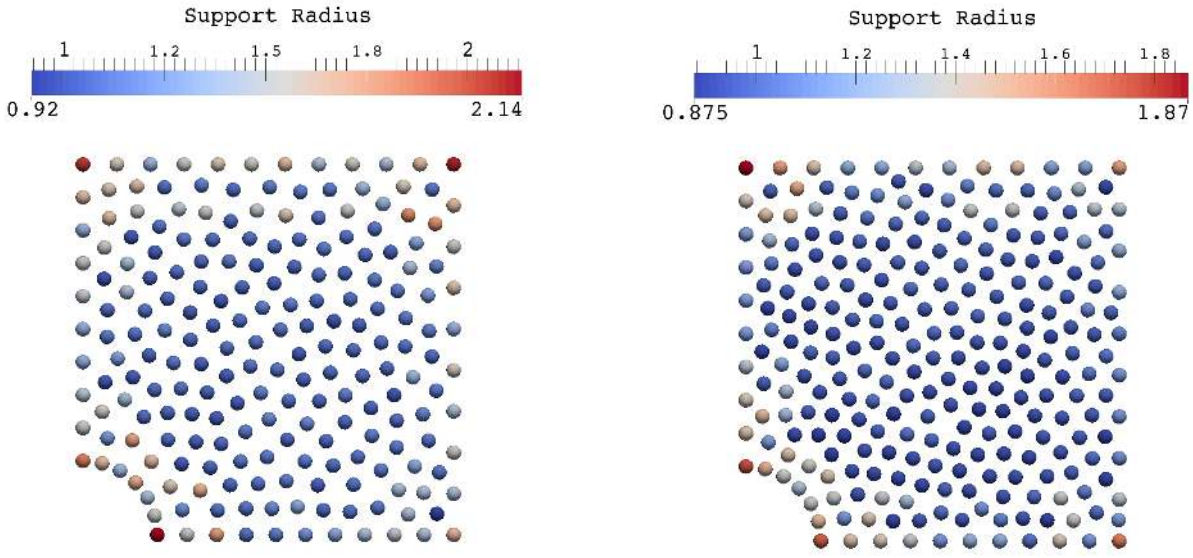
The parameters used for this study are given in Table 4.2. Four discretizations were considered and are shown in Figure 4.13, with the particles' support domain shape being rectilinear. The particles are colored according to the norm of their support size. The determination of the support domains was done using Algorithm 16 and the shape functions are constructed to be linearly consistent.

Figure 4.14 depicts the error computed using Equation 4.24 as a function of the discretizations' characteristic length given by Equation 4.28. The slope of the line is 2.5, indicating a convergence rate slightly greater than the optimal rate of two.



(a) 68 particles (b) 96 particles

Figure 4.12: Plate with hole



(a) 124 particles (b) 152 particles

Figure 4.13: Particle configurations for plate with hole

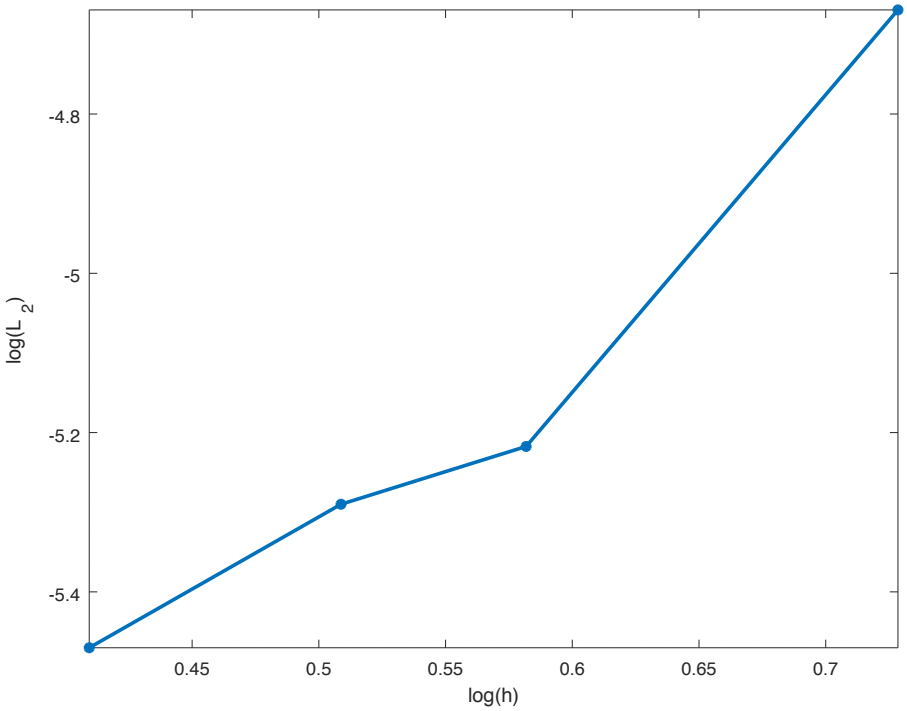


Figure 4.14: L_2 displacement error norm for quarter plate

CHAPTER 5: MECHANICS OF SOLIDS

5.1 Formulation of Governing Equations

Following the standard textbook formulations, such as [7], the formulation of solid mechanics is reviewed. Consider a domain, as shown in Figure 5.1, Ω bounded by Γ, where Γ is composed of traction (Γ_t) and displacement (Γ_u) boundaries. The traction and displacement

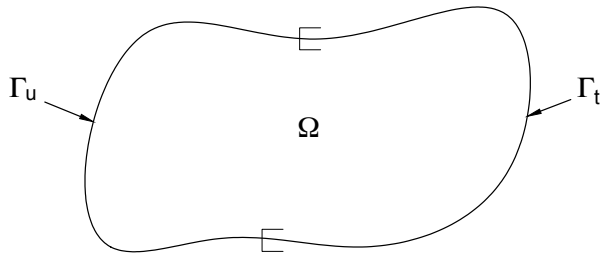


Figure 5.1: Arbitrary domain

boundaries are disjoint i.e.,

$$\Gamma = \overline{\Gamma_u \cup \Gamma_t}$$

$$\emptyset = \Gamma_u \cap \Gamma_t.$$

The governing PDE for static equilibrium or elastostatics is given as

$$\sigma_{ji,j} + b_i = 0, \quad \text{in } \Omega \tag{5.1}$$

subject to the following boundary conditions

$$\begin{aligned} u_i &= \bar{u}_i, \quad \text{on } \Gamma_u \\ n_j \sigma_{ji} &= \bar{t}_i, \quad \text{on } \Gamma_t \end{aligned} \tag{5.2}$$

where σ_{ji} is the Cauchy stress tensor, b_i is the body force per unit volume, $\bar{u}_i$ is the prescribed displacement on the displacement boundary Γ_u, and n_j is the outward normal defined on the traction boundary Γ_t.

The remainder of the work will be restricted to the assumptions of linear elasticity as they pertain to the research conducted. For linear elasticity, the stress and the strain are related by Hooke's law,

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl}, \tag{5.3}$$

where C_{ijkl} is the elasticity tensor and ε_{kl} is the strain tensor. The term linear elasticity corresponds the relationship between the stress and strain. This does not preclude the use of a non-linear strain measure, that is, a non-linear relationship between the strains and displacements. However, this work will employ the infinitesimal strain measure, as the failure theory used in the next chapter relies on the linear kinematic relationship. The kinematic relationship between the strains and displacements is given as the symmetric part of the displacement gradient i.e.,

$$\varepsilon_{kl} = \frac{1}{2}\left(u_{k,l} + u_{l,k}\right). \tag{5.4}$$

5.2 Solution Methodology

Equation 5.1 represents the Conservation of Linear Momentum in the strong form. While numerical methods such as the Finite Difference Method and Collocation Method seek to solve the strong form directly, the consistency requirement of the trial solution must be equivalent to the order of the PDE. Differing from the strong form where the solution must satisfy equilibrium at every point in the domain as well as adhere to the boundary condition, the weak form only requires the solution to satisfy the equilibrium in an averaged sense. There are two approaches to develop the weak form. The first approach is referred to as the weighted residual method. Notable variants of the weighted residual method include the point collocation method, subdomain collocation method, and the Galerkin method. The second approach is based on variational principles that seek to find a solution that minimizes a functional. The two approaches can be shown to result in the same weak form provided the Euler equations of the variational principle are the same as the original problem's governing equations [70], which is the case for elasticity problems.

To develop the weak form using the Galerkin method, the strong form in Equation 5.1 is multiplied with an arbitrary test function $\delta u_i(\mathbf{x})$ and integrated over the domain resulting in:

$$\int_{\Omega} \sigma_{ji,j} \delta u_i d\Omega + \int_{\Omega} b_i \delta u_i d\Omega = 0. \tag{5.5}$$

Performing integration by parts on the first integral in Equation 5.5 and applying the Divergence Theorem leads to,

$$\int_{\Omega} \sigma_{ji}\delta u_{i,j} d\Omega - \int_{\Gamma_t} \delta u_i \sigma_{ji} n_j d\Gamma_t - \int_{\Gamma_u} \delta u_i \sigma_{ji} n_j d\Gamma_u - \int_{\Omega} b_i \delta u_i d\Omega = 0, \tag{5.6}$$

where the integration over the boundary was separated into the summation of integrals over the traction and displacement boundaries. To complete the derivation of the weak form the trial and test functions must be defined. In the Galerkin method the trial and test functions are chosen as:

$$\begin{aligned} \mathcal{U} &= \{u_i | u_i = \bar{u}_i, \forall \mathbf{x} \in \Gamma_u; u_i \in \mathbb{H}^1(\Omega)\} \\ \mathcal{V} &= \{\delta u_i | \delta u_i = 0, \forall \mathbf{x} \in \Gamma_u; \delta u_i \in \mathbb{H}_0^1(\Omega)\} \end{aligned} \tag{5.7}$$

where $\mathcal{U}$ is the set of trial functions in the Sobolev space, $\mathbb{H}^1(\Omega)$, that take on the prescribed values on the displacement boundary Γ_u. The test functions space, $\mathcal{V}$, is defined as the set of functions in the Sobolev space, $\mathbb{H}_0^1(\Omega)$, which vanish on Γ_u. Here the Sobolev space is a collection of functions that are square integrable and whose first derivatives are also square integrable i.e.,

$$\mathbb{H}^1(\Omega) = \left\{ v | \left(\int_{\Omega} |v|^2 \mathrm{d}\Omega \right)^{\frac{1}{2}} < \infty; \left(\int_{\Omega} |\partial v / \partial x_i|^2 \mathrm{d}\Omega \right)^{\frac{1}{2}} < \infty, i = 1, \ldots, d \right\} \tag{5.8}$$

Applying the definitions of the test and trial functions the weak form of Equation 5.1 becomes

$$\int_{\Omega} \sigma_{ij}\delta u_{i,j} d\Omega + \int_{\Omega} b_i \delta u_i d\Omega - \int_{\Gamma_t} \delta u_i t_i d\Gamma = 0, \tag{5.9}$$

where the integration over the traction integral was recast in terms of the traction boundary condition. The gradient of the test function, $\delta u_{i,j}$, from Equation 5.9 can be restated as

$$\delta u_{i,j} = \delta \varepsilon_{ij} = \frac{1}{2} \left(\delta u_{i,j} + \delta u_{i,j} \right). \tag{5.10}$$

The symmetrization of $\delta u_{i,j}$ comes from the fact that the stress tensor, σ_{ji}, is symmetric and the double contraction of a skew symmetric tensor with a symmetric tensor is zero. Substitution of Equations 5.3, 5.4, and 5.10 into Equation 5.9 results in the following:

$$\int_{\Omega} \delta \varepsilon_{ij} C_{ijkl} \varepsilon_{kl} d\Omega = \int_{\Omega} b_i \delta u_i d\Omega + \int_{\Gamma_t} \delta u_i t_i d\Gamma. \tag{5.11}$$

Equation 5.11 is the weak form of the governing equations. During the formulation of the weak form an assumption on the kinematic admissibility of the test functions was used, but this assumption is not valid for meshfree approximation functions due to the lack of the kronecker delta property. Furthermore, since the trial functions are not interpolants, the imposition of essential boundary conditions is less straightforward when compared to approximation schemes equipped with the kronecker delta property. The penalty method is an approach to enforce the essential boundary conditions for meshfree methods that will be used in this work.

As the end goal of discretizing the weak form is to develop a formulation applicable for computer implementation, Voigt notation will be used instead of indicial notation. The stress tensor is composed of nine components uniquely defining the stress at any point in the domain. From the Conservation of Angular Momentum the Cauchy stress tensor is

symmetric thereby resulting in $\frac{1}{2}n\left(n+1\right)$ unique stress components allowing for the second rank tensor to be recast as a one-dimensional vector given as:

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_{xx} & \sigma_{yy} & \sigma_{zz} & \sigma_{yz} & \sigma_{xz} & \sigma_{xy} \end{Bmatrix}^T. \tag{5.12}$$

This transformation of a higher order tensor to a column matrix is referred to as Voigt notation [7]. Similar to the conversion of the stress tensor in Equation 5.12, the tensorial strain ε_{ij} can be recast as a column matrix through the kinematic Voigt rule resulting in the following expression:

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_{xx} & \varepsilon_{yy} & \varepsilon_{zz} & 2\varepsilon_{yz} & 2\varepsilon_{xz} & 2\varepsilon_{xy} \end{Bmatrix}^T. \tag{5.13}$$

The factor of two applied to the shear strains stems from the requirement that the strain energy be equivalent in indicial notation and Voigt notation. The tensorial shear strains in Equation 5.13 are commonly replaced with engineering shear strains resulting in the following:

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_{xx} & \varepsilon_{yy} & \varepsilon_{zz} & \gamma_{yz} & \gamma_{xz} & \gamma_{xy} \end{Bmatrix}^T. \tag{5.14}$$

The constitutive relationship in Equation 5.3 is expressed in Voigt notation as:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon}, \tag{5.15}$$

where the 6 x 6 matrix $\mathbf{D}$ is the Voigt form of C_{ijkl} and is commonly referred to as the elasticity matrix of moduli. Further discussion on the form of $\mathbf{D}$ will occur in the next section and for now the only information needed is the matrix is symmetric. Continuing

with transformation of the indicial notation to matrix notation, the engineering strain is related to the displacements by the following equation:

$$\boldsymbol{\varepsilon} = \mathcal{L}\mathbf{u}, \tag{5.16}$$

where $\mathbf{u}$ is the the displacement vector,

$$\mathbf{u} = \begin{Bmatrix} u_1 & u_2 & u_3 \end{Bmatrix}^T, \tag{5.17}$$

and where $\mathcal{L}$ is the matrix defined as:

$$\mathcal{L} = \begin{bmatrix} \partial/\partial x_1 & 0 & 0 \\ 0 & \partial/\partial x_2 & 0 \\ 0 & 0 & \partial/\partial x_3 \\ 0 & \partial/\partial x_3 & \partial/\partial x_2 \\ \partial/\partial x_3 & 0 & \partial/\partial x_1 \\ \partial/\partial x_2 & \partial/\partial x_1 & 0 \end{bmatrix}. \tag{5.18}$$

The approximation to the displacement field can be expressed as,

$$\mathbf{u}(\mathbf{x}) = \sum_{I}^{card(\Lambda)} \mathbf{N}_I(\mathbf{x})\, \hat{\mathbf{u}}_I, \tag{5.19}$$

where the summation is taken over all the particles that contribution at $\mathbf{x}$. $\mathbf{N}$ is the matrix of shape functions defined as:

$$\mathbf{N}_I(\mathbf{x}) = \Psi_I(\mathbf{x})\,\underline{\underline{\mathbb{I}}} = \begin{bmatrix} \Psi_I(\mathbf{x}) & 0 & 0 \\ 0 & \Psi_I(\mathbf{x}) & 0 \\ 0 & 0 & \Psi_I(\mathbf{x}) \end{bmatrix}, \tag{5.20}$$

and $\hat{\mathbf{u}}$ is the vector of nodal coefficients expressed as:

$$\hat{\mathbf{u}} = \begin{Bmatrix} \hat{u}_{1I} & \hat{u}_{2I} & \hat{u}_{3I} \end{Bmatrix}^T. \tag{5.21}$$

Substitution of Equation 5.19 into Equation 5.16 leads to the following expression for the strain:

$$\boldsymbol{\varepsilon} = \sum_{I}^{card(\Lambda)} \mathbf{B}_I \hat{\mathbf{u}}_I, \tag{5.22}$$

where $\mathbf{B}_I$ is commonly referred to as the strain displacement matrix and is given as

$$\mathbf{B}_I = \begin{bmatrix} \Psi_{I,1} & 0 & 0 \\ 0 & \Psi_{I,2} & 0 \\ 0 & 0 & \Psi_{I,3} \\ 0 & \Psi_{I,1} & \Psi_{I,2} \\ \Psi_{I,2} & \Psi_{I,3} & 0 \\ \Psi_{I,1} & 0 & \Psi_{I,3} \end{bmatrix}. \tag{5.23}$$

The Galerkin weak form from Equation 5.11 can be recast using Voigt notation as:

$$\int_{\Omega} \delta \left(\mathcal{L}\mathbf{u}\right)^T \mathbf{D} \left(\mathcal{L}\mathbf{u}\right) \mathrm{d}\Omega - \int_{\Omega} \delta\mathbf{u}^T \cdot \mathbf{b} \mathrm{d}\Omega - \int_{\Gamma_t} \delta\mathbf{u}^T \cdot \mathbf{t}_{\Gamma} \mathrm{d}\Gamma - \int_{\Gamma} \left(\mathbf{u} - \bar{\mathbf{u}}\right)^T \cdot \boldsymbol{\alpha} \int_{\Gamma} \left(\mathbf{u} - \bar{\mathbf{u}}\right) \mathrm{d}\Gamma = 0. \tag{5.24}$$

The last integral in Equation 5.24 is the penalty term used to enforce the essential boundary conditions, with $\boldsymbol{\alpha}$ representing the diagonal matrix of penalty parameters i.e.,

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix}. \tag{5.25}$$

The penalty parameters can vary with position and in direction, but are generally kept constant in terms of spatial location. The ability to modify the penalty parameter with direction is necessary for boundary conditions applied along an edge in one direction, but not another. Therefore Equation 5.25 will be rewritten as,

$$\boldsymbol{\alpha} = \alpha \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}, \tag{5.26}$$

where

$$s_i = \begin{cases} 1 & \text{if } u_i \text{ is prescribed on } \Gamma_u, \\ 0 & \text{if } u_i \text{ is not prescribed on } \Gamma_u, \end{cases} \qquad i = 1, \ldots, d. \tag{5.27}$$

Substitution of Equation 5.19 into Equation 5.24 and noting $\delta\hat{\mathbf{u}}_I$ is arbitrary leads to the following linear system:

$$\left(\mathbf{K} + \mathbf{K}^{penalty}\right)\hat{\mathbf{u}} = \mathbf{F} + \mathbf{F}^{penalty}, \tag{5.28}$$

where $\mathbf{K}$ is the global stiffness matrix, $\mathbf{K}^{penalty}$ is the global penalty matrix, $\mathbf{F}$ is the global force vector, and $\mathbf{F}^{penalty}$ is the global penalty force vector. The nodal contributions to the global matrices are computed as follows:

$$\mathbf{K}_{IJ} = \int_{\Omega} \mathbf{B}_I^T \mathbf{D}\mathbf{B}_J d\Omega, \tag{5.29}$$

and

$$\mathbf{K}_{IJ}^{penalty} = \int_{\Gamma_u} \mathbf{N}_I^T \boldsymbol{\alpha} \mathbf{N}_J d\Gamma. \tag{5.30}$$

The global contributions to the forcing vectors are:

$$\mathbf{F}_I = \int_{\Gamma_t} \mathbf{N}_I \bar{\mathbf{t}} d\Gamma + \int_{\Omega} \mathbf{N}_I \mathbf{b} d\Omega, \tag{5.31}$$

and

$$\mathbf{F}_I^{penalty} = \int_{\Gamma_u} \mathbf{N}_I \boldsymbol{\alpha} \bar{\mathbf{u}} \mathrm{d}\Gamma. \tag{5.32}$$

These final equations are the fundamental equations with the assumptions of small strains or more specifically small rotations and a linear relationship between the stress and strain. In the context of FEM, the penalty terms in Equation 5.28 are not common as the finite element shape functions possess the kronecker delta property so imposition of boundary conditions

are straightforward using methods such as static condensation. The next section will discuss the constitutive relationship.

5.3 Mechanics of Laminated Composites

A composite material is composed of at least two constituents that are combined at the macroscopic level and are insoluble in one another. The first component is the reinforcement phase, which is embedded in the second component known as a matrix phase. In laminated composites, fibers are a common form of the reinforcement phase. The fiber reinforcements are characterized as long unidirectional structural elements exhibiting a high modulus of elasticity. The matrix is generally composed of a continuous material which surrounds the fibers and serves as a mechanism to distribute the loads amongst the fibers. A single layer of unidirectional or woven fibers embedded in a matrix is referred to as a lamina. An oriented stack of lamina where each layer can vary in orientation and material properties is a laminate [33]. Figure 5.2 shows three individual lamina with varying fiber orientation that are combined to form a single laminate.

Until now there have been no assumptions or restrictions on the formulation to laminate composites. The only assumptions are of linear elasticity and infinitesimal strain. Both of these are common practice when conducting an analysis of a variety of materials. This section will specialize the governing equations developed in the last section for laminated composites. This specialization will occur in the constitutive relationship.

The elastic material parameters for orthotropic materials are defined in terms of their Young's moduli, shear moduli, and Poissons' ratios. The form of the elasticity matrix can

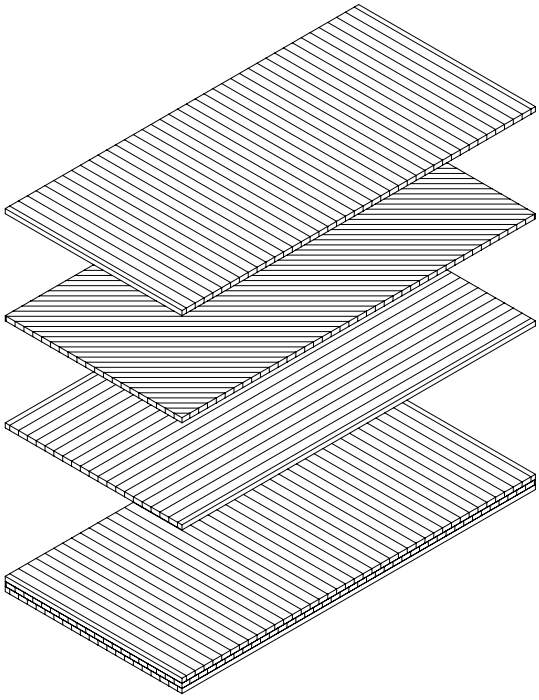


Figure 5.2: Laminated composite

be expressed in terms of the parameters as:

$$\mathbf{D}' = \begin{bmatrix} \frac{1}{E_{1'}} & -\frac{\nu_{1'2'}}{E_{2'}} & -\frac{\nu_{1'3'}}{E_{3'}} & 0 & 0 & 0 \\ -\frac{\nu_{2'1'}}{E_{1'}} & \frac{1}{E_{2'}} & -\frac{\nu_{2'3'}}{E_{3'}} & 0 & 0 & 0 \\ -\frac{\nu_{3'1'}}{E_{1'}} & -\frac{\nu_{3'2'}}{E_{2'}} & \frac{1}{E_{3'}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{G_{2'3'}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{G_{1'3'}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{G_{1'2'}} \end{bmatrix}, \tag{5.33}$$

where the prime indicates these material parameters are measured with respect to the three axes of material symmetry. Symmetry of the elasticity matrix reduces the number of independent material parameters to nine. The relationship between stress and strain in the

primed coordinate system is simply:

$$\boldsymbol{\sigma}' = \mathbf{D}'\boldsymbol{\varepsilon}'. \tag{5.34}$$

The relationship in the global basis is:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon}. \tag{5.35}$$

To develop the relationship between $\mathbf{D}'$ and $\mathbf{D}$ a formulation for the transformation tensors is needed. A tensor is transformed by the following equation:

$$\sigma'_{ij} = Q_{ki}Q_{mj}\sigma_{km}, \tag{5.36}$$

where Q_{ki} is an orthogonal transformation tensor. Equation 5.36 can be expressed in matrix form as:

$$\boldsymbol{\sigma}' = \mathbf{T}\boldsymbol{\sigma} \tag{5.37}$$

where $\mathbf{T}$ denotes the transformation matrix from the unprimed to primed coordinate system and is given as

$$\mathbf{T} = \begin{bmatrix} Q_{11}^2 & Q_{12}^2 & Q_{33}^2 & 2Q_{12}Q_{13} & 2Q_{11}Q_{13} & 2Q_{11}Q_{12} \\ Q_{21}^2 & Q_{22}^2 & Q_{33}^2 & 2Q_{22}Q_{23} & 2Q_{21}Q_{23} & 2Q_{21}Q_{22} \\ Q_{31}^2 & Q_{32}^2 & Q_{33}^2 & 2Q_{32}Q_{33} & 2Q_{31}Q_{33} & 2Q_{31}Q_{32} \\ Q_{21}Q_{31} & Q_{22}Q_{32} & Q_{23}Q_{33} & Q_{22}Q_{33}+Q_{23}Q_{32} & Q_{21}Q_{33}+Q_{23}Q_{31} & Q_{21}Q_{32}+Q_{22}Q_{31} \\ Q_{11}Q_{31} & Q_{12}Q_{32} & Q_{13}Q_{33} & Q_{12}Q_{33}+Q_{13}Q_{32} & Q_{11}Q_{33}+Q_{13}Q_{31} & Q_{11}Q_{32}+Q_{12}Q_{31} \\ Q_{31}Q_{31} & Q_{12}Q_{22} & Q_{13}Q_{23} & Q_{12}Q_{23}+Q_{13}Q_{22} & Q_{11}Q_{23}+Q_{13}Q_{21} & Q_{11}Q_{22}+Q_{12}Q_{21} \end{bmatrix}. \tag{5.38}$$

This transformation is valid for the Voigt form of the stress tensor, but the relationship does not hold for the Voigt form of the strain tensor. This is due to the kinematic Voigt rule used to transform the strain tensor i.e., the factor of two applied to the shear strains. However, if the same Voigt rule is applied to the strain tensor as done for the stress tensor or,

$$\boldsymbol{\epsilon} = \begin{Bmatrix} \varepsilon_{11} & \varepsilon_{22} & \varepsilon_{33} & \varepsilon_{23} & \varepsilon_{13} & \varepsilon_{12} \end{Bmatrix}, \tag{5.39}$$

then the same transformation applied to the stress in Equation 5.37 can be applied to $\boldsymbol{\epsilon}$. The two forms of the strain tensor in Voigt notation can then be cast as:

$$\boldsymbol{\varepsilon} = \mathbf{R}\boldsymbol{\epsilon}, \tag{5.40}$$

where $\mathbf{R}$ is

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}. \tag{5.41}$$

The relationship between the strain in the primed and unprimed basis can be stated as:

$$\boldsymbol{\varepsilon}' = \mathbf{RTR}^{-1}\boldsymbol{\varepsilon} \tag{5.42}$$

Substitution of Equations 5.42 and 5.37 into 5.34 and simplifying yields:

$$\boldsymbol{\sigma} = \mathbf{T}^{-1}\mathbf{D}'\mathbf{RTR}^{-1}\boldsymbol{\varepsilon}. \tag{5.43}$$

The elasticity matrix in Equation 5.29 can be expressed as

$$\mathbf{D} = \mathbf{T}^{-1}\mathbf{D}'\mathbf{RTR}^{-1}\boldsymbol{\varepsilon}, \tag{5.44}$$

allowing for the mechanics of laminated composites with any fiber orientation to be analyzed.

CHAPTER 6: EXPLICIT FRACTURE MODELING

Modeling a crack in a continuum model requires embedding some form of discontinuity into the continuous formulation. Models not based on a continuum approach, such as molecular dynamics, are able to naturally evolve to open discontinuities in a material. However, the number of molecules required for molecular dynamics simulations capable of modeling practical engineering fracture problems results in an intractable problem with high computational expense. Analytical solutions, such as the Westergaard's solution, have been developed for analyzing the mechanics of fracture, but these models are limited by assumptions in the geometry, constitutive model, loading conditions, and description of the crack. In order to understand the effect fracture has on a general specimen, computational techniques are employed.

Early efforts to employ numerical techniques to model fracture relied on mesh-based methods such as the Finite Element Method. FEM treats discontinuities in the topology as internal boundaries that must conform to element edges. For problems where the topological structure of the system evolves, such as crack propagation, this requires the mesh to be recomputed at each step of the simulation and the field values to be interpolated to the new mesh [56]. While much work has been done on mesh generation, the procedure is computationally inefficient and depending upon complexity of the domain, requires manual

correction of the mesh. The problem of updating the field values to the new mesh has potential loss of information from step to step and is an additional computational burden.

To reduce the burden of re-meshing and eliminate the requirement of the mesh conforming to the boundaries of the discontinuities, Belytschko and Black [5] and Moës [48] developed the extended finite element method (XFEM). The fundamental idea behind XFEM is that the displacement field is enriched with additional terms to model discontinuity. This enrichment is incorporated into the displacement field upon the nucleation or propagation of a crack. The topology of the domain remains fixed, with discontinuities captured by the displacement.

Meshfree methods do not require a topological map as in FEM, making them well-suited for the simulations involving fracture. In addition to not needing the connectivity information *a priori*, meshfree methods allow for increased continuity of the approximation functions. This work employs a meshfree method for discretizing the governing PDEs, so the approaches for incorporating discontinuity into the approximation will be elaborated on.

6.1 Meshfree Methods for Modeling Fracture

Two general techniques have been developed for incorporating discontinuity into the formulation. The first approach approach referred to as the visibility criterion [8], incorporates the discontinuity by modifying the topology of the domain, but maintains a continuous displacement field. The second approach incorporates the discontinuity by enriching the displacement field with special functions that incorporate discontinuities into the function approximation.

6.1.1 Visibility Method

The visibility method truncates the support domain of the meshfree shape functions. In the visibility method, a point $\mathbf{x}$ is in the support of node $\mathbf{x}_i$ if $\phi(\mathbf{x} - \mathbf{x}_i) > 0$ and if the point is visible from node $\mathbf{x}_i$ when the boundaries are opaque. Figure 6.1 illustrates the effect the visibility condition has on a shape function. The visibility method has been successfully used for incorporating a discontinuity in the displacement field [8, 58, 59]. However, additional discontinuities in the meshfree shape functions covering the crack tip occur. Despite these unintended discontinuities, it has been shown that the Galerkin approximation using moving least squares interpolants based on the visibility criterion is convergent [35]. For certain problems, the visibility approach produces better results than conforming shape functions [36]. In order to restore continuity to the shape functions near the crack tip, the transparency and diffraction methods were developed [51]. While these methods removed the discontinuity of the shape functions near the crack tip, their implementation is not easily extended to three dimensions.

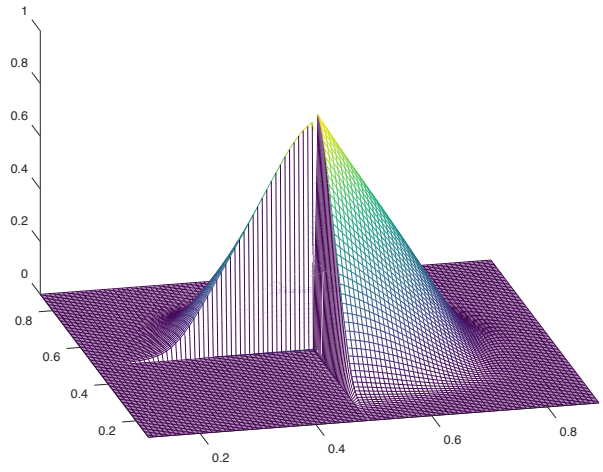


Figure 6.1: Shape function with visibility condition.

6.1.2 Enrichment Techniques

Meshfree Galerkin methods offer the ability to enrich the function space approximation. The standard approach uses monomials, but the approximation space can be enriched by embedding near-tip asymptotic functions in the displacement field [6, 23]. This enrichment comes in two forms, either extrinsic or intrinsic.

Extrinsic enrichment augments the meshfree shape functions to include the terms of the near-tip asymptotic expansion for the displacement field at a crack tip:

$$u^{\rho}(\mathbf{x}) = \sum_{I} \Psi_I(\mathbf{x})\, d_I + \sum_{K}^{N_c} \sum_{i=1}^{N} \Phi_i \Psi_K e_{iK}, \tag{6.1}$$

where Ψ are the meshfree shape functions, N is the number of enrichment functions added to the approximation space, N_c correspond to the number of cracks, e_{iK} are unknown parameters associated with crack K. Generally four enrichment functions are added given as

$$\Phi(\mathbf{x}) = \left[\sqrt{r}\sin\left(\tfrac{\theta}{2}\right) \quad \sqrt{r}\cos\left(\tfrac{\theta}{2}\right) \quad \sqrt{r}\sin(\theta)\cos\left(\tfrac{\theta}{2}\right) \quad \sqrt{r}\sin(\theta)\sin\left(\tfrac{\theta}{2}\right) \right], \tag{6.2}$$

where r and θ are the polar coordinates associated with a basis centered at the crack tip depicted in Figure 6.2. The extrinsic approximation requires the finding of the unknown nodal parameters as well as e_{iK}, which are the unknowns associated with the enrichment. In Equation 6.2, the enrichment field around the crack tip is introduced only for particles whose support covers the crack tip. In practice, this means that the enriched region gets smaller and smaller as the discretization is refined, yielding suboptimal convergence rates. In theory it is desirable to keep the size of the enriched region constant as the discretization is

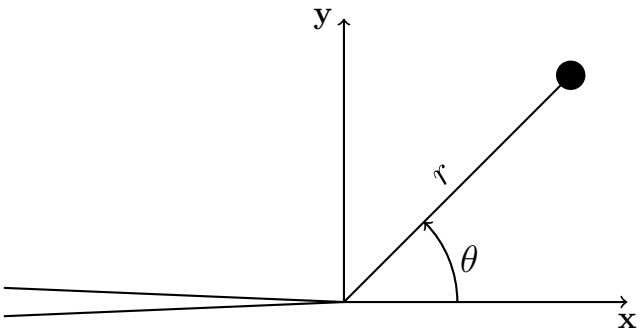


Figure 6.2: Polar coordinates used in enrichment terms

refined. This would lead to optimal convergence, but can cause severe ill-conditioning of the global stiffness matrix [54]. A specifically designed local preconditioner can be constructed to alleviate this issue [54], but this induces a non-negligible computational cost.

Intrinsic enrichment augments the monomial basis terms in the meshfree function approximation with trigonometric functions extracted from the asymptotic field around the crack tip:

$$\mathbf{P}(\mathbf{x}) = \left[1 \quad x \quad y \quad \sqrt{r}\cos\left(\tfrac{\theta}{2}\right) \quad \sqrt{r}\sin\left(\tfrac{\theta}{2}\right) \quad \sqrt{r}\sin(\theta)\cos\left(\tfrac{\theta}{2}\right) \quad \sqrt{r}\sin(\theta)\sin\left(\tfrac{\theta}{2}\right)\right] \tag{6.3}$$

with r and θ corresponding to the Figure 6.2. While this approach does not involve additional unknowns as in the extrinsic enrichment method, the increased size of the basis requires greater computational effort in the inversion of the moment matrix. The approximation functions generally must be the same throughout the domain. Otherwise, discontinuities at the subdomain interfaces can occur, thereby requiring the shape function calculation to be performed using the trigonometric basis throughout the entire domain. This results in a high computational cost and ill-conditioning of the moment matrix. Belytschko *et al.* introduced a ramp function at the subdomain interfaces to handle the discontinuity [23]. This allows

for subdomains near the crack tip to use the enriched basis and those subdomains not near the crack tip use a revert back to the monomial basis.

6.2 Modeling Fracture of Laminated Composite

In laminated composites the failure can be categorized as intra-lamina or inter-lamina. Intra-lamina failures, such as matrix failure, fiber failure, and failure at the interface between the matrix and fibers, occur strictly inside a single lamina. Inter-lamina failure occurs at the interface of two lamina, i.e. delamination. The complexity of modeling fracture is further increased by the orientation of the fibers, which dictates the trajectory of the crack. This effect may result in unphysical crack orientations in mesh-based analysis, as the orientation of the mesh can influence the crack propagation direction. A method known as the Particle Splitting Crack Algorithm (PSCA) [58, 59] explicitly models cracks and changing topology and is formulated to reduce or eliminate any bias from the domain discretization. While the PSCA was able to reduce the effects the discretization has on the solution, certain cases still prove challenging. This work will expand upon the PSCA using techniques discussed in §4 and eliminate the domain decomposition from having any significant impact on the solution.

6.2.1 Laminated Composite Failure Criteria

The objective of a damage model is to provide a means for predicting the failure of a material. Within the context of solid mechanics a multitude of failure theories have been proposed. Failure theories for composite materials are not as mature as those for isotropic materials. As this work is focused on composite materials, those failure models that are not applicable will not be discussed. For composite materials, the failure is difficult to model

due to the micro-mechanical structure. At a microscopic level, the strength of the material is dictated by both constituents as well as the strength of the bond between the two. This makes modeling failure a multi-scale problem. Multi-scale modeling uses knowledge from a more refined level to better predict the coarser model's behavior. At the continuum scale, the effects of the fibers in the lamina are accounted for in the material properties of a linear elastic model. At the microscopic level, the fiber and matrix are resolved, but the fibers are of a size much smaller than the part itself. In order to resolve the fibers within parts of practical dimension, the number of degrees of freedom introduced would result in a system that is impractical to solve. Furthermore, the true locations of the fibers in the composite are not known, only the orientation of the fibers. The homogenization of the fibers into the material properties works well prior to cracking. However, due to the absence of the explicit fibers, the continuum model is presented with a significant challenge of predicting the correct crack morphology. In order to use continuum mechanics to model the physical crack propagation in laminate fibrous composites, the physical constraints pertaining to the micro fiber-matrix system must be accounted for, resulting in a multi-scale problem.

Failure theories such as the Maximum Stress, Maximum Strain, Tsai-Hill, and Tsai-Wu are highly dependent on the fiber orientation. Therefore, these methods are not applicable to an arbitrary configuration. This work will employ the Onset Theory, previously coined the Strain Invariant Failure Theory (SIFT), originally proposed by Gosse [28] for predicting the onset of failure for composite systems. The key component of the Onset Theory is the dehomogenization of the matrix and fiber phases. This allows for the computation of the strain field within each constituent separately. Given these strain fields the failure can be classified based on which constituent failed. The Onset Theory is restricted to linear

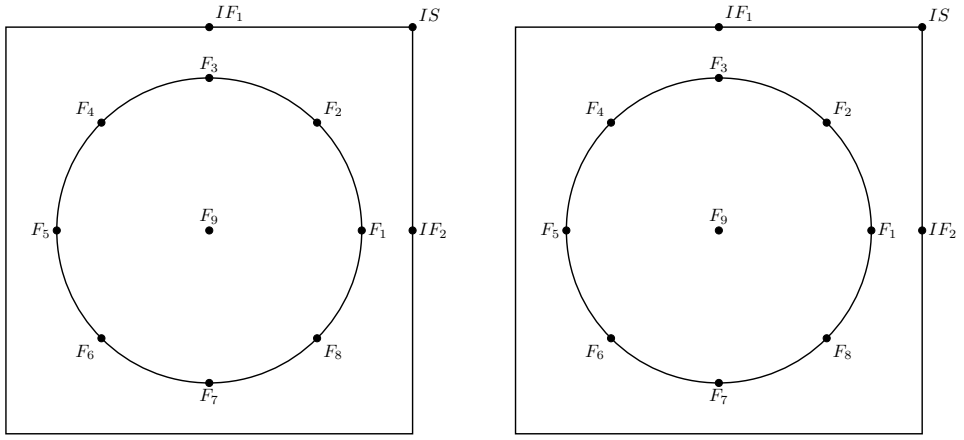
elasticity, hence the measure of strain used is the engineering strain defined in §5. The Onset Theory relies on strain as opposed to stress for determining the onset of failure. This is because the stress varies significantly for determining the failure of polymers depending on whether the failure is due to dilatation or distortion [14].

To properly account for the local effects of the fibers, fiber-matrix interface, and the internal thermally induced strain, the states of strain are micro-mechanically enhanced using the methods described in [13, 28, 60] and is given by,

$$\boldsymbol{\varepsilon} = \mathbf{M}\left(\varepsilon_{applied} - \alpha\Delta T\right) + \mathbf{A}\Delta T + \alpha\Delta T. \tag{6.4}$$

The superscript k represents kth point of interest in the representative volume element (RVE), $\boldsymbol{\varepsilon}$ is the micro-mechanical strain, and α is the thermal expansion coefficient vector of fiber or matrix medium. $\mathbf{M}$ is the strain amplification matrix, $\bar{\varepsilon}$ is the external applied strain, and $\bar{\alpha}$ is the effective thermal expansion coefficient. $\mathbf{A}$ is the thermal enhancement vector and ΔT is the temperature difference between the temperature at curing and operating temperature. Two common configurations for the representative volume element denoting the points of interest are given in Figure 6.3. This enhancement takes place at each evaluation point used to integrate the weak form.

The micro-mechanically enhanced strains are used to compute two scalar values, which are compared to critical values determined from laboratory experiments or molecular models. The two scalar values are invariants of the enhanced strain. The first invariant characterizes



(a) Square cell (b) Hexagonal cell

Figure 6.3: Critical points for single cell

the failure associated with changes in volume or dilatational failures and is defined as

$$J_1 = \varepsilon_1 + \varepsilon_2 + \varepsilon_3, \tag{6.5}$$

where ε_i are the principal strains. The second characterizes failures associated with shear deformation or distortional failures is defined as

$$J_2 = \frac{1}{6}\left((\varepsilon_1 - \varepsilon_2)^2 + (\varepsilon_2 - \varepsilon_3)^2 + (\varepsilon_1 - \varepsilon_3)^2\right). \tag{6.6}$$

6.2.2 Particle Splitting Crack Algorithm

The (PSCA) was developed in two dimensions and used to simulate thermo-mechanical ductile fracture under finite deformations [58]. The method was later expanded to directed surfaces in three dimensions and used to model fracture in laminated composites [59].

The algorithm begins by decomposing the domain with a set of particles through the thickness. Each set of particles through the thickness is referred to as a filament. The algorithm proceeds by conducting a linear elastic analysis. The results of this analysis are used to compute a scalar damage value, described in §6.2.1 for each particle in the system. With the scalar damage value for each particle, the overall damage of the filament is determined. If the filament exceeds the allowable damage, then the filament is split and the crack propagated. In order to propagate the crack, a search is conducted to locate the filament with the next highest damage level. This filament is assumed to be the next crack tip. The failed filament is then split by splitting each particle in the filament. Once particles have split, a visibility condition is enforced by inserting a crack panel along the direction from the failed filament to the next crack tip. The visibility criteria described earlier is then used to update the topology. This procedure is illustrated in Figure 6.4.

6.2.3 Physically Consistent Crack Propagation

The micro-mechanical enhancement allows the determination of material failure and whether it is matrix or fiber at the continuum level. However, the use of an external damage model does not preclude unnatural crack propagation; therefore one must incorporate external knowledge to prevent unphysical crack formation from occurring. If a failure is in the matrix phase, the crack should be restricted from crossing fibers. Consider the lamina in Figure 6.5 with fibers represented by the gray areas and matrix as the white area. Since the morphology of the crack is guided by a scalar value that does not provide information about the physical orientation of the fibers, an unnatural crack trajectory can occur as shown in Figure 6.5a. To prevent this, Simkins *et al.* [59] proposed a discrimination on candidate

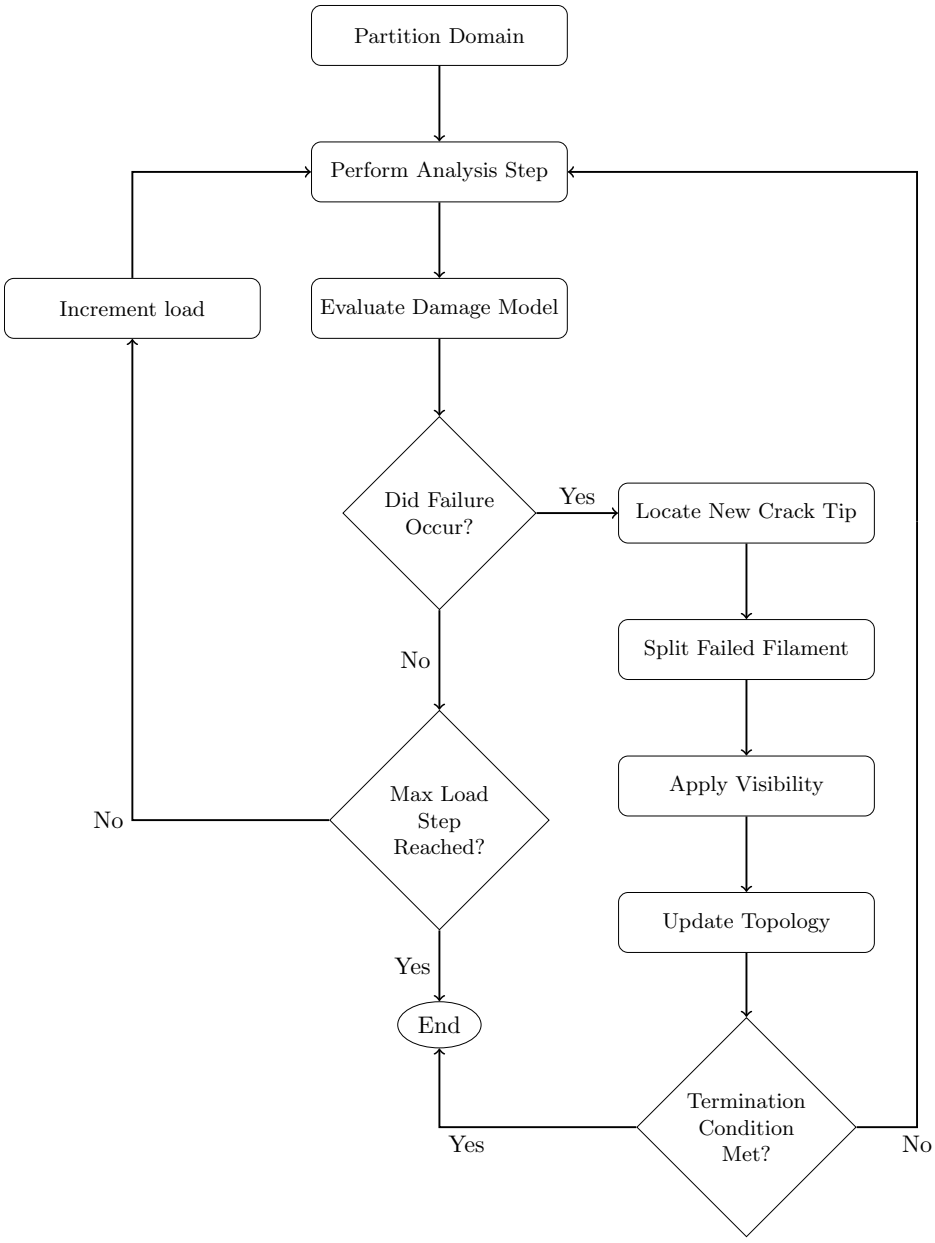
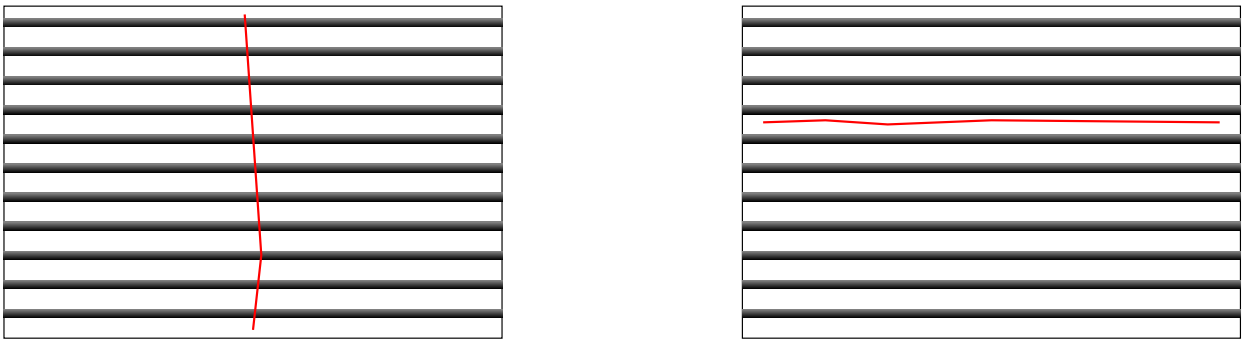


Figure 6.4: Particle splitting crack algorithm flowchart

crack tip particles that lead to unphysical cracking. The remaining potential new crack tips lead to a crack path shown in Figure 6.5b. For a single ply laminate or lamina there is no failure due to delamination. Generally, parts are constructed of multiple lamina and therefore the failure due to delamination should be accounted for. In a multi-ply laminate, an intra-lamina crack does not propagate from one laminate to another. Upon reaching the



(a) Unphysical correct crack path

(b) Physically correct crack path

Figure 6.5: Illustration of physcially consistent and inconsistent crack morphology.

inter-ply interface, the crack results in a delamination. In the PSCA this is handled by restricting the meshfree particles to which a crack may propagate. Once a crack reaches this interface, the crack may only propagate in such a way as to cause delamination. Within the original PSCA the candidate particles are chosen from the existing particles and used as crack initiates and maintained as it propagates. As the crack propagates and searches for new crack directions, particles which were not in the candidate subset determined when the crack was initiated are ignored. An unnatural crack propagation can occur if the damage model is restricted to choose candidates for the new crack tip from existing particles. The proposed work is to develop a robust method to determine the crack direction that is consistent with the material constraints. In order to provide crack path directions that adhere to the physical constraints of the material dictated by the fiber orientation, the set of rules developed by Simkins et al. will be expanded upon. These heuristics will need to account for the failure mode, matrix or fiber, as well as the fiber orientation. In the case of intra-ply matrix failure, which is the most likely scenario, the crack path is restricted to form along the fiber direction and not cross fibers. A challenge with enforcement of this rule is that the particle discretization might not lend itself to locating a particle along the fiber direction.

To address this issue, a method that adapts the discretization through particle insertion is presented. This method will adapt the discretization refinement method presented in §4.3.

6.2.4 Modified Particle Splitting Crack Algorithm

Following a similar process as the original PSCA, the modified algorithm begins by partitioning the domain and conducting an analysis for the initial load step. The damage model is then evaluated at each filament, with those filaments exceeding the prescribed max damage value tagged as failed. These filaments are sorted with respect to their utilization determined as the max ratio of their damage with respect to the critical value for the respective failure mode. The state of the filament with the highest damage that has failed is then determined. The filament state is an identifier dictating how a filament is treated upon failure and whether an initiation or propagation event should occur. The possible states are listed below.

- *Undeformed* The undeformed state refers to filaments that have not failed and have no crack associated with them. This is the state that all filaments are initialized to at the start of the simulation.The filaments are further classified as to whether they are located on a boundary of the domain.

- *Crack tip* A crack tip filament refers to a filament that a crack has terminated at. The filament has not necessarily failed, but is associated with a crack resulting from a nearby filament that has failed. All crack-tip filaments are considered as boundary filaments.

- *Crack face* A crack-face filament is a filament that has failed and is the result of splitting a failed filament that was either a crack tip or located on the boundary. All crack-face filaments are considered as boundary filaments.

Given the filament state the algorithm continues with either an initiation event or propagation event. Initiation occurs when either there are no cracks currently formed or when the failure occurred away from existing crack tips.

6.2.4.1 Initiation

If the state of the failed filament with the largest utilization is undeformed and located on the boundary the algorithm may proceed in one of two ways depending upon whether the failure occurred in the matrix phase or fiber phase. If the failure was in the fiber phase, a similar procedure to the original PSCA is followed. This procedure finds a set of candidate filaments to form the next crack tip. This set of candidates is composed of filaments that interact with the failed filament through their particles' support domains expressed as,

$$\mathcal{C} = \bigcup_{I}^{N_{pf}} \Pi^I \tag{6.7}$$

where $\mathcal{C}$ denotes the set of candidate filaments, N_{pf} is the number of particles in the failed filament, and Π^I is the set of particles interacting with particle I as defined in Equation 3.4. The set of candidate particles is then filtered to remove and candidates that result in unphysical cracks. The first filter removes candidates that are undeformed boundary filaments. This allows for cracks to merge, but prevents cracks forming along exterior boundaries of the domain i.e., along an edge. The second filter removes crack face filaments from the

candidate list. The third filter detects those filaments that are not directly visible from the failed filament. This filter differs from the initial determination of the candidate set as two set filaments may have particles sharing evaluation points, but themselves are visible to one another. From the remaining candidates the filament with the highest damage corresponding to the same failure modes as the current filament is selected as the next crack tip with its associated state updated. The failed filament is then split and a crack is initiated from the failed filament to the new crack tip. For failure occurring in the matrix phase the algorithm begins by determining the same filter candidate set used in the fiber failure case. Instead of choosing a candidate from the set as the new crack tip, the candidate with the largest damage value in the same mode as the failed filament is used to determine whether to propagate along the positive fiber direction or negative fiber direction. This is done by computing the signed projection of the vector from the failed filament located at $\mathbf{x}_I$ to the candidate candidate filament located at $\mathbf{x}_J$ along the fiber direction. The expression for the end point of the crack and subsequent new crack tip is given as:

$$\mathbf{x}_{new} = \mathbf{x}_I + \operatorname{sgn}\left(\mathbf{r}_{IJ} \cdot \hat{\mathbf{d}}_f\right) \lambda \mathbf{d}_f, \tag{6.8}$$

where $\operatorname{sgn}(\cdot)$ is the sign function defined as follows:

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases} \tag{6.9}$$

The vector from the failed filament location to the candidate filament is $\mathbf{r}_{IJ} = \mathbf{x}_J - \mathbf{x}_I$. The fiber direction is denoted as $\hat{bd}_f$, which is assumed to be normalized. The distance from the failed filament location, $\mathbf{x}_I$, along the fiber direction, $\hat{\mathbf{d}}_f$, is represented here as λ. This distance is chosen to be the shortest distance from the failed filament to any other filament in the candidate set $\mathcal{C}$. This prevents excessively large cracks from forming at once, but may result in a longer run time for the simulation. Given the location of the new crack tip and end point of the crack, a new crack tip filament is created and inserted into the discretization. The failed filament is then split forming two crack face filaments followed by the insertion of a crack from the failed filament to the new crack tip. Following the initiation, the stochastic particle evolution algorithm described in §4.1 is used to ensure the quality of the discretization is maintained. The only particles not constrained in the particle evolution algorithm are those particles associated with undeformed non-boundary filaments within a given proximity to the new crack tip. Following the particle placement, the support domains for those particles affected by the motion are updated using Algorithm 16. Handling failure occurring at undeformed non-boundary filaments is done using the same approach as the undeformed boundary case except the failed filament is marked as a crack tip and not split.

6.2.4.2 Propagation

If the failed filament is an existing crack tip or in close proximity to an existing tip a propagation event is triggered. The propagation can be modeled in the same manner as the initiation event occurring at a crack tip. The only alterations to the initiation method an additional filter to be applied to the set of candidates and the splitting of the crack tip. The additional filter removes those candidates that would result in a crack propagating back

on itself. The splitting of the crack tip for propagation differs from the filament splitting performed for an initiation event with respect to the split direction. In the initiation scenario the splitting direction was taken to be the vector perpendicular to the crack direction. Here the split direction is the weighted sum of the perpendicular vectors associated with the previous crack that terminated at the failed crack tip and the new crack being formed. The weight associated with each perpendicular vector is taken to be the crack size. For the case of matrix failure the split direction is the same for initiation and propagation. An illustration of this process is illustrated in Figures 6.6-6.8. In Figure 6.6 the crack tip has been determined as the failed filament with the failure in the matrix phase. The dashed line indicates the allowed crack direction i.e., fiber direction. Here there are no filaments along

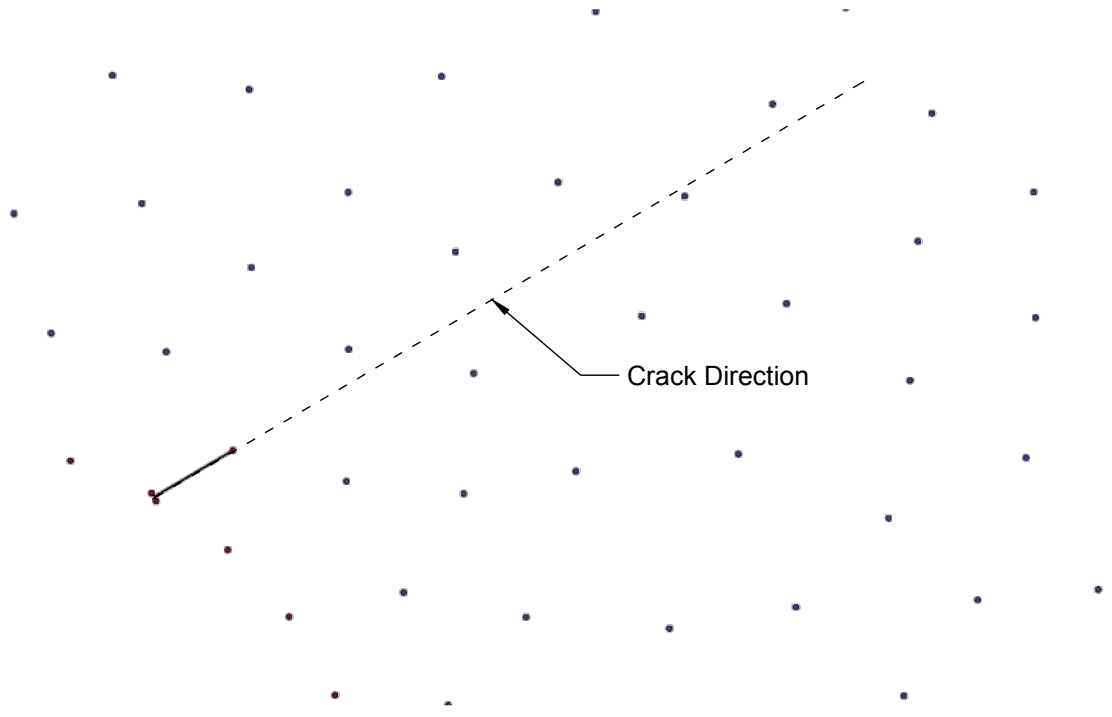


Figure 6.6: Propagation event with direction

this direction, therefore a new filament is inserted and the surrounding particles distributed

using the stochastic approach with the result shown in Figure 6.7. The crack tip is then split

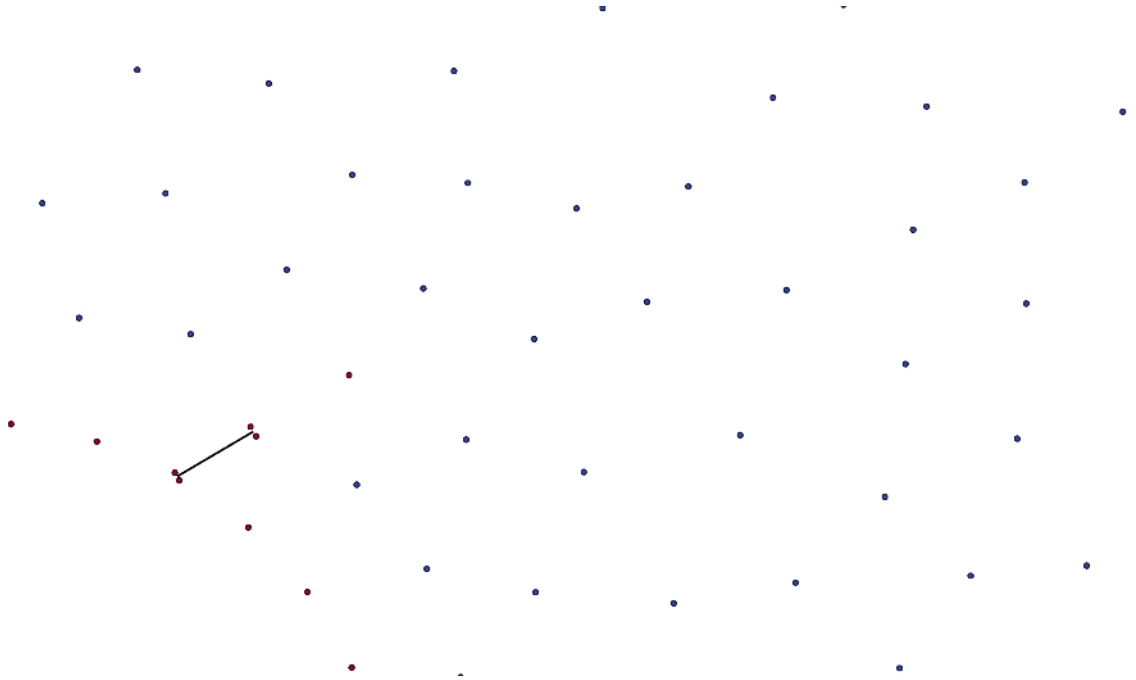


Figure 6.7: Propagation insertion event

and the visibility condition enforced as shown in Figure 6.8.

6.3 Numerical Examples

The following section will demonstrate the ability to model fracture patterns that are physically consistent with the material constraints. The model problem will be a plate with a hole in the center as shown by Figure 6.9. The material will be orthotropic and the properties are given in Table 6.1. As the goal of this work is to demonstrate that the modified particle splitting crack algorithm can be used to model the fracture of laminated composites in a physically consistent manner, no attempts were made to ensure the material parameters align perfectly with experiments. The initial discretization is shown in Figure 6.10. Three different fiber orientations will be considered, with fibers aligned at 0, 30, and 90

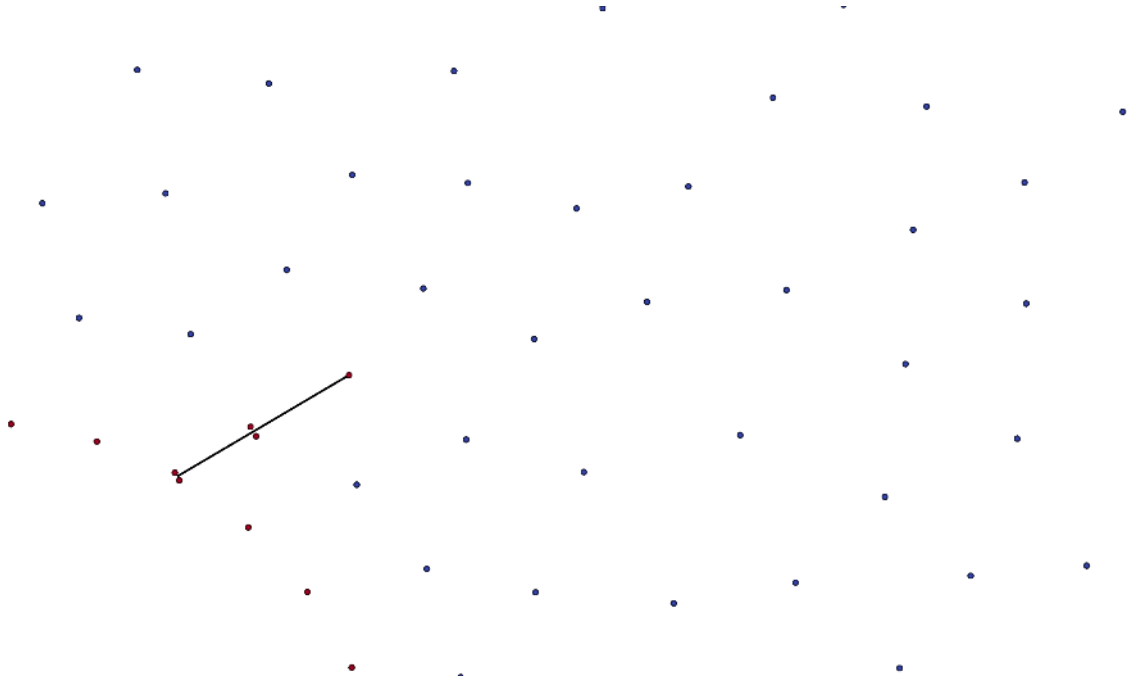


Figure 6.8: Propagation split event

degrees measured with respect to the x axis as shown in Figure 6.9. The plate is subjected to an applied displacement at $x = L/2$ and is constrained along the x-direction along the edge located at $x = -L/2$. To constrain the model against the remaining two remaining rigid body modes the corner at $(-L/2, -H/2)$ is fixed as shown in Figure 6.9. The applied displacement is incrementally increased for each simulation step until the specimen can no longer carry any load.

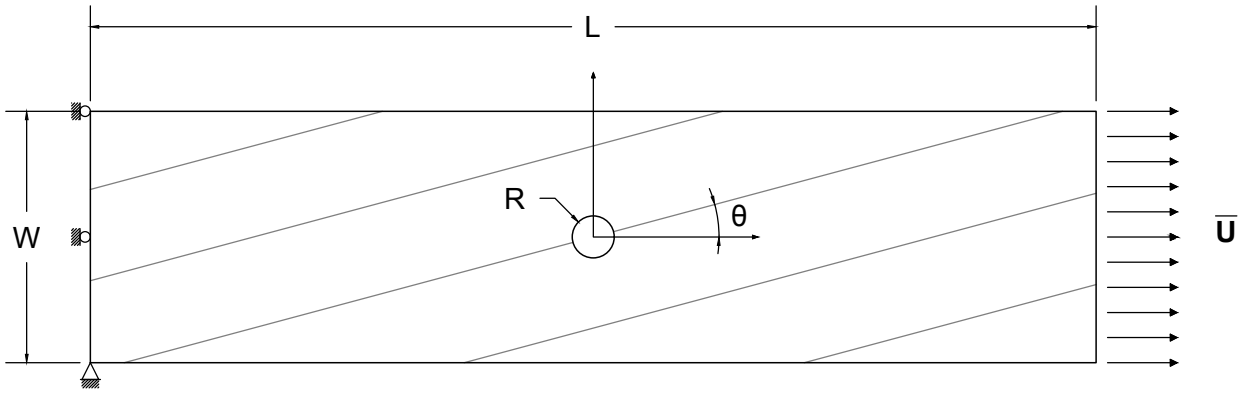


Figure 6.9: Open hole tension model for single ply lamina

Table 6.1: Material properties

E_1	E_2	ν_{12}	G_{12}
2.13 x 10^7	1.13 x 10^7	0.3	580 x 10^3

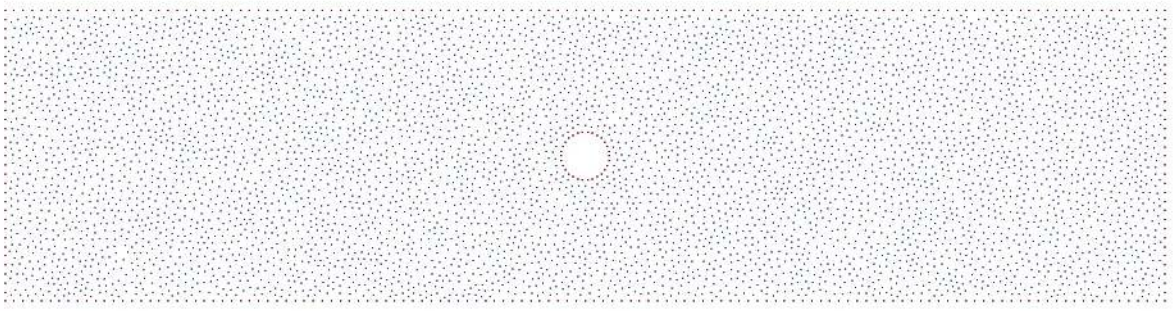


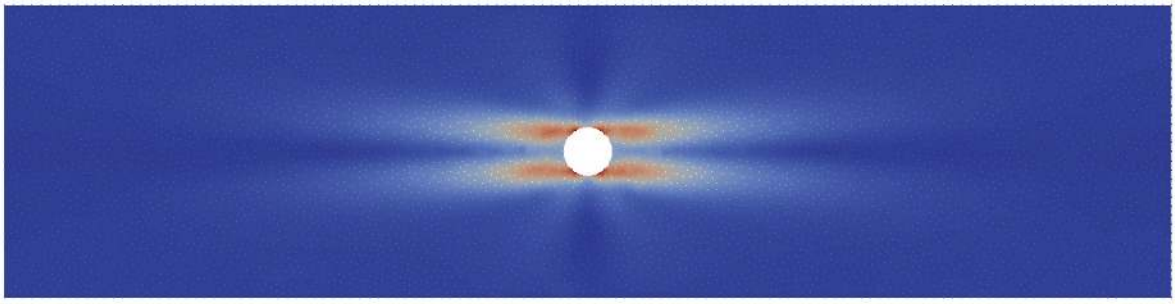
Figure 6.10: Open hole tension example discretization

6.3.1 0 Degree Lamina

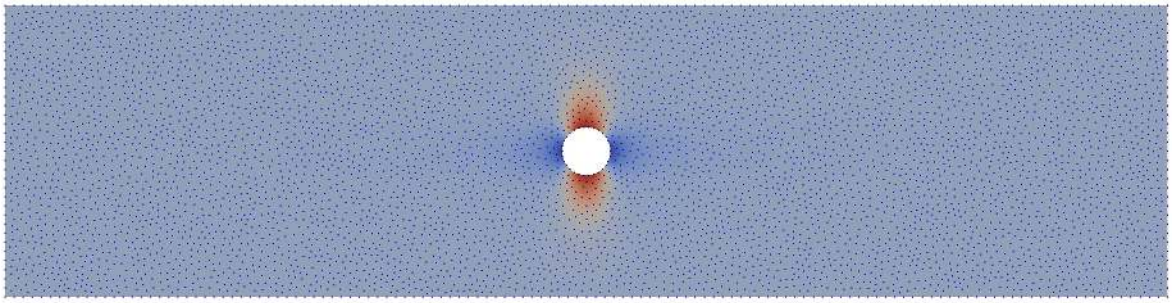
For the zero degree configuration with the failure occurring in the matrix phase the fracture is restricted to propagate along the x- axis. This is in contrast to the result that would be expected for an isotropic material where the highest damage should occur at the top and bottom of the hole and propagate along the y-axis. The distortional and dilatational damage is depicted prior to the first failure occurring in Figures 6.11a and 6.11b respectively. Since the failure is in the matrix phase the crack is force to propagate along the fiber direction. The resulting crack path and for both distortional and dilatational is shown in Figures 6.12a and 6.12b respectively.

6.3.2 30 Degree Lamina

For the configuration with the thirty degree fiber orientation, the failure predicted by Onset theory states the failure occurs in the matrix phase. The distortional and dilatational damage is depicted prior to the first failure occurring in Figures 6.13a and 6.13b respectively.

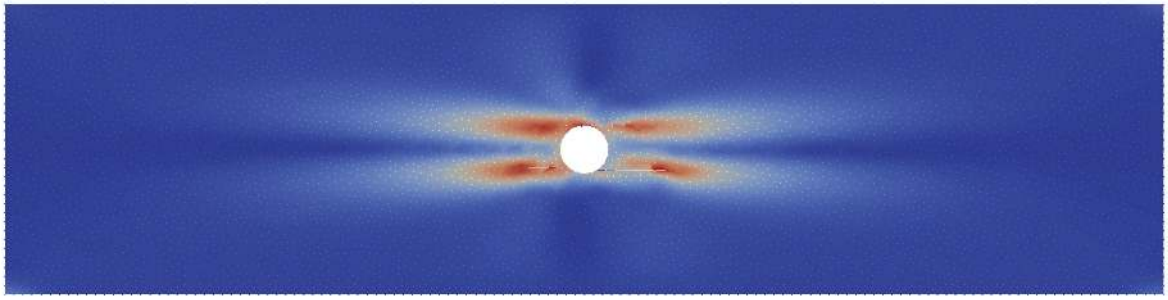


(a) Distortional strain invariant

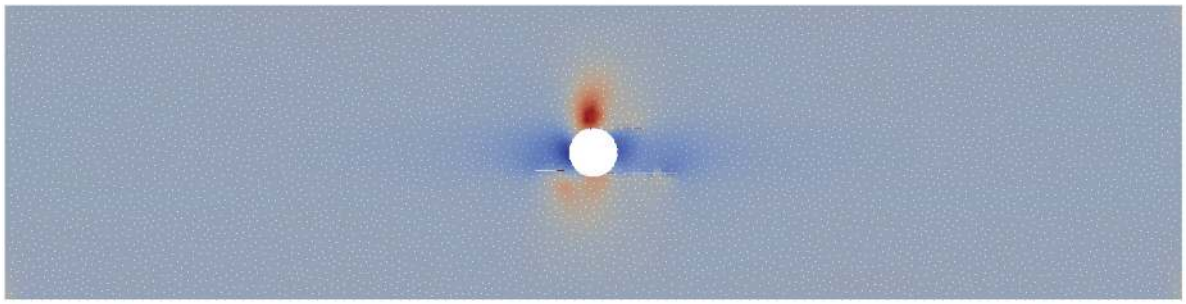


(b) Dilatational strain invariant

Figure 6.11: Damage prior to the onset of failure for the 0 degree lamina

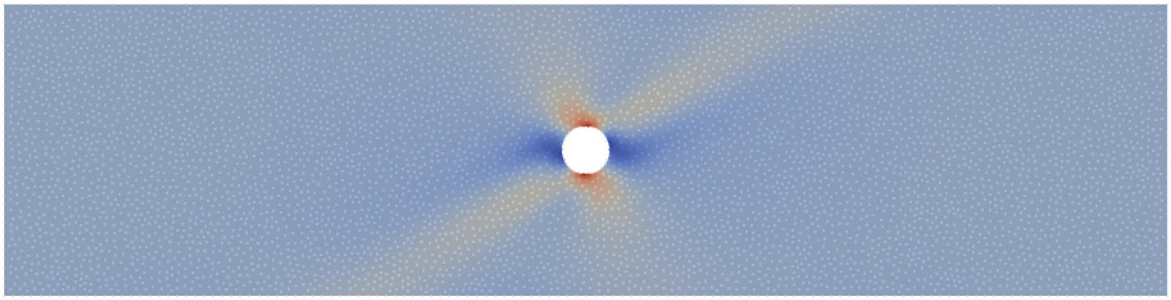


(a) Distortional strain invariant

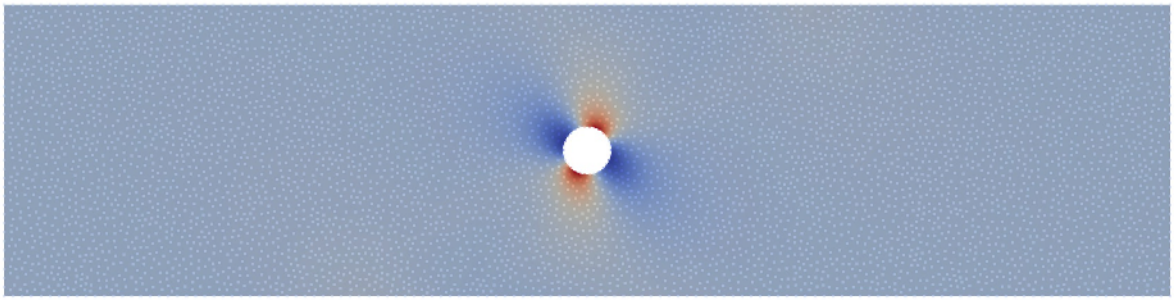


(b) Dilatational strain invariant

Figure 6.12: Damage after failure for the 0 degree lamina



(a) Distortional strain invariant



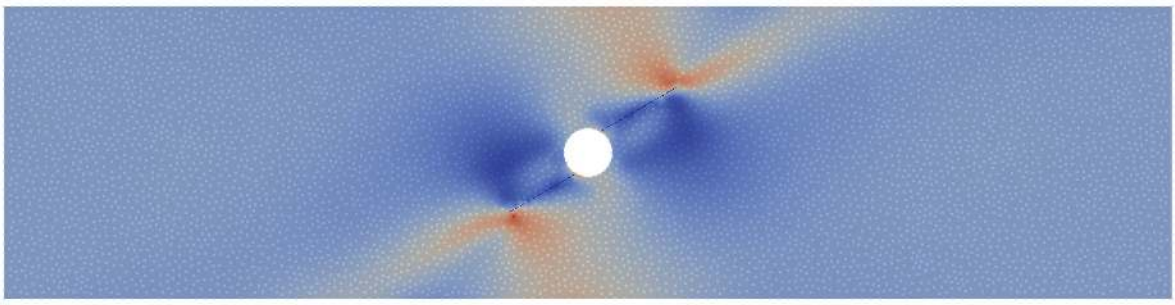
(b) Dilatational strain invariant

Figure 6.13: Damage prior to the onset of failure for the 30 degree lamina

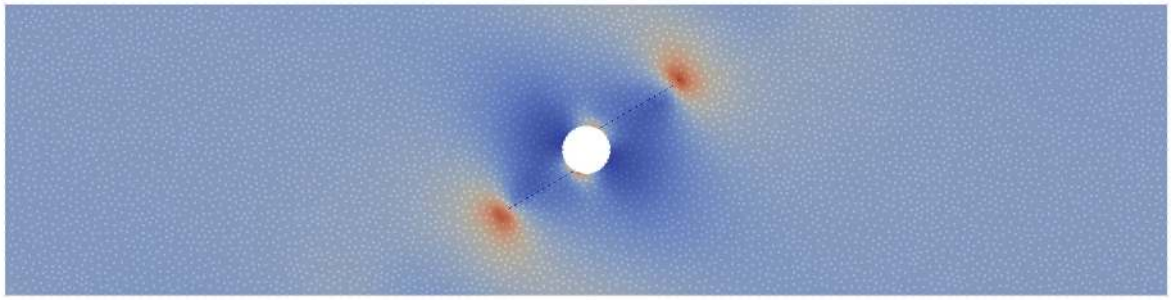
Since the failure is in the matrix phase the crack is force to propagate along the fiber direction. The resulting crack path and for both distortional and dilatational is shown in Figures 6.14a and 6.14b respectively.

6.3.3 90 Degree Lamina

For the configuration with the ninety degree fiber orientation, the failure predicted by Onset theory states the failure occurs in the matrix phase. The distortional and dilatational damage is depicted prior to the first failure occurring in Figures 6.15a and 6.15b respectively. Since the failure is in the matrix phase the crack is force to propagate along the fiber direction, which in this case corresponds the vertical axis. The resulting crack path and for both distortional and dilatational is shown in Figures 6.16a and 6.16b respectively.

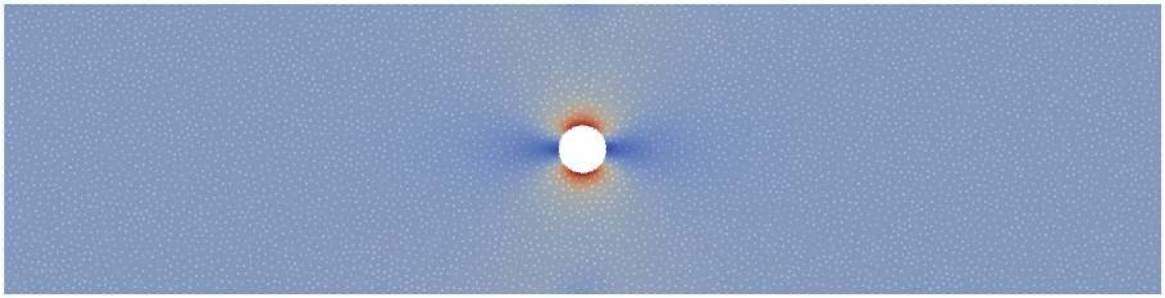


(a) Distortional strain invariant

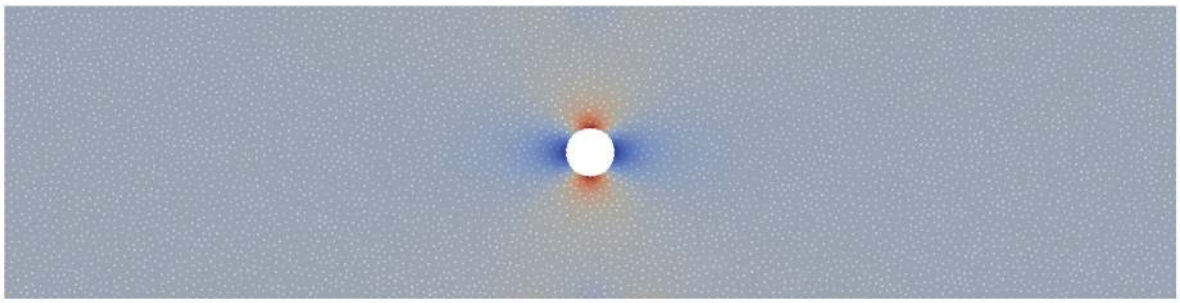


(b) Dilatational strain invariant

Figure 6.14: Damage after failure for the 30 degree lamina

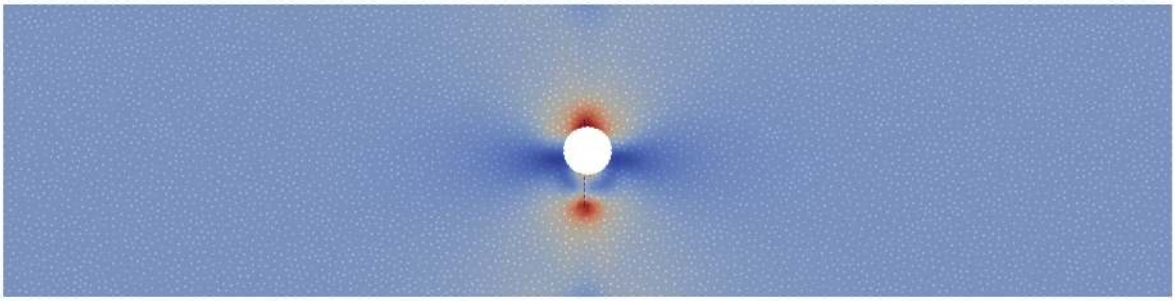


(a) Distortional strain invariant

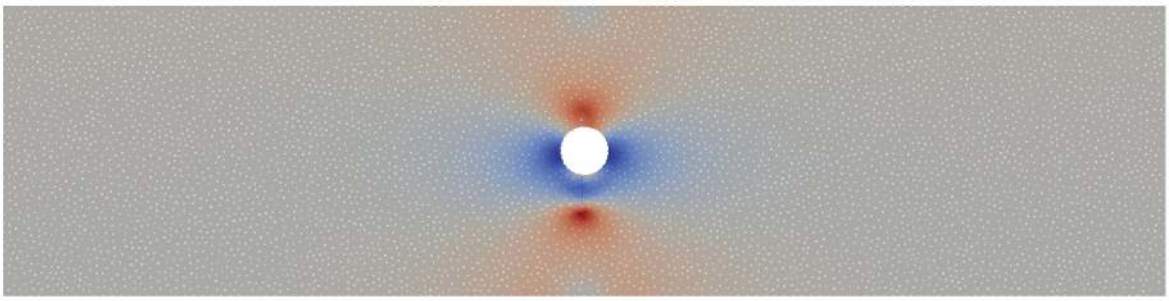


(b) Dilatational strain invariant

Figure 6.15: Damage prior to the onset of failure for the 90 degree lamina



(a) Distortional strain invariant



(b) Dilatational strain invariant

Figure 6.16: Damage after failure for the 90 degree lamina

6.4 Discussion

The objective of this chapter is to illustrate the application of the simulated annealing algorithm in the context of modeling fracture in laminated composites in a physically consistent manner. Using the approach in [59] as a foundation, a modified version of the original PSCA was developed. Unlike the original PSCA where existing particles were moved to align with the fiber direction, the modified version creates new filaments at the desired location. This eliminates the need for dense discretizations. This is made possible with the simulated annealing algorithm and ability to determine particle attributes, such as the size of the support domains. The effectiveness of this approach was illustrated with three examples. The correctness of these results with respect to experimental data was beyond the

scope of this work. However, the results do agree with previous numerical studies performed by Simkins *et al.* Further validation with experimental results would require tuning of the material properties and failure parameters in accordance with laboratory tests. It should be noted that the actual positions of the cracks are impossible to accurately determine, at least in a continuum model, so the goal of modeling fracture is not to state the exact geometry of the fracture. Instead, the numerically generated fracture paths should capture the complex physical phenomena occurring at the sub-macroscopic level. While this section focused solely on modeling fracture in laminated composites, the applicability of the approach presented is not limited to these materials. For an isotropic material where there is no need to restrict fracture paths along specific directions, the ability to insert new particles such that the quality of the discretization is not compromised could be advantageous. For instance, fracture algorithms such as those described in [52, 58, 59] which restrict the crack paths to existing particles require fine discretizations. This is necessary to avoid a single crack resulting in cascading failure occurring due to the creation of an excessively large crack. However, using techniques presented here, the coarse discretization could be used by introducing new particles via the simulated annealing algorithm to control the crack size while still maintaining the crack path to coincide with particles. The ability to introduce new particles to the discretization could also be used to better exploit failure models. Many failure models depend upon derived quantities such as stresses or strains to dictate when failure has occurred. These values are not directly available at the particles, instead they are computed at the integration points and projected to the particles. Instead of projecting these quantities to the particles to determine failure, the integration points themselves can be used to form new cracks. To adapt this to existing fracture algorithms requiring cracks to coincide with parti-

cles, new particles could be introduced at the location corresponding to the highest damage using the techniques presented here, such that an admissible discretization is maintained.

CHAPTER 7: CONCLUSION

While FEM has proven to be an invaluable tool for engineers, certain classes of problems prove challenging and may be better addressed using a meshfree method. However, the use of meshfree methods is hindered by several practical limitations. The computational inefficienies of meshfree methods related to determining adjacency information and meshfree domain discretization were addressed in this work.

The formulation of the Reproducing Kernel Particle meshfree method was reviewed. Following this development, challenges stemming from unique characteristics of the approximation functions were discussed along with existing techniques to address them.

The computational expense associated with determining adjacency information presents a performance bottleneck for meshfree methods. The present work defined the three adjacency queries that commonly arise when employing a meshfree method. An overview of several data structures was provided and the approaches to applying these within the context of meshfree methods were discussed. In addition to the discussion on existing data structures, a new data structure was presented with the associated algorithms for construction, searching, and dynamic insertion. The numerical results show the grid data structure is best both in memory and speed for Case 2 searches. Alternatively, of the Case 1 structures, the newly proposed Support Tree demonstrated strong performance in both memory usage and search speed, and scaled well with increasing problem sizes.

An approach to address the challenge of constructing a particle distribution that is suitable for use with a meshfree method was developed. This included techniques for determining the spatial positions of the particles as well as particle attributes necessary for constructing the meshfree approximation functions. The spatial positions of the particles were determined by a two step procedure. The first stage involved transforming the boundary representation into a volumetric set of points. The use of the tree structure allowed for sharp features to be captured without requiring the entire domain to be highly refined. Following the initial point generation, the second stage involved moving the points such that the points are well-distributed and cover the entire domain. This optimal distribution is determined as the configuration that minimizes the total potential energy of the system while constraining the points to stay within the boundary. This constrained optimization problem was solved using a stochastic approach known as simulated annealing. The simulated annealing algorithm allowed for easy application of the constraints at the cost of a high computational expense. To alleviate this computational expense, the spatial partitioning structures were utilized.

The concept of domain discretization was applied to refinement of an existing discretization. Employing the discretization technique with meshfree approximants in the solution of a boundary value problem presented an integration challenge. In meshfree methods, the newly added nodes do not come with additional integration points as the newly added elements do in FEM. This integration challenge was addressed with a hierarchical grid technique. This technique allowed for an adaptive integration scheme that also reduces the integration error that occurs due to support domain and integration cell misalignment. The domain refinement paired with the hierarchical grid integration technique was shown to perform well in convergence studies. In addition to the application of the discretization technique

to h-refinement, its application towards modeling physically consistent fracture of laminated composites was demonstrated. A modified version of the Particle Splitting Crack Algorithm was proposed. The difficulties associated with aligning particles along fiber directions were eliminated by a dynamic insertion routine. This routine proved effective in simulations of fracture in laminated composites.

7.1 Recommendations for Future Research

While this work addresses several deficiencies of meshfree methods, there remains opportunity for improvement. With regard to the computational expense associated with the adjacency queries in meshfree methods, the study of heuristics and build factors in the construction of the acceleration data structures warrants further work. The stochastic particle placement algorithm used in the domain discretization, h-refinement, and fracture modeling could be enhanced. In order to develop a more robust algorithm, a thorough investigation into the termination condition, particle attributes, and cooling schedule as they pertain to the stochastic particle placement algorithm is an opportunity for further research.

REFERENCES

[1] N. R. Aluru. A point collocation method based on reproducing kernel approximations. *International Journal for Numerical Methods in Engineering*, 47(6):1083–1121, 2000.

[2] S. N. Atluri and T. Zhu. A new meshless local petrov-galerkin (mlpg) approach in computational mechanics. *Computational Mechanics*, 22(2):117–127, 1998.

[3] I. BABUSKA and J. M. MELENK. The partition of unity method. *International Journal for Numerical Methods in Engineering*, 40(4):727–758, 1997.

[4] Stephen Beissel and Ted Belytschko. Nodal integration of the element-free galerkin method. *Computer Methods in Applied Mechanics and Engineering*, 139(1):49 – 74, 1996.

[5] T. Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45:601 – 620, 1999.

[6] T. Belytschko, Y. Krongauz, M. Fleming, D. Organ, and W.K. Liu. Smoothing and accelerated computations in the element free galerkin method. *Journal of computational and applied mathematics*, 74:111–126, 1996.

[7] T. Belytschko, W.K. Liu, and B. Moran. *Nonlinear Finite Elements for Continua and Structures*. Nonlinear Finite Elements for Continua and Structures. Wiley, 2000.

[8] T Belytschko, Y.Y. Liu, and L. Gu. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering*, 37:229–256, 1994.

[9] T. Belytschko, D. Organ, and Y. Krongauz. A coupled finite element-element-free galerkin method. *Computational Mechanics*, 17(3):186–195, 1995.

[10] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[11] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry*. Springer-Verlag Berlin Heidelberg, 2008.

[12] A. Bowyer. Computing dirichlet tessellations*. *The Computer Journal*, 24(2):162–166, 1981.

[13] David L. Buchanan, Jonathan H. Gosse, Jeffrey A. Wollschlager, Andrew Ritchey, and R. Byron Pipes. Micromechanical enhancement of the macroscopic strain state for advanced composite materials. *Composites Science and Technology*, 69(Experimental Techniques and Design in Composite Materials (ETDCM8) with Regular Papers):1974 – 1978, 2009.

[14] J.M. Caruthers and G.A. Medvedev. Nonlinear viscoelastic behavior of glassy polymers and its effect on the onset of irreversible deformation of the matrix resin in continuous fiber composites. In *ICCM International Conferences on Composite Materials*, 01 2009.

[15] Gary Chartrand. *Introductory Graph Theory*. Dover, New York, 1985.

[16] Jiun-Shyan Chen, Michael Hillman, and Marcus Rüter. An arbitrary order variationally consistent integration for galerkin meshfree methods. *International Journal for Numerical Methods in Engineering*, 95(5):387–418, 2013.

[17] Jiun-Shyan Chen, Cheng-Tang Wu, Sangpil Yoon, and Yang You. A stabilized conforming nodal integration for galerkin mesh-free methods. *International Journal for Numerical Methods in Engineering*, 50:435–466, 2001.

[18] S. De and K. J. Bathe. The method of finite spheres. *Computational Mechanics*, 25(4):329–345, Apr 2000.

[19] J. Dolbow and T. Belytschko. Numerical integration of the Galerkin weak from in meshfree methods. *Computational mechanics*, 23, 1999.

[20] Qiang Du, Max Gunzburger, and Lili Ju. Meshfree, probabilistic determination of point sets and support regions for meshless computing. *Computer Methods in Applied Mechanics and Engineering*, 191(13-14):1349 – 1366, 2002.

[21] C.Armando Duarte and J.Tinsley Oden. An h-p adaptive method using clouds. *Computer Methods in Applied Mechanics and Engineering*, 139(1-4):237 – 262, 1996.

[22] Sonia Fernández-Méndez and Antonio Huerta. Imposing essential boundary conditions in mesh-free methods. *Computer methods in applied mechanics and engineering*, 193(12):1257–1275, 2004.

[23] M. Fleming, Y. A. Chu, B. Moran, and T. Belytschko. Enriched element-free galerkin methods for crack tip fields. *International Journal for Numerical Methods in Engineering*, 40:1483 – 1504, 1997.

[24] Hermann G. Fries, Thomas-Peterand Matthies. Classification and overview of meshfree methods. *Informatik-Berichte der Technischen Universität Braunschweig*, 2003-03, 2004.

[25] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. *SIGGRAPH Comput. Graph.*, 14(3):124–133, July 1980.

[26] Akira Fujimoto and Kansei Iwata. *Accelerated Ray Tracing*, pages 41–65. Springer Japan, Tokyo, 1985.

[27] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389, 1977.

[28] Jonathan H. Gosse and Stephen Christenson. Strain invariant failure criteria for polymers in compostie materials. *AIAA-2001-1184*, 2001.

[29] M. Griebel and M. A. Schweitzer. A particle-partition of unity method-part ii: Efficient cover construction and reliable integration. *SIAM Journal of Scientific Computation*, 23:1655–1682, 2002.

[30] M. A. Griebel, M.and Schweitzer. *Geometric Analysis and Nonlinear Partial Differential Equations*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[31] Michael Griebel and Marc Alexander Schweitzer. A particle-partition of unity method for the solution of elliptic, parabolic, and hyperbolic pdes. *SIAM Journal on Scientific Computing*, 22(3):853 – 890, 2000.

[32] Antonio Huerta and Sonia Fernández-Méndez. Enrichment and coupling of the finite element and meshless methods. *International Journal for Numerical Methods in Engineering*, 48(11):1615–1636, 2000.

[33] Autar K. Kaw. *Mechanics of Composite Materials*. CRC Press, November 2005.

[34] M. S. Klaas, O.and Shephard. Automatic generation of octree-based three-dimensional discretizations for partition of unity methods. *Computational Mechanics*, 25(2):296–304, Mar 2000.

[35] P. Krysl and T. Belytschko. Element-free Galerkin method: convergence of the continuous and discontinuous shape function. *Computer Methods in Applied Mechanics and Engineering*, 148:257–277, 1996.

[36] Petr Krysl and Ted Belytschko. Esflib: A library to compute the element free galerkin shape functions, 1999.

[37] C.L. Lawson. Software for {C1} surface interpolation. In John R. Rice, editor, *Mathematical Software*, pages 161 – 194. Academic Press, 1977.

[38] J.E. Lennard-Jones. On the determination of molecular fields. —ii. from the equation of state of a gas. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 106(738):463–477, 1924.

[39] Shaofan Li and Wing Kam Liu. Meshfree and particle methods and their applications. *Applied Mechanics Review*, 55(1):1–34, 2002.

[40] Shaofan Li and Wing Kam Liu. *Meshfree particle methods*. Springer, 2004.

[41] Shaofan Li, Dong Qian, Wing kam Liu, and Ted Belytschko. A meshfree contact-detection algorithm. *Computer Methods in Applied Mechanics and Engineering*, 190:3271–3292, 2001.

[42] Xiang-Yang Li, Shang-Hua Teng, and Alper ÃIJngÃűr. Biting: advancing front meets sphere packing. *International Journal for Numerical Methods in Engineering*, 49(1-2):61–81, 2000.

[43] G.R. Liu. *Meshfree methods Moving beyond the finite element method*. CRC press, 2009.

[44] W.K. Liu, S. Jun, S. Li, J. Adee, and T. Belytschko. Reproducing kernel particle methods for structural dynamics. *International Journal for Numerical Methods in Engineering*, 38:1655–1679, 1995.

[45] W.K. Liu, S. Jun, and Y.F. Zhang. Reproducing kernel particle methods. *International Journal for Numerical Methods in Fluids*, 20:1081–1106, 1995.

[46] W.K. Liu, S. Li, and T. Belytschko. Moving least square reproducing kernel method. part i: Methodology and convergence. *Computer Methods in Applied Mechanics and Engineering*, 143:422–453, 1997.

[47] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, dec 1977.

[48] N. Moes, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46:131–150, 1999.

[49] Per olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM Review*, 46:2004, 2004.

[50] E. ONATE, S. IDELSOHN, O. C. ZIENKIEWICZ, and R. L. TAYLOR. A finite point method in computational mechanics. applications to convective transport and fluid flow. *International Journal for Numerical Methods in Engineering*, 39(22):3839–3866, 1996.

[51] D. Organ, M. Fleming, T. Terry, and T. Belytschko. Continuous meshless approximations for nonconvex bodies by diffraction and transparency. *Computational Mechanics*, pages 225 – 235, 1996.

[52] Bo Ren and Shaofan Li. Modeling and simulation of large-scale ductile fracture in plates and shells. *International Journal of Solids and Structures*, 49(18):2373 – 2393, 2012.

[53] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH Comput. Graph.*, 14(3):110–116, July 1980.

[54] Marc Alexander Schweitzer. Stable enrichment and local preconditioning in the particle-partition of unity method. *Numerische Mathematik*, 118(1):137–170, 2011.

[55] Irving H. Shames and Francis A. Cozzarelli. *Elastic and inelastic stress analysis*. Taylor & Francis, 1997.

[56] Mark S Shephard, Nabil AB Yehia, Gary S Burd, and Theodore J Weidner. Automatic crack propagation tracking. *Computers & Structures*, 20(1):211–223, 1985.

[57] Kenji Shimada and David C. Gossard. Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications*, SMA '95, pages 409–419, New York, NY, USA, 1995. ACM.

[58] D. C. Simkins and S Li. Meshfree simulations of thermo-mechanical ductile fracture. *Computational Mechanics*, 38:235–249, 2006.

[59] Jr. Simkins, Daniel C., Nathan Collier, and Joseph B. Alford. Meshfree modeling in laminated composites. In Michael Griebel and Marc Alexander Schweitzer, editors, *Meshfree Methods for Partial Differential Equations VI*, volume 89 of *Lecture Notes in Computational Science and Engineering*, pages 221–233. Springer Berlin Heidelberg, 2013.

[60] T.E. Tay, S. H. N. Tan, V. B.c C. Tan, and Jonathan H. Gosse. Damage progression by the element-failure method (EFM) and strain invariant failure theory (SIFT). *Composties Science and Technology*, 65:935–944, 2005.

[61] S. Timoshenko and J.N. Goodier. *Theory of Elasticity, by S. Timoshenko and J.N. Goodier, ... 2nd Edition.* McGraw-Hill Book Company, 1951.

[62] Gregory J. Wagner and Wing Kam Liu. Hierarchical enrichment for bridging scales and mesh-free boundary conditions. *International Journal for Numerical Methods in Engineering*, 50(3):507–524, 1 2001.

[63] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Trans. Graph.*, 25(3):485–493, July 2006.

[64] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes*. *The Computer Journal*, 24(2):167–172, 1981.

[65] Holger Wendland. *Scattered data approximation*, volume 17. Cambridge university press, 2004.

[66] Xiang yang Li, Shang-Hua Teng, Alper Ungor, and Alper Ungor. Point placement for meshless methods using sphere packing and advancing front methods. In *International conference on Computational Engineering Science*, 2000.

[67] Hanzhou Zhang and Andrei V. Smirnov. Node placement for triangular mesh generation by monte carlo simulation. *International Journal for Numerical Methods in Engineering*, 64(7):973–989, 2005.

[68] A. L. Zheleznyakova and S. T. Surzhikov. Molecular dynamics-based unstructured grid generation method for aerodynamic applications. *Computer Physics Communications*, 184:2711–2727, December 2013.

[69] T. Zhu and S. N. Atluri. A modified collocation method and a penalty formulation for enforcing the essential boundary conditions in the element free galerkin method. *Computational Mechanics*, 21(3):211–222, 1998.

[70] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method*, volume 1. Butterworth-Heinemann, Oxford, 5 edition, 2000.

APPENDIX A: COPYRIGHT PERMISSIONS

A.1 Copyright Permission Chapter 3

Reprinted by permission from Springer Customer Service Centre GmbH: Springer Nature Computational Mechanics Efficient searching in meshfree methods, James Olliff, Brad Alford, Daniel C. Simkins, 2018, advance online publication, 1 January 2018 https://doi.org/10.1007/s00466-018-1574-9

SPRINGER NATURE LICENSE TERMS AND CONDITIONS

Jun 20, 2018

This Agreement between University of South Florida ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	4355900580323
License date	May 25, 2018
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Computational Mechanics
Licensed Content Title	Efficient searching in meshfree methods
Licensed Content Author	James Olliff, Brad Alford, Daniel C. Simkins
Licensed Content Date	Jan 1, 2018
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	<501
Author of this Springer Nature content	yes
Title	Efficient Adjacency Queries and Dynamic Refinement for Meshfree Methods with Applications to Explicit Fracture Modeling
Instructor name	James Olliff
Institution name	University of South Florida
Expected presentation date	May 2018