Delft University of Technology

Masters Thesis - Final Report

An Efficient 3D-model for the Micro-Scale Analysis of Geometrically Imperfect Fibre-Reinforced Composites

Author:
Anirudh BHARATH
(5098947)

Supervised By:
Dr. Boyang CHEN

June 16, 2022

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Definition
1/2/3-D	1/2/3-Dimensional
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CFRP	Carbon Fibre Reinforced Polymer
CUF	Carrera Unified Formulation
DOF(s)	Degree(s) of Freedom
EVP	Elastic Visco-Plastic
FE(M)	Finite Element (Modelling)
FN	Functional Nucleus
FRP	Fibre-Reinforced Polymer
HM	High Modulus
IGA	IsoGeometric Analysis
MC	Multi-Celled
MMC	Metal Matrix Composites
MRVE	Multiple-Fibre Representative Volume Element
NURBS	Non-Uniform Rational Basis Splines
PPS	PolyPhenylene Sulphide
RV	Representative Volume
RVE	Representative Volume Element
SC	Single-Celled
SME	Simplified Micromechanics Equations
UD	Uni-Directional
XFEM	Extended Finite Element Method

Acknowledgements

I would like to thank my thesis supervisor Dr. Boyang Chen for his unwavering support and encouragement through the course of this thesis. Your guidance through the difficult moments and your impeccable judgement were instrumental in helping me complete this thesis, and your patience, reassurance, and motivation were invaluable both in times of strife and success. I would also like to thank the team of Professors at the Faculty of Aerospace Engineering for maintaining such a high standard of education and from whom I have gained a lifelong appreciation and understanding of Aircraft Structures and Design.

I am also grateful for my incredible group of friends who are always present for me, have provided me with abundant technical advice and knowledge, have helped me take crucial practical and professional decisions, and most importantly, have helped me maintain a positive mentality. They have been a source of joy, enthusiasm, inspiration and motivation on a daily basis and are instrumental in helping me achieve my goals. Particularly, I would like to thank my close friend Anant Chandra who has been a staunch companion and my closest ally through our time at Delft, with whom I have conquered many obstacles and celebrated many victories.

Finally, I thank my parents and my brother for helping me realise my childhood dreams with abundant love, encouragement, and unconditional support at all times.

Anirudh Bharath
Delft, June 2022

Abstract

This work develops a stand-alone Finite Element-based Unit Cell for the modelling of Fibre Reinforced Composites, capable of capturing 3D stress states and geometrical imperfections at the micro-scale. The project considers different beam and 3D elements for this purpose and draws comparisons between them based on computational efficiency, accuracy, and ease of implementation. A new setup is proposed to greatly reduce computational effort and facilitate user interaction and parametrisation by drawing from some of the most promising concepts identified from the present literature. The thesis presents such an FE model which can model a Multi-Fibre-matrix Unit Cell including codes for the generation and perturbation of the geometry, multi-scale meshing, solving, and post-processing. Some typical results for Micro-scale Unit Cells are presented to validate and demonstrate the capabilities of the model, and avenues for improvement and development of this concept are also detailed.

1 Introduction

Composites materials are used in critical structural components in the aerospace and allied industries. They are fast replacing conventional metal-based designs due to their superior strength-to-weight ratio and the ability to tailor the material properties for different load cases. In such high performance engineering fields, Fibre-Reinforced Polymers (FRPs) are the most prevalent [27] since they provide the most superior advantage in the directional properties, which can be tailored to resist the primary load components with high efficiency. The reinforcement materials commonly found in the FRPs include Glass Fibres, Carbon Fibres, Naturally occurring fibrous materials, or Aramid Fibres. The matrix material is usually a Thermoplastic or Thermoset polymer, and these materials are used in different combinations based on the desired properties.

FRPs can be classified into several categories [28], which can be broadly placed under 'short' and 'long' fibre composites. For critical structures and applications, long fibre composites are generally preferred due to their superior stiffness and strength. Long fibre composites can further be classified into Uni-directional (UD) and Woven Fabrics. Woven fibres are very commonly used in secondary structural elements due to their ease of manufacturing and layup. Woven fabrics also reduce the anisotropy of the structure by providing comparable mechanical properties in the primary orthogonal directions since they are made up of fibres running in mutually perpendicular (warp and weft) directions. This is beneficial for structures dealing with several different load-cases, and with relatively high secondary stresses. UD composites provide the opportunity to tailor the material strength in a single direction with the greatest efficiency, and are used in primary structural members in the aerospace industries.

Composite materials are expensive to manufacture as compared to the metal counterparts, making experimental study tedious and destructive. There has hence been a large effort in the past century to study the composites through mathematical formulation and simulation. However, due to their inherent non-homogeneity, they are difficult to model mathematically for the purpose of structural analyses. Composite structures also experience a large scatter in material properties due to inconsistencies in manufacturing processes (macro scale) and also minor defects and variation of the structure in the micro scale. Current structural models reduce the material properties to the macro scale by semi-empirical homogenisation processes which fail to capture these variations. Hence, for the purpose of an accurate simulation of the mechanical properties and damage tolerance of composites, it is essential to use a multi-scale modelling method which can capture the stress state at the micro, meso, and macro scale.

In this thesis, an attempt has been made to identify the different methodologies used to model Fibre Reinforced Composites at the micro and meso length scales, for the purpose of capturing the effect of the micro-level inhomogeneity on the behaviour of the composite. A brief definition of the most practical method of micro-scale study is presented in Chapter 2 - The Unit Cell, followed by an account of the early evolution of the Unit Cell model over the late 1900s in Chapter 3. The current state of composite modelling at the micro and meso scale is depicted in Chapter 4. Several conclusions are drawn based on the state of the literature in Chapter 5, with an aim to identify the existing gaps in current literature and posit the possible benefits of previously explored models which have scope for further expansion. In Chapter 6, the relevant points from the summary are condensed into essential requirements for a new mathematical model, and a few solution paths are explored. The Final solution developed in this thesis is presented in detail in Chapter 7. The various Results and figures validating this model are shown in Chapter 8 The study concludes with a revision of the novel characteristics of the current model and different options for future considerations in Chapter 9.

2 The Unit Cell

To fully understand the behaviour of composites under different loading conditions, it is essential to study the microscopic interactions between the discrete phases of the composite. The heterogeneity arising from the different phases results in microscopic load transfer mechanisms, areas of localised stress concentration and damage, and stochastic variations which cannot be captured consistently with analytical modelling. For this purpose, the Unit Cell can be used to study these micro-scale interactions, by capturing the distinct phases present in the material.

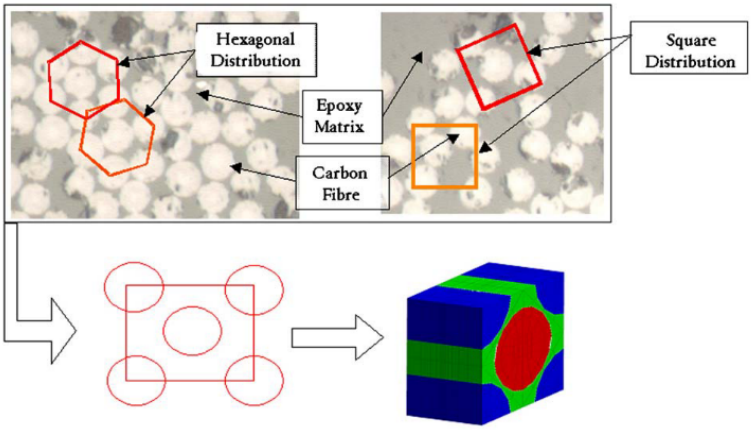

Figure 1: *A standard single fibre unit cell for UD Fibre reinforced composites.* [1]

The Unit Cell can be understood as the smallest possible periodically occurring structural domain that can be composed to model an entire material or a large area of relevance within the material through regular arrangements of the Unit Cell. The Unit Cell for a fibre reinforced composite, in its simplest form, contains a single fibre and matrix domain, possibly extended to capture symmetric representations of neighbouring fibres (Fig. 1). While theoretically such a unit cell can be used to model the entire composite, it would require immense computational effort due to the severely reduced scale employed in modelling a single fibre domain. For the purpose of capturing the behaviour of the complete macroscopic structure within feasible bounds of scale, it is possible to define a sub-domain of the structure, known as the Representative Volume Element (RVE), where the mechanical properties captured over this defined domain remains representative of the material over a bulk scale [29]. The Representative Volume Element does not necessarily capture all the material properties of the bulk material, but is rather intended for the study of specific behaviours of the material which are relevant to the critical aspects of the composite structure, and which rely significantly on the effect of the microstructure of the composite. For this reason, an RVE can range from consisting of a single Unit Cell, to a large domain of fibres with several unit cells. The RVE can often include realistic variations in properties which cannot be captured by a periodic domain such as stochastic variations and distributions of material, defects, and properties. A detailed description of the Unit Cell and RVE and their applications can be found in the work by Li and Sitnikova [29].

The following sections depict the development of the Unit Cells over different Representative Volumes based on the properties to be derived from them, including stress states, propagation and accumulation of damage, load transfer, defects, and stochastic variation, both in two and three dimensional formulations. While different approaches have been used to formulate and study the unit cell, the Finite Element Modelling (FEM) approach has been overwhelmingly preferred due to its flexibility in capturing discrete phases and properties, and the studies presented herein are primarily restricted to FEM-based formulations.

3 Unit cells before 2000

Composite unit cells have been studied relatively recently in the history of composite mechanical formulations. Particularly, for Fibre reinforced composites, initial studies focus on the interaction of a single fibre, or localised periodic groups of fibres, with the surrounding matrix. The studies are mostly restricted to square and hexagonal packing of fibres over the cross-section of the composite, assuming them to be uniformly spaced, for the purpose of exploiting the symmetry and periodicity to choose the smallest possible representative domain. The objective of this period of research is to analyse the different possible unit cells by identifying the symmetries present, refining the boundary conditions on such a domain for the accurate transposition of macroscopic loading and strains to the micro-scale, and to motivate the efficacy of the simple unit cell idealisations and assumptions, by comparing their mechanical response to the global or effective material properties of the macro-level composite.

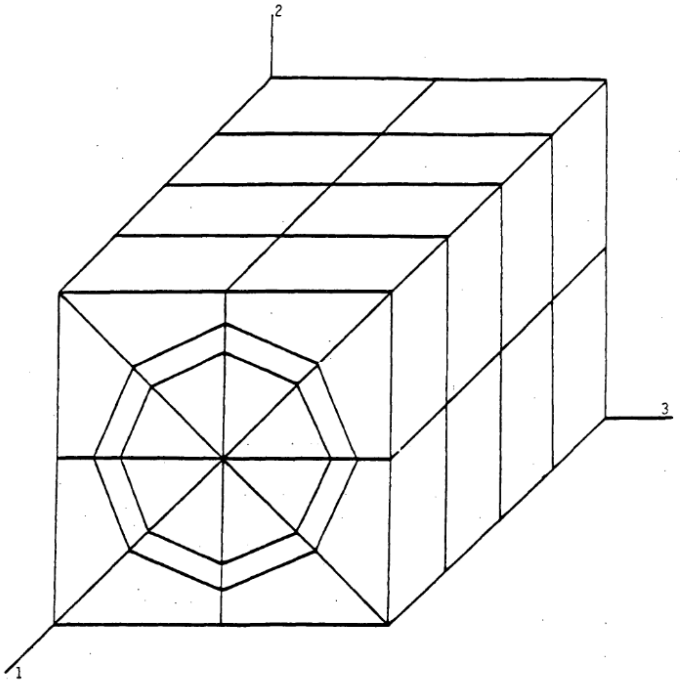

Figure 2: *Discretisation of the fibre matrix domain with 3D elements* [2]

One of the earliest developments of the unit cell for a Fibre reinforced composite is conducted by NASA [2]. This work is a comprehensive introduction to the study of micromechanics using an RVE of 1-9 fibres for Uni-directional (UD) FRPs. The work used 3D models for a single fibre and matrix configuration. A combination of wedge and prism elements are used to model the fibre and matrix, with refinement on the interfacial region (Fig. 2). Two different mesh sizes are considered, with a finer mesh preferred for this application. For the purpose of getting more comprehensive data on effective transverse properties involving the interaction of multiple fibres, a 9-cell model is created by joining 9 of the single cell models. The authors note the significant increase in computational effort required for such a 9-cell model with the present elements (Fig. 3). The elements used are capable of only linear interpolation (6-noded wedge and 8-noded brick). It was found that the single cell model provides very good agreement with the multi-cell model and experimental data for most of the hygral, thermal, and mechanical properties for isotropic-fibre and isotropic-matrix composites. The multi-cell model is used to accurately predict the properties of orthotropic-fibre composites, as well as some other transverse shear and Poisson ratios. The authors recommend this element for the purpose

of accurate modelling of stress-fields, damage analysis, and also for further expansion with random fibre orientations, inherent defects, and other stochastic variations observed in experimental studies. However, the authors note the significant investment of computing time in such a model, requiring the use of more efficient elements.

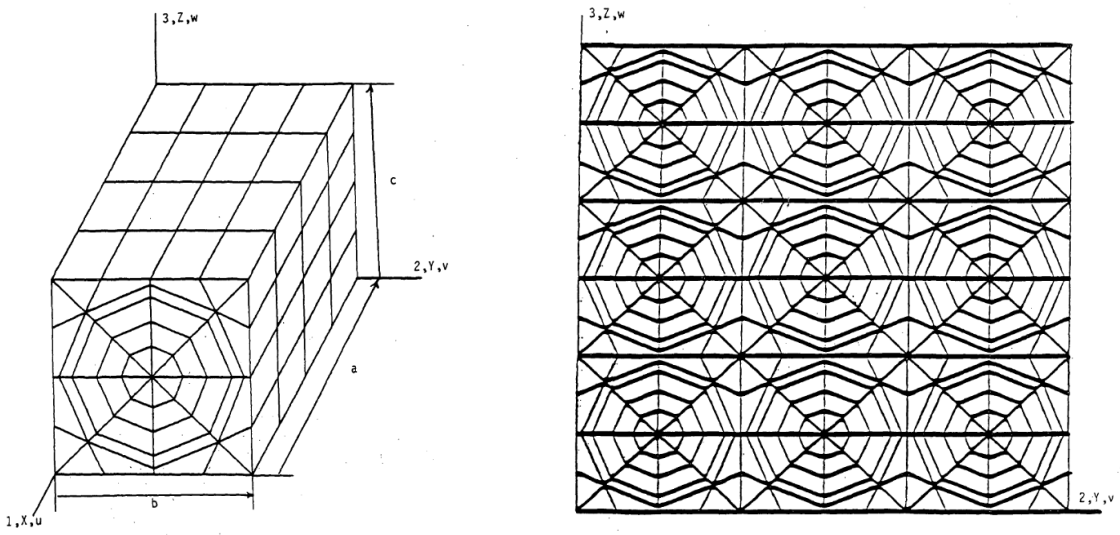

Figure 3: *Refined mesh for a) single fibre unit cell; b) RVE combining 9 unit cells* [2]

This work was one of the only early attempts at studying the micromechanics with a 3D unit cell. However, due to the high computational effort involved, a 2D representative element was preferred for the purpose of running several parametric studies efficiently. The 2D element is capable of capturing the periodicity of the fibre square and hexagonal fibre layouts, while focusing on the fibre-matrix interaction under transverse normal and shear stresses. Further, the necessity to study the failure modes of composites through micromechanical damage evolution was considered essential, and a 2D unit cell is capable of capturing such transverse damage propagation with refined meshes and parameterised local material properties.

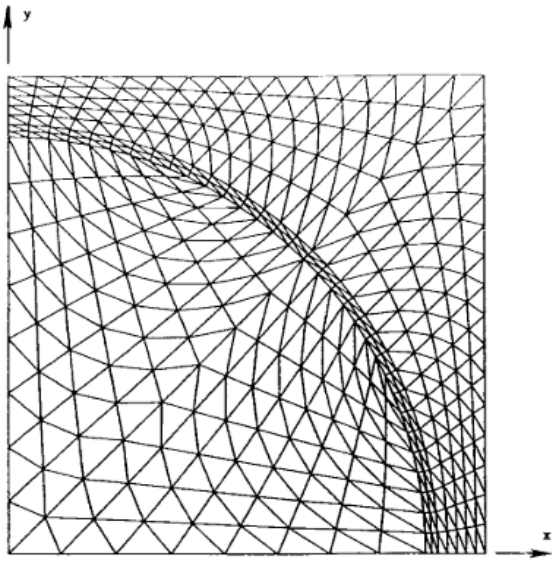

Figure 4: *A 2-dimensional unit cell with a quarter fibre and surrounding matrix* [3]

The use of such a 2D unit cell was shown by King et. al. [3]. This work uses a 2D quarter fibre unit cell for a UD FRP to study the yielding and propagation of crack growth along the fibre-matrix interface under longitudinal shear loading. The mesh consists of triangular elements which are refined around the interface region. It can be seen from Fig. 4 that it is possible to generate a more refined mesh for a 2D domain as compared to the 3D domain around the same era. The maximum normal stress failure criterion was used as it showed good agreement with experimental results. The effect of interfacial bond strength, matrix properties, and fibre volume fraction were studied on the failure of the fibre matrix interface. The results show good agreement with the experimental results, especially when considering the bond strength to be equivalent to the strength of the matrix. This showed the efficacy of a 2D quarter fibre unit cell in capturing fibre-matrix interface failure propagation (Fig. 5). This study can be further expanded to the 3D domain with better crack propagation and failure theories, such as cohesive elements or other damage models.

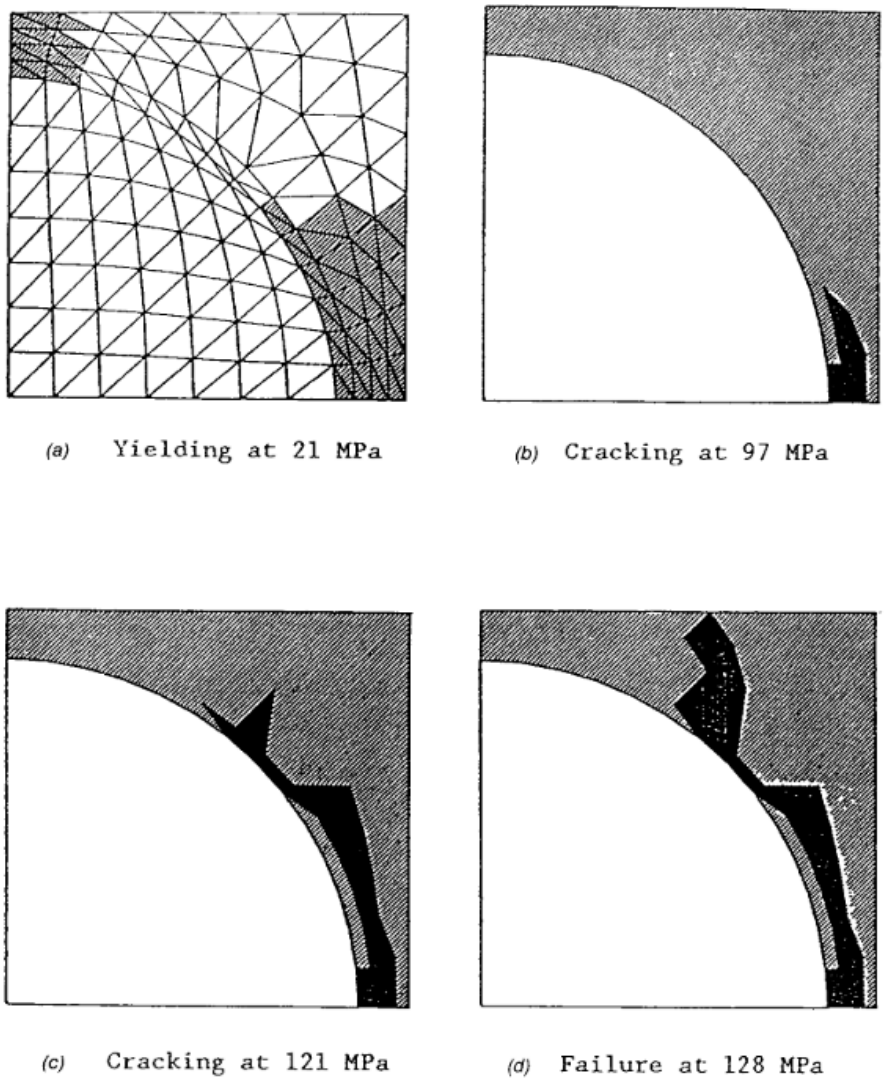

Figure 5: *An example of the ability to capture damage near the fibre matrix interface in the transverse direction* [3]

A similar unit cell and symmetry logic has been expanded to capture the effective material properties of different reinforcement types commonly used in Metal Matrix Composites (MMC) by Shen et. al. [4]. This paper considers different forms of reinforcements in composites including spherical, platelet, whisker, and cylindrical reinforcements, focusing on metal-matrix composites. Each type is modelled

with a 2D unit cell representing a cross-section along the length of the fibre (Fig. 6). The unit cell uses symmetry to model a quarter of the reinforcement geometry for computational efficiency. The authors are interested in studying the effect of the shape of the reinforcement and the distribution (using reinforcement volume fractions from 0-60 percent) against the elastic response of the composite. The effective elastic modulus is compared for the different reinforcements, and it is shown that the whisker (short fibre) reinforcement shows the most increase in rigidity. The results are validated against existing literature from numerical and analytical macro-simulations. This validated unit cell is then used to analyse the properties of materials with 2 distinct types (sizes) of reinforcements, particularly focusing on unit cylinder-type reinforcements. The unit cell is expanded to capture different configurations of the reinforcements. Further, the model is applied to analyse the material under thermal pre-stress, commonly associated with the manufacturing of MMCs. The micromechanics captures the reduction in material properties on thermally pre-stressed composites for cylindrical and spherical microstructures. The work also showed the ability of cracked or broken reinforcements in contributing to load transfer and hence in increasing the stiffness of a ductile matrix. The use of such a unit cell can be further expanded to capture long fibre composite behaviour, however it fails to capture 3D failure modes. Plastic strain is modelled in the matrix, and interfacial compliance is also used with this model, which could be beneficial in the study of long-fibre composites. The study also does not consider stochastic variation in the microstructure with random distribution of fibres, but can be expanded for this purpose.

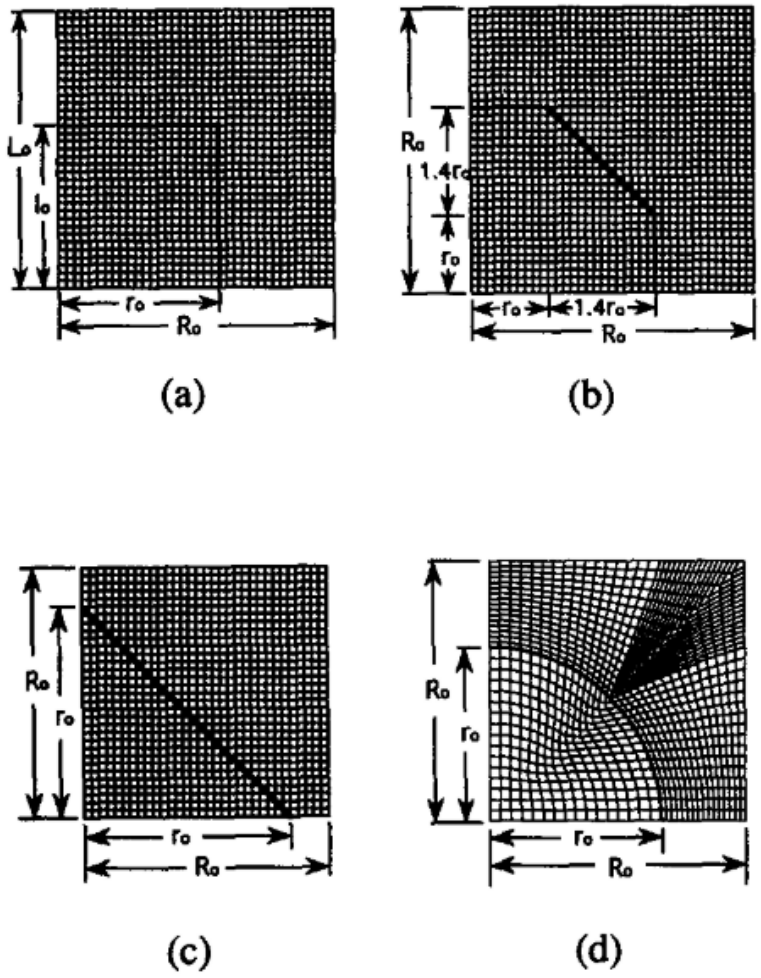

Figure 6: *Different reinforcements modelled in a 2D plane parallel to the reinforcement axis: a) cylinder; b) truncated cylinder; c) double cone; d) sphere* [4]

Apart from the study of damage evolution, more effort has been put into refining the Unit Cell to

more accurately represent the actual stress states and boundary conditions observed in reality. Sun et. al. [5] have improved on previous work in the literature by choosing the appropriate boundary conditions and unit cell geometry to capture both square and hexagonal layout of fibres (Fig. 7). All load cases studied were 2-dimensional in nature and it was possible to reduce the model to a 2D mesh, with the mechanics and boundary conditions accounting for the third direction. The model is used to obtain the effective material properties for a Boron-Aluminium composite as well as Graphite-Epoxy composite, which are then compared against macroscopic properties observed through experimental analysis. The model shows good agreement, however, the authors note that the wide scatter of data for graphite epoxy composites creates a difficulty in exactly replicating the results with a simple element. While both the 2D hexagonal and square layups can adequately predict the elastic properties of the material, they can be further expanded to the 3D regime involving stochastic variations to properly capture the micromechanics interaction and damage evolution of the composite.

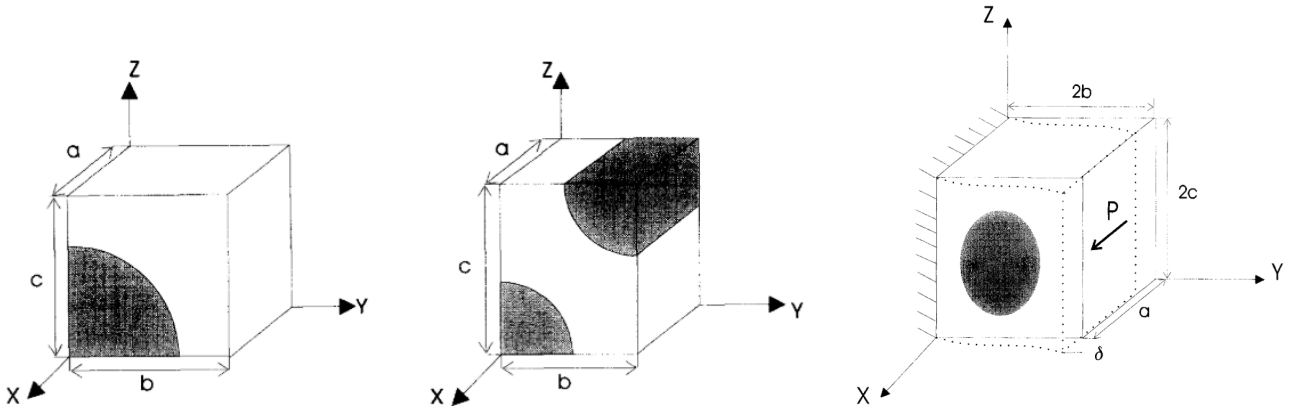

Figure 7: *Unit Cell for a) square packing under tensile load; b) hexagonal packing under tensile load; c) square packing under shear load* [5]

The shape of the unit cell was further examined by Shuguang Li [6] by studying the various symmetries existing in the single fibre unit cell model. While both square and hexagonal regular arrangements of fibres are considered through the cross-section, the author focuses on hexagonal packing and its symmetries as it was not found to be a well researched topic. The paper goes further to also present the boundary conditions arising from the symmetries for the unit cell. These symmetries can serve useful to reduce computational effort for the unit cell analysis and enable finer meshes. The paper arrives at an ideal 2D trapezoidal unit cell (Fig. 8) as the optimum for computing effective properties of the material, since it most fully utilises the symmetries present in the periodic arrangement of the fibres and can capture all the properties of the whole unit cell. The paper also presents formulations to express the mechanical response of the trapezoidal unit cells as effective macroscopic mechanical properties of the composite. However, the study is restricted to 2D cells and does not consider random distributions or damage modes. No actual parametric analysis has been considered here to show the efficacy of such unit cells against conventional models.

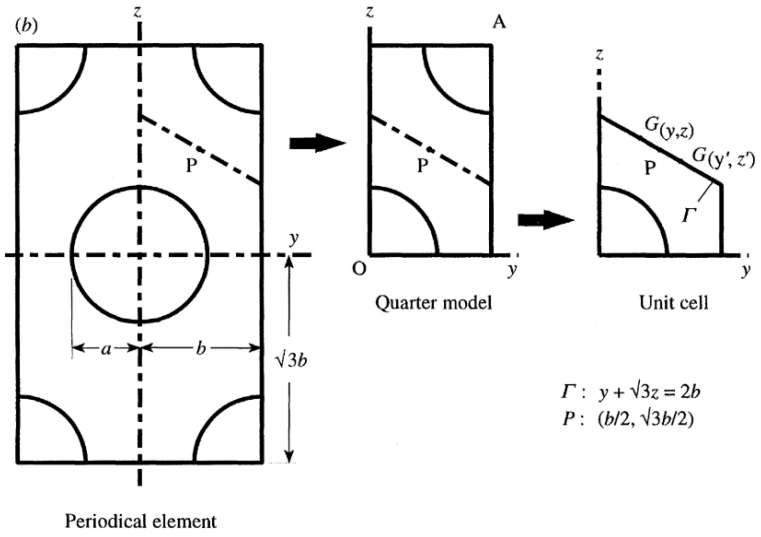

Figure 8: *Derivation of an optimum Unit cell from a standard periodical element for hexagonal packing of fibres* [6]

It is evident from the state of the literature so far that the focus of study before the 2000s in the field of Unit cells has been on 2-Dimensional layouts, with emphasis on obtaining the effective macroscopic mechanical properties of the material studied. While some focus has been drawn to the initiation and propagation of damage in the matrix and fibre regions of the composite, no attempts have been made to develop a comprehensive damage and plasticity model for the composite. One of the most evident trends through this span of literature has been the work involved to reduce the computational effort of the model, by simplifying and reducing the unit cell model with use of symmetry. This is due to the necessity of creating fine meshes to accurately capture the stress states in the micro-scale. Also, the amount of computational time required grows exponentially on introducing multi-scale finite element formulations to study the effect of micromechanical stresses on macroscopic geometry, which is the eventual intended application of such unit cells. This has not yet been attempted up to this point due to a lack in computational power. Apart from this, there is limited literature available on the use of multiple fibres in a unit cell or RVE. The micromechanics of load transfer between neighbouring fibres, and the inclusion of manufacturing voids and defects have hence not been considered. This also precludes the study of randomly oriented/distributed fibres in the cross-sectional plane, which can more accurately mimic realistic fibre distributions, as compared to the periodic formations used here. The next section depicts the improvements in these regimes in the coming years.

4 Recent Developments on the Unit Cell

To fully understand the behaviour of a composite structure, it is necessary to model the micro and macro scale simultaneously. The unit cell has been developed for this purpose through the late 1980s and 90s, as described in the previous section. Large improvements in computational power over the next two decades have allowed for the implementation of such a multi-scale system, albeit requiring significant computational effort. Several efficient multi-scale homogenisation techniques have been devised for efficient computation, and the unit cells are carefully selected for each unique case, so as to limit the domain of study to the area of interest. The unit cell can generate a complete picture of the stress states through a single fibre and the surrounding matrix, which is essential to understand the micromechanics of damage initiation and growth in the composite. Also, the behaviour of the composite material post-damage or in the presence of defects is possible using this approach, and has been a growing area of interest in recent years. The unit cell also allows for capturing micro-scale variations in the reinforcement geometry. There have already been attempts at modelling different regular reinforcement geometries such as platelets, short fibres, cylinders, and so on [4]. However, for the case of long fibre composites, it is also possible to consider different fibre trajectories, fibre distribution through the matrix, and random variations in such parameters as is commonly observed in experimental and production samples. For such cases, recent theories including Multiple Fibre Representative Volume Elements (MRVE) have grown more popular due to the need for capturing inter-fibre variations and the resulting change in load transfer.

In this section, the recent developments in Representative volume element and unit cell design and applications will be covered, with particular focus on 3-dimensional models. Both woven and straight fibre composites are shown, with intent to highlight the similarity between the unit cell approaches. Particular focus is has been shown to the study of damage in uni-directional composite and laminates, as it highlights the importance of a comprehensive unit cell.

4.1 Woven FRP unit cells

Woven composites are commonly used in structural applications due to their ease of handling and natural strength in primary and secondary loading directions. While several analytical and semi-empirical formulations exist to model the structural mechanics and damage in a woven composite, it is essential to model the interactions at the micro-level. A typical woven composite consists of bundles of fibres which are woven in different patterns in 2 perpendicular directions, and then impregnated with the matrix material. This forms two apparent phases in the meso-scale, including the resin impregnated fibre bundles or tows, and the 'neat' or pure resin pockets between the weave patterns and tows. The properties of the fibre-matrix tow can be simplified by homogenising the domain, analogous to the unit cells for UD FRPs, where the single fibre surrounded by matrix is akin to a tow surrounded by matrix.

To efficiently model a unit cell for a woven composite, it is necessary to identify the key features of the tow and matrix geometry. On drawing an analogy with the unit cell for a single fibre shown in [2], the distinguishing features for a woven tow as compared to a UD fibre is the presence of well defined periodic undulations in the fibre trajectory (depending on the pattern of the weave). Apart from the oblong shape of the tow-cross-section, these leads to an inherently 3-dimensional geometry which requires 3D unit cells to model. Early attempts to model this geometry have been made by [7], where the tows are modelled as elliptical or rectangular tubes (Fig. 9) and were developed for applications in manufacturing. The work stops only at the geometric modelling phase for use as a manufacturing aid, but proposes that this geometry can further be developed for Finite Element Modelling. The model presents the opportunity to create a distinct representative volume element to model the tows and the weave structure, and to fill the gaps with a separate matrix element either through 3D elements or

analytical constraints.

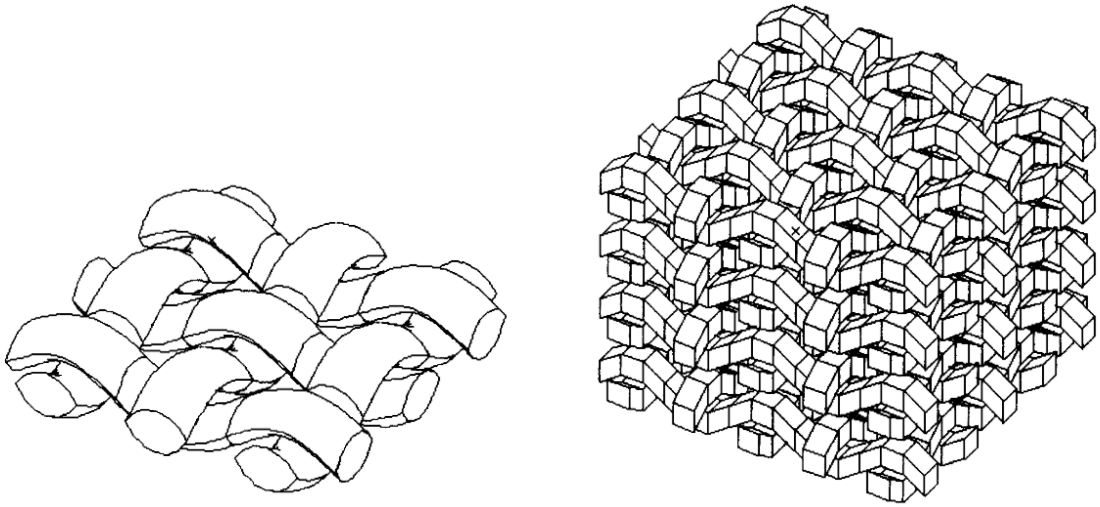

Figure 9: *A representation of: a) smooth fibre tows; b) modelled woven tows* [7]

The idea of a simple unit cell for a woven composite was presented by Whitcomb [30], where a unit cell was modelled in an individual tow as a sequence of 3D rectangular prism elements which are later smoothed to more accurately mimic the shape of actual tows in a woven fabric by using wedge elements (Fig. 10). This model was used to compare different degrees of 'waviness' or undulations in the tows with respect to the normalised mechanical properties of the unit cell; the results showed that the undulations play a significant role in the stiffness behaviour and local strain contours of the tows. This work was developed to consider different weave patterns and study the effect on stress distribution in the unit cell [31]. The author demonstrates the possibility of using simplified elements to model woven fabrics and capture 3D stress states to a reasonable accuracy, but this method fails to consider the interaction of the warp and weft tows with each other, and the effect of failures arising from resin pockets. Later, Tang and Whitcomb [32] further developed this element to capture damage progression in these unit cells and compared the effect of different weave types on the damage progression. The maximum stress failure theory is used for the sake of simplicity, and a degradation model is used for the reduction of properties within the element. The work concludes that there is a significant effect of the weave pattern on the damage progression, and that the inclusion of geometrical non-linearity in the analysis shows no significant effect until the point of damage initiation irrespective of the degree of undulation of the tow.

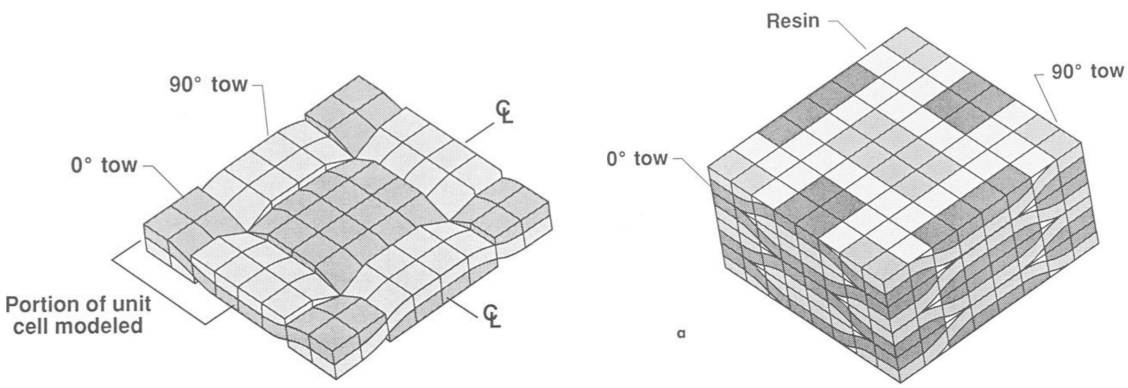

Figure 10: *3D mesh for the unit cell: a) fibre tows; b) tow and matrix* [7]

Due to the popularity of woven fibre composites, there has also been a lot of effort to improve the accessibility and implementation of such elements by using commercially available software and integrating the CAD and CAE processes. An early work by Sun et. al. [8] uses an element similar to that shown in figure of Whitcomb along with other unit cells for popular textile-like composites. The authors aim to integrate the process of CAD and CAE for fibre reinforced composites for the purpose of capturing the heterogeneity at the unit-cell scale. Different fibre configurations are considered, including 2D fabric weave, 2D basket weave, 3D tri-axial braid, and 3D UD composite (Fig. 11). Tetrahedral elements are used to mesh all components. The CAD modelling is improved by using Boolean operation algorithms to independently model each phase. The scale of the meshing is roughly 2 elements in the thickness direction of the fibre and 4 elements through the thickness of the surrounding matrix (approximately 5000 elements per unit cell). It can be assumed that the weave-pattern considers the fibre to be equivalent to the tow, while in the 3D UD composite each fibre-element represents an individual fibre, highlighting clearly the analogy between the RVEs. The objective of the paper is solely to demonstrate the feasibility of such an approach for the modelling of heterogeneity at the unit-cell level. The unit cells are analysed under in-plane shear and in-plane tensile loads. The paper shows the deformation and stress results, focusing on the 3D stresses captured within the fibre. No comparisons or validations are demonstrated as it was beyond the scope of this work to show the numerical efficacy of the model.

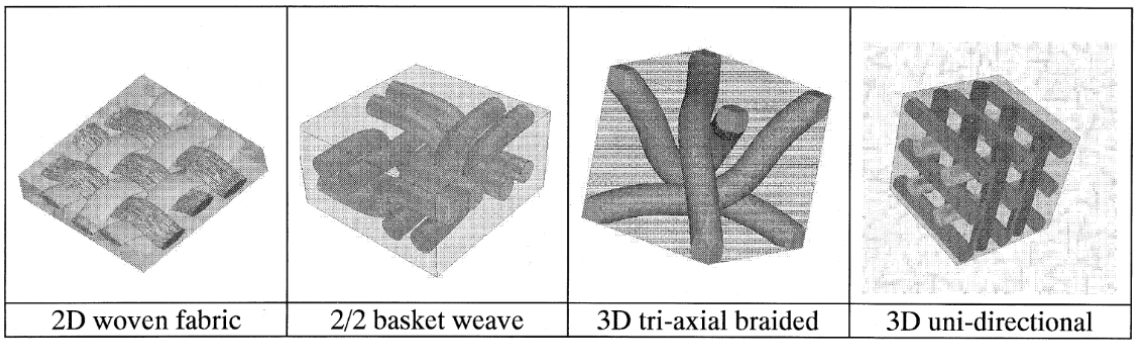

Figure 11: *Different fibre weaves analysed by [8], with analogous unit cell formulations for a fibre-based and tow-based unit cell*

It is evident from the trends in the literature that there is a need for more refined models for capturing minute interactions at the micro scale, particularly for the study of specific products or cases used in practical applications. An example of this is the work by Dixit et. al. [9] for the modelling of a 5-harness satin weave comprised of a matrix of isotropic PolyPhenylene Sulphide matrix and yarns(or tows) of T300JB carbon fibres impregnated with PPS matrix. The model unit cell for this model displays the minimum representative volume element which can capture all the features of the weave and maintain periodicity (5x5 tows per cell). The model is meshed with 8-noded brick elements (50000 elements per cell) with a reduced order of integration (single gauss point at centroid), which shows the increasing demand for highly refined meshes. Yarn and matrix pockets are meshed separately and combined. The effective material properties of this RVE are found by conducting an in-plane loading analysis. Peak stresses and stress distributions within the fibre and matrix are shown. The effective material properties are verified against theoretical and experimental sources. The paper then investigates the effect of weave parameters such as yarn spacing, yarn width, and fabric thickness on the effective material properties. There is no discussion on stochastic variation of properties and the micromechanics of the tow/yarn or surrounding matrix pockets. Figure 12 shows this complex 3D 'voxel' discretisation which demands high computational resources for a very small domain of modelling.

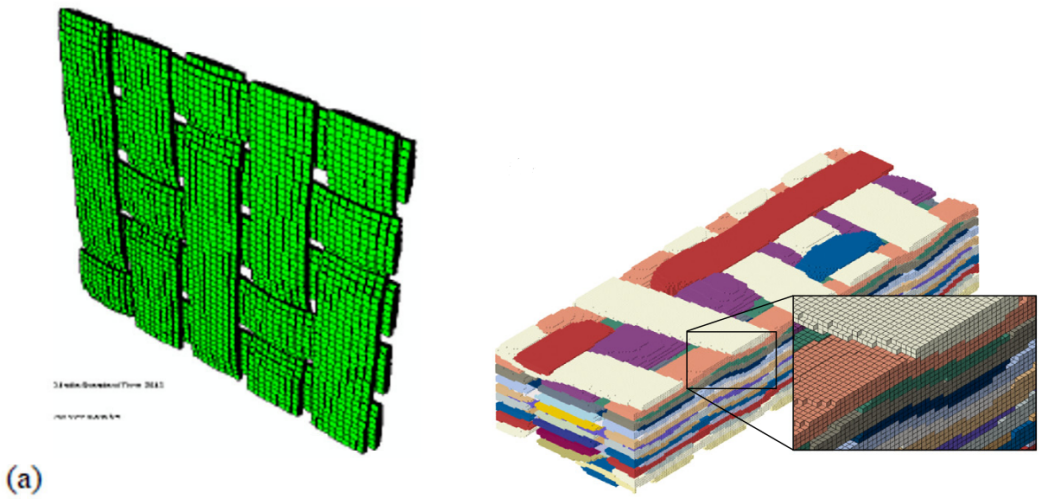

Figure 12: *High density meshes observed in 3D woven composite unit cells* [9, 10]

A comprehensive review on these 3D weave structures and their modelling methods are described by Ansar et. al. [33]. The authors succinctly conclude that the FE analyses of woven composites follow two major subdivisions, both with significant drawbacks. The most common method employed is the use of a homogenised unit cell with averaged properties, which produce smoothed 3D stress states over a large domain and are incapable of capturing the local variations of stress at the micro-scale, hence proving inadequate for accurate failure analysis. The alternative is to model the geometry with fine 3D mesh elements to capture the stress states at a micro level (Fig 12). This has been shown to be highly computationally demanding, and is deemed unsuitable for full-scale analyses by the authors.

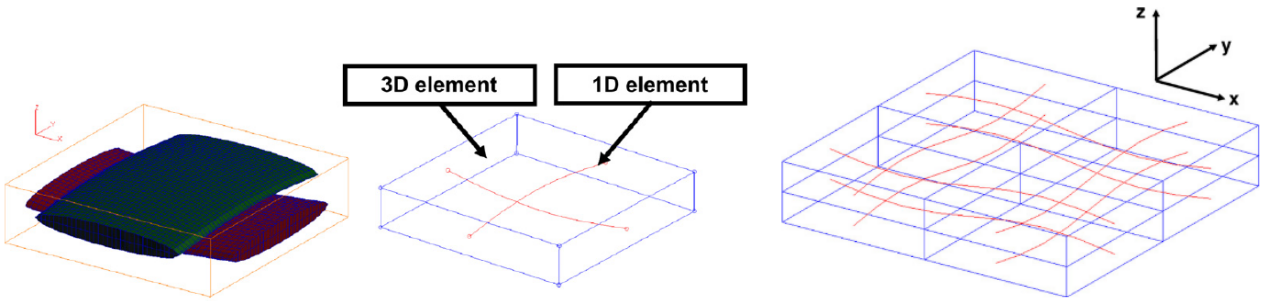

Figure 13: *An example using the binary mesh theory: a) modelling the tows with curved spring elements in a 3D matrix; b) a periodic unit cell* [11]

A possible solution to the high computational inefficiency of the 3D microstructure mesh is the binary model for woven FRPs, first described by Cox et, al. [34, 35, 36]. Here, the tows and matrix are treated separately, as before, but the tows are modelled as 1-Dimensional spring elements with no transverse or shear rigidity (Fig 13) . The authors propose that this model could significantly reduce the computational effort while still producing realistic values for global deformation under loading including the effect of damage on the composite. A wire of infinitesimal thickness is given the mid-line properties of the tow as a series of springs, and the rest of the medium is filled with elements which are anisotropic in nature and represent an averaged effect of the fibre bundle encapsulated in matrix and surrounded by matrix pools as explained in [37]. This model was further improved in [38] by averaging the stresses over a gauge length equal to or greater than the tow width, thus eliminating any singularities and also lowering the dependence on the mesh sizing, while retaining local stress state data. Römelt and Cunningham [11] compare the binary method to the 3D mesh method and

present the efficacy of this method over the linear range and also point out a loss of effectiveness at the onset of damage and non-linearity due to the small size of the unit cell. In his review of such works, Bogdanovich [37] claims that this model, while struggling to capture highly localised failure, is an excellent method to model woven composites with high computational efficiency due to its ability to capture common damage modes, include stochastic variation, and capture all essential geometric data. The model can even be expanded to include beam-like formulations using a similar concept of modelling the tow neutral axis as lines surrounded by a 3D matrix.

4.2 Damage and Failure in UD composite Unit cells

The use of the unit cell allows for the study of the micromechanics of damage and failure modes in composite structures. Composite structures are capable of retaining some structural properties even after undergoing micro-scale failures such as matrix plasticity, matrix cracks, fibre breakage, and fibre-matrix debonding. While designing composite structures, it is essential to understand the new pathways for load transfer on the failure of the region of a composite, and hence calculate the residual strength. For this purpose, the unit cell proves an ideal domain for analysis.

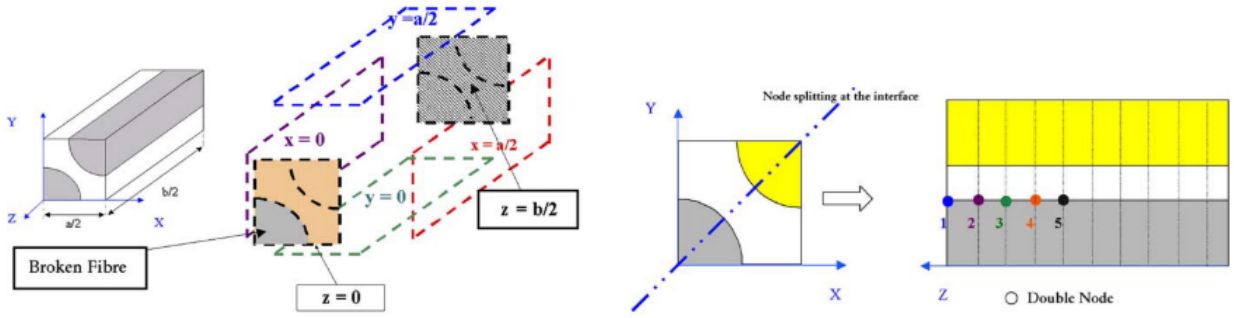

Figure 14: *Unit cell for studying load transfer mechanism in a UD composite post-damage with: a) C2 fibre breakage; b) debonding along fibre-matrix interface* [1]

Different unit cell designs are often used to capture specific types of damage based on the area of focus for the study. Particularly for the case of UD composite unit cells, it is possible to represent the complete structure by exploiting the periodicity and using unit cells that capture the domain of a single fibre and its immediate neighbourhood, as seen up till now. For some cases however, it is necessary to use multiple fibre unit cells either for the purpose of capturing stochastic variations in position, or to capture interactions between the fibres. While these models are computationally expensive, such cases are used only for modelling the damage within a single unit cell, as opposed to conducting a multiscale analysis. An example of such an application is the work by Blassiau et. al. [1] who have studied the stress distribution along the length and cross-section of fibre neighbourhoods surrounding a region of damage. The paper uses 3D unit cells to model the load transfer and stress states of a UD FRP with damaged fibres (Fig. 14). The locations and distributions of the damaged fibres studied here range from C-2 (every alternate fibre is damaged) to C-infinity (only a single fibre is damaged). Different unit cell shapes and sizes were chosen based on the symmetries observed in the damaged fibre layouts. The element also captures fibre matrix debonding by introducing duplicate nodes along the axis of the fibre-matrix interface. A coefficient of load transfer is defined and plotted based on axial position along the fibre and distance from broken fibre. This work shows the efficacy of this model by presenting a comprehensive parametric description of load transfer mechanisms in an FRP with broken fibres with different degrees of debonding while remaining in the linear elastic regime. It does not consider random distribution of fibres, or random fibre breaks. The author proposes developing the model to include elastic-plastic and visco-plastic behaviour in the matrix, and introducing multi-scale techniques to use

the unit cell in more detailed 3D analysis of failure propagation.

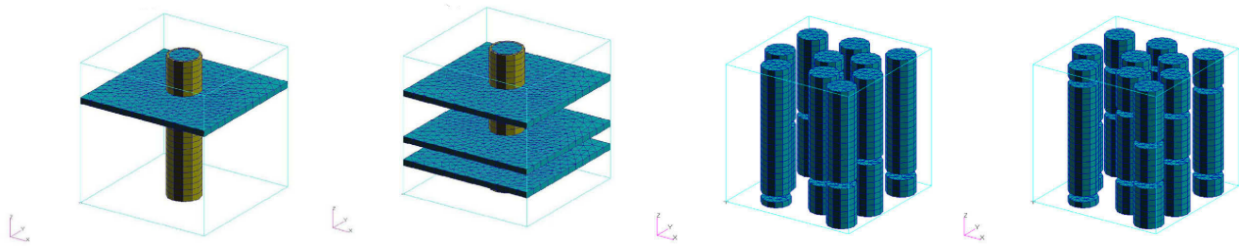

Figure 15: *Unit cell models with different iterations of fibre numbers and damagable layers* [12]

A similar model has been used by Wang et. al. [12] to study the effect of different unit cell configurations on the ability to capture damage evolution in a UD FRP. The model introduces damage in specific location by using failure models only in designated layers within the model, such as at the fibre-matrix interface, specified cross-sectional position along the fibre, or failure plane within the matrix. The unit cells studied include iterations of the number of such damage layers over a single or multi-fibre Unit cell (Fig. 15). The analysis captures the 3D stress states in both the fibres and matrix pre and post damage initiation. The model employs two different methods to capture damage in the failure layers: the weakening element model, and the cohesive element model. The authors find that the two models show very similar results in terms of stress distribution and static results, while they differ in capturing the dynamic changes in stress or stress evolution, the damageable layer showing sudden changes in stress while the cohesive elements showing a more gradual evolution of damage. Nano-scale inhomogeneity in the material properties of the fibre were introduced to emulate real fibres. These provided regions with weaker elements which provided sources for failure initiation. The authors conclude that the inclusion of fibre inhomogeneity does not affect the final results of the strength and stiffness of the material post-damage, but it affects the nature of damage propagation and the emulation of the damage initiation within the fibre. Several other damage modes were studied in the matrix and interfacial layers. The paper also shows the importance of considering multiple fibres for the Unit cell for the purpose of studying the propagation of failure since a broken fibre induces high stresses in neighbouring fibre which leads to propagation of the failure. The work can further be expanded to consider the 3D propagation of cracks or stresses, since the damage is currently localised layers. The model also appears to use a coarse mesh as compared to previous studies in 2D domains, possibly to aid large parametric studies including complex multi-fibre RVEs. The study also shows that it is possible to capture different damage modes with a single-fibre unit cell, which may be sufficient for the purpose of a multi-scale study. While some elements of stochastic distribution of multiple fibres and damage zones were included, this was not the focus of the study.

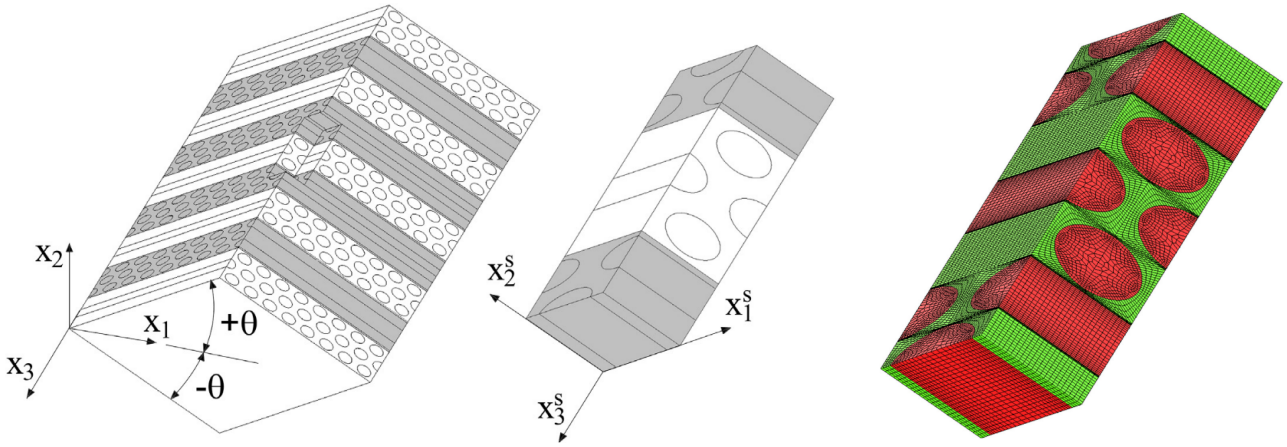

Figure 16: *Unit cell capturing multiple layers of an angled ply: a) representation; b) fine mesh density* [13]

Another application of the unit cell model is to study the effect of the microstructure on different macroscopic properties and failure modes in commonly found composite laminates including cross-ply, angled ply, and quasi-isotropic layups. The common approach for such cases is to include a unit cell that captures all the layers involved in the laminate, such as in the work of Romanowicz [13]. This work is an attempt at studying the stress distribution at the meso-scale through FRPs using layups of angled UD ply. The unit cells are chosen based on the on the square and hexagonal packing of the fibres, and they represent a 3D periodic element through multiple layers of the composite, with fibres depicted in all directions involved (Fig. 16). Different ply orientations are considered for the analyses, and the stress-strain response of each composite is shown against the variation in ply-angle, and plasticity parameters. The study uses a continuous plasticity model (Drucker-Prager plasticity model), and the domain of the analysis ranges from the linear elastic regime through plasticity, to failure. Different failure mechanisms for the composite are described, while not considering the interaction of different failure modes for the simplicity of the model. The effect of fibre-packing, ply-thickness, and presence of interfacial bonding are also studied and presented. This work only uses 3 fibres per ply through the thickness of the laminate, which is an idealised assumption, taken due to the lack of sensitivity of the model for increase in number of fibres through the thickness of the lamina. However, the results demonstrate that the model is sensitive to fibre packing. The authors show the predicted strength of the laminates match experimental results, and the plasticity and damage model is able to capture the high ductility of the 45 degree orientations. This model cannot be expanded for larger ply thicknesses due to the unit cell assumed, and the model also does not account for randomness in the distribution of fibres across the thickness. The computational efficiency of the model can also be improved as seen by the high number of elements used (Fig. 16).

While such unit cells are beneficial for the study of specific angled ply composites, It is difficult to generalise the model for different angles and layups due to the unit cell having to capture all the different layer data simultaneously. Instead, it might be preferable to utilise a model with a simple representative unit cell such as that shown in Fig. 17. Such models capture the periodicity within a single layer by utilising a single or double fibre unit cell. These unit cells can capture all the damage models and interactions as described previously, while also having the capability of using modified boundary conditions to be stacked and effectively form a multi-fibre RVE.

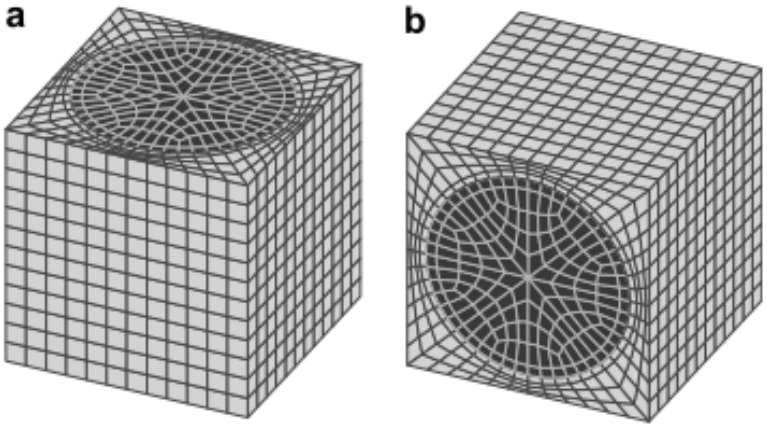

Figure 17: *Cubic unit cell with a single fibre periodic entity* [14]

To utilize this approach, it is necessary to use a substructuring scheme which is capable of capturing the different layers and orientations involved. Some of the work in this domain has been described by Matsuda et.al. in several studies. One of the early studies [14] describes the interlaminar stresses in a multi-layered (cross-ply) composite, where each layer consists of UD Carbon fibre reinforced polymer. The study focuses on 2 load cased of in-plane tensile load, along the global on-axis direction, and along the 45 degree off-axis direction. The domain considered involved two adjacent layers (0-90 orientation), each with an N x N array of fibres (Fig. 18). Due to the regular distribution of fibres, it was possible to model each fibre matrix region with a 3D single fibre cubic unit cell, discretised with 4320 elements (Fig. 17). The work concludes that the use of such a detailed mesh is necessary only around the region of focus for the analysis, since for the case of inter-laminar stress, the concentration of high stresses is predicted only to be within one fibre-width of the inter-laminar interface. A sub-structuring method was employed to further increase the computational efficiency of the model, with homogenisation techniques used to solve the boundary value problem. The authors present that this technique can be expanded to evaluate other localised microscopic regions of stress concentrations, such as free-edge stresses in laminates, and for other fibre orientations and angles.

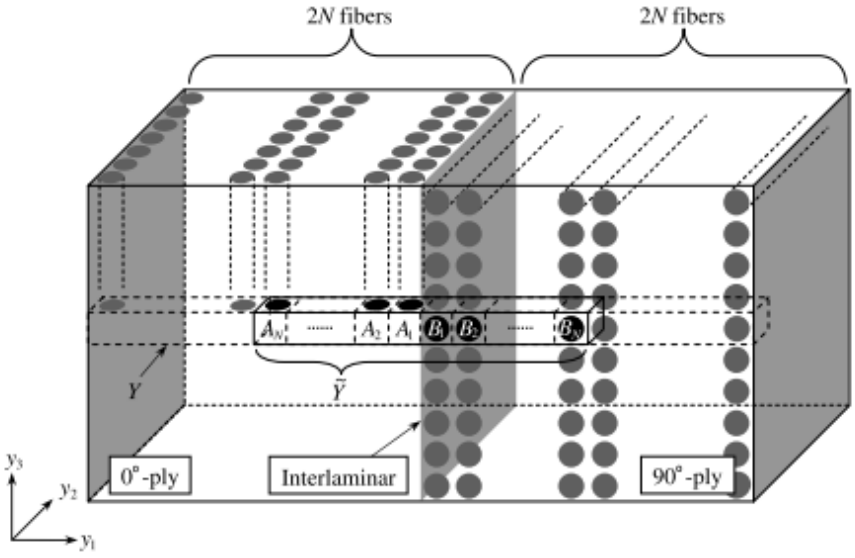

Figure 18: *Substructure including semiunit periodic cell for a cross-ply laminate* [14]

This work was further expanded by the same group [39]. Here, the region of focus is the free

edge of the laminate under in-plane on-axis tensile loading. A 3D stress state was obtained for the free edge using a similar cubic unit cell and sub-substructure as described in [14]. The results show accumulation of interlaminar shear stresses on these free-edges, leading to possible regions of damage initiation. The investigation of such damage is yet to be conducted, but can be considered by expanding the formulation of the current unit cells used. The author also recommends the use of the element for further study of the free-edge and interlaminar stresses under off-axis loading. The substructure established here can also be expanded for different orientations of composites as shown in [15], In this work, a unit cell was developed to model a quasi-isotropic laminate comprised of UD CFRP plies. The unit cell comprises of the intersection of a hexagonal prism with axis orientated parallel to the thickness direction of the lamina, such that the intersection of the geometry with equal sections of 2 adjacent fibres is captured (Fig. 19). This is then meshed with 3D brick and prism elements. Different iterations of the shape are used for modelling the different orientations of the quasi-isotropic layup. A sub-structuring method has been used in conjunction with the homogenisation theory for the Elastic Visco-Plastic (EVP) model to reduce the number of active degrees of freedom. The analysis shows the near-linear behaviour of the composite in the macroscale, in tandem with the accumulation of visco-plastic deformation localised in the matrix regions in the narrow gaps between fibres. This model was also capable of capturing the interlaminar shear stresses between different ply angles. The authors conclude that such a unit-cell model was essential in capturing this micro-scale behaviour. Such a technique can be further expanded to different layups of the laminates, and the study can also be expanded to capture different orientations and packing of fibres in the thickness direction of the laminate. Further failure modes and inherent defects can be modelled to accurately depict the multi-scale behaviour of laminates with the scatter in material properties and different failure modes.

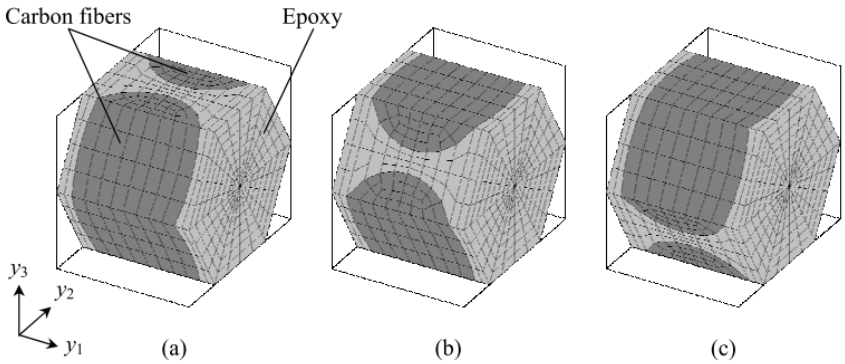

Figure 19: *Hexagonal unit cells capturing different fibre orientations in quasi-isotropic laminates* [15]

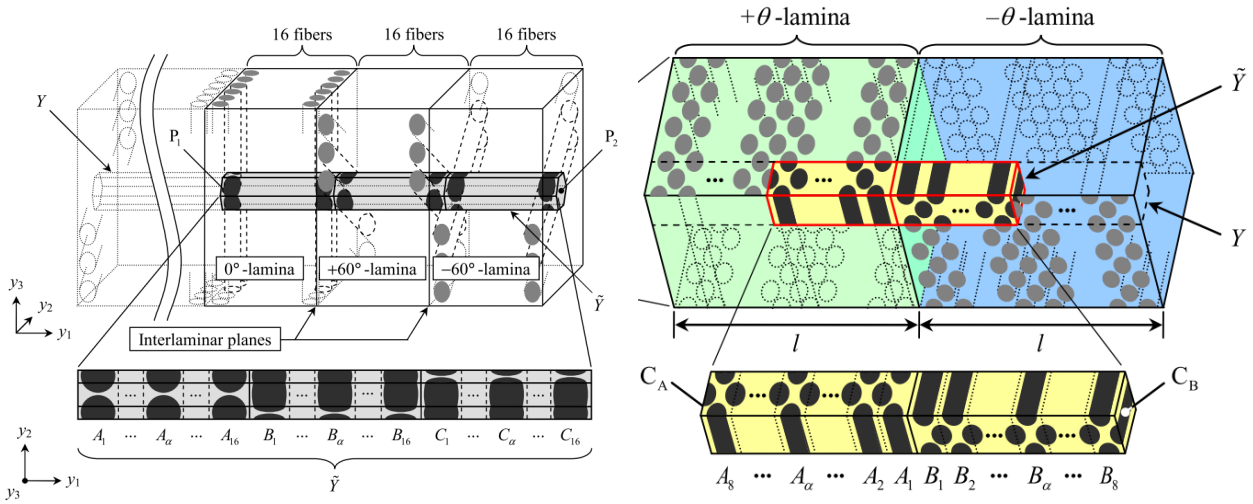

Figure 20: *Substructure employed for modelling: a) quasi-isotropic; b) angled ply laminates* [15, 16]

There is also a strong need to be able to capture different angled plies to compare their effective behaviours as mentioned by Romanowicz [13]. Matsuda et. al. [16] have extended their subtructuring approach to provide an alternative to the multi-layer unit cell prescribed by Romanowicz. A two-fibre RVE considering periodic hexagonal packing is used instead, which is capable of being integrated with the substructuring approach shown in Fig. 20. This study focuses on angled UD ply FRP laminates under uni-directional tensile strain loading with constant strain rates. The Poisson's ratio in the EVP range was studied, and the evolution of this property with increasing tensile strain was presented. These are then used to model a larger domain involving multiple angled ply using efficient sub-structuring methods. The analysis shows negative EVP poisson's ratio for angled ply composites only in the range of ±15 – 40 degree orientations, with increasingly negative values on onset and progression of visco-plastic deformation. The authors propose a micro-mechanic reasoning for this observed result based on the multi-scale analysis performed, and attribute it to the interaction of the different orientations of the angled ply at their interface, with induced compressive loads observed perpendicular to fibre orientations. The unit cell used for this purpose uses a finely meshed domain (Fig. 21), leading to possible computational inefficiency, and the model does not account for stochastic variation of fibre distribution through the thickness of the angled ply. While some stress distribution through the composite is presented at the micro-scale to demonstrate induced normal compressive stress, introduction of damage or defects has not been considered for this study.

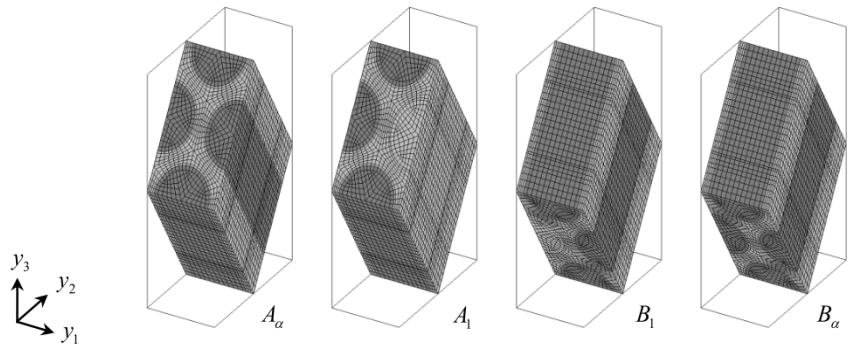

Figure 21: *Angled ply substructure cells for ± 30 degree orientations* [16]

There have been several other attempts at modelling damage with unit cells, including the presence

of stochastically distributed fibre positions, including multiple layers and fibres in the unit cell, and by studying different types of composites such as textile-based, woven, braided, and other reinforcement types. Some of these will be shown in the next section, which is focused on highlighting the increased interest in stochastic variations.

4.3 Irregularity in fibre distributions

Most of the unit cell models studied so far consider periodic packing of fibres in the cross-section of the lamina. However, in practice, the fibres are distributed randomly in the cross-section as seen in Fig. 22 and also the shape of the fibres need not be strictly circular. Fibres can also exhibit undulations along their length due to manufacturing processes or structure of the composite. This variation could significantly affect the global mechanical properties and also the failure and behaviour of the composite post-damage, and this is reflected in the large scatter in material data for composites. Due to the improvements in unit cell design and computational performance, there has been a growing interest in studying these random variations at the micro scale. A few examples of unit cell and RVE design are presented in this section.

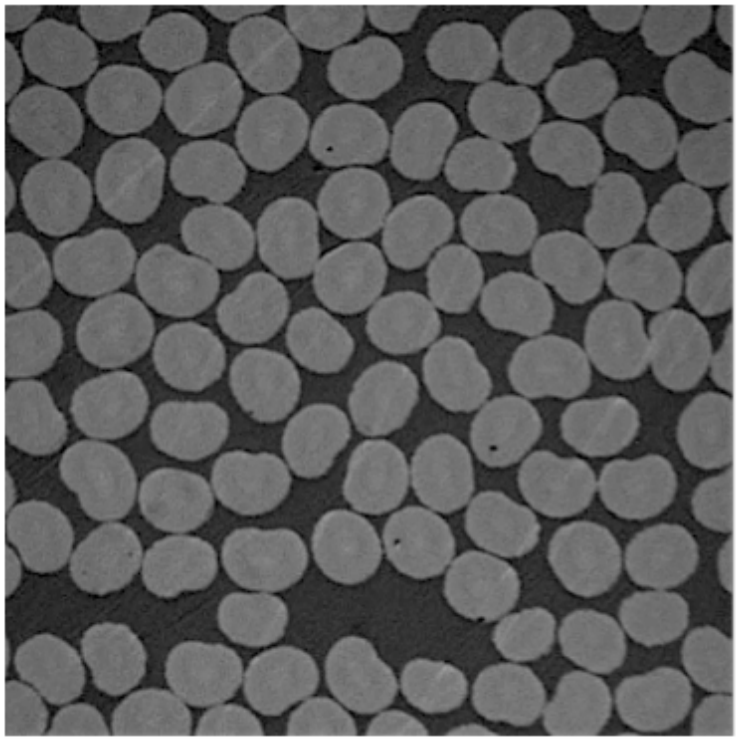

Figure 22: *Irregularity in fibre distribution* [17]

One of the methods used to capture the realistic irregularity in the fibre distribution is through image-based reconstructions of actual composites as presented by Okabe et. al. [18]. This work assesses the locations and modes of damage initiation under in-plane tension for the case of a UD FRP. The analysis considers 3 type of cross-sectional profiles, including the uniform square and hexagonal packing of fibres, and a non-uniform, image-based distribution of fibres (Fig. 23). The unit cell consists of several fibres surrounded by matrix distributed over a 2D surface emulating the cross-sectional profile. The damage in the matrix is simulated using a single parameter visco-plastic model. The analyses considers transverse loading and shows the effect of the different fibre orientation on the evolution of the damage in the matrix and interfacial stresses along the circumference of the fibre-matrix boundary.

The authors conclude that low tensile strength is observed in regions surrounding fibres parallel to the loading direction. Also, the model accurately captures the realistic failure initiation observed under experimental analysis. The paper does not consider the 3D stress states, and a similar model expanded to 3D would require a very large computational effort.

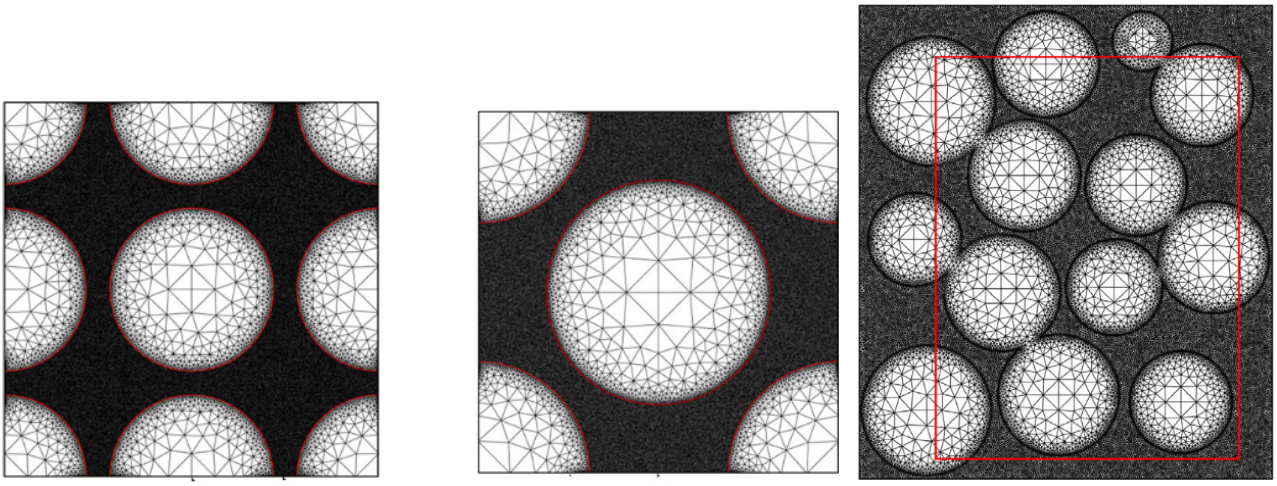

Figure 23: *2D unit cells for a) periodic square; b) periodic hexagonal; c) Realistic image* [18]

While this method is very useful for emulating the behaviour of a realistic composite material, it is limited to a a countable number of microstructures which can be imaged, and also focused on a single region of the whole composite. This process of microscopic imaging is also destructive for the composite sample analysed and expensive due to the equipment required. Instead, it is possible to model unit cells using stochastic methods to simulate the natural variations in different composite materials and also obtain a range of mechanical property values which might reflect the scatter observed in experimental setups. One example of such a study is shown by D'Mello and Waas [17]. This paper aims to find an optimal sizing for a multi-fibre, 2D unit cell. The paper asserts the strong correlation between the damage on a global(macro) scale and the failure at a local(micro) scale. The model captures a random orientation of fibres in the cross-sectional plane of the composite, considering different number of fibres in the unit cell, ranging from 6 to 180 fibres per cell (Fig. 24). The orientation within each unit cell is varied over 10 random iterations to obtain data on the effect of fibre distribution. It was found that the random distribution of fibres played a significant role in the scatter of material properties for all unit cells. Also, it was found that a larger unit cell showed more compliance for a constant volume fraction as compared to smaller cells. Hence the authors conclude that for all analyses, convergence studies must be conducted to ensure that the small size of the RVE does not over-estimate the stiffness or strength of the material.

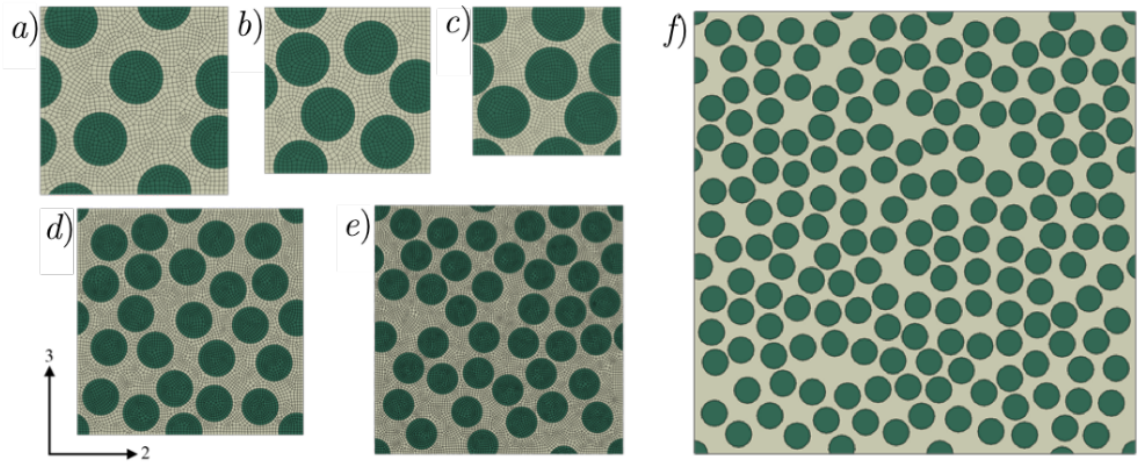

Figure 24: *Different unit cell iterations and sizes for a stochastically distributed 2D unit cell* [17]

The introduction of stochastic variation has also been seen in 3D Representative Volume Elements. However, this often requires larger size of RVEs involving a very fine 3D mesh. An early attempt to capture variation of fibre positions in the matrix cross-section for UD straight fibre composites is shown by Aghdam and Dezhsetan [19]. A microscopic RVE of some fibres and matrix are considered, where the cross-section is further subdivided into sub-cells of dimension (R x C). These sub-cells can either contain the material properties of a single fibre, or of pure matrix. This distribution is randomised with the constraint of the volume fraction imposed (Fig. 25). The structure is then modelled analytically considering only linear strain terms. This results in a set of equations for which closed form solutions can be obtained for certain load cases. The model is verified under isotropic material properties, and then expanded for the thermal and mechanical responses of a SiC/Ti composite system. The Elastic Modulus, Shear Modulus, Poisson's ratio, and longitudinal coefficient of thermal expansion of the effective material are evaluated with respect to different fibre volume fractions. The results are averaged over 6 different random fibre matrix orientations. This model is limited to allotting a cuboidal representative volume for each fibre, and does not capture any fibre matrix interactions, failure behaviour, and damage effects.

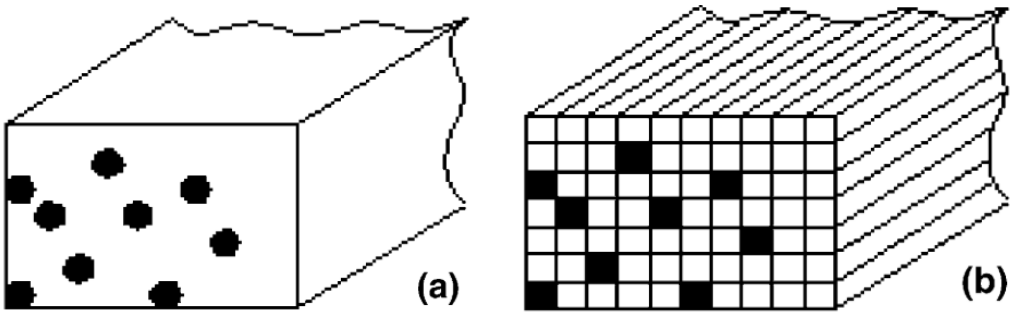

Figure 25: *Random distributions of fibres through the cross-section: a) ideal; b) simplified cuboidal model* [19]

Some other attempts at modelling simplistic 3D fibres with some randomness can be found in the literature such as those seen in previous sections [12]. Swolfs et. al. [20] have explored the scope of the 3D RVE thoroughly for the purpose of studying stress concentrations, damage propagation, and effects of fibre breakage including randomised distribution of fibres. The advantage of the 3D RVE used over the randomised 3D unit cell of Wang et. al. [12] is the use of a fully 3-Dimensional model (Fig. 26) without limiting damage to localised layers of elements. However, this model requires very high mesh

densities leading to severe computational load. The random distribution is compared to regular square and hexagonal arrays over several parameters such as stress concentration factors, ineffective fibre length, and stress distributions with local fibre breakage. The model has also shown the importance of proper modelling within the fibre domain, by including anisotropy representative of a realistic carbon fibre. They also show the vast differences in the stress distribution on randomising the fibre distribution, and the authors advocate such a model for the purpose of studying the influence of this distribution on the failure characteristics of the composite. This research group has comprehensively utilised this RVE method for studying the microscopic stress distributions on randomising the fibre distribution and breakage parameters as can be found in Ref. [40].

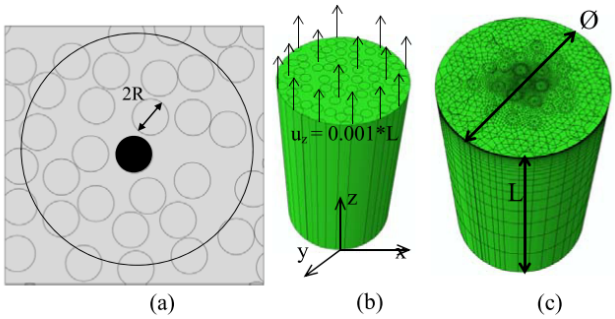

Figure 26: *a) Random distribution of fibres; b) Extracted cylindrical domain; c) Meshed 3D RVE* [20]

Another popular application of the Randomised RVE is the M2RVE which utilises a multi-fibre multi-layer model to include random distributions over a multi-ply composite. An example of this work is on the brittle fracture study of Ceramic matrix fibre composites by Daggumati et. al. [21]. The RVE compares the failure behaviours captured by a single fibre unit cell, multi-fibre RVE (Uni-directional), and cross-ply multi-layer RVE including random distribution (Fig. 27). These FEM models are compared against Experimental data, showing the conformity of the random distributions to the scatter observed in practice. This form of Representative Volume Element has gained a lot of popularity in recent years, and represents the trend of approaching the ideal model for the complete evaluation of a Fibre reinforced composite structure, while sacrificing computational efficiency to capture more structural detail.

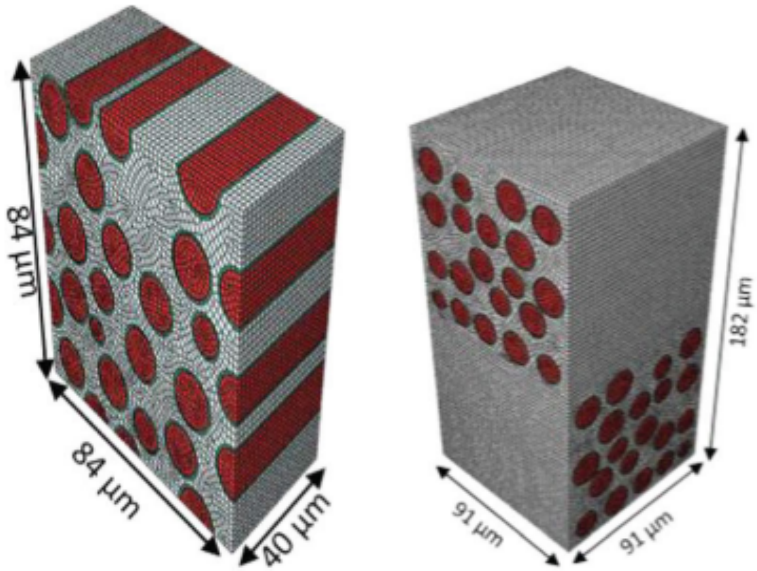

Figure 27: *a) A randomised Multifibre RVE; b) Extended M2RVE for a Cross-ply laminate* [21]

The applications described so far for straight fibre unit cells consider perfectly straight fibres, for the 2D and 3D case. In reality, the fibres often show some undulation over the length, which can significantly affect the stiffness and strength of the composites. While the attempts to model periodic and regular undulations have already been seen for woven composites, the 'waviness' of the fibres in UD composites are much more erratic and require stochastic modelling to capture them. The 3D unit cells serve as ideal opportunities to capture these undulations. Catalanotti and Sebaey [22] have developed an approach to create a multi-fibre RVE and capture the perturbations of fibre waviness along the length of the fibre. The orientation of the fibre is also distributed randomly through the thickness, and such a model is generated using Rhino CAD and meshed using ABAQUS (Fig. 29). The fibres are modelled using analytical Bezier curves (Fig. 28) which allow them to be perturbed consistently and the perturbed fibre possess the precise analytical geometrical data needed for meshing.

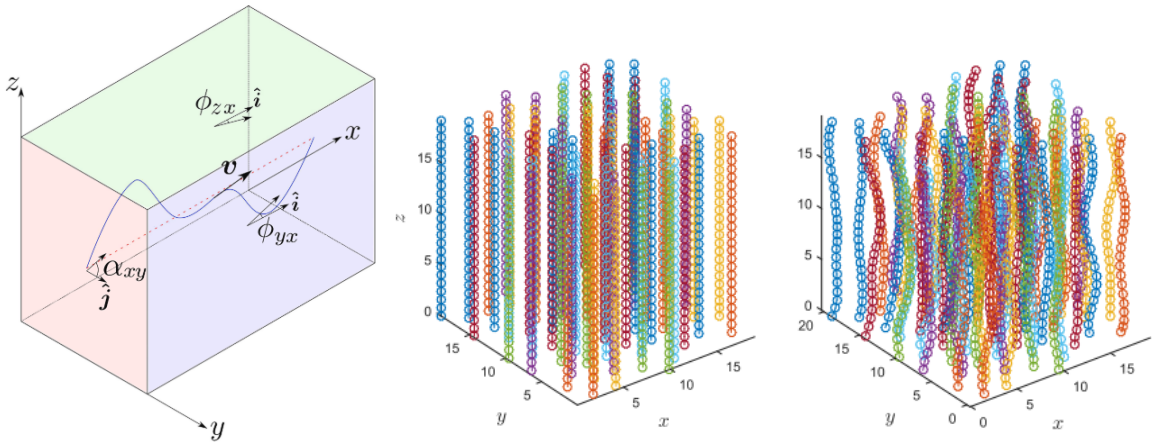

Figure 28: *a) Altered effective fibre angles due to undulations; b) initial configuration; c) perturbed fibres* [22]

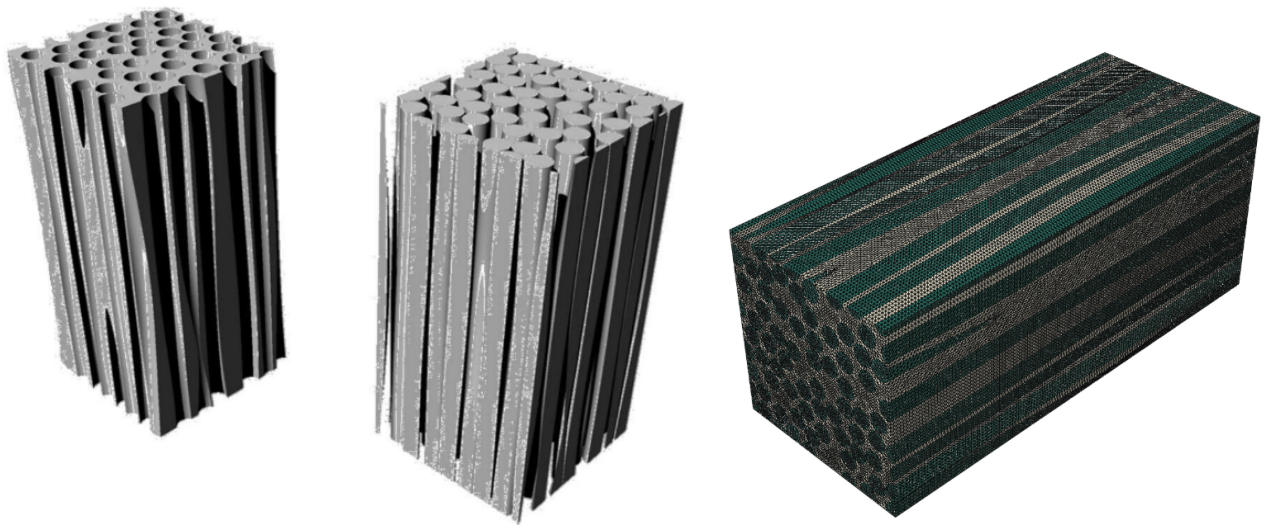

Figure 29: *a) Matrix and Fibre domains modelled in Rhino CAD; b) Complete ABAQUS 3D mesh* [22]

A comprehensive conclusion to the current state of the research in unit cell design and composite analysis is given by Bargmann et. al. [23]. This work serves as a valuable reference for 3D unit cell design for various composite and heterogeneous materials for micro-mechanical analysis. With respect to long fibre reinforced composites, the authors describe the basic unit cell for a single fibre element, involving both square and hexagonal packing, stating their low computational effort due to

inherent symmetries and assumption of periodicity and regular distribution of fibres. However, for a complete analysis involving the influence of fibre distribution, such as transverse loading and damage studies, it has been recommended to consider multi-fibre unit cells, or to introduce randomness in the distribution of the fibres. The work also covers other work on randomly oriented fibres and the algorithms used to generate such RVEs, including cross-sectional image-based modelling, perturbation based modelling of regular patterns, and sequential adsorption algorithms. Further, the work shows the growing interest in introducing and studying the variations of material properties in 3-dimensional geometries, including variation in fibre geometry and orientation along the length of the fibre, in application such as woven, braided, or natural fibres which inherently possess waviness or arbitrary undulations. Several works have been cited to study the effects of both regular or periodic undulations and also stochastic undulations along the fibre. The models used to analyse such materials involve multi-fibre RVEs with computationally expensive discretisation. There has been some effort to reduce the computational load by simplifying the wavy fibres with sphero-cylinders, or discrete spheres, (Fig. 30) but this loses significant data on the exact geometry of the fibre.

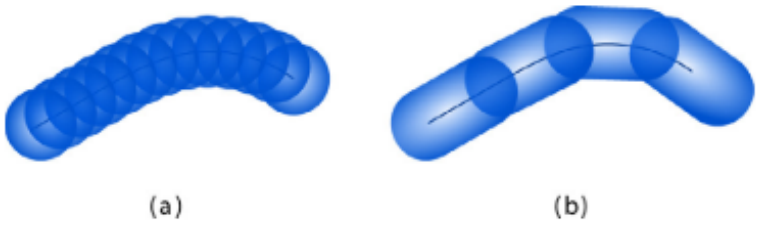

Figure 30: *Attempts at simplifying the undulated fibre with: a) spheres; b) sphero-cylinders* [23]

The conclusion drawn from the historical development and the current state of the literature of unit cell theory is that there has yet to be a comprehensive solution to the modelling of such composites where the 3D orientation of the fibres and undulating geometry can be captured including stochastic effects with simplified models requiring low computational effort. The conclusions of the research group of Swolfs et. al. on their evaluation of the leading models for UD fibre composites in the present literature [40] align with this statement, and further comment that the development of such models for the accurate representation of fibre, matrix, and interface properties with realistic statistical distributions is paramount for the development of new composite materials and is the appropriate direction of development for composite modelling theory.

5 Summary of the Literature

A detailed literature study has been conducted on the development of microstructural composite analysis, with specific focus on the use of unit cells to capture periodic representative domains involving the discrete materials phases in the composite. The study has focused on covering the progress in Fibre Reinforced Composite unit cells, including Unidirectional or Laminated composites, and Woven fibre composites. The literature was classified into two major time periods of development, pre-2000s and post-2000s, capturing the initial phase of development and culminating in the current areas of focus. Some of the important conclusions drawn from the state of the literature, and the observed trends are presented here.

- The progress in unit cell study in the period before 2000s focuses on improving the computational efficiency by optimising the shape and boundary conditions of the unit cell to most efficiently utilise the existing symmetries of the ideal microstructure. While the inability to scale up to a refined multi-scale model was primarily driven by the low available computational power of the era, it also highlights the inherent difficulty in modelling the microscopic domain accurately without the use of very fine meshes or very small representative domains.
- The early studies show the efficacy of a single fibre unit cell element by adequately replicating the global material properties of the bulk material and validating them against experimental results. These studies also demonstrate the validity of assuming periodic square or hexagonal fibre packing through the cross-section, particularly for the purpose of replicating the effective global material behaviour.
- These simple fibre unit cells have also been used to study the highly localised progression and effects of damage related to fibre-matrix interface failures and fracture within the reinforcement. However, the single fibre unit cell lacks the ability to fully predict the stress states and capture more complex damage modes in the adjacent fibre-neighbourhood, requiring the implementation of multiple fibres in the unit cell.
- With the increasing availability of computational power in recent years, there has been more focus on developing multi-fibre unit cells. Two dimensional RVEs were preferred due to the significantly lower computational effort. They have been applied to study the behaviour of the composite under transverse loads, including fracture and failure propagation, but cannot be used to simulate realistic applications as the load transfer mechanisms and failure of composites are dependent on the 3D stress states of the material. Also, it is impractical to assume purely transverse in-service loads for realistic simulations and hence the 2D unit cells are limited purely for the purpose of academic investigation.
- The use of 3D unit cells has been popular for the modelling of woven fibre reinforced composites due to the inherent 3-dimensional geometry of the microstructure. The primary techniques used to model the woven fibres is to consider the fibre tows as a homogenised domain surrounded by an isotropic matrix, which is analogous to the fibre-matrix unit cell for UD composites. The refined 3D unit cell mesh required for the accurate representations of 3D stress states requires very large computational effort, preventing the implementation of multi-scale analyses. This has triggered several avenues of research into developing efficient elements and substructures for the modelling of woven fibres, including the use of low-order beam-like curvilinear elements to model the midlines of the tows.
- The modelling of damage in composites is one of the primary applications for the unit cell model, particularly in recent years. The necessity of 3D stress state data through the Representative volume has been discussed, however, the use of 3D unit cells with periodic fibres can only estimate

the failure model for an ideal composite. Fibre reinforced composite materials show significant scatter in material properties, particularly in terms of failure strength due to the inherent randomness in the distribution of fibres. There is a strong desire to capture this variation by including stochastic distributions of fibres in the cross-sectional domain. Apart from capturing stress concentrations and crack propagation due to differing fibre gaps, there is also an attempt to model the effect of a localised failure on the residual strength of the composite. This can also capture pre-existing defects present in the material due to manufacturing inconsistencies, further predicting realistic failure ranges.

- The stochastic variation of micro-structure geometry has been extended to 3-dimensions in recent studies, to capture the natural undulations observed in composite fibres. This undulation is shown to have a significant effect on the stiffness of the composites. While there are different methods to capture the complete fibre geometry and apply the undulations, they all require a large domain of analysis along the length of the fibre, leading to further computational requirements. The current scope of the literature is aimed at reducing the model complexity by simplifying the fibre domain with discrete geometrical elements such as sphero-cylinders. There is also a strong similarity between the problems faced in the modelling of such undulating fibres, and the periodic undulations captured in Woven composites.

- The main trend in unit cell development is towards the complete 3-dimensional modelling of practical composite structures. A very common application of UD fibre composites is in the form of laminated layups, which involve multiple layers of differently oriented UD plies. The application of the unit cell models to capture such layers is very popular. Several approaches are used to model the layers, including using a simple single fibre unit cell, stacking several single fibres over multiple layers to create a more complex unit cell, or to include randomness in the fibre locations by creating a multi-fibre multi-layer RVE. The increasing complexity of the unit cell however, is an imminent result of the attempt to include multiple realistic features, and this leads to a significant reduction in number of parameters which can be varied and studied simultaneously.

- Another possible approach to model a large region or complete structure is by the use of multi-scale sub-structuring schemes which allows for the creation of sub-unit cells with repeating simple geometries. The current models however, still require highly refined meshes with simple unit cells incapable of capturing randomness or undulations. This approach can be expanded to capture realistic phenomena to more accurately model the complete composite.

- An important observation from the current direction of research is the need for the development of a portable element which can seamlessly integrate CAD and CAE modelling. Since the formulation of the unit cell is governed primarily by geometric peculiarities, the CAD development of the element needs to be well integrated with the meshing and evaluation tools. This is even more applicable for the case of sub-structuring techniques, due to geometrical data involving global positions of each sub-structure. There have been several attempts to generate models using existing CAD-CAE packages so as to easily vary geometrical and mechanical parameters simultaneously. This approach is also beneficial for modelling realistic structures with the developed unit cell.

From this literature study, several literature gaps and trends have been identified. A solution has been proposed in the following section, addressing specific gaps and identifying the important areas of focus for a new fibre composite unit cell.

6 Proposed Solutions

From the several conclusions drawn previously, it is possible to extract the important requirements for the development of a new unit cell for the modelling of fibre reinforced composites.

- Accurate modelling of effective global properties
- Ability to capture three dimensional stress states.
- Low computational effort.
- Ability to capture interactions of multiple neighbouring fibres.
- Compatibility with different applications such as laminated or woven composites.
- Inclusion of stochastic variation through the cross-section and along the length of the fibre.
- Capture damage and plasticity in the fibre and matrix domain.
- Ease of creating and parametrising the geometry through CAD-CAE integration.

An ideal starting point for the development of such an element has been provided by Caruso [2], whose work was one of the earliest comprehensive studies on a 3-dimensional unit cell with simple geometric design, reduced computational effort, and ability to accurately capture the global properties of the composite. Here, the fibre-matrix domain for a single fibre has been modelled, with the ability to be combined to form a periodic multi-fibre domain (Fig. 3). In later works involving the single fibre unit cell, it is common to use a truncated cuboid or a cubic element (Figs. 7,20), however the author demonstrates the possibility of using an elongated domain with the ability to capture stress fields along the fibre. This elongated geometry of the unit cell also provides the opportunity for introducing stochastic variations in the fibre path and orientations. Due to the simplistic slender structure of the fibre, the unit cell structure can be further simplified to be represented by 1-D line elements along the centroid of the fibre cross-section, to describe the position and path of the fibres through the composite. This representation allows for the seamless integration with CAD modelling to prescribe the fibre paths of several fibres in a representative domain, but also allows for the possibility of introducing computationally efficient finite elements such as refined Beam elements. An application of this method has been seen in the modelling of fibre tows in woven composites, and also for the purpose of capturing fibre path perturbations in UD fibre reinforced composites. This analogous representation is shown in Fig. 31).

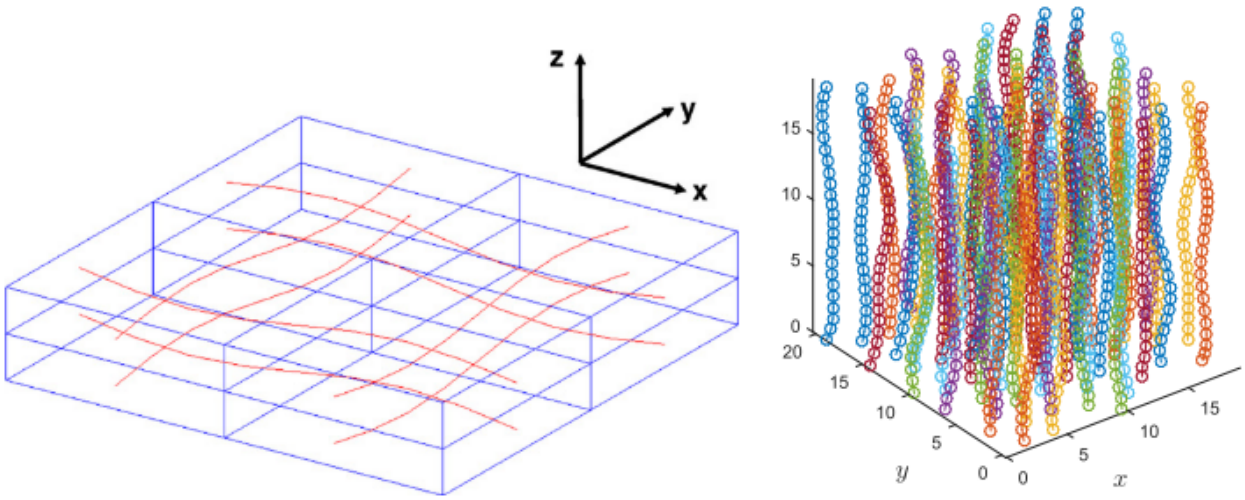

Figure 31: *One-dimensional line geometry used to represent a) woven fibre tows; b) UD Fibres with undulations* [11, 22]

The unit cell proposed here is based on a single-fibre matrix domain with an elongated cuboidal geometry such as that described in Fig. 2, where the whole cuboid is treated as a beam, with the mid-line being initially defined as the centroid of the fibre. This beam path can now be discretised with 3D beam finite elements which are capable of capturing the different material properties and geometry of the fibre and matrix through the cross-section, and also capable of modelling 3D stress states. Non-linear strain formulations and enriched elements can also be included to fully model the fibre matrix domain beyond the elastic limit. The beam neutral axis can be varied stochastically to simulate the variation in fibre path and fibre arrangement through the cross-section, hence more accurately capturing the realistic material scatter observed. Apart from the accurate modelling of the composite properties, it is also important for such an element to be user-friendly and implementable in both commercially viable software, and in stand-alone simulations, with the ability to integrate CAD and CAE. The model can be scripted to be easily described by defining the bounding domain of the Representative Volume Element and describing the fibre paths with splines or similar constructions [22] using a CAD software or by numerical formulation, and automatically identify and define the element discretisation, boundary conditions, contact with neighbouring fibres, description of beam cross-section geometry and its variation along the length of the fibre, and different domains along the cross-section. This is prepared model can then be utilised to perform finite element analysis, including pre- and post-damage studies.

While beam elements are generally defined by analytical formulations through the cross-section based on shear deformation theories such as the Euler-Bernoulli, Timoschenko, or Higher order models, it is also possible to model them using 3D formulations which can capture the exact strains as observed in conventional brick elements. Some such formulations are described below.

6.1 Geometrically Exact 3D Beam

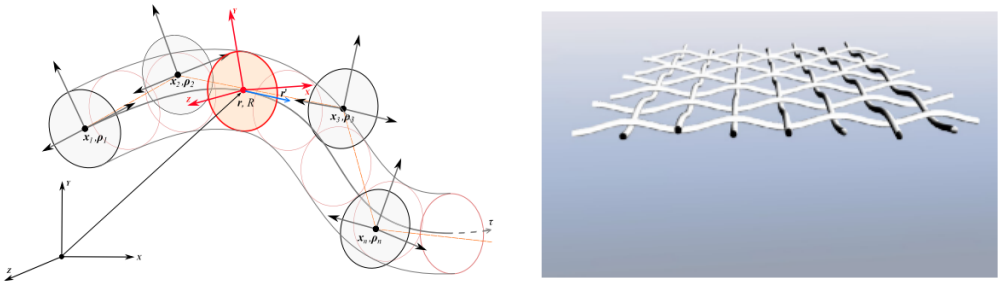

Figure 32: *The Isogeometric model for the Geometrically exact beam: a) independent positions and geometrical state vectors of cross-section planes; b) application for a woven mesh including contacts similar to woven FRPs* [24]

The geometrically exact formulation of beams treats the cross-section of the beam plane at different locations along the beam as independent degrees of freedom which are not bound to the orientation of the neutral axis in a perpendicular constraint or a fixed shear constraint as described by common shear deformation theories. Instead, the cross-section plane is evaluated using a co-rotational formulation based on the Cosserat Rod formulation ([41]) which can derive the exact strains from the displacements and rotations relative to the state of the domain in the previous iteration, and iterate these displacements over several steps to ensure a minimisation of the variation in potential energy based on the strains calculated and the material stiffness components. This model is capable of a complete description of the beam cross-section at each point and can hence generate a 3D stress state as required by the unit cell. The model has been expanded to include a fully defined definition of the beam mid-line using Isogeometric modelling by Tasora et. al. [24] and also include the detection and implementation of contact models for multiple such beam elements. An example of this is shown by Tasora et. al. by modelling a woven system (Fig. 32) , similar to the geometry of a woven fibre composite, demonstrating the ability of this model to be implemented in fibre reinforced composite unit cells. This model can be further expanded to include damage and plasticity through the cross-section, including fibre-breakage, localised voids, and other defects. However, due to the abstracted nature of the cross-sectional geometry, it is difficult to visualise propagation of crack-based failures, as there is a need for a discretisation of the cross-section to capture this failure mode.

6.2 Carrera Unified Formulation

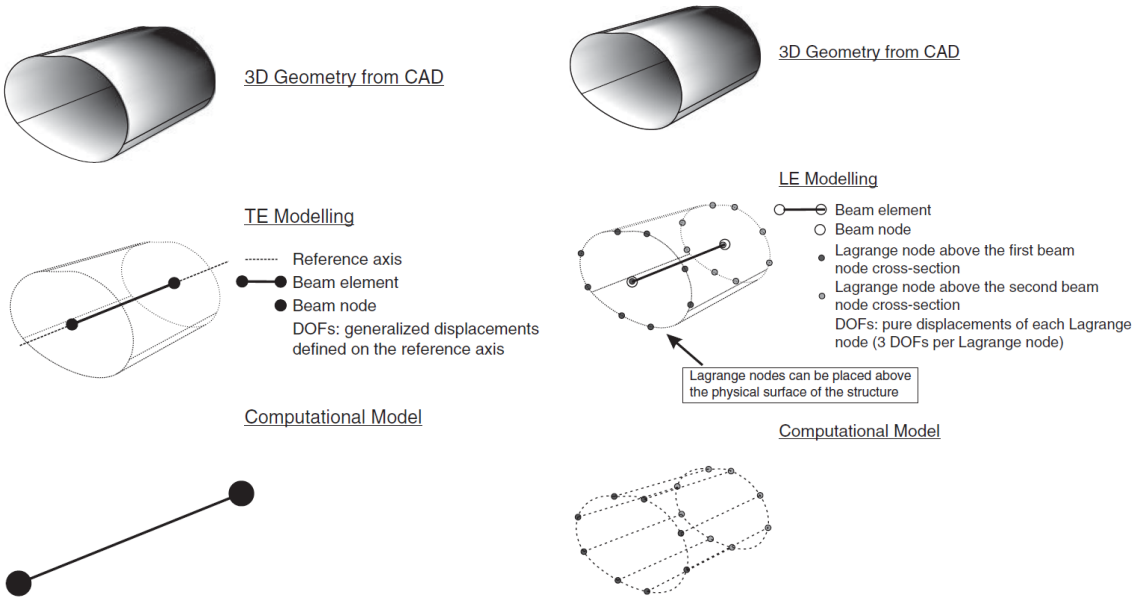

Figure 33: *The Carrera Unified Formulations for a non-uniform beam such as a fibre tow: a) Taylor Expansion model; b) Lagrange Element method* [25]

The limitation of the Geometrically exact 3D model and other beam models lies in the inability to discretise the cross-section to use popular failure propagation analysis such as XFEM or cohesive zone modelling. The Carrera Unified Formulation (CUF) [25] allows for a conventional description of the beam cross-section at defined points along the beam neutral axis by discretising the cross-section with 2-D elements which are constrained by the necessary strain displacement model ascribed to the beam, with the free degrees of freedom available through the cross-section to be treated as nodal degrees of freedom for the beam element. These degrees of freedom are constrained using different techniques, primarily using Taylor series polynomials or Lagrange polynomials to define the state of the deformation field through the cross-section of the beam (Fig. 33). In the case of the Taylor series method, the displacements along the cross-section are allowed to have a higher order polynomial function as compared to the linear values generally ascribed to 2-D beams, with the coefficients of the polynomials treated as nodal degrees of freedom, and the displacements constrained by the strain formulation used. The domain of the cross-section can be captured by integrating these displacement fields over the Gaussian quadrature points through the domain for regular shapes, and by discretising the domain to regularised elements for the case of non-uniform cross-sections. The more common approach however, is to use Lagrange polynomials [42] to describe the cross-sectional deformation, as it is possible to discretise the domain into simple 2D or 3D finite element meshes with only displacement-based degrees of freedom being used to completely model the cross-section. This is approach is also popular in modelling complex geometries over the cross-section. The CUF method, particularly the Lagrangian approach can very accurately model multi-phase cross-sectional profiles such as a single fibre unit cell or a fibre tow in a woven composite with very high efficiency, as it is possible to most effectively distribute the degrees of freedom to the areas of focus, and interpolate the intermediate fields from the computed output. The implementation of non-linearity, failure, and damage models are strongly motivated by the CUF formulation, however, there is limited literature in these fields available at the present stage, creating a barrier to the rapid prototyping and implementation of the formulation to a standard unit cell.

6.3 Difficulties in Implementation

While the Cosserat beam formulation is capable of being adapted to capture the required properties for a composite unit cell formulation, it poses several practical challenges for implementation due to the limited literature available. The range of studies with such models focus on specific applications involving continuous material properties and simple strain models, making it difficult to identify the method of implementation for the particular case of a composite unit cell. The geometrically-exact isogeometric formulation is a very rarely explored area, with sparse mentions in the literature from localised author groups, and while it is possible to find applications with similarities to that of the composite fibre tow, the lack of detailed information on the methodology used in these works of literature makes it difficult to reproduce and implement for a unit cell.

The difficulty in the approach stems from the combination of the non-linearity of the strain tensor with the use of 3D displacements and rotations of the cross-sections in the formulation. The model is defined using a parametric centre-line $r(\tau)$ where τ is generally implemented as a NURBS function parameter, and the cross-sectional orientation of the model is described with a set of quaternions $\rho(\tau)$. Generally, with iso-geometric or FE models, the parameter τ is continuous, and hence the values of ρ and r need to be interpolated from the nodal values associated with the parameter τ. For the positions, it is possible to use simple IGA shape or basis functions which are already implemented for the NURB or B-spline. The nodal degrees of freedom describing the incremental change in position are the conventional translation or displacement DOFs δx. However, for the cross-sectional orientation, the rotational degrees of freedom θ are used to update the model geometry and need to be converted to the quaternion form to conform with the formulation. An exponential map is described in [24]:

$$\rho = \exp([0, \delta\theta]) = \left\{ \cos\left(\frac{\|\delta\theta\|}{2}\right), \frac{\delta\theta}{\|\delta\theta\|} \sin\left(\frac{\|\delta\theta\|}{2}\right) \right\} \tag{1}$$

This map defines the nodal quaternions which are now to be interpolated over the parametric domain to obtain a fully defined geometry. It is not possible to use standard basis functions alone to interpolate quaternions as there is no direct correspondence between the change in parametric position and the change in individual quaternion values over the domain, hence leading to inconsistencies and erroneous rotation values between nodes and consequently, leading to improper stiffness and stress-strain fields in the beam. The quaternion interpolation function uses a co-rotational system using nodal averages ($\underline{\rho}$) defined with another exponential map including the basis functions (indicated here as $N(\tau)$) as follows:

$$\rho(\tau) = \underline{\rho} \exp\left(\text{ pure } \sum_i N_i(\tau) \operatorname{imag}\left(\log\left(\underline{\rho}^* \rho_i\right)\right) \right) \tag{2}$$

where the (*) denotes a conjugate quaternion. By taking the derivative (shown by a dot over the function) of this mapped shape function, it is possible to obtain the function for the beam curvature $\kappa(\tau)$:

$$\kappa(\tau) = \sum_i J_{s\tau}^{-1} \dot{N}_i(\tau) \operatorname{imag}\left(\log\left(\rho(\tau)^* \rho_i\right)\right) \tag{3}$$

where J is the Jacobian for converting between the global coordinate system to the local parametric system. The curvature function along with the positions $r(\tau)$ fully define the beam and can be used in computing the internal strains and stresses, for use in further development of the solution. The

combination of these three equations are very difficult to understand mathematically, as there is no clear reference provided in [24] and related papers elaborating on the exponential map or its implementation of the code. An interpretation of this formulation and the complete parametric IGA model is developed in the 'SplineIGA_Halton.py' and can be found in the online repository in Github [43]. However, it is presumed that this section of the formulation is not fully defined due to possible errors, lack of further data for validation and comparison from the references, and practical limitations due to the complexity of the mathematics involved, and needs further clarification for the mapping involved. It is currently very difficult to verify the accuracy of this section of the formulation and its implementation in code as this study could not find much relevant literature or data to compare against.

Another area of difficulty in this model involves the non-linear solution procedure, which uses the conventional FE method by developing the tangential stiffness matrix to solve for the displacements and rotations which is used to update the geometry in each time step. The displacements and rotations are converted to position and quaternion data, and are used to create the internal force vector f_{int} from the governing stress-strain and energy equations. The tangential stiffness matrix is obtained by differentiating the internal force vector with respect to the degree of freedom vector for the system:

$$\boldsymbol{K}_t = -\nabla_q \boldsymbol{f}_{int} = -\left[\frac{\partial \boldsymbol{f}_{int}}{\partial q}\right] \tag{4}$$

The authors mention that this differentiation must be performed numerically. Due to the number of degrees of freedom involved with the model, a direct differentiation scheme was not feasible, and other numerical methods were considered for this code. A numerical function was first created by mapping the linear weightage of each DOF on the terms of the stiffness matrix. The partial derivatives of each term would simply be the corresponding coefficients of these linear functions, hence creating an efficient method of numerical differentiation. In the most recent iteration of the code, a Halton sampling scheme is used to most efficiently cover and perturb the vector space of the DOF vector in a pseudo-random manner and to generate the internal force vector data for several perturbations of each DOF. This data is then fed to a Linear Regressions algorithms which outputs the coefficients or weightage of each DOF on the stiffness matrix hence allowing for the computation of its vector derivative and creating the Tangential stiffness matrix. Currently, this method has been fully implemented in code, however, there are errors of singularity and asymmetry in the final tangential stiffness matrix, indicating problems in this methodology of numerical differentiation, or in the original formulation as indicated previously.

The CUF model described previously has also been briefly studied in this thesis and attempts have been made to encode the formulation as described in [25] and other related work. While some success has been achieved with this methodology, it still poses several practical problems in implementation. Although it is more widely explored and presented in the literature, it uses formulation which is different from conventional FE constructs due to the use of a more complex Stiffness matrix structure. The construction of this matrix requires implicitly following a set of functional equations to create a Functional Nucleus (FN) for the structure, which is then assembled based on the scheme of cross-sectional enrichment adopted. Both the Taylor and Lagrange polynomial methods have been coded for a simple straight beam (can be found in the online repository in Github [43]), although both methods faced problems in achieving consistent results. A possible reasoning for this difficulty could be ambiguity in the implicit formulation for the Functional Nucleus presented in [25], as there were minor textual and mathematical inconsistencies observed in comparison with other work by the same group of authors. However, this avenue has not been pursued in detail in this thesis and has been left open for future exploration as the CUF model needs to be paired with a fully defined 3D center line theory such as the Geometrically-exact model for sufficiently modelling the FRP.

These are the primary areas of focus for future implementation of this beam theory for the purpose

of modelling FRP, and although the present research has shown it to be a very efficient solution for the 3D micro-scale modelling of FRP stress, imperfection, and damage, several practical difficulties have necessitated the use of an alternate strategy to approach this literature gap.

7 The Final Unit Cell model

From the principles proposed in the CUF and Geometrically exact models, the possibility of using a distinct center-line and boundary domain setup was considered in tandem with more conventional 3D finite elements to model the fibre-matrix interactions. Geometrically, the fibres are fully represented by just their shape and their center-line, while the matrix domain can be fully defined with the information of the boundary of the domain (34). Such a geometry could provide the necessary usability and ease of CAD integration required, while allowing for easy modelling with no investment in developing unique mathematical formulations. The formulation described in the following sections can describe the fibre mid-line and cross-sectional profile using an independently developed modelling routine, and an algorithm can be implemented to produce a mesh using higher order wedge elements to populate the cross-section and capture the different phases through the profile, in a model similar to that depicted by [2]. The focus of this Unit Cell model is to be able to capture single and/or multiple fibres in a single Representative domain, with each fibre having individual geometric characteristics to simulate a realistic FRP. The model can be enriched to include perturbations of neutral axis and fibre path while also providing scope for future implementation of damage models, crack-propagation, and other heterogeneities observed in composite materials, while retaining the computational simplicity and producing 3D stress states required for this application. This approach is yet to be investigated in the literature, and it is believed that it can appropriately satisfy the requirements for a comprehensive composite unit cell as concluded by this study, and present a very low practical barrier for implementation at any level of academic study, while providing the opportunity for future improvements and scaling.

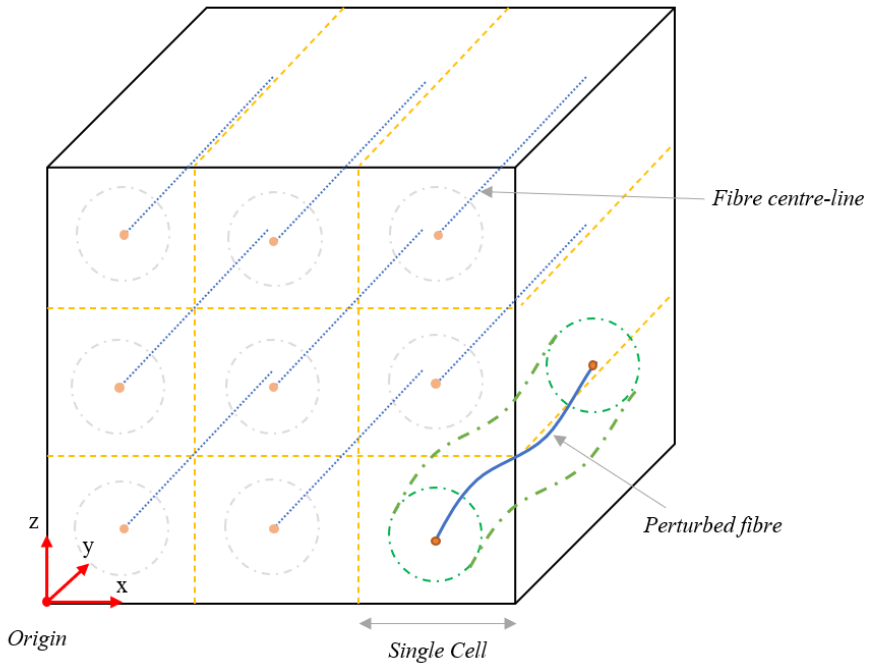

Figure 34: *Representation of the Unit Cell model with 9 cells*

The objective of this model is to provide a stand-alone setup which receives the boundary description of the geometry along with the load cases, and outputs the displacement and stress field in a 3D region including the fibres and matrix. This is setup is essential for conducting virtual component testing on a Representative Volume of a UD Fibre reinforced polymer including fibre undulations, so as to obtain the critical structural constants needed to create a simplified representative beam element. The model

consists of four major routines; the External Mesher: responsible for ordering the boundary nodes and defining the locations of straight fibre paths; the Internal Mesher: reads the external nodes, identifies the fibre mid-nodes and generates the fibre and matrix domains with internal nodes connected by wedge elements; the Perturbation routine: reads the fibre mid-nodes data and returns a stochastically perturbed set; and the Wedge Solver: conventional 3D Linear and Quadratic wedge, can read nodal and element data, material properties for fibre and matrix domain, loads, prescribed displacements, boundary conditions, can output selected effective global material properties based on Caruso 1984 [2], also includes post-processing output with nodal, stress, and strain data in the vtk file format. These routines are designed to work in congruence with each other, and are required to be called by a main routine which assigns where the necessary basic variables are defined and can be parametrically iterated over to generate the necessary data.

7.1 External Meshing Routine

The external boundary of the domain is described by a series of nodes which are then arranged in a regular order through a set of outer connectivity arrays. The objective of this routine is to provide the relevant boundary nodes and arrangements to the internal meshing code so as to automatically detect the fibre midlines and cell boundaries, and populate each individual cell with the relevant internal nodes and elements. The following steps are involved in defining the boundary nodes:

- The primary input of the code is the bounding region of the Representative Volume Element. This region is presently a cuboidal geometry with dimensions x_G, y_G, z_G; representing the three global coordinate axes, and with the fibre profile traversing the 'y' direction.
- The number of fibre-matrix cells along the coordinate axes are taken as inputs (nCellX, nCellY, nCellZ) and are used to generate the boundaries of the individual cuboidal cells within the RV. The dimensions of these boundaries are read as B, L, H (representing the breadth, length, and height) along the x, y, and z directions respectively.
- The number of discrete segments along each boundary of the cell are defined as nL, nB, and nH. This sizing is extended to the rest of the geometry and defines the global mesh size.
- This data is first used to compute the number of external nodes corresponding to different facets of the cell boundary geometry. The number of nodes required for a linear mesh and quadratic mesh (using wedge or brick element) is computed separately and used to initiate the arrays which store the ordered sequence of linear and quadratic nodes. This allows the subsequent routines to decide the set of nodes based on the linear or quadratic wedge defined for a particular analysis.

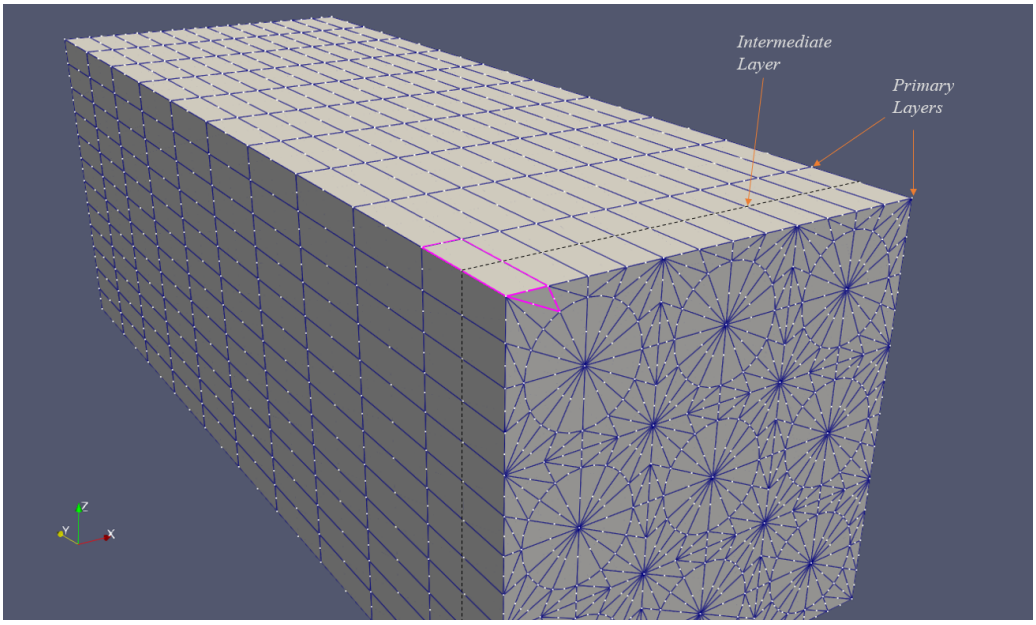

Figure 35: *The primary and intermediate (only quadratic) layers of nodes*

- The RV is assumed to be divided into 'layers' , with each layer representing the set of nodes with a constant y-coordinate (on the x-z plane). These layers are identically imposed at regular intervals along the length of the domain. The boundary nodes in a layer are populated in a structured manner, with the lower left corner of the face being defined as the origin, followed by a raster-like movement towards the positive x-direction, generating a node at fixed intervals along the line $z = nZ(H/nH)$; where nZ is the nth node in the z-direction (perpendicular to the line) directly above the layer origin node. The number of nodes along this line is represented by tNodesXr, which can take one of two values depending on the z-position, as the lines representing the nodes along the edge of a cell boundary (along x-direction) will have more nodes than the other z-positions. Once the nodes of a layer are generated, the next array entry is the first node of the next layer, and this pattern is extrapolated for all the layers in the domain. This forms the BoundaryNodesLin array.

- For the case of a quadratic mesh, a second array BoundaryNodesQuad is populated using a very similar algorithm, adjusting the positions to the midpoint between two adjacent linear nodes. This data is also stored by arrangement in the same layerwise format (iterating over the nodes along the x-direction, followed by z-direction, and then the y-direction.

- The remaining boundary nodes are in the form of the quadratic nodes which lie between the previously defined layers in the intermediate layers (Fig. 35). The pattern followed for this is similar to the Linear Boundary nodes, offset in the y-direction by half the element size along the y-axis, and the result is stacked in the BoundaryNodesQuad2 array.

- The next stage of the code involves generating a connectivity matrix identifying the individual nodes belonging to particular cells, to ease the process of generating internal nodes. This division retains the layer-like structure from the previous steps, but isolates the layers over the a single cell, with the number of linear nodes in a layer of the cell represented by tLayerNodes. This step uses pattern-based algorithms spotting the repeating sequences of node numbers in each x-z face of the cell, also accounting for the nodes shared by adjacent cells. The sequence of nodes in a single layer is read from the bottom-right corner of the x-z face, moving anti-clockwise around the perimeter of the layer. This covers the 4 edges of the quadrilateral formed by the layer, and a distinct pattern is identified for each edge. The node number of the first node of each edge is calculated and stored in the variables 'seed_' and this is used to seed the iterative counter which

moves to the adjacent node in a regularly spaced sequence. This sequence can be easily derived due to the regular nature of the stacking of the Boundary node arrays.

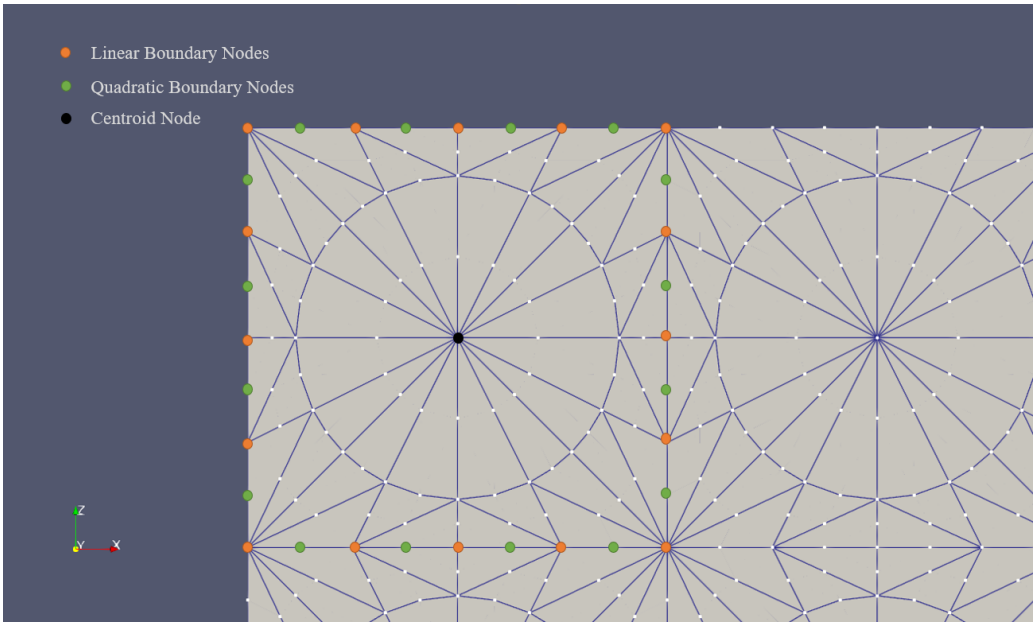

Figure 36: *The Linear, Quadratic, and Centroid nodes for a single layer of a single cell belonging to the boundary mesh*

- The same procedure is repeated for the Quadratic nodes on these faces as well as the intermediate face quadratic nodes, resulting in three arrays, namely BConnLin, BConnQuad, and BConnQuad2 which are of size (nLayers x tLayerNodes x nCells) where nLayers and nCells are the number of layers in a cell and the number of cells in the domain respectively. The structure of these arrays is designed for easy identification of a particular boundary node number when the cell number, layer number, and circular position are known, as is needed by the internal meshing routine.

- The other major task of this routine is to populate the fibre-centroids in the domain in the forms of nCells lines along the y-direction containing the $2nL + 1$ nodes each. The nodes are again segregated as Linear (nL+1 nodes per line) or Quadratic (nL nodes per line), and are stacked in separate matrices. The algorithm iterates over the number of cells along the Y, X, and Z directions in that order, with the x- and z-coordinates being the geometric centroid of the current cell in relation to the global origin, and the y-coordinate calculated as a function of the corresponding layer number.

- On creating the two arrays of Centroid nodes (CentroidListLin, CentroidListQuad), the final algorithm creates connectivity matrices for these nodes, indicating the ordered list of nodes for a particular cell. The number of the first node of each cell is computed and stored in the 'seed_' variable and it is iteratively incremented by one to represent the subsequent nodes along the line. This results in the CentroidConnLin and CentroidConnQuad arrays which possesses the data regarding the set of centroid nodes belonging to a particular cell.

The code returns these key Node-List and Node-Connectivity matrices to the Internal mesher, along with certain key variables indicating the number of nodes in particular geometric entities. An important observation from the mesh geometry is that the boundary described by the external code, when viewed in 3D, is exactly analogous to a construction of conjoined thin rectangular tubes meshed with 2-D linear (4-node) or quadratic (8-node) quadrilateral elements, such as a plate or shell element.

This indicates the possibility of being able to read the mesh data from such a domain meshed with quad-shelled tubes indicating a fibre-matrix cell of variable shapes and sizes over the span of the RVE. This extension could simplify future integration of this micro-scale solver with CAD and FEM software for rapid multiscale modelling of such domains by adding a simple routine to reconcile the two node-numbering schemes.

7.2 Internal Meshing Routine

The internal mesh of the RV is defined by a set of nodes which divide the region into the fibre and matrix domains, and are arranged to form a linear or quadratic wedge element mesh. The objective of this routine is to populate the internal nodes and elements by demarcating the outer circumference of the fibre domain while maintaining compatibility with the external boundary mesh defined in the previous routines. The steps employed in this routine are described below:

- The code is responsible for calling the external meshing routine, and hence reads all parametric variables as inputs from the main routine. The geometric parameters of the RV are packaged in arrays and read as inputs: Cells (indicating the number of cells in each direction), nIntElem (indicating the number of elements in each cell), Dims (containing the global dimensions of the RV).
- The code also reads check-values (0-False, 1-True): quadElems indicating the use of quadratic wedge elements, and perturbCheck indicating the perturbation of the fibre geometry.
- This routine is also responsible for initiating the perturbation of the fibre geometry, and hence reads the base geometry in the form of the fibreRadius, kf (fibre volume fraction), and rhomaxval (maximum fibre perturbation radius).
- On unpacking the variables from the geometry arrays, this data is fed to the external mesher, returning the list and connectivity arrays of the boundary nodes. The boundary and internal nodes are stacked in a GlobalNodeCoords array which is initiated with an arbitrarily large number of rows, as it is difficult to know the total number of internal nodes at this stage. The other global arrays are also initiated in a similar manner at his point, including the Global Element Connectivity matrix (GlobalElemConn) with an arbitrary number of rows and 15 columns (indicating the 15 nodes in a quadratic wedge element), and the list of all element tags (GlobalElemTag) which is a vector containing the material tag for all element numbers, (with 1 indicating a fibre element and 0 indicating a matrix element). These arrays are initiated as global variables so that they can be called, updated, and edited within all necessary functions.
- The Boundary nodes are now updated to the Global node list through the function updateGlobal, which accepts an array of nodal coordinates and appends them to the current Global node list, while also updating the count of total added nodes (gPos). Prior to this, the code, if the perturbCheck variable is set to True, the code calls the fibrePerturb routine to perturb the Centroidal coordinates of the fibre. This routine is further explained in the upcoming sections.
- The number of boundary nodes of each belonging to different categories (linear, quadratic, centroidal, and others) are identified, and the total number of Boundary nodes are stored in the variable tBoundNodes (value is computed differently if quadElems is True).
- Now the interior nodes are to be populated. The current position of the pointer (gPos) in the global node list is saved in the variable intstart, which represents the node number of the first internal node. This value is used as the seed number for the computation of the element connectivity matrices further along the routine.

- The interior nodes are populated over a series of loops, with the primary loop being an iteration over each cell in the domain. This allows the nodes to be arranged in a consistent manner within each cell, with all nodes which are assigned to the same unique domain being arranged in a continuous block of the array. As the nodes are internal to each cell, it is important to note that there are no shared internal nodes, with the cell interactions only captured at the boundary nodes. This distinction allows for ease of future multi-scale modelling, and is an important feature of this modelling system. The following explanation shall cover the stacking system of the nodes belonging to a particular cell.

7.2.1 Arrangement of linear nodes belonging to a cell:

- The nodes within a cell are arranged in a similar layer-wise fashion as the external boundary nodes.
- The first step is to create the nodes (linear) along the fibre boundary. Each fibre node corresponds to a boundary node, and lies on the line joining the fibre centroid and the boundary node (Figure 41). These boundary nodes are extracted from the LayerConn (Layer-wise connectivity), which is a cell-wise element of the BoundaryConnLin array from the external mesher. These nodes are passed into the 'fibrenodes' function, in charge of creating and splitting the lines joining each of these nodes to the layer centroid.
- The code uses the distance and section formula to determine the location of the fibre boundary node. Considering an edge node A (x_a, y_a, z_a) , and fibre centroid node B (x_b, y_b, z_b):
  - $Dist = |A - B| = \sqrt{((x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2)}$
- The position of the fibre boundary node will lie at a distance equal to the fibreRadius away from the centroid along this line. To obtain the vector of this resultant node R, given points A, B, Dist, and fibreRadius:
  - $ratio = fibreRadius/dist$ (The ratio of the section created by point R on the line between A and B)
  - $D = A - B$ (difference of vectors)
  - $R = ratio * D + B$ (section formula)
- The values of R represent the global coordinates of the newly populated internal node on the fibre boundary, and is stored and returned via an interim array INodeCoords (internal node coordinates), and the number of nodes being created is updated and returned from this function (nodesadded). These are updated to the global matrix via the updateGlobal function.
- It is of note that the number of created linear nodes per layer for the fibre boundary are equal to the number of external boundary nodes per layer (tLayerNodes). This helps in calculating the total linear nodes per cell, knowing the number of layers (nLayers).
- The connectivity matrices for the fibre and matrix are initiated. The fibre domain is first meshed with the linear wedge elements. Each element is comprised of 6 nodes, as shown in the figure 37. Nodes 1 through 3 lie on one layer, while the other 3 nodes are on the subsequent layer. Hence the corresponding nodes are exactly tLayerNodes apart, warranting computation for the node numbers of only the first layers.

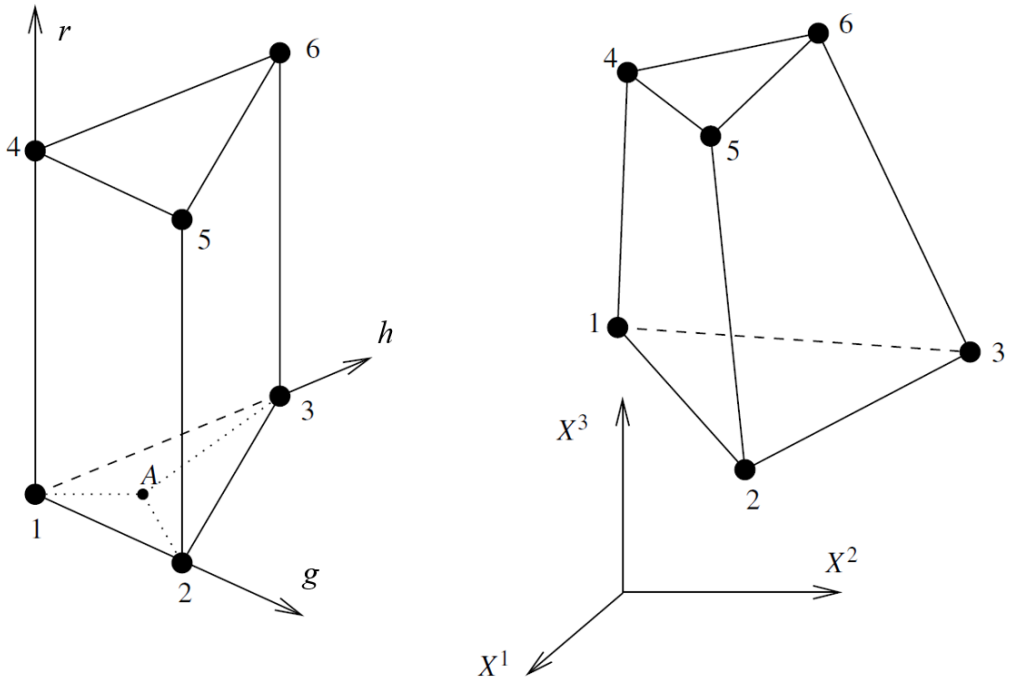

Figure 37: *The Linear Wedge Element*[26]

- On one layer, the nodes belonging to an element form a triangle, with the first node being assigned to the centroid, and the second and third nodes belonging to the list of fibre boundary nodes on this layer, in the descending order of node number (Figure 38). The first node is identified using the Centroid connectivity matrix of the present cell, with this value being added to the total number of boundary nodes (tBoundNodes) which precede it in the global array. This node number, along with the corresponding entry in the adjacent layer, is constant for the remaining elements of the fibre domain, as they all share a common first node.

- The second and third nodes are usually consecutively positioned in the list of fibre nodes. The value of the node number is deduced by counting from the first interior node of the current cell, stored in the variable seedC. This variable is initially set to intstart (defined previously as the value of the first global interior node), and is subsequently updated to the next vacant position at the end of each cell iteration. The value depends on the current element in the layer, and the current element layer (module) being meshed.

- The elements are numbered in the anti-clockwise direction (similar to the layer nodes), with the second and third nodes of the element being numbered based on the number of iterations completed from the seed node, their values being separated by one. Since the elements enclose a complete circular domain, the last and first element share a common node, which is accounted for by identifying the relevant iteration number for this case and using a different count for the common node.

- The same pattern is seen in node 4, 5, and 6 of the wedge, with the appropriate spacing term added. The node numbers of all elements belonging to a cell are stored in a temporary matrix, and are updated to the global Element Connectivity matrix using the updateGlobalEl function. This function takes the input of the temporary connectivity matrix, the number of nodes per element being updated, and the tag of the element, and updates the GlobalElemConn and GlobalElemTag arrays.

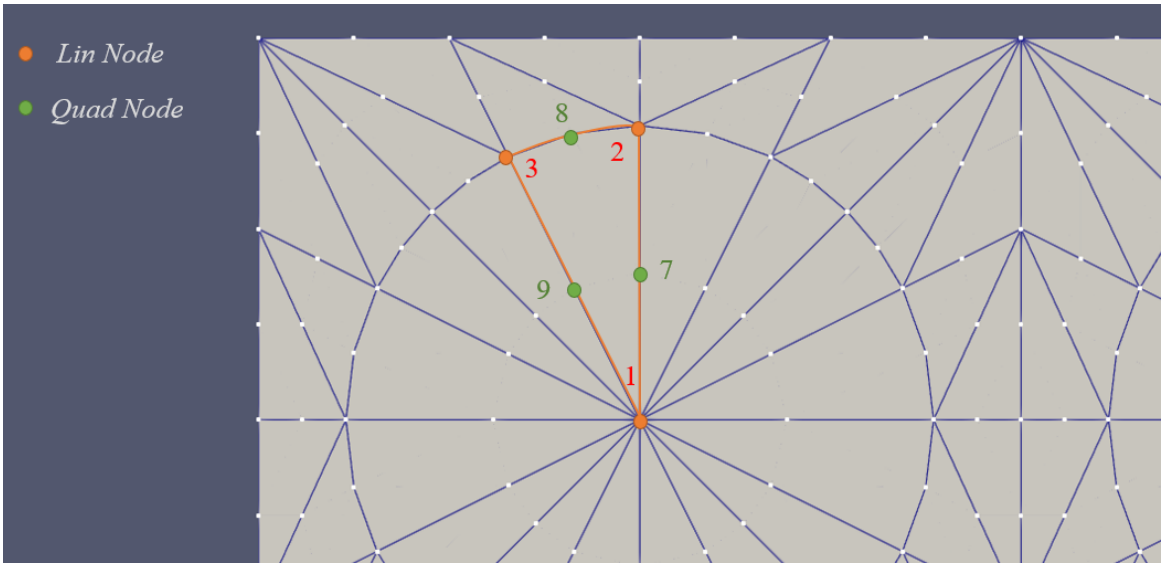

Figure 38: *Node numbering scheme for fibre elements*

7.2.2 Connectivity of Matrix elements:

- The matrix region is also meshed with wedge elements, involving a connectivity between the external boundary nodes and the internal fibre nodes. The assignment of these elements are performed in the same cell loop as before. The layer-wise approach is used here, with a loop iterating over the different element layers or modules in a cell. Within one module, the assignment of elements moves roughly in a similar anti-clockwise fashion, starting from the seed node of the layer. Also, the list of boundary nodes and internal nodes for the cell are split into two layers per module using the splitlayers routine, for ease of handling.

- From the geometry seen (Figure 39), on extending the element borders of the fibre elements, a set of quadrangles are formed in the matrix. These can be imagined as consisting of two triangular regions, which on extension to the adjoining layer, can be formed into two wedge elements.

- These quad formations are identified and the nodes are stored in temporary arrays (sq1 and sq2) in both layers. The quads are labelled (from node 0 to node 3) starting from the lower boundary node value (from the split-layer arrays), moving in an anticlockwise direction over the interior nodes. The quads are iterated over in a circular manner, with appropriate measures taken to account for the last element as previously seen.

- This quad is to be split to form two triangular regions. This split is done along the diagonal of the quad, creating two possible split methods along two opposing diagonals. The distance measuring algorithm is used between the opposing nodes to find the diagonal of greatest length, and this diagonal is considered as the split. This is seen to generally result in more equivalently sized and shaped elements, which can avoid issues of spurious stress concentrations over element borders. Based on the diagonal chosen, the nodes along the diagonal are stored in an array (sl – split line), while the opposing corner nodes are stored in the a different array (sc – split corner). This is used for creating the connectivity of the two elements, with the sl nodes being shared by both elements and sc nodes being independent (within the quad).

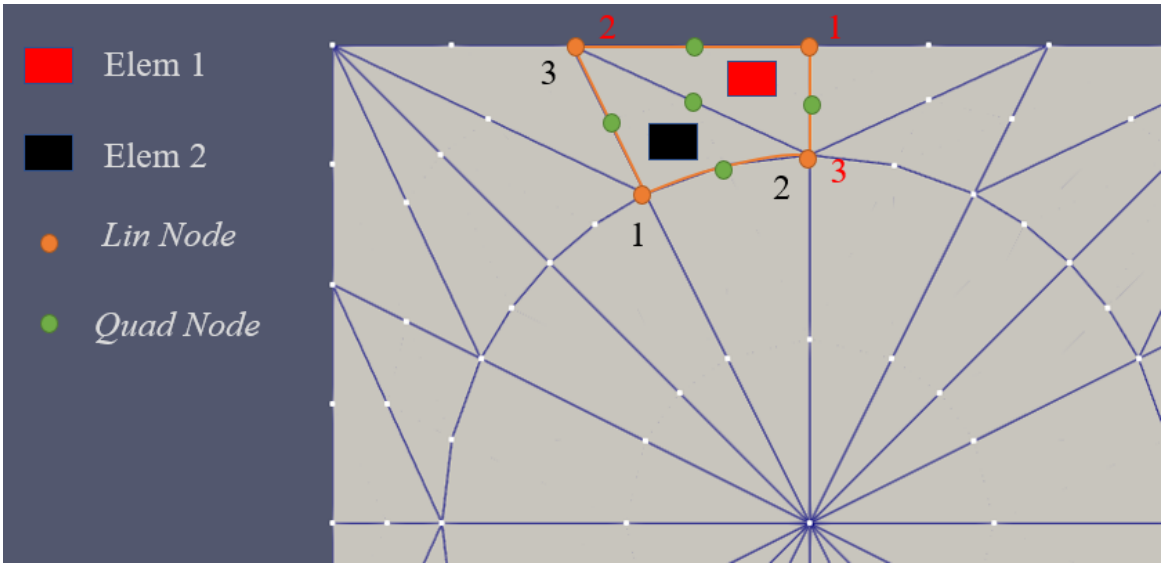

Figure 39: *Node numbering scheme for matrix elements*

- Two interim connectivity matrices, conn1 and conn2 are used to store the node numbers of the two wedge elements formed. While there are a few configurations and arrangements possible, the two ordering schemes chosen are shown in the figure (Figure 39), chosen so as to maintain element normal consistency with the fibre nodes. This step is made easy by the use of the split line and corner arrays, in tandem with the sq arrays.
- The two elements are updated in the ElConnMLin array as consecutive entries, with the array gaining 2 rows per iteration over the quads. The array is updated to the global element array with the tag 0 (indicating matrix) using the updateGlobalEl function.

7.2.3 Quadratic Nodes:

For the case of quadratic elements, each element edge is populated by an additional central node. For this purpose, it is necessary to identify all node pairs and split them to create the third node. This is done in a categorical manner, by creating five different functions for each type of mid-node to be created (quadnodes1 – quadnodes5). Each individual set of nodes is fully populated for a particular cell before stacking it in the global coordinate matrix, following which the next set is created.

- The first set of nodes, termed the 'inner ring nodes' refer to the quadratic nodes lying on the triangular edge radiating from the centroid node. The algorithm to create this node (quadnodes1) follows the same order as the fibre edge node arrangements and uses the section formula to identify the centrepoint of the line joining each fibre edge node with the centroid. The function returns all the inner ring nodes for the current cell arranged in a layerwise manner as before, and this is updated to the global node list.

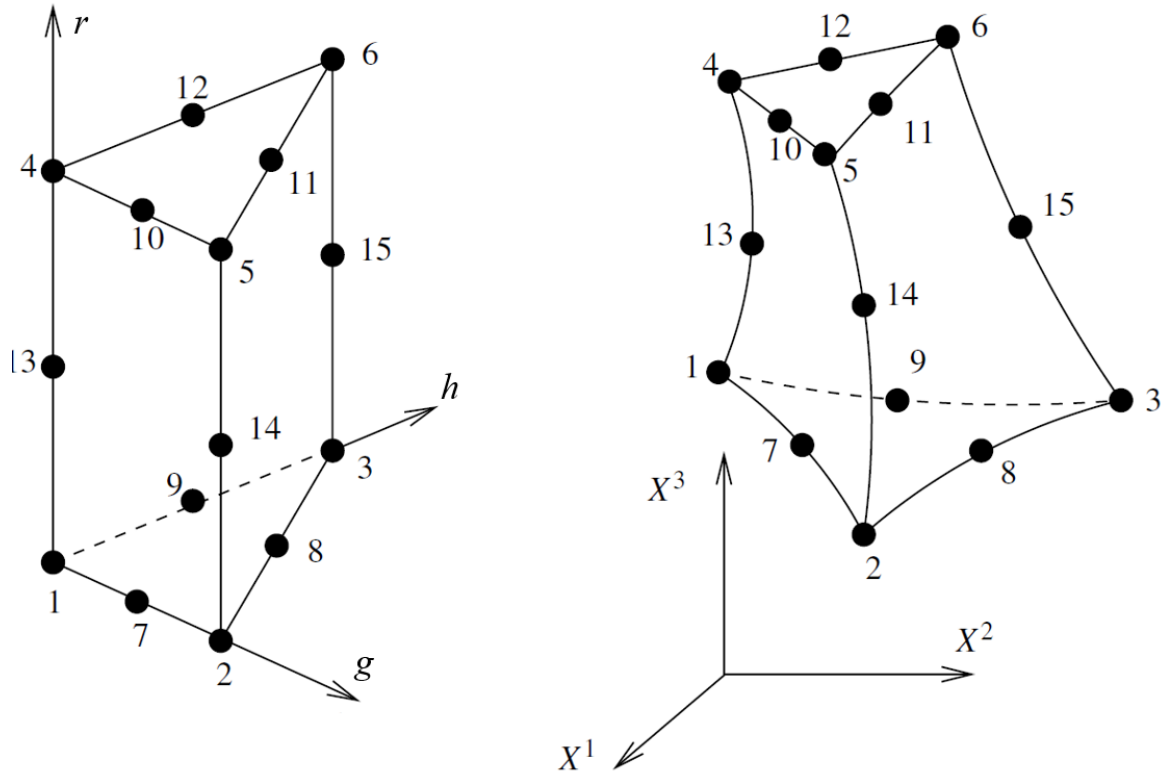

Figure 40: *The Quadratic Wedge Element*

- The next set of nodes, termed the 'outer ring nodes' refer to the quadratic nodes lying on the fibre boundary, bisecting the line joining the linear boundary nodes. These divisions also follow the same ordering as the boundary nodes, to facilitate the logical ordering of the connectivity matrix.

- The quadratic nodes in the matrix are created next. These nodes are of two distinct type, the first being the 'primary matrix nodes', which refer to the quadratic nodes lying along the line extended from the fibre element-edge connected to the centroid. These are created by identifying the linear fibre boundary nodes in the defined sequential manner, pairing them with the relevant linear cell boundary node (there exists a one-to-one correspondence between the two sets of nodes as noted previously), and using the section formula to bisect the line joining the two points.

- The second type of matrix node, refered to hear as 'split matrix nodes' are the midnodes along the element-edge identified as the diagonal or split line, which is the last remaining set of quadratic nodes to be created on the primary layers. The predominant boundary lines are identified again in the same manner as described in the linear element connectivity formulation, with the node pairs along the diagonal taken as the two points which are then to be bisected. The position of this node need not follow a clear pattern unlike the other nodes, and instead depends on the diagonal chosen to form the split matrix elements.

- There is one more set of quadratic nodes to be formed, which represents node-numbers 13-15 in the reference wedge element, found in the layer between the primary layers. These nodes, termed as 'fibre mid nodes' follow the pattern formed by the primary layer linear nodes, and lie on the edge joining the corresponding linear nodes across adjacent layers. The relevant nodes are identified in the layerwise pattern, and their coordinates are bisected to populate these interspersing layers.

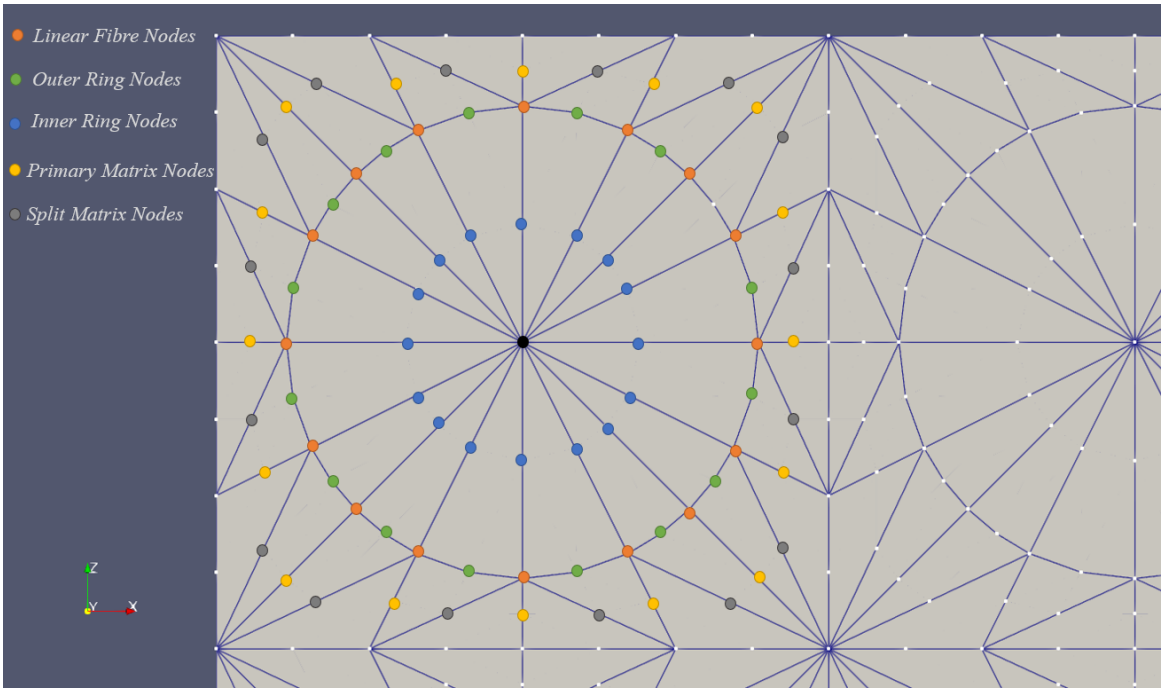

Figure 41: *The Internal Node categories*

At the start of the quadratic node formations, and at the end of the creation and updating of each quadratic node set, The position of the pointer in the current Global node array (gPos) is stored in an list (QList). This acts as an important seed for the creation of the connectivity matrices as described in the next section.

7.2.4 Connnectivity of quadratic elements:

The linear wedge elements are created by forming a connectivity matrix including 6 columns per row, representing six nodes per element (numbers 1-6). Due to the numbering scheme employed for the standard quadratic wedge element (Figure 40), the quadratic nodes are numbered after the linear nodes, from number 7 to 15. This allows for the creation of a separate quadratic connectivity set, carrying the node numbers corresponding to nodes 7 to 15 in the global element array, with nine nodes per element. Two matrices are created for each cell, ElConnFQuad and ElConnMQuad, representing the Fibre and Matrix quad elements. Each of these matrices have 9 columns corresponding to the quadratic nodes in an element, and have rows equal to the number of fibre and matrix elements in a cell. The number of elements have already been computed for the linear case, using the number of elements per layer, and the number of layers per cell.

- The fibre quadratic nodes are added to the connectivity first. Two iterative loops are created, the first cycling over the element modules or layers within the cell, and the second iterating over each element within the module. For each element, the element number is computed based on the current iteration and stored as elnum, while an empty vector of length 9 (qconn) is created to temporarily store the 9 quadratic nodes of that element. Following the order of node arrangements as described by the standard quadratic wedge definition, it is trivial to identify the position of the nodes with the help of the stored seed values in QList.
- The fibre nodes utilise contain nodes from three distinct quadratic sets, the inner ring nodes (set 1), Outer ring nodes (set 2) and the fibre mid nodes (set 5). Node numbers 7-9 lie on a

single layer, nodes 10-12 on the second layer of the module at congruent locations, and nodes 13-15 on the intermediate layer consisting of the fibre mid nodes. Nodes 7,9,10,12 are part of the inner ring nodes which are computed from the seed QList[0], Nodes 8 and 11 are part of the outer ring nodes, computed from QList[1] and the fibre mid nodes are computed using the QCentroids list (Position list of Quadratic centroids) and QList[4].The elnum variable is used to identify the distance of the element from the different seeds across all the modules. An additional if condition is used to handle the boundary case of the last element within a module, to ensure the appropriate connectivity of the shared nodes with the first element.

- Each qconn list is updated to the ElConnFQuad array, and subsequently updated to the GlobalElemConn array using the updateGlobalEl function, with the first input being the ElConnFQuad array, the second input being the number of nodes to be updated (9) and the third input with the tag for fibre (1). The 9 node argument triggers an alternate condition within the updateGlobalEl function, causing the 9 nodes to be appended to the existing 6 nodes of the linear elements. This also triggers the update for the global variable gEq, which represents the current number of updated quadratic elements in the global array, and hence the current pointer location in the quadratic part of the global array.

- The quadratic matrix nodes are now connected in the quadratic elements. This process follows the same principle as the linear method, by identifying each quadrilateral matrix cell to be split into two triangular domains and subsequently two wedge elements. The boundary nodes for the cell are now read and sorted, with the linear node layers in an element module being split using the splitlayers function, and the quadratic node layers being derived and stored in arrays (L1Q, L1Q2, L2Q2) using the LayerConnQ and LayerConnQ2 arrays (from the external meshing routine arrrays BConnQuad and BConnQuad2). It is important to note here that LayerConnQ stores the node numbers for the nodes lying on the intermediate or quadratic layers along the cell-boundary, while LayerConnQ2 stores the values of the quadratic nodes along the primary layers, placed between two linear boundary nodes along the layer.

- These separated layer nodes can be used to identify the nodes on a particular quad-domain. A total of five node sets are created for every 2 wedge elements in the matrix, sq1 and sq2 containing the linear nodes in layers 1 and 2 respectively (4 nodes each), qsq1 and qsq2 containing the quadratic nodes in the respective layers, and qsqm with the quadratic nodes in the intermediate layer. The arrangement of nodes in qsqm mirrors that of the linear arrays. The arrays qsq1 and qsq2 contain 5 nodes each as can be seen in the figure. The arrays are adjusted accordingly for the last element pair in the module.

- The primary diagonal is computed again here for the sake of convenience, and the sc and sl lists are created as before, indicating the shared and independent nodes between the elements. This is useful to have for the sake of ordering the nodes in the intermediate layer. Similar arrays (qsa, qsb) are now created for the quadratic nodes, one for each layer. A simple observation shows the node ordering for the quadratic nodes based on the orientation of the primary diagonal.

- Each loop updates the ElConnMQuad array with two wedge elements, and at the end of a cell, this array is updated to the GlobalElemConn array, with the number of updated nodes set to 9, to trigger the quadratic setup, and with a tag of 0, indicating matrix elements.

7.2.5 Identifying nodes on geometric faces:

For future ease of applying loads and constraints, a function (findFaceNodes) has been added to find all nodes which lie on a single face of the cuboidal RVE. Due to the simple nature of the cuboidal geometry, this face can be represented by two parameters, the axis normal to the face, and the coordinate location

of the intersection of the face along this axis. A simple coordinate search is used to find all nodes on a face, and this list of nodes is returned as the function output. For the purpose of this analysis, the nodes belonging to all 6 faces of the cuboidal RV are found, and stored as nested lists within the list 'Faces'.

On completing the iterations over all cells, the domain has now been fully meshed based on a given set RV dimensions and subsequent boundary node set. This mesh is returned to the main solver in its simplest form, with the GlobalNodeCoords (C), GlobalElemConn (E), GlobalElemTag (Etag), list of face-nodes (Faces), a job name created for output file names (jobname), and the parameters of the RVE (Dims, Cells). The code is also designed to output a vtk file using the vtk_output_format function from the WedgePost.py code, formatted in to plot the linear and/or quadratic mesh. This vtk file is output with '_meshL' or '_meshQ' suffix to the jobname.

7.3 The Wedge Element

The Representative Volume in its simplest form is defined by a cuboidal domain representing a matrix, interspersed with beam-like lines (straight or perturbed) representing the fibre domain, in tandem with a fibre radius. For such a cylindrical domain, wedge elements are suitable as they can divide the domain in equal sections with low variation in element shape and size. It is also possible to enrich the wedge elements with higher order interpolation functions by including more intermediate nodes along the wedge edges. For the sake of this thesis, Linear and Quadratic Wedge Elements are considered, and their respective formulations are presented here. This formulation is used to develop the Wedge_Quadratic_UnitCell_Solver.py code, and the relevant functions will also be detailed here.

7.3.1 Geometry and Interpolation:

The Linear Wedge element consists of 6 nodes, with 3 nodes on the vertices of both triangular faces. The general element coordinates, displacement, and other fields are calculated or assigned to the these nodes, and can be interpolated within the element to fully define the domain. Assume a field f_e which has the nodal values $f_1, f_2, ...f_n$; where n is the number of nodes in the element. The relations between the nodal values and interpolated internal value at a given position x, y, z are given using the following terms:

$$f(x,y,z) = \sum_{i=1}^{n} N_i(x,y,z) f_i \tag{5}$$

where N_i are the shape functions responsible for representing the contribution of each node to the internal interpolation. The order of these shape functions is related to the number of nodes in the element, with higher orders of interpolation requiring additional intermediate nodes in the geometry. Due to the variability in the shape and orientation of the elements, the use of a master element with parametric coordinates is preferred for flexibility in meshing and modelling. The parametric coordinates g, h, r are used here, borrowing from the element library documentation used by ABAQUS [44]. The domain of the parametric coordinates define the boundaries of the wedge element as follows:

$$\begin{aligned} 0 \leq g, h, r \leq 1; \\ g + h \leq 1 \end{aligned} \tag{6}$$

The following shape functions are used for the linear element:

$$\begin{aligned} N_1 =& \frac{1}{2}(1-g-h)(1-r) \\ N_2 =& \frac{1}{2} g(1-r) \\ N_3 =& \frac{1}{2} h(1-r) \\ N_4 =& \frac{1}{2}(1-g-h)(1+r) \\ N_5 =& \frac{1}{2} g(1+r) \\ N_6 =& \frac{1}{2} h(1+r) \end{aligned} \tag{7}$$

The following shape functions are used for the quadratic element:

$$\begin{aligned}
N_1 =& \frac{1}{2}\left((1-g-h)(2(1-g-h)-1)(1-r)-(1-g-h)\left(1-r^2\right)\right)\\
N_2 =& \frac{1}{2}\left(g(2g-1)(1-r)-g\left(1-r^2\right)\right)\\
N_3 =& \frac{1}{2}\left(h(2h-1)(1-r)-h\left(1-r^2\right)\right)\\
N_4 =& \frac{1}{2}\left((1-g-h)(2(1-g-h)-1)(1+r)-(1-g-h)\left(1-r^2\right)\right)\\
N_5 =& \frac{1}{2}\left(g(2g-1)(1+r)-g\left(1-r^2\right)\right)\\
N_6 =& \frac{1}{2}\left(h(2h-1)(1+r)-h\left(1-r^2\right)\right)\\
N_7 =& 2(1-g-h)g(1-r)\\
N_8 =& 2gh(1-r)\\
N_9 =& 2h(1-g-h)(1-r)\\
N_{10} =& 2(1-g-h)g(1+r)\\
N_{11} =& 2gh(1+r)\\
N_{12} =& 2h(1-g-h)(1+r)\\
N_{13} =& (1-g-h)\left(1-r^2\right)\\
N_{14} =& g\left(1-r^2\right)\\
N_{15} =& h\left(1-r^2\right)
\end{aligned} \tag{8}$$

The derivatives of these shape functions against their parametric coordinates, and subsequently the global coordinates are also necessary for the computation of the final stiffness matrix. The shape functions(N) and the values of the first derivatives of the shape functions (dNg,dNh,dNr) are also computed and returned by the 'shapefuns' function for a particular set of parametric coordinates. To obtain the derivatives with respect to the global coordinates, it is necessary to compute the Jacobian matrix. A function 'Jacobian' is created for this purpose, which returns the Jacobian and its determinant, inverse Jacobian, and the global derivatives.

$$\begin{Bmatrix} \partial N_i/\partial g \\ \partial N_i/\partial h \\ \partial N_i/\partial r \end{Bmatrix} = \underbrace{\begin{bmatrix} \partial x/\partial g & \partial y/\partial g & \partial z/\partial g \\ \partial x/\partial h & \partial y/\partial h & \partial z/\partial h \\ \partial x/\partial r & \partial y/\partial r & \partial z/\partial r \end{bmatrix}}_{\mathbf{J}} \begin{Bmatrix} \partial N_i/\partial x \\ \partial N_i/\partial y \\ \partial N_i/\partial z \end{Bmatrix} \tag{9}$$

The Jacobian matrix ($\mathbf{J}$) can be computed using the relation between the global and local coordinates. As the formulation used here is iso-parametric, the coordinate transformation uses the element shape functions as described previously.

$$\mathbf{x} = \sum_{i=1}^{n} N_i(g,h,r)\mathbf{x_i} \; ; \; \frac{\partial \mathbf{x}}{\partial \mathbf{g}} = \sum_{i=1}^{n} \frac{\partial N_i(g,h,r)}{\partial \mathbf{g}} \tag{10}$$

where $\mathbf{x}$ represents (x,y,z); $\mathbf{g}$ represents (g,h,r).

These derivatives are now used to create the strain-displacement relation matrix, termed as $\mathbf{B}$, using the procedure outlined in Quek and Liu [45]. This matrix can be created using the 'Elmat' function

given a set of shape functions and global derivatives of the shape functions. The linear strains in a domain can be written as the sum of partial global derivatives ($\mathbf{L}$) of the local displacements in the following way:

$$\varepsilon = \mathbf{L}U \tag{11}$$

where ε is the strain vector $\{\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \varepsilon_{yz}, \varepsilon_{xz}, \varepsilon_{xy}\}$; U is the displacement vector $\{u, v, w\}$; and $\mathbf{L}$ is the partial differential operator for linear strains:

$$\mathbf{L} = \begin{bmatrix} \partial/\partial x & 0 & 0 \\ 0 & \partial/\partial y & 0 \\ 0 & 0 & \partial/\partial z \\ 0 & \partial/\partial z & \partial/\partial y \\ \partial/\partial z & 0 & \partial/\partial x \\ \partial/\partial y & \partial/\partial x & 0 \end{bmatrix} \tag{12}$$

Since the displacement vector for this element can be written using the shape functions and the nodal displacements in the following manner:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \sum_{i=1}^{n} \begin{bmatrix} N_i & 0 & 0 \\ 0 & N_i & 0 \\ 0 & 0 & N_i \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ w_i \end{Bmatrix} \tag{13}$$

It is possible to write the $\mathbf{B}$ as the action of the $\mathbf{L}$ operator on the shape function array, resulting in the following output:

$$\mathbf{B}_i = \mathbf{L}\mathbf{N}_i = \begin{bmatrix} \partial N_i/\partial x & 0 & 0 \\ 0 & \partial N_i/\partial y & 0 \\ 0 & 0 & \partial N_i/\partial z \\ 0 & \partial N_i/\partial z & \partial N_i/\partial y \\ \partial N_i/\partial z & 0 & \partial N_i/\partial x \\ \partial N_i/\partial y & \partial N_i/\partial x & 0 \end{bmatrix} \tag{14}$$

Here, i represents the number of nodes in an element, and consequently the number of blocks of the $\mathbf{B}$ matrix. The number of columns in the element $\mathbf{B}$ array equates to the number of degrees of freedom (dofs) per node times the number of nodes per element. In this case, each node is allowed only pure translation, and hence 3 dofs representing the tri-axial displacements $\{u_i, v_i, w_i\}$ are assigned per node.

From this strain-displacement relation for an element, it is possible to write the strains and stresses in terms of the material matrix of the element $\mathbf{D}$, the $\mathbf{B}$ matrix, as well as the unknown degrees of freedom (displacements) of the nodes U. This can now be used to compute the element stiffness matrix K_e for the structure [45].

$$K_e = \int_{V_e} B^T D B dV \tag{15}$$

where the material matrix D is written as shown for an isotropic material with Young's Modulus E, Poisson's Ratio ν, and Shear stiffness $G = E/(2(1+\nu))$

$$D = \begin{bmatrix} 2G+\lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2G+\lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2G+\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{bmatrix} \text{ ; where } \lambda = \frac{E\nu}{((1+\nu)(1-2\nu))} \tag{16}$$

The D matrix is computed and returned using the 'Material' function for each element individually, with the input as a nested list of Young's Modulus and Poisson's ratio pairs for each material involved in the structure, and the tag of the current element material.

The integral over the volume of the element required to compute the stiffness matrix is computed numerically using Gauss-quadrature formulation. This entails the weighted summation of the integrand as computed over a list of n_{GP} (represented as ngp in the solver) Gauss points at predefined parametric locations within the elements. To convert the domain of the integral from global coordinates to parametric volume, the determinant of the Jacobian array (detJ) is used.

$$K_e = \sum^{n_{GP}} B^T D B w_{GP} |J| \tag{17}$$

7.3.2 Assembling element and Global matrices:

The main routine of the code uses an element loop which cycles over all the elements in the mesh as received from the internal meshing output. For each element, the material properties are read and the **D** matrix is created based on the tag of the element. For the purpose of this exercise, the individual fibre and matrix domains are considered to be isotropic. The coordinates and connectivity of nodes in the current element are also extracted using the 'Elemnodes' function. The next step is the creation of the element matrices using the formulation described previously. In the solver, this operation is done by looping over the set of Gauss points in the function 'GaussLoop' for each individual element. The set of Gauss Points involves 3 locations along the triangular domain and 2 locations along the length the wedge, resulting in 6 points. This number is chosen for providing the option of extrapolating the field variables to the linear nodes of the elements, to generate continuous fields during post processing (Guido Dhondt [26]). This procedure will be briefly mentioned in a forthcoming sub-section. It is possible to alter the integration scheme by simply adding the required integration points and weights in the function 'GaussPts' and setting the Tngp variable to the required total number of Gauss Points. The parametric location of the coordinates currently used parallel to the trianglular faces are taken to be either $g = h = 1/6$ or $[g, h] = [1/6, 4/6]$ or $[4/6, 1/6]$; and along the length $r = \pm 1/\sqrt{3}$; with each point weighted at $wt = 1/6$.

In the Gauss point loop, the current Gauss point is selected and defined by the variables g, h, r, wt from the list of points, and these values are sent to the 'shapefuns' function to obtain the shape function values and their local derivatives. This is passed to the 'Jacobian' function to obtain the necessary Jacobian data as well as the global shape function derivatives at the current Gauss point. This is passed into the 'Elmat' function to create the **B** matrix as described previously. The element stiffness matrix (Kel) is created as an empty array of size eldof x eldof (number dofs in the element = nodes per elem x dof per node) prior to the Gauss loop, and this is cumulatively added for each Gauss point as shown in Equation 17. An empty matrix is also created for the element load vector (Pel) which can be populated according to applied point loads later in the procedure. It has been initiated at this stage to allow for the possibility of writing a routine to include surface and body forces which require element level gaussian integration similar to the stiffness matrix.

The next stage is the assembly of the element matrices to the global stiffness matrix (BigK). This utilises a combination of the node number ordering as well as the connectivity matrix to make cumulatively add the contribution of each element to the stiffness term of every degree of freedom in its domain. These degrees of freedom are assumed to be numbered as per the node numbering multiplied by a factor of 3 (as there are consistently 3 dofs per node throughout the mesh). This procedure is executed in the 'Assembly' function. Firstly, two identical arrays (rows, cols) are formed by using the connectivity matrix for this element to collect the dof numbers associated with this element. Let the ith entry in the connectivity matrix be a reference to node n. The degrees of freedom associated with this node will be numbered $3n, 3n+1, 3n+2$. These values are now placed in the positions $3i, 3i+1, 3i+2$ in the 'rows' array. Similarly, for an index j, the jth entry of the connectivity matrix and the subsequent dof numbers will be added to the corresponding positions in the 'cols' array. Indices ii and jj are used to represent the indexing of these new arrays, with the range equalling the number of dofs in the element. To write the assembly algorithm, it is necessary to show the correspondence between the global and local stiffness matrices, and this can now be depicted as follows:

$$Kel[ii, jj] \longmapsto BigK[rows[ii], cols[jj]] \tag{18}$$

With this correspondence established, it is now possible to cumulatively add the corresponding terms from the element matrix to the global matrix (initially populated with zeros) in an efficient way by grouping together rows and columns and minimising the frequency of array accessing. The 'BigP' matrix is now populated using the 'PointLoads' function to give the global force vector for the structure.

7.3.3 Loads and Constraints:

The loads, constraints and boundary conditions (clamped nodes) are recorded using the arrays 'Loadset', 'Constset', and 'Clampset' respectively. The next step is to enforce these constraints on the structures. The 'Loadset' list stores a list of instructions containing the node number, dof, and Load value to be applied. This is used by the PointLoads function as mentioned previously. The set of constraints are recorded in a list 'Constset' which primarily is used to prescribe displacements to the structure. It is stores a set of lists each consisting of a node number, dof number, and magnitude of applied displacement, as instructions for the final trimming and constraining function. The 'Clampset' list just contains the set of node numbers which are to be clamped, as it implies that the displacement on all dofs of these nodes are to be set to zero.

The Clamp and Constrain set are fed to the function 'TrimBC' along with the BigP vector. This function returns a list of boolean arrays (bu : constrained and bk : free), where the pertinent values are set to True, while the rest are maintained False. It also returns the integer arrays with similar information for the sake of slicing the BigK matrix (sliceu and slicek) identifying the constrained and free degrees of freedom. The last output from this function is an updated displacement vector (Xvals) which lists the current displacements at the timestep zero for each dof. By default they are set to zero, and are updated based on the Constset list.

7.3.4 Solution:

The system can now be solved for static linear displacement with a single timestep based on the equivalence of internal and external force.

$$KU = F \tag{19}$$

where K is the global stiffness matrix, U is the global displacement, and F is the global load vector.

This system is solved in the conventional method of segregating the constrained and free degrees of freedom and simplifying the involved matrices using the lists generated by the 'TrimBC' function, the procedure for which can be found in (Ref.[45]). To greatly improve the solution efficiency and memory allocation of the solution process, the stiffness matrix is created and operated as a sparse matrix using the 'scipy.sparse' and 'pypardiso' libraries.

7.3.5 Post Processing:

The main output of the solver up to this point is the solved displacement vector for the static structural analysis conducted. The code is also capable of using this data to generate the strain and stress fields over the element gauss points, and can output the geometry data, displacements, and stresses and strains in the form of a vtk file for use in commercially available post processing software.

The 'postProcessing' check variable can be set to '1' or 'True' to trigger the stress and strain computations. The vtk formatting function 'vtk_outputgauss_format' and the mesh deformation function 'ElemDisp' are imported from the 'WedgePost.py' module. The procedure involves the element-wise computation of the new mesh coordinates after the static deformation step and the corresponding element stress and strains. It is necessary to generate some of the previously obtained matrices again at his point by reusing existing functions. The parametric Gauss points are generated for the reference element using the GaussPts function as described previously, and the Elemnodes function is used to extract the element nodes and connectivity. The element data is passed to the 'ElemStrainGP' function to calculate strains and stresses at the gauss points of the element. Here, the shape functions, derivatives, and Jacobians are recalculated for each gauss point to generate the **B** matrix. The strains and stresses can now be calculated from the strain-displacement matrix as follows:

$$\varepsilon_{el} = BU_{el} \; ; \; \sigma_{el} = D\varepsilon_{el} \tag{20}$$

At this point, it is also possible to generate exact nodal stress and strain values using the 'Extrapolation' and 'GPtoNode'. For this, however, it is necessary to have a one-to-one correspondence between each gauss point and a node in the element, and hence the number of gauss points and the ordering need to be changed to equal the number and position of the nodes. The formulation for this procedure is borrowed from Guido Dhondt [26], where the author mentions the necessity of first computing stress, strain and other field variables over the gauss points as the accuracy of these results rely on the integration scheme, which is highest at the gauss points. Particularly, this method is necessary for the detailed application of 3D stress states such as damage and failure analysis in composites.

On considering any one component of strain ε, the strain at the jth Gauss Point ε_j^{GP} can be written as the interpolation of the nodal strains using the shape functions computed at the parametric coordinates of the jth Gauss Point $[g, h, r]_j^{GP}$ as follows:

$$\varepsilon_j^{GP} = \sum_i^n N_i^{GP} \varepsilon_i^{ND} \tag{21}$$

where ε_i^{ND} are the nodal strains; N^j are the shape functions at the jth Gauss point. On stacking

all the strain or other field value of each gauss point in a vector $\{\varepsilon\}^{GP}$, it is possible to rewrite the equations in matrix form:

$$\{\varepsilon\}^{GP} = [A]\{\varepsilon\}^{ND} \tag{22}$$

where $[A]$ is the matrix of shape function values (coded as the variable 'Extrapmat'), with each row consisting the shape function values at that particular Gauss point. Since the values of strain and/or stress are already known at the Gauss Points as described previously, it now possible to find the exact nodal strains using the inverse relation:

$$\{\varepsilon\}^{ND} = [A]^{-1}\{\varepsilon\}^{GP} \tag{23}$$

where $[A]^{-1}$ is coded as the variable 'Extrapmatinv'. The current function 'Extrapolation' calculates and returns the Extrapmat and Extrapmatinv for each case to be passed on to the 'GPtoNode' function. While this functionality has been provided for future use, this thesis will not be exploring the stress and strain data of the Unit Cell in detail and hence to reduce computation effort (requiring 15 Gauss points for quadratic elements), this feature has not been used.

The output from the post processing section is returned in 2 vtk files representing the deformed mesh (filename = 'jobname_disp_output.vtk') and the Gauss point field data format (filename = 'jobname_Gauss_output.vtk'). It is also possible to view the deformations over the Gauss points using software such as ParaView, as the Gauss displacements are also written to the vtk file as the displacement field.

7.4 Effective Material Properties

One of the important results and method of verification for the Unit cell model formulated in this thesis is the ability to predict the global material properties of a Uni-directional composite based on the average stresses and strains experienced over the micro-mechanical domain. This depicts a good interaction between the fibre and matrix elements and validates the FEM used. The approach followed here is based on the work of Caruso [2]. These values are computed in the Wedge_Quadratic_UnitCell_Solver.py code following the post-processing section. The approach used to calculate the effective properties is by applying a prescribed displacement on a face of the RV and measuring the load or displacement response at other faces to compute the material constants. Since the loaded faces and probed faces will change based on the value required to be computed, a separate coded (MainRoutine_Validation.py) is written to call the solver with the appropriate variables. The selection of the load and probe faces are passed in terms of face numbers, which are assigned in the Internal Meshing function ('Faces' array), and this order can be changed based on user preference. For the sake of this thesis, the order being used is as follows:

$$\begin{aligned} Faces[0] &\longmapsto surface|y = 0 \\ Faces[1] &\longmapsto surface|y = y_G \\ Faces[2] &\longmapsto surface|z = 0 \\ Faces[3] &\longmapsto surface|z = z_G \\ Faces[4] &\longmapsto surface|x = 0 \\ Faces[5] &\longmapsto surface|x = x_G \end{aligned} \tag{24}$$

Based on this key, the required face numbers are passed to the solver by the MainRoutine_Validation as lists for load application ('Loadfaces'), clamped boundaries ('Clampfaces') and the axes along the direction of load application ('Loadaxes'), where $\{0, 1, 2\}$ represent the directions $\{x, y, z\}$. It is possible to iterate over each of these parameters as well as other parameters such as Fibre volume fraction and cell size, which refers to the number of fibre-matrix sections are present in one Unit Cell.

The following global properties are evaluated using in this code:

- Longitudinal Young's Modulus: A uniform displacement u is applied on the face $y = y_G$ along the fibre direction (+y). The face $y = 0$ is clamped, and the resultant force over this face (F) is computed using the Reaction force routine. The Longitudinal Young's Modulus is then computed as:

$$E_{L11} = \frac{F y_G}{A u} \tag{25}$$

  where A is the area of the clamped face. This simulation is triggered by passing a True value to the array entry findEL[0]. The solver uses specific functions to generate the necessary values described above. The total force acting on a face along a defined direction is computed by the NetForce function, which iterates over all the relevant nodal dofs of the updated BigP vector belonging to a particular face and sums up the individual force values. Since BigP has been updated with the calculated reaction forces, it is possible to obtain the net force F required to compute E_{L11}.

- Poisson's Ratio: This value is computed in the same simulation as the Longitudinal Young's Modulus with the same input parameters and findEL[0] set to True. To calculate the Poisson's ratio, the total displacement in the direction perpendicular to load application u_x is compared to the prescribed displacement along the fibre length as follows:

$$\nu_{L12} = \frac{v_{avg}/x_G}{u/y_G} \tag{26}$$

The value of v_avg is interpreted as the sum of average total displacement across the faces $x = 0, x_G$. These average displacements are computed using the AvgDisp function with the nodal displacements taken from the updated Xout vector probed at the relevant nodal degrees of freedom lying on the faces.

- Transverse Modulus: This evaluates the modulus of elasticity in the matrix dominant direction perpendicular to the fibre path. This simulation is run when the findEL[1] array element is set to True. Similar to the Longitudinal modulus, a prescribed displacement v is applied in the +x direction, this time to the $x = x_G$ face, while the face $x = 0$ is clamped. Since the structure is orthotropic, it is possible to interchange the x and z directions along the breadth and height to obtain the same result. The reaction force at the clamped face is recorded using the NetForce function, and the following formulation is used:

$$E_{L22} = \frac{Fx_G}{Av} \tag{27}$$

where A is the area of the face $x = x_G$

- Transverse Poisson's Ratio: This value is also computed with the findEL[1] calculation. under the same conditions as described previously, the sum of average displacements along the surfaces $z = 0, z_G$ is compared against the prescribed displacement as follows:

$$\nu_{L23} = \frac{w_{avg}/z_G}{v/x_G} \tag{28}$$

where w_{avg} is calculated using the AvgDisp function.

- In-plane Shear Modulus: This value is computed on setting the findEL[2] array entry to True. The structure is clamped at the $z = 0$ face with a shearing prescribed displacement applied on the face $z = z_G$ in the +y direction (fibre direction), with x and z being interchangeable. It is necessary to ensure that the loaded surface moves parallel to the clamped surface without any out of plane deformation. To achieve this, a second set of displacements is prescribed by forcing $w = 0$ along the +z direction at the face $z = z_G$. The reaction shear force (F) is measured along the fibre direction on either the clamped or loaded face, and the shear modulus is calculated as:

$$G_{L12} = \frac{F/A}{u/x_G} \tag{29}$$

For pure shear calculations, it is necessary for the face $y = 0$ to deform linearly and remain planar after the displacement. However, due to the use of higher order elements in this model, it is difficult to achieve this.

- Transverse Shear Modulus: This value is computed on setting findEL[3] to True. The structure is clamped along the face $z = 0$ and loaded along the face $z = z_G$. The prescribed displacement is applied in the x direction while maintaining the same parallel constraint as used for the in-plane shear modulus. On finding the shear reaction force at the clamped face, the formula used to calculate the transverse shear modulus as:

$$G_{L23} = \frac{F/A}{w/x_G} \tag{30}$$

The solver function (validation_solver) in the Wedge_Quadratic_UnitCell_Solver.py code returns the results of the effective material properties according to the input array findEL, which is a list of size = 4 set to zero by default. On setting either findEL[0] or findEL[1] to True or 1, the code outputs two material properties (Tensile modulus and Poisson's ratio). For the next two test cases, the code outputs a single Shear modulus value and the second output is set to 0. If no value of findEL is set to True, both return values of the function are 0. However, all calls of the validation_solver function generate the post-processing vtk files and displacement solution as described previously by default.

7.5 Perturbation of Fibres

The objective of this model is to demonstrate the ability to capture micro-scale geometric imperfections of fibre-matrix composites which are commonly occurring due to manufacturing methods currently in use. For this, it is necessary to show the ability of the model to generate such geometric imperfections as a part of the stand-alone generator and solver without the use of any additional CAD or CAE software. Since the structure of the Unit cell is defined by the cell dimensions and centroid positions, it is possible to perturb the fibre centroid positions to create undulations in the geometry prior to generating the internal mesh. There are several possible methodologies to achieve this, with the objective being to maintain representative variables in the model capable of capturing realistic arrangements and geometry. One of the popular approaches is the use of splines, where the control points of the spline are perturbed based on a stochastic model in such a way that the fibre angle misalignment can be quantified and equated with experimental statistics, as described by Catalanotti and Sebaey [22], and further expanded upon in more recent research including experimental and FEM simulations ([46, 47, 48]). A brief explanation of such a stochastic model is presented here. The fibre is represented by a center line as shown in Figure 42 is taken to be oriented along the x-axis prior to perturbation in this example. The fibre center line is attributed with three parameters, $\{\alpha_{xy}, \phi_{yx}, \phi_{zx}\}$ which represent the fibre angle, out-of-plane misalignment angle, and in-plane misalignment angle respectively. Knowing the unit vector for any at any point along the center line curve, it is possible to deduce the misalignment angles using the vector projection angles as denoted in Ref [22]. These misalignment angles can be statistically compared to experimental results.

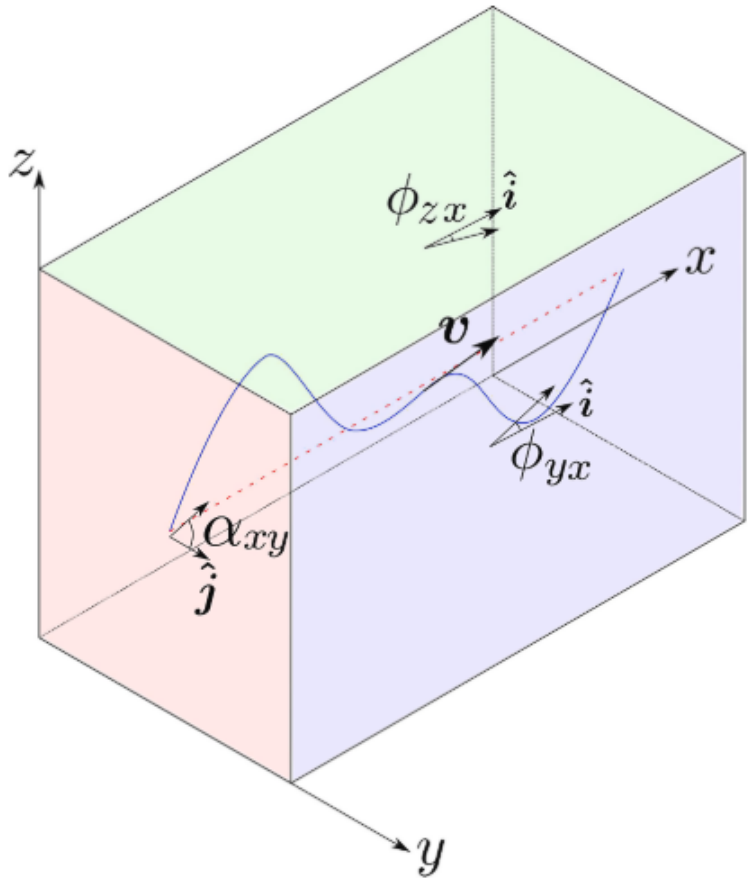

Figure 42: *Fibre mid-line path with the misalignment angles*[22]

The center line can be mathematically modelled as spline curved, with the use of control points

for the perturbation. For the present Unit Cell, since the centroid points are already generated and interpolated using element shape functions, it is sufficient to perturb the position of these points for the purpose of creating undulations. The centerlines are not perturbed along the fibre direction as this does not have any effect on the final geometry, and hence a two-parameter system can be used to represent the points. A polar coordinate system is chosen to represent the perturbed point positions to provide a controlled distribution of points across all directions (θ) in the plane perpendicular to the fibre direction. The use of polar coordinates also provides control over the amplitude of perturbation or the distance from the original straight fibre layout in the form of the radius parameter (ρ). This parameter is shown to have significant effect on the loss in stiffness of the FRP. The authors of Ref. [22] describe an iterative procedure to perturb the lattice of centroid points over several fibres, by creating a randomised array of the centroids to iterate over, with one random point being perturbed every iteration. This allows the overall statistics of the Unit Cell to be controllable by tailoring the displacement of each subsequent iteration to approach the desired statistic.

For this thesis, the objective is to demonstrate the ability of the model to capture changes in stiffness due to small perturbations in the fibre layout, while also providing a base for future expansion of the model to capture realistic undulations and stochastic models. The model has been coded using the basic geometric features from the Reference [22] and subsequent work, with several simplifications for practical implementation. The perturbation routines are written in the code 'Perturbation_Random.py' and are called from the solver 'MainRoutine_Perturbation.py', which is a variation of the 'MainRoutine_Validation.py' adapted for single or low parameter variation runs including perturbation. The Perturbation.py code contains routines for the generation of the Bezier curves and Control points for possible future use, but these are not used for the current thesis. The iterative model from [22] is also used here, with the control points in their original order, to fully implement the stochastic model, it is possible to rearrange this array in several suitable ways. For each iteration, there is a displacement imposed upon the control points based on the polar parameters, described as follows:

$$u_i = \rho_i\{cos\theta_i, sin\theta_i\} \tag{31}$$

where the two coordinates represent the in-plane axes systems perpendicular to the fibre direction. In the code, this represents the $\{x, z\}$ direction displacements. The displacement along y or the fibre direction in the code is zero.

The angular position of the displacement is described using a random variable function as follows:

$$\theta_i = 2\pi\mu \tag{32}$$

where $\mu \in rand[0, 1]$ is a random variable with a uniform distribution over the range between zero and one, and hence θ_i covers the domain from 0 to 2π.

The radius parameter is also expressed in terms of a continuously distributed random variable, and is described as follows:

$$\rho_i = \lambda_i\bar{\rho}_i \tag{33}$$

where $\bar{\rho}_i$ is the maximum possible radius of the displacement based on geometrical constraints within the system; λ_i is a random variable between 0 and 1, following a continuous distribution based on the user's input. It is possible to use $\lambda_i = \mu$ as mentioned in Ref.[22], and this shall also be used in this thesis for demonstrating the ability of the current model to captures such variations. This results in a

radius function where $\rho_i \in [0, \bar{\rho_i}]$ is a uniformly distributed random variable. The value of $\bar{\rho_i}$ is decided based on the dimensions of each cell in the RV, and cannot exceed the difference between the side length of the fibre-matrix cell and the fibre radius. This calculation is performed in the singleruns.py code and the value 'rhomax' is passed to the 'rhomaxk' function in the Perturbation_Random.py code. The function subtracts a buffer amount from the actual maximum radius to avoid zero volume mesh elements in the matrix. In more advanced geometries, it is necessary to consider aspects such as contact between adjacent fibres as in Ref. [22], which generally requires higher number of iterations and computations to achieve the necessary statistical distribution.

Future Scope for stochastic model:

In this setup, it is possible to implement the realistic statistical models in the future, the methodology is briefly described here and found in further detail in several publications [22, 46, 47, 48] including FEM and experimental results. The general methodology involves assuming a statistical distribution of fibre misalignment angles based on experimental observations. From [22], the most appropriate distribution is the von Mises distribution with the following distribution:

$$g(\phi; \mu, \kappa) = \frac{1}{2\pi I_0(\kappa)} \mathrm{e}^{\kappa \cos(\phi - \mu)} \tag{34}$$

where μ is the mean direction (which has a most likely estimate of 0 in this case as the fibre original direction has no misalignment), ϕ is the misalignment angle, I_p is the modified Bessel function of order p, and κ is the concentration parameter which is the critical value which defines the degree of misalignment.

Consider that the perturbation algorithm has completed N iterations over the centroid points and randomly perturbed them as described thus far. Given this set of perturbed fibre centroid coordinates, it is possible to generate a set of misalignment angle values using vector projections as mentioned previously. On fitting these angles to the von Mises distribution, it is possible to obtain a Maximum Likelihood estimate for the concentration parameter for the current configuration as follows:

$$\hat{\kappa} = A^{-1}(\bar{C}) \tag{35}$$

where A can be written as a function of κ as follows:

$$A(\kappa) = I_1(\kappa)/I_0(\kappa) \tag{36}$$

and $\bar{C}$ is the Mean resultant length of the misalignment angles with a zero mean, expressed as:

$$\bar{C} = \frac{1}{N} \sum_{i=1}^{N} cos\phi_i \tag{37}$$

The inverse function of A can be computed numerically using Newton iterations to obtain $\hat{\kappa}$. It is now possible to minimise the Standard Error of the Maximum Likelihood Estimate of the concentration parameter with respect to experimental values $(SE(\kappa(\hat{\lambda}_i)))$ using different statistical methods, as described in [22], hence creating the best fit for the experimental distribution. This method uses several complex stochastic concepts for which a direct closed-form solution or numerical approximation cannot be found readily in the available literature, and needs to be derived from core principles using esoteric mathematics. For this purpose, a further investigation of this methodology is beyond the scope of this report, but is definitely an approach which is viable in the future with the current model.

8 Results

The wedge element-based stand-alone 3D FE model with internal and external meshing algorithms for fibre-matrix RV geometry has been fully described including its implementation in code. This section now demonstrates the ability of the code to model a straight fibre composite with fibre undulations effectively at the micro-scale. The solver has been setup to primarily generate results for effective material properties of the Unit Cell, and hence the results here will be presented in this form. The results presented here are some preliminary simulations intended to demonstrate the capabilities and deficiencies of the developed unit cell model and to present a validation for the model concept and FEM code developed. The results are also intended to demonstrate the scope of the model for further development for predicting the realistic micromechanical behaviour of imperfect composites with high efficiency.

The verification of Effective material properties of the composite was conducted against the values computed by Caruso [2] for a fibre reinforced composite material. The material considered here is a high modulus epoxy resin (HM-epoxy) reinforced with Boron fibres, where both the fibre and matrix are considered to be independently isotropic. This requires just material variables per material to populate the D matrix as described in section:

$$\begin{aligned} E_f &= 58.0 \text{ Mpsi } ; \nu_f = 0.20 \\ E_m &= 0.75 \text{ Mpsi } ; \nu_m = 0.35 \end{aligned} \tag{38}$$

where the subscripts f and m refer to the fibre and matrix respectively. The list of effective material properties which are considered here are described in section 7.4. The material properties are calculated for different fibre volume fractions covering the range $v_f = \{0.622,\ 0.466,\ 0.224,\ 0.069\}$. To obtain these fibre volume fractions, the Unit Cell domain size is changed while the fibre diameter is maintained constant. For the Boron fibre, the diameter is assumed to be $d_f = 0.0056in$. The results are depicted in Tables 1 and 2, listing high and low volume fractions respectively, each covering 2 iterations of v_f values.

Table 1: *Effective Material Properties for different mesh types and high fibre volume fractions compared against Caruso* [2] (with $v_f = 0.622, 0.466$)

v_f	Mesh Type	E_{L11}	ν_{L12}	E_{L22}	ν_{L23}	G_{L12}	G_{L23}
0.622	Present SC 2x2	36.3827	0.2495	4.4675	0.4338	1.2237	0.7862
	Present MC 2x2	36.4216	0.2346	4.4507	0.2744	1.1648	0.6825
	Present SC 4x4	36.4072	0.2414	4.3712	0.3620	1.2236	0.7083
	Present MC 4x4	36.4455	0.2326	4.3993	0.2586	1.1644	0.6423
	Ref. [2] SC	36.4	0.251	4.35	0.413	1.35	1.49
	Ref. [2] MC	36.4	0.245	4.60	0.212	1.35	1.49
	Ref. [2] SME	36.4	0.257	3.39	0.340	1.26	0.722
0.466	Present SC 2x2	27.4492	0.2722	2.5839	0.4875	0.7389	0.5261
	Present MC 2x2	27.4830	0.2563	2.5654	0.3473	0.6949	0.4410
	Present SC 4x4	27.4676	0.2666	2.4793	0.4158	0.7354	0.4888
	Present MC 4x4	27.5011	0.2551	2.4862	0.3397	0.6911	0.4217
	Ref. [2] SC	27.4	0.273	2.31	0.467	0.795	0.921
	Ref. [2] MC	27.4	0.269	2.44	0.296	0.795	0.921
	Ref. [2] SME	27.4	0.280	2.30	0.372	0.856	0.515

The tables show a comparison between the material properties of different Unit Cell specifications for the current model (referred to as 'Present' in the Mesh Type column) against different specifications used by Caruso (referred to as 'Ref. [2]' in the Mesh Type column). The two primary mesh types used both by Caruso and the present model are the single-celled model (SC) which uses a single fibre-matrix module in the Unit Cell, and the multi-celled model (MC) which uses a 3x3 grid of fibre-matrix modules in the Unit Cell (Figure 43). The work by Caruso also covers the same cases of a 1 and 9 cell RV, with a mesh comprising of 8-noded brick and 6-noded wedge elements. Additionally, Caruso also presents the Effective material properties calculated using analytical equations, which is termed as Simplified Micromechanics Equations (SME) which are further elaborated in Ref. [2]. For the current model, two different mesh sizes are considered, referred to as '2x2' and '4x4'. This represents the number of divisions along the line representing the side of a single cell along the breadth (nB) and height (nH) along the x and z directions, and having a constant 10 divisions along the length (nL) or fibre direction. It can be observed that Caruso uses only 4 divisions along the length, as this does not have much effect on the prediction of effective material properties for straight fibre cases. However, 10 divisions are used consistently in this thesis to adequately capture perturbation of the fibre centroid line in subsequent analyses. All results for elastic moduli are presented in 'Mpsi' while the Poisson's ratios are non-dimensional.

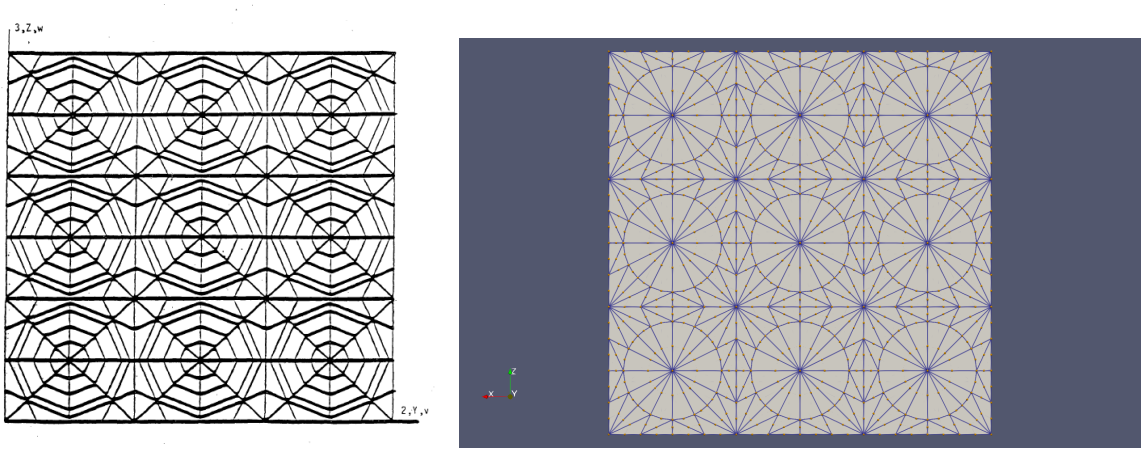

Figure 43: *Multi-cell (9 cell) mesh used in a) Caruso [2], b) Present* (4x4 MC), v_f =0.466

A preliminary observation of the data shows good agreement in general of the current model with both the results of Caruso and the presented analytical formulae, hence validating both the Unit Cell and FE model developed in this thesis. The simulations for Longitudinal Modulus (E_{L11}) is notable as it shows very low deviation in changing the model or mesh in the current analyses, and also in the simulation of Caruso. The results are all in excellent agreement with analytical models, and this provides an excellent baseline for conducting tests to predict loss in tensile/compressive stiffness along the fibres after introducing perturbations. There is also very good agreement between the 2x2 and 4x4 mesh observed, indicating that the cross-sectional mesh density does not significantly affect the simulated longitudinal stiffness of the Unit Cell. The only point of interest is the slightly higher longitudinal stiffness predicted by the MC model over the SC model over all volume fractions in the current unit cell, which is only reflected from the Caruso paper in the $v_f = 0.224$ section of Table 2, possibly due to the truncation of the decimal expansion in that work.

On observing the data of the transverse tensile modulus (E_{L22}), there is a larger discrepancy between the 2x2 and 4x4 mesh sizes, indicating that the matrix-dominant behaviour is more sensitive to the cross-sectional mesh density. The values simulated by Caruso also show a significant deviation from the calculated analytical SME values over all volume fractions. However, the proximity of the present results to the SME values are seen to vary depending on the volume fraction. For higher volume

Table 2: *Effective Material Properties for different mesh types and low fibre volume fractions compared against Caruso* [2] (with $v_f = 0.224, 0.069$)

v_f	Mesh Type	E_{L11}	ν_{L12}	E_{L22}	ν_{L23}	G_{L12}	G_{L23}
0.224	Present SC 2x2	13.5904	0.3095	1.4121	0.4989	0.4224	0.3266
	Present MC 2x2	13.6212	0.2915	1.3800	0.4162	0.3788	0.2814
	Present SC 4x4	13.5993	0.3079	1.3591	0.4485	0.4181	0.3159
	Present MC 4x4	13.6298	0.2914	1.3378	0.4114	0.3749	0.2762
	Ref. [2] SC	13.5	0.311	1.20	0.417	0.445	0.523
	Ref. [2] MC	13.6	0.309	1.24	0.359	0.446	0.524
	Ref. [2] SME	13.6	0.316	1.41	0.417	0.522	0.357
0.069	Present SC 2x2	4.7168	0.3333	0.9995	0.4526	0.2974	0.2468
	Present MC 2x2	4.7490	0.3118	0.9708	0.4071	0.2314	0.2223
	Present SC 4x4	4.7195	0.3340	0.9833	0.4273	0.2942	0.2400
	Present MC 4x4	4.7514	0.3125	0.9596	0.4002	0.2291	0.2193
	Ref. [2] SC	4.72	0.337	0.875	0.380	0.321	0.349
	Ref. [2] MC	4.72	0.337	0.876	0.372	0.322	0.350
	Ref. [2] SME	4.72	0.340	1.01	0.434	0.376	0.298

fractions, both the present model as well as Caruso predict a higher stiffness as compared to the SME values, particularly for the case of $v_f = 0.622$ as seen in Table 1. The present multi-cell model also appears to converge to a lower stiffness values for this volume fraction value, possibly due to the greater compliance of the quadratic element used here as compared to the linear elements used by Caruso. For the lower values of v_f, the present model shows much better agreement with the SME values as compared to the work by Caruso, with the SC model tending to predict results closer to the SME values, while the MC model shows a slightly lower stiffness value. Another property that is matrix-dominant is the Poisson's ratio under longitudinal loading (ν_{L12}). Here, the present model generally predicts values lower than those simulated by Caruso, however the trend of showing lower ν values for MC simulations is also seen in the present model. An observation which can be made generally over the majority of the data set in all material properties is that the SME values tend to agree closer to the SC models, however, for the case of ν_{L23}, the present model shows closer approximations with the MC model, particularly for the lower fibre volume fractions. There is no significant conclusion that can be drawn from this data apart from the fact that both the present model and Caruso's model show large differences in the value predicted between SC and MC models. Similarly, the in-plane shear stiffness G_{L12} shows some deviation between the SC and MC values, with the SC values remaining closer to SME results while the MC values are slightly lower. The present results again predict a lower stiffness as compared to Caruso, with the present model predicting closer to SME results than Caruso only for the high fibre volume fraction $v_f = 0.622$. The cross-sectional mesh density does not have a significant effect on in-plane shear stiffness through all volume fractions. The lower stiffness predicted by the present model can be attributed to the use of quadratic elements which simulate a more realistic response to shear loading which causes undulating contours perpendicular to the shear plane (Figure 44) as opposed to the simple shear case where this edge generally remains linear with no relative slipping. The capturing of this additional compliance is also reflected in the transverse shear modulus G_{L23} with the present values consistently predicting lower values of stiffness than the simulations by Caruso. Caruso notes this deficiency of the model used in [2] at predicting shear stiffness accurately. The present model shows much better results in computing transverse stiffness with values much closer to expected SME results, particularly for the SC case as mentioned previously. On summarising all the data, it is possible to assert that the present model performs very well against both the data from Caruso and SME, and is particularly suited to simulate medium to high volume fraction ranges and is

adept at capturing shearing modes with more realistic deformations than the linear models previously used without the need for high mesh densities.

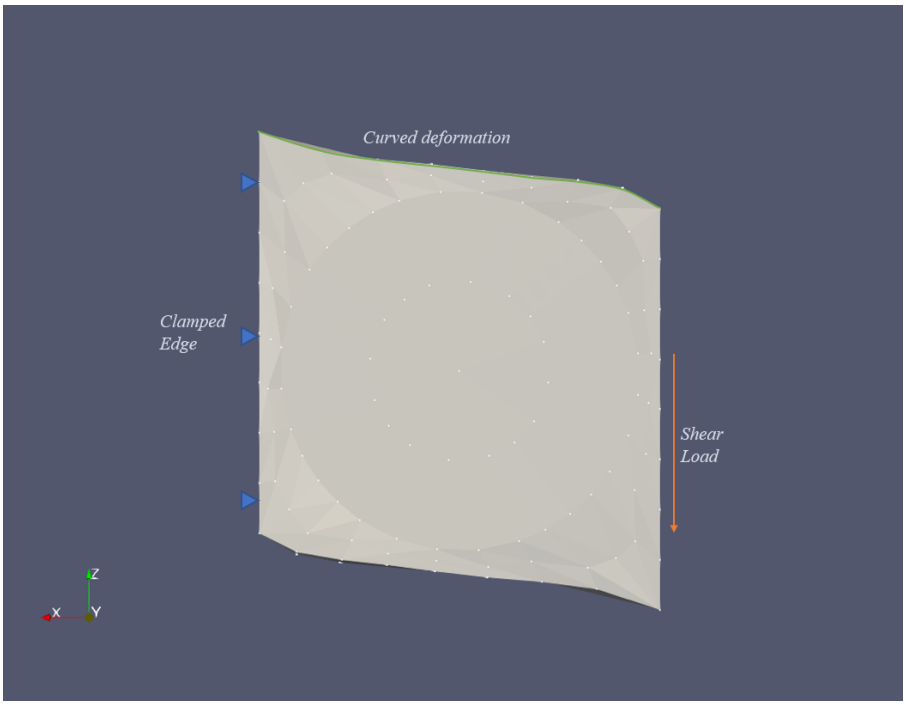

Figure 44: *Curvilinear deformation due to plain shear loading in a G_{L23} simulation*, 4x4 SC, v_f =0.622

With the model now validated for straight fibres, it is possible to demonstrate some basic properties observed on undulating the fibres. The undulations are introduced by perturbing the fibre centroid line as described in Section 7.5 and the analysis runs over different volume fractions consistent with the data used so far. All material properties are consistent with the Caruso work and previous results, with a $d_f = 0.0056in$ and Boron-HM Epoxy isotropic materials. The primary output is the compressive stiffness of the RV on the introduction of the undulations in the fibre. The perturbation is controlled by first identifying the maximum allowable perturbation ($\bar{\rho}$) depending on the volume fraction used, and defining five radius values as analyses input points as follows:

$$\rho_n = \frac{n}{4}\bar{\rho} \text{ ; where } n \in \{0, 1, 2, 3, 4\} \tag{39}$$

As the generation of these perturbed fibres are random, 10 different iterations are considered for each value of ρ_n and v_f, from which the mean is calculated and used to predict the loss in stiffness in the composite RV. Further, since the sample size selection is difficult to decide for such a complex system where the correlation between the perturbation input and the stiffness output is not clearly defined, the sample size 10 is chosen for practical reasons related to total computational run time of the parametric study, and the 95% confidence interval is also recorded.

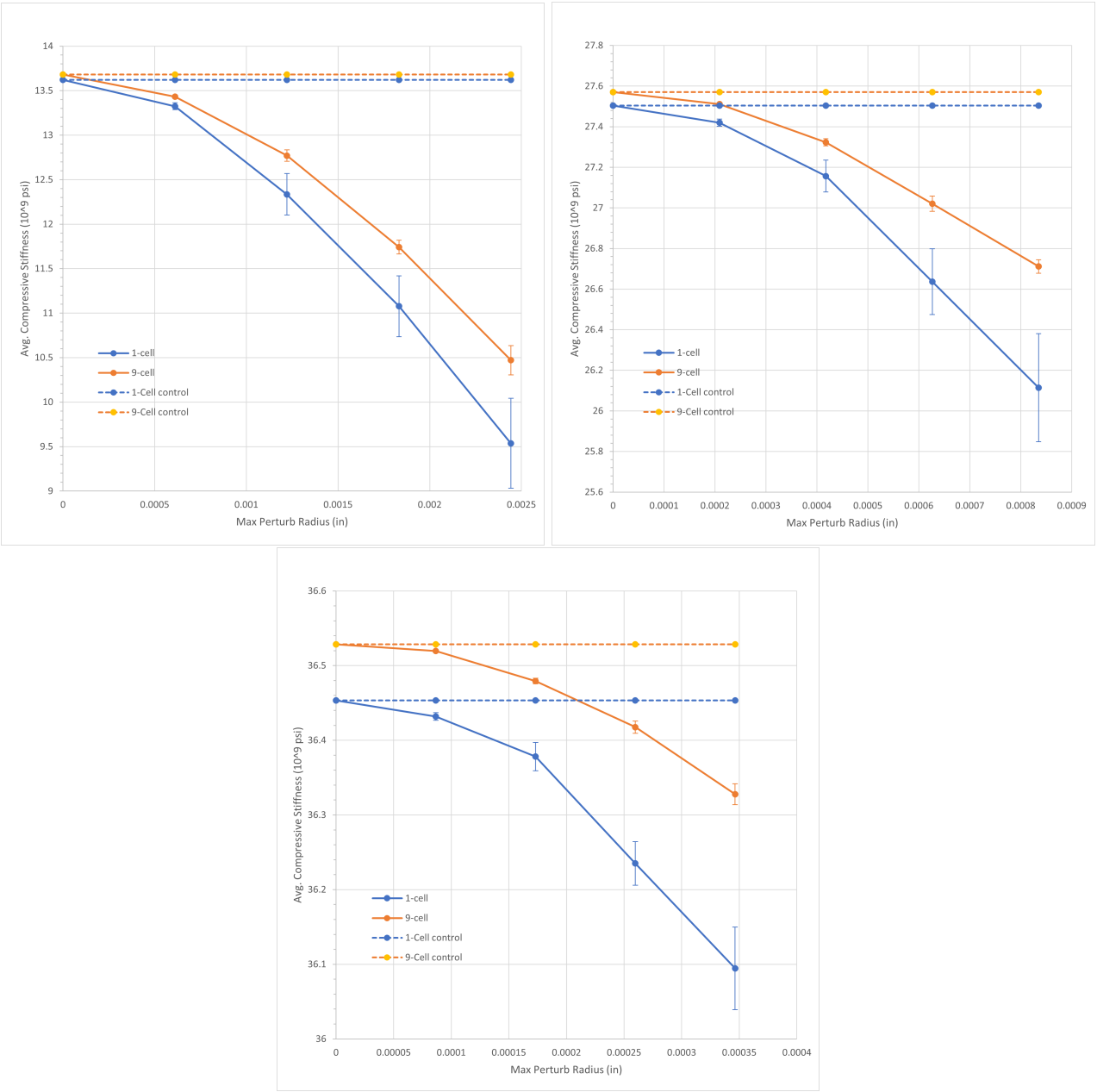

Figure 45: *Predicted Average Compressive Stiffness values against different perturbation radii, with different fibre volume fractions* v_f = a) 0.224; b) 0.466; c) 0.622

Figure 45 depicts three plots, with each plot showing the loss in average compressive stiffness (E_{C11}) over increasing radius of perturbation. The dotted lines signify the control values without perturbation consisting of straight fibres. The compression loading is applied in a very similar manner to the longitudinal Elastic modulus testing, as described in Section 7.4, with a compressive longitudinal prescribed displacement, and zero-displacements prescribed in the other two directions on the loaded face, to ensure a displacement purely along the fibre axis. It is immediately evident that there is a significant loss in compressive stiffness on creating small fibre undulations as compared to purely straight fibres as would be predicted, signifying that the natural undulations in manufactured FRP composites will experience lower stiffness values than conventionally calculated. More importantly, an increase in the degree of perturbation of the fibre shows a non-linear loss in stiffness over all values of fibre volume fractions. The differently coloured plot lines show the single celled and multi cell (9-cell) models to compare the effects of an equivalent perturbation on the two cases. Also, the possible

uncertainty in the calculated mean due to the limited sample size is captured by the error bars depicted in all the plots as a vertical bar at every data point. This represents the 95% confidence interval for the predicted value of compressive stiffness, and is a good indication of the expected scatter in the data due to the random perturbations. In general, the effect of perturbation is more significant in the single celled model as can be seen from the steeper descent of the graph in all three volume fraction cases. It can also be opined that the deviation or scatter of the compressive stiffness values is much higher for the 1-cell case as can be seen from the large error bar range. This is possibly due to the presence of minor localised out-of-plane deformations in the cell due to secondary loads and bending in areas of large perturbation. These deformations are much less pronounced in the multi-celled case due to conflicting perturbations in neighbouring cells and hence they show higher stiffness values as well as lower scatter as can be seen over all fibre radii.

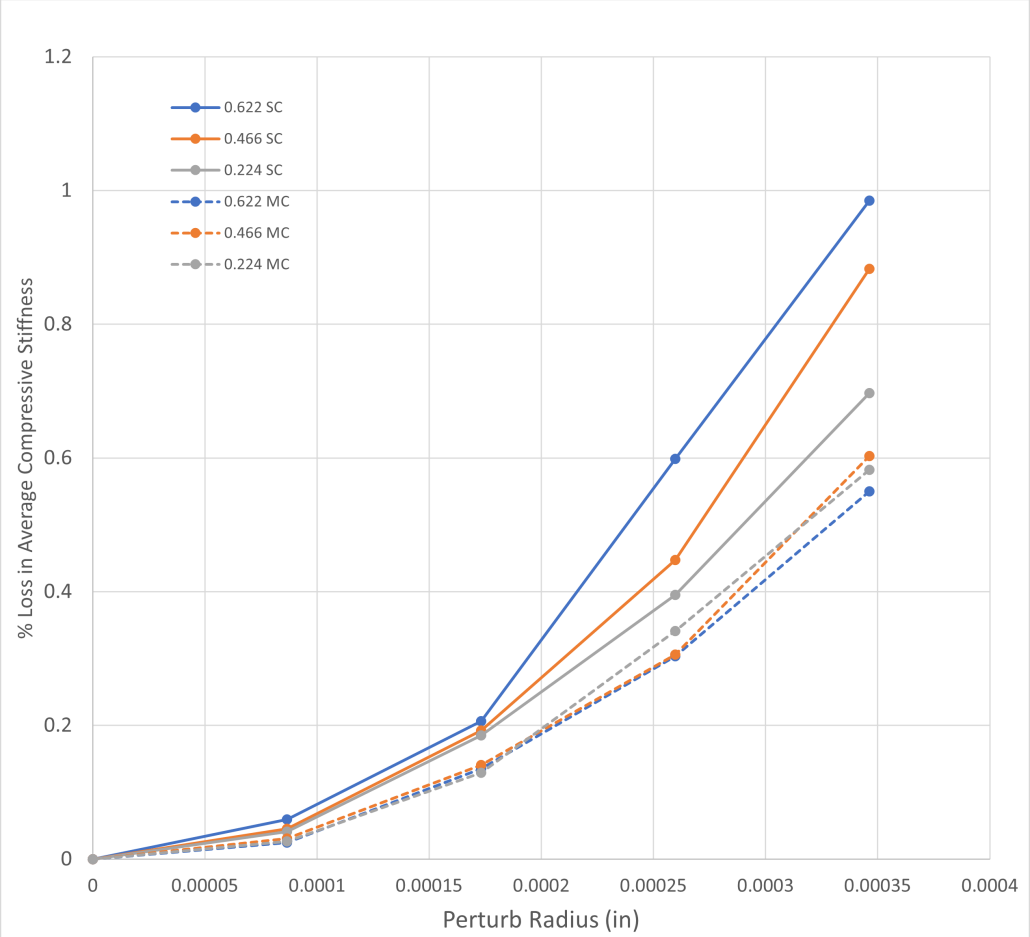

Figure 46: *Percentage Loss of Average Compressive Stiffness against different perturbation radii over different fibre volume fractions*

A point of note is that the value of maximum perturbation radius and the subsequent radius values are significantly lower for high volume fractions due to the limited difference between fibre diameter and cell-size, hence the results presented in Fig.45 are difficult to compare directly. Also, due to large changes in the base stiffness value over different fibre radii, it is difficult to immediately gauge the relative loss in stiffness on increasing the perturbation. To remedy this, a second set of data is simulated, with the maximum radius of perturbation maintained as constant over all fibre volume fractions, by selecting the maximum perturbation value which is valid for $v_f = 0.622$ ($\sim 0.06 d_f$), as the lower volume fractions are capable of accommodating higher perturbation. The results are presented in terms of the Percentage loss in compressive stiffness with respect to increasing perturbation radius, and

are presented graphically in Figure 46. This provides a direct comparison between the different volume fractions and cell-models. The high volume fractions show a steep loss in stiffness with very small increments of perturbation, with nearly 1% loss in Compressive stiffness with a maximum perturbation amplitude of only 6% of the fibre diameter for the single cell cases shown in the solid lines. Even for the multi cell models, the loss in stiffness is significant, albeit similar over all volume fractions, to the degree of loss averaging over 0.5% for the same maximum perturbation radius. The increase in loss of stiffness appears to be parabolic and this non-linear relation is not currently accounted for in standard semi-analytical homogenisation equations and methods.

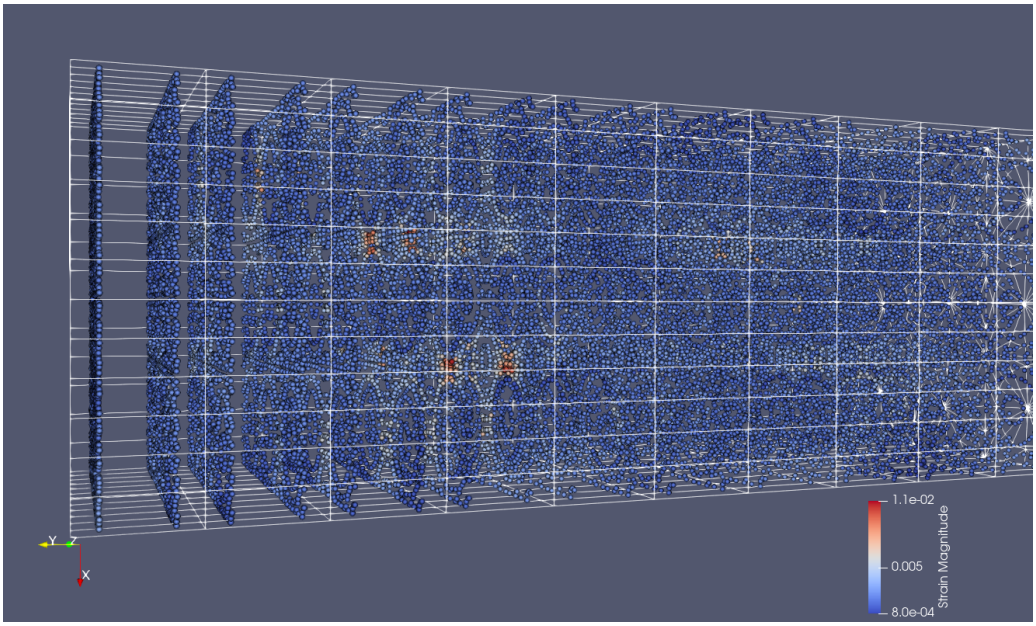

Figure 47: *Concentration of strains in the matrix regions due to fibre undulations* v_f=0.466, MC 3x3

An important aspect of the current model is the ability to capture 3D stress and strain states, particularly for future development of damage models. While it is seen that Multi-celled cases show a slightly lower sensitivity to perturbation in terms of loss of compressive stiffness, the effect of these perturbations are still reflected clearly in the local stresses and strains, particularly in the matrix regions. An example of this is shown in Figure 47 where the local strains in the regions highlighted in red are visibly higher nearly by a factor of two or more than the surrounding matrix regions. The figure depicts the strains in the composite as calculated at the Gauss points which creates some sparse gaps in between Gauss nodal layers as can be seen in the image, however this can be remedied by using the Extrapolation algorithm defined in Section 7.3. The values are exported from the 'gauss' labelled vtk file and imported into Paraview for visualisation. It can be posited that the local concentration of stresses can be attributed to reduced distance between neighbouring fibres in the region, leading to a reduction in matrix material locally and higher secondary stresses induced. This data also demonstrates the ability of the code to capture 3D stress states and hence satisfy an important requirement of the thesis.

9 Conclusion and Future Scope

This report has covered the scope of the literature, identified literature gaps, and has attempted to show different methodologies used to propose a solution. The conclusions of this thesis are presented here:

- In this thesis, the scope of the literature has been studied in the field of composite micromechanical analysis. It has been identified that conventional semi-analytical models of micro-mechanical stress states and damage models are not sufficient for consistent prediction of fibre composite properties.
- The current path of development in the literature is by creating Finite Element or similar mathematical models with increasing complexity and highly discretised domains which can accurately capture the fibre-matrix interactions, albeit at the cost of high computational power.
- Moreover, the modelling at micro-scale includes geometric and material non-linearities and anisotropy which requires significant investment in developing unique geometries and mathematical models for each case.
- A solution is proposed in this thesis to overcome these literature gaps, by developing an efficient scalable model which defines the fibre profile with a set of mid-lines or centroidal nodes, while populating the surrounding material with a 3D mathematical model capable of capturing fibre undulation, 3D stresses, damage, in addition to predicting the global properties of the simulated composite. This model is targeted at uni-directional fibre-reinforced composites, and can be expanded to model woven fibre composite tows as well.
- The development of a 3D enriched beam finite element was initially considered and attempted, by modelling each fibre-matrix pair with a single beam defined by its mid-line using a Geometrically exact strain formulation while capturing 3D stress states by using the Carrera Unified Formulation to enrich the cross-section. While it has the potential to greatly reduce computational time while capturing 3D stress states, several practical issues were identified with the implementation of the model in code due to the limited research available and the mathematical complexity involved.
- The final presented solution is a Unit Cell model where a Representative Volume consisting of independent fibre-matrix cells are created, with the fibre being defined by a centroidal line and the rest of the domain being defined by a set of external domain boundaries. This model is meshed with quadratic 3D wedge elements and has been fully implemented in code.
- The thesis has achieved the objective of developing a stand-alone system capable of handling the geometry modelling, meshing, solution, and generating post-processing data of a multi-fibre unit cell with no requirement for additional CAD modelling or meshing tools.
- The model has been validated against existing literature and has shown to be capable of simulating the realistic material properties of a UD FRP.
- It is observed that the present model is able to capture transverse shear and other matrix-dominated properties at a higher level of accuracy as compared to previous studies.
- The study also demonstrates the capability of the model to include pseudo-random or stochastic perturbations in the fibre geometry with modules which are already integrated with the existing code, and hence with no requirement for additional CAD modelling.
- The perturbation of fibres shows that there is a significant drop in Compressive stiffness of the composite on introduction of perturbation, with the stiffness sharply decreasing on increasing the maximum amplitude of the perturbation.

- The report demonstrates the distinct predictions obtained on using multiple fibre domains in the Unit cell, indicating that the effect of perturbation is reduced on increasing the number of fibres, possibly due to an averaging out of the asymmetric properties.
- The Unit cell code also meets the requirement of generating 3D stress and strain data adequately through the post-processing routines. The results indicate the pertinence of 3D stress states due to the presence of high local strain concentrations at regions of higher fibre perturbation amplitude in the Multi-cell composite, due to interaction of neighbouring fibres. Although the overall loss in global stiffness is significantly lower than that shown by the single cell model, the more significant effect of the undulations are seen in stress concentration pockets which can act as regions of failure initiation.

The report also recognises some key points of focus for improvement of the model in the near future:

- The formulation used currently for the perturbation uses random variables to perturb the radius of perturbation, and this can be improved to match the stochastic distribution of misalignment angles as described in Section 7.5 and Ref. [22].
- The wedge element can be further improved by using higher orders of interpolation to more accurately capture out of plane and shear deformation. The quality of the mesh also needs to be studied and improved for certain cases of fibre volume fractions due to the creation of very high aspect ratio elements.
- Due to the calculation of 3D stress states, it is possible to introduce damage and plasticity models in the matrix elements to more accurately capture the composite in the failure regime. It is also possible to enrich the element with fracture initiation and propagation functions.
- It is possible to use this Unit Cell itself as a homogenised element which can be used as a method of discretisation in a multi-scale composite simulation. Due to the distinction in the boundary and internal nodes, it is possible to ascribe the homogenised properties of the super-element to just the boundary nodes and the fields that are captured over the model can then be used for further microscale analysis on the Unit cell.
- The maximum volume fractions and perturbation of the fibre is currently restricted due to the packing scheme of the cuboidal cells used in the current model. It is possible to modify the external and internal meshing code slightly to have a more efficient fibre packing.
- The model can be used to generate Effective material properties and characteristic behaviour under different types of loading to be used as a coupon testing bed in developing a more efficient beam Finite Element model.
- The Geometrically exact model can capture non-linearities and rotations with a high degree of precision while maintaining a 3D beam geometry. Additional work needs to be done to consolidate the areas of practical difficulty including the co-rotational quaternion interpolation, and the numerical differentiation of the internal force vector to create the tangential stiffness matrix. It is also possible to solve the system explicitly using the internal and external force vector, however this would need more refined time-steps in the non-linear simulation.
- The Carrera Unified formulation can also be expanded upon to fully populate the cross-section of the beam corresponding to the nodal positions. This method shows good promise even with preliminary attempts, and it is possible to develop a Lagrangian mesh accross the cross-section to replace the beam mid-line DOFs of the Geometrically exact beam and hence obtain a true strain CUF model.

The thesis has been completed by delivering on the core objectives of the problem statement as outline by the extensive survey of the literature and the gaps identified consequently. Although the final methodology used is different from the original intention of a fully defined 3D beam model, the current Unit Cell has shown to be a positive step in the direction of developing an efficient, scalable, stand-alone model capable of modelling Fibre reinforced composites at the micro-scale while also providing ample avenue for future research.

Supporting Data:

The final code iteration or best attempt of all approaches discussed here are uploaded to the Git repository (https://github.com/anirudh9298/MastersThesisCodes) and are concurrent to the report as of June 16, 2022.

References

[1] S. Blassiau, A. Thionnet, and A. R. Bunsell, "Micromechanisms of load transfer in a unidirectional carbon fibre-reinforced epoxy composite due to fibre failures. Part 1: Micromechanisms and 3D analysis of load transfer: The elastic case," *Composite Structures*, vol. 74, no. 3, pp. 303–318, 2006.

[2] Caruso and J. J., "Application of finite element substructuring to composite micromechanics. M.S. Thesis - Akron Univ., May 1984," no. August 1984, 1984.

[3] T. R. King, D. M. Blackketter, D. E. Walrath, and D. F. Adams, "Micromechanics Prediction of the Shear Strength of Carbon Fiber/Epoxy Matrix Composites: The Influence of the Matrix and Interface Strengths," *Journal of Composite Materials*, vol. 26, no. 4, pp. 558–573, 1992.

[4] Y. L. Shen, M. Finot, A. Needleman, and S. Suresh, "Effective elastic response of two-phase composites," *Acta Metallurgica Et Materialia*, vol. 42, no. 1, pp. 77–97, 1994.

[5] C. T. Sun and R. S. Vaidya, "Prediction of composite properties from a representative volume element," *Composites Science and Technology*, vol. 56, no. 2, pp. 171–179, 1996.

[6] S. Li, "On the unit cell for micromechanical analysis of fibre-reinforced composites," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 455, no. 1983, pp. 815–838, 1999.

[7] J. A. Hewitt, D. Brown, and R. B. Clarke, "Computer modelling of woven composite materials," *Composites*, vol. 26, no. 2, pp. 134–140, 1995.

[8] W. Sun, F. Lin, and X. Hu, "Computer-aided design and modeling of composite unit cells," *Composites Science and Technology*, vol. 61, no. 2, pp. 289–299, 2001.

[9] A. Dixit, H. S. Mali, and R. K. Misra, "Unit cell model of woven fabric textile composite for multiscale analysis," *Procedia Engineering*, vol. 68, pp. 352–358, 2013.

[10] S. D. Green, M. Y. Matveev, A. C. Long, D. Ivanov, and S. R. Hallett, "Mechanical modelling of 3D woven composites considering realistic unit cell geometry," *Composite Structures*, vol. 118, no. 1, pp. 284–293, 2014.

[11] P. Römelt and P. R. Cunningham, "A multi-scale finite element approach for modelling damage progression in woven composite structures," *Composite Structures*, vol. 94, no. 3, pp. 977–986, 2012.

[12] H. W. Wang, H. W. Zhou, L. Mishnaevsky, P. Brøndsted, and L. N. Wang, "Single fibre and multifibre unit cell analysis of strength and cracking of unidirectional composites," *Computational Materials Science*, vol. 46, no. 4, pp. 810–820, 2009.

[13] M. Romanowicz, "A mesoscale study of failure mechanisms in angle-ply laminates under tensile loading," *Composites Part B: Engineering*, vol. 90, pp. 45–57, 2016.

[14] T. Matsuda, D. Okumura, N. Ohno, and M. Kawai, "Three-dimensional microscopic interlaminar analysis of cross-ply laminates based on a homogenization theory," *International Journal of Solids and Structures*, vol. 44, no. 25-26, pp. 8274–8284, 2007.

[15] K. Goto, T. Matsuda, and N. Kubota, "Fully-Modeled unit cell analysis for macro/micro elastic-Viscoplastic behavior of quasi-Isotropic CFRP laminates," *Key Engineering Materials*, vol. 626, pp. 512–517, 2015.

[16] T. Matsuda, K. Goto, N. Ohno, Y. Kawasaki, and S. Miyashita, "Multiscale analysis for negative through-the-thickness poisson's ratio of elastic-viscoplastic angle-ply CFRP laminates," *ICCM International Conferences on Composite Materials*, vol. 2015-July, no. July, pp. 19–24, 2015.

[17] R. J. D'Mello and A. M. Waas, "Influence of unit cell size and fiber packing on the transverse tensile response of fiber reinforced composites," *Materials*, vol. 12, no. 16, 2019.

[18] T. Okabe, M. Nishikawa, and H. Toyoshima, "A periodic unit-cell simulation of fiber arrangement dependence on the transverse tensile failure in unidirectional carbon fiber reinforced composites," *International Journal of Solids and Structures*, vol. 48, no. 20, pp. 2948–2959, 2011.

[19] M. M. Aghdam and A. Dezhsetan, "Micromechanics based analysis of randomly distributed fiber reinforced composites using simplified unit cell model," *Composite Structures*, vol. 71, no. 3-4, pp. 327–332, 2005.

[20] Y. Swolfs, L. Gorbatikh, V. Romanov, S. Orlova, S. V. Lomov, and I. Verpoest, "Stress concentrations in an impregnated fibre bundle with random fibre packing," *Composites Science and Technology*, vol. 74, pp. 113–120, 2013.

[21] S. Daggumati, A. Sharma, A. Kasera, and N. Upadhyay, "Failure Analysis of Unidirectional Ceramic Matrix Composite Lamina and Cross-Ply Laminate under Fiber Direction Uniaxial Tensile Load: Cohesive Zone Modeling and Brittle Fracture Mechanics Approach," *Journal of Materials Engineering and Performance*, vol. 29, no. 4, pp. 2049–2060, 2020.

[22] G. Catalanotti and T. A. Sebaey, "An algorithm for the generation of three-dimensional statistically Representative Volume Elements of unidirectional fibre-reinforced plastics: Focusing on the fibres waviness," *Composite Structures*, vol. 227, no. June, p. 111272, 2019.

[23] S. Bargmann, B. Klusemann, J. Markmann, J. E. Schnabel, K. Schneider, C. Soyarslan, and J. Wilmers, "Generation of 3D representative volume elements for heterogeneous materials: A review," *Progress in Materials Science*, vol. 96, pp. 322–384, 2018.

[24] A. Tasora, S. Benatti, D. Mangoni, and R. Garziera, "A geometrically exact isogeometric beam for large displacements and contacts," *Computer Methods in Applied Mechanics and Engineering*, vol. 358, p. 112635, 2020.

[25] E. Carrera, M. Cinefra, E. Zappino, and M. Petrolo, *Finite Element Analysis of Structures through Unified Formulation*, vol. 9781119941217. 2014.

[26] G. Dhondt, *The Finite Element Method for Three-dimensional Thermomechanical Applications*. John Wiley & Sons, Ltd, 2004.

[27] C. Soutis, "Fibre reinforced composites in aircraft construction," *Progress in Aerospace Sciences*, vol. 41, no. 2, pp. 143–151, 2005.

[28] K. L. Edwards, "An overview of the technology of fibre-reinforced plastics for design purposes," *Materials and Design*, vol. 19, no. 1-2, pp. 1–10, 1998.

[29] S. Li and E. Sitnikova, *Representative Volume Elements and Unit Cells: Concepts, Theory, Applications and Implementation*. Woodhead Publishing, 2019.

[30] J. D. Whitcomb, "Three-dimensional stress analysis of plain weave composites," *ASTM Special Technical Publication*, no. 1110, pp. 417–438, 1991.

[31] J. Whitcomb and X. Tang, "Effect of tow architecture on stresses in woven composites," *Collection of Technical Papers - AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, vol. 3, no. c, pp. 2309–2323, 1999.

[32] X. Tang and J. D. Whitcomb, "Progressive failure behaviors of 2D woven composites," *Journal of Composite Materials*, vol. 37, no. 14, pp. 1239–1259, 2003.

[33] M. Ansar, W. Xinwei, and Z. Chouwei, "Modeling strategies of 3D woven composites: A review," *Composite Structures*, vol. 93, no. 8, pp. 1947–1963, 2011.

[34] B. N. Cox, W. C. Carter, and N. A. Fleck, "A binary model of textile composites-I. Formulation," *Acta Metallurgica Et Materialia*, vol. 42, no. 10, pp. 3463–3479, 1994.

[35] J. Xu, B. N. Cox, M. A. McGlockton, and W. C. Carter, "A binary model of textile composites-II. The elastic regime," *Acta Metallurgica Et Materialia*, vol. 43, no. 9, pp. 3511–3524, 1995.

[36] M. A. McGlockton, B. N. Cox, and R. M. McMeeking, "A Binary Model of textile composites: III high failure strain and work of fracture in 3D weaves," *Journal of the Mechanics and Physics of Solids*, vol. 51, no. 8, pp. 1573–1600, 2003.

[37] A. E. Bogdanovich, *Multi-scale modeling, stress and failure analyses of 3-D woven composites*, vol. 41. 2006.

[38] Q. D. Yang and B. Cox, "Predicting failure in textile composites using the Binary Model with gauge-averaging," *Engineering Fracture Mechanics*, vol. 77, no. 16, pp. 3174–3189, 2010.

[39] K. Goto and T. Matsuda, "Three-dimensional analysis of microscopic stress distribution at a free edge of a cross-ply CFRP laminate," *ICCM International Conferences on Composite Materials*, pp. 1–6, 2011.

[40] C. Breite, A. Melnikov, A. Turon, A. B. de Morais, C. Le Bourlot, E. Maire, E. Schöberl, F. Otero, F. Mesquita, I. Sinclair, J. Costa, J. A. Mayugo, J. M. Guerrero, L. Gorbatikh, L. N. McCartney, M. Hajikazemi, M. Mehdikhani, M. N. Mavrogordato, P. P. Camanho, R. Tavares, S. M. Spearing, S. V. Lomov, S. Pimenta, W. Van Paepegem, and Y. Swolfs, "Detailed experimental validation and benchmarking of six models for longitudinal tensile failure of unidirectional composites," *Composite Structures*, vol. 279, no. March 2021, 2022.

[41] P. Jung, S. Leyendecker, J. Linn, and M. Ortiz, "A discrete mechanics approach to the cosserat rod theory—part 1: static equilibria," *International journal for numerical methods in engineering*, vol. 85, no. 1, pp. 31–60, 2011.

[42] A. Pagani and E. Carrera, "Unified formulation of geometrically nonlinear refined beam theories," *Mechanics of Advanced Materials and Structures*, vol. 25, no. 1, pp. 15–31, 2018.

[43] "Github repository : MastersThesisCodes." https://github.com/anirudh9298/MastersThesisCodes, 2022. Accessed: 15-06-2022.

[44] "ABAQUS library of solid elements: Triangular, tetrahedral, and wedge elements." https://abaqus-docs.mit.edu/2017/English/SIMACAETHERefMap/simathe-c-tritetwedge.htm, 2017. Accessed: 06-04-2022.

[45] G. R. Liu and S. S. Quek, *The Finite Element Method: A Practical Course*. Butterworth-Heinemann, 2003.

[46] T. A. Sebaey, G. Catalanotti, and N. P. O'Dowd, "A microscale integrated approach to measure and model fibre misalignment in fibre-reinforced composites," *Composites Science and Technology*, vol. 183, no. June, p. 107793, 2019.

[47] L. F. Varandas, G. Catalanotti, A. R. Melro, R. P. Tavares, and B. G. Falzon, "Micromechanical modelling of the longitudinal compressive and tensile failure of unidirectional composites: The effect of fibre misalignment introduced via a stochastic process," *International Journal of Solids and Structures*, vol. 203, pp. 157–176, 2020.

[48] S. Gomarasca, D. M. Peeters, B. Atli-Veltin, and C. Dransfeld, "Characterising microstructural organisation in unidirectional composites," *Composites Science and Technology*, vol. 215, no. September, p. 109030, 2021.

A Appendix : MainRoutine_Perturbation.py

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 23 13:48:04 2022

@author: aniru
MainRoutine_Perturbation.py

Single or small instance runner of Wedge Unit Cell solver
including perturbation setup
"""

import numpy as np
import csv

import Wedge_Quadratic_UnitCell_Solver as solverQ

def domaincalc(r,kf):
    af = np.pi*r**2
    ad = af/kf
    a = np.sqrt(ad)
    return a

OutputList = []
FvfList = [0.466,0.224] #[0.622,0.466,0.224,0.069]

Cellsizes = [[1,1,1],[3,1,3]]
Celltypes = [[0.0,0.0,0.0,0.0,0.0,0.0],[1.0,1.0,1.0,1.0,1.0,1.0]]

LoadList = [[1,1,1],[3],[5,5],[5,5]]
AxisList = [[1,0,2],[2],[1,0],[2,0]]
ClampList = [[0],[2],[4],[4]]
PrescribedDispList = [[-0.00012,0,0],[0.00012],[0.00012,0],[0.00012,0]]



perturbCheck = 1

fibreRadius = 0.0056/2

rhomax = 0.000346355417501517 # (domaincalc(fibreRadius, FvfList[0])/2) - fibreRadius
# <-- use domain calc method for normal runs
for rhoiter in range(5):
    rhomaxval = (rhomax/4)*(rhoiter)
    for iteration in range(2):
        Cells = Cellsizes[iteration]
        nIntElem = [10,4,4]
        OutputList.append(Celltypes[iteration][0:2])

        for fibreVolumeFraction in FvfList:

            squareside = domaincalc(fibreRadius, fibreVolumeFraction)
            # rhomax = squareside/2 - fibreRadius
            # rhomaxval = rhomax/4
            domainLengthY = 0.06 #inches
```

```
            domainWidthX = squareside*Cells[0] #inches
            domainHeightZ = squareside*Cells[2] #inches

            Dims = [domainWidthX,domainLengthY,domainHeightZ]

            Matproplist = [[0.75,0.35],[58,0.2]]


            if (rhoiter == 0):
                numtests = 1
                perturbCheck = 0
            else:
                numtests = 10
                perturbCheck = 1

            for testnum1 in range(numtests):
                print(f'\n{Cells}, rhoiter:{rhoiter}, pertrub?: {perturbCheck},
    rhomaxval:{rhomaxval}, testnum1:{testnum1}\n')
                OutList = []
                OutList.append(fibreVolumeFraction)
                testnum=0
                findEL = [0,0,0,0]
                findEL[testnum] = 1
                Loadfaces = LoadList[testnum]
                Clampfaces = ClampList[testnum]
                Loadaxes = AxisList[testnum]
                PrescribedDisps = PrescribedDispList[testnum]
                O1,O2 = solverQ.validation_solver(Cells,nIntElem,fibreRadius,
    fibreVolumeFraction,Dims,Matproplist,Clampfaces,Loadfaces,Loadaxes,
    PrescribedDisps,findEL,perturbCheck,rhomaxval)
                OutList.append(O1)
                if (testnum<2):
                    OutList.append(O2)

                OutputList.append(OutList)

    OutputList.append([2,2])
    OutputList.append([rhoiter,rhomaxval,rhomax,squareside])

if (1):

    with open(f"CompressiveStiffnessWavyALLfvf_0.000346_MINMAXrho_{nIntElem[0]}x{
    nIntElem[1]}x{nIntElem[2]}.csv", 'w') as f:
      #f"CompressiveStiffnessWavy{FvfList[0]}_{(rhomaxval*1e3):4}rhomaxk_{nIntElem
    [0]}x{nIntElem[1]}x{nIntElem[2]}.csv", 'w'
    # using csv.writer method from CSV package
        write = csv.writer(f)

        write.writerows(OutputList)
```

Listing 1: Main routine for setting up Perturbation models and calling all routines

B Appendix : MainRoutine_Validation.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 27 18:57:02 2022

@author: aniru
```

```
Looped runner for Caruso Verification

"""
import numpy as np
import csv

import Wedge_Quadratic_UnitCell_Solver as solverQ

def domaincalc(r,kf):
    af = np.pi*r**2
    ad = af/kf
    a = np.sqrt(ad)
    return a

OutputList = []
FvfList = [0.622,0.466,0.224,0.069]
# LoadList = [1,3,5,5] #Face numbers to load and clamp as listed in internal mesher
# ClampList = [0,2,4,4]
# AxisList = [1,2,1,2] # Load Axis
Cellsizes = [[1,1,1],[3,1,3]]
Celltypes = [[0.0,0.0,0.0,0.0,0.0,0.0],[1.0,1.0,1.0,1.0,1.0,1.0]]

LoadList = [[1],[3],[5,5],[5,5]]
AxisList = [[1],[2],[1,0],[2,0]]
ClampList = [[0],[2],[4],[4]]
PrescribedDispList = [[0.00012],[0.00012],[0.00012,0],[0.00012,0]]

]]

for iteration in range(2):
    Cells = Cellsizes[iteration]
    nIntElem = [10,4,4]
    OutputList.append(Celltypes[iteration])
    fibreRadius = 0.0056/2
    for fibreVolumeFraction in FvfList:

        squareside = domaincalc(fibreRadius, fibreVolumeFraction)

        domainLengthY = 0.06 #inches
        domainWidthX = squareside*Cells[0] #inches
        domainHeightZ = squareside*Cells[2] #inches

        Dims = [domainWidthX,domainLengthY,domainHeightZ]

        Matproplist = [[0.75,0.35],[58,0.2]]



        OutList = []
        OutList.append(fibreVolumeFraction)
        for testnum in range(0,4):
            findEL = [0,0,0,0]
            findEL[testnum] = 1
            Loadfaces = LoadList[testnum]
            Clampfaces = ClampList[testnum]
            Loadaxes = AxisList[testnum]
            PrescribedDisps = PrescribedDispList[testnum]
            O1,O2 = solverQ.validation_solver(Cells,nIntElem,fibreRadius,
    fibreVolumeFraction,Dims,Matproplist,Clampfaces,Loadfaces,Loadaxes,
    PrescribedDisps,findEL)
            OutList.append(O1)
            if (testnum<2):
                OutList.append(O2)
```

```
        OutputList.append(OutList)

if (1):

    with open(f"CarusoValidation{nIntElem[0]}x{nIntElem[1]}x{nIntElem[2]}.csv", 'w')
    as f:

    # using csv.writer method from CSV package
        write = csv.writer(f)

        write.writerows(OutputList)
```

Listing 2: Main routine for setting up Validation models and calling all routines

C Appendix : Wedge_Quadratic_UnitCell_Solver.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun  7 23:19:01 2021
Keep it going....
@author: aniru

Wedge Element - Quadratic
"""
'''------------------------------------------------
                QUADRATIC
   ------------------------------------------------'''
import numpy as np
from numpy import linalg as la

from scipy.sparse import lil_matrix # sparse matrix format as linked list
import scipy.linalg as sla
from pypardiso import spsolve #Intel MKL Pardiso parrallel(shared memory) sparse
    matrix solver

from time import time

def shapefuns(g,h,r):

    N = np.array([0.5*((1-g-h)*(2*(1-g-h)-1)*(1-r)-(1-g-h)*(1-r**2)),
                  0.5*(g*(2*g-1)*(1-r)-g*(1-r**2)),
                  0.5*(h*(2*h-1)*(1-r)-h*(1-r**2)),
                  0.5*((1-g-h)*(2*(1-g-h)-1)*(1+r)-(1-g-h)*(1-r**2)),
                  0.5*(g*(2*g-1)*(1+r)-g*(1-r**2)),
                  0.5*(h*(2*h-1)*(1+r)-h*(1-r**2)),
                  2.0*(1-g-h)*g*(1-r),
                  2.0*g*h*(1-r),
                  2.0*(1-g-h)*h*(1-r),
                  2.0*(1-g-h)*g*(1+r),
                  2.0*g*h*(1+r),
                  2.0*(1-g-h)*h*(1+r),
                  1.0*(1-g-h)*(1-r**2),
                  1.0*g*(1-r**2),
                  1.0*h*(1-r**2)])

    dNg = np.array([1/2 - (r - 1)*(g + h - 1) - r**2/2 - ((r - 1)*(2*g + 2*h - 1))/2,
                    r**2/2 - g*(r - 1) - ((2*g - 1)*(r - 1))/2 - 1/2,
                    0,
                    ((r + 1)*(2*g + 2*h - 1))/2 + (r + 1)*(g + h - 1) - r**2/2 + 1/2,
                    ((2*g - 1)*(r + 1))/2 + g*(r + 1) + r**2/2 - 1/2,
```

```
                    0,
                    (r - 1)*(2*g + 2*h - 2) + 2*g*(r - 1),
                    -2*h*(r - 1),
                    2*h*(r - 1),
                    - (r + 1)*(2*g + 2*h - 2) - 2*g*(r + 1),
                    2*h*(r + 1),
                    -2*h*(r + 1),
                    r**2 - 1,
                    1 - r**2,
                    0])

    dNh = np.array([1/2 - (r - 1)*(g + h - 1) - r**2/2 - ((r - 1)*(2*g + 2*h - 1))/2,
                    0,
                    r**2/2 - h*(r - 1) - ((2*h - 1)*(r - 1))/2 - 1/2,
                    ((r + 1)*(2*g + 2*h - 1))/2 + (r + 1)*(g + h - 1) - r**2/2 + 1/2,
                    0,
                    ((2*h - 1)*(r + 1))/2 + h*(r + 1) + r**2/2 - 1/2,
                    2*g*(r - 1),
                    -2*g*(r - 1),
                    (r - 1)*(2*g + 2*h - 2) + 2*h*(r - 1),
                    -2*g*(r + 1),
                    2*g*(r + 1),
                    - (r + 1)*(2*g + 2*h - 2) - 2*h*(r + 1),
                    r**2 - 1,
                    0,
                    1 - r**2])

    dNr = np.array([- ((g + h - 1)*(2*g + 2*h - 1))/2 - r*(g + h - 1),
                    g*r - (g*(2*g - 1))/2,
                    h*r - (h*(2*h - 1))/2,
                    ((g + h - 1)*(2*g + 2*h - 1))/2 - r*(g + h - 1),
                    g*r + (g*(2*g - 1))/2,
                    h*r + (h*(2*h - 1))/2,
                    g*(2*g + 2*h - 2),
                    -2*g*h,
                    h*(2*g + 2*h - 2),
                    -g*(2*g + 2*h - 2),
                    2*g*h,
                    -h*(2*g + 2*h - 2),
                    2*r*(g + h - 1),
                    -2*g*r,
                    -2*h*r])


    return N, dNg, dNh, dNr


def Mapping(xa,g,h,r):

    N,dNg,dNh,dNr = shapefuns(g,h,r)
    x = N[None,:] @ xa

    return x

def ParamCoords():
    X = np.array([[0,0,-1],
                  [1,0,-1],
                  [0,1,-1],
                  [0,0,1],
                  [1,0,1],
                  [0,1,1],
                  [0.5,0,-1],
                  [0.5,0.5,-1],
```

```
                    [0,0.5,-1],
                    [0.5,0,1],
                    [0.5,0.5,1],
                    [0,0.5,1],
                    [0,0,0],
                    [1,0,0],
                    [0,1,0]],dtype = float)
    for i in range(X.shape[0]):
        pt = X[i,:]
        g = pt[0]
        h = pt[1]
        r = pt[2]
        N,dNg,dNh,dNr = shapefuns(g,h,r)
        for j in range(X.shape[0]):
            if j==i:
                print(N[j],j)
            elif N[j]!=0:
                print(f"Error i={i} j={j}, N[j] = {N[j]}\n")
            else:
                print('ok')
    return X


'''-----------------------------------------------
                QUADRATIC
   -----------------------------------------------'''


def Material(Matproplist,tag):

    E = Matproplist[tag][0]
    nu = Matproplist[tag][1]

    G = E/(2*(1+nu))
    lam = nu*E/((1+nu)*(1-2*nu))
    Cmat = np.array([[2*G+lam, lam, lam, 0, 0, 0],\
                     [lam, 2*G+lam, lam, 0, 0, 0],\
                     [lam, lam, 2*G+lam, 0, 0, 0],\
                     [0, 0, 0, G, 0, 0],\
                     [0, 0, 0, 0, G, 0],\
                     [0, 0, 0, 0, 0, G]])
    return Cmat


def Jacobian(elnum, NodeCoords, N, dNg, dNh, dNr, errF):

    localderivmat = np.array([dNg,dNh,dNr])
    J = localderivmat @ NodeCoords
    detJ = la.det(J)
    if (detJ <= 0):
        print("invalid Jacobian",elnum,detJ)        #ngp
    Jinv = la.inv(J)
    globalderivmat = Jinv @ localderivmat
    dNx = globalderivmat[0,:]
    dNy = globalderivmat[1,:]
    dNz = globalderivmat[2,:]

    return detJ,dNx,dNy,dNz,J,Jinv



def Elmat(elnum,NodeCoords,N,dNx,dNy,dNz,nodes,dofn):
```

```
    B = np.zeros([6,nodes*dofn])  # 6nodes per element
    for i in range(nodes): #6 nodes
        Bi = np.array([[dNx[i],0,0],
                       [0,dNy[i],0],
                       [0,0,dNz[i]],
                       [0,dNz[i],dNy[i]],
                       [dNz[i],0,dNx[i]],
                       [dNy[i],dNx[i],0]])
        B[:,i*dofn:(i+1)*dofn] = Bi

    return B


def Elemnodes(elnum,Coords,Elemconn,nodes):

    NodeCoords = np.zeros([nodes,3]) #nodes = 6 -> Lin wedge
    Conn = Elemconn[elnum,:]
    for i in range(nodes):
        NodeCoords[i,:] = Coords[Conn[i],:]

    return NodeCoords,Conn

def EleStiff(elnum,B,D,stifchk,ngp,g,h,r):

    K = B.T @ D @ B

    # if stifchk:
    #     print("checking symmetry")
    #     nullmat = K-K.T
    #     if np.amax(nullmat) > 1e-10:
    #         print (np.amax(nullmat),ngp,g,h,r)
    return K


def GaussPts(Tngp):
    r1 = 1/6
    r2 = 1/(3**0.5)
    wgp = np.array([r1,r1,r1,r1,r1,r1])
    gpt = np.array([[r1,r1,-r2],
                    [4*r1,r1,-r2],
                    [r1,4*r1,-r2],
                    [r1,r1,r2],
                    [4*r1,r1,r2],
                    [r1,4*r1,r2]])

    return gpt,wgp,Tngp


def GaussLoop(elnum,NodeCoords,D,nodes,dofn):

    eldof = nodes*dofn
    Kel = np.zeros([eldof,eldof])
    Pel = np.zeros([eldof])
    gpt,wgp,Tngp = GaussPts(6)

    for ngp in range(Tngp):

        g,h,r = gpt[ngp,:]
        wt = wgp[ngp]
        errF = 0
        stifchk = 0

        N,dNg,dNh,dNr = shapefuns(g, h, r)
```

```
            detJ,dNx,dNy,dNz,J,Jinv = Jacobian(elnum, NodeCoords, N, dNg, dNh, dNr, errF)
            B = Elmat(elnum, NodeCoords, N, dNx, dNy, dNz, nodes, dofn)

            kk = EleStiff(elnum, B, D, stifchk, ngp, g, h, r)

            Kel += kk*detJ*wt

        return Kel,Pel


def BCs(NodesClamp,Constset,NDOF,dofn):
    bu = np.zeros([NDOF])
    for i in NodesClamp:
        bu[i*dofn:(i+1)*dofn] = 1

    bu, Xvals = Constrain(Constset,NDOF,dofn,bu)

    bu = bu.astype(bool)

    bk = ~bu

    return bk,bu,Xvals


def Constrain(Constset,NDOF,dofn,bu):
    Xvals = np.zeros([NDOF])
    for nd in Constset:
        node = nd[0]
        dof = nd[1]
        const = nd[2]
        # if const == 0:
        bu[dofn*node+dof] = 1
        Xvals[dofn*node+dof] = const
    return bu, Xvals

def TrimBC(BigP,NodesClamp,Constset,NDOF,dofn):

    bk,bu,Xvals = BCs(NodesClamp,Constset,NDOF,dofn)
    slicek = np.array([i for i in range(bk.shape[0]) if (bk[i])])
    sliceu = np.array([i for i in range(bu.shape[0]) if (bu[i])])
    # Kk = BigK[bk,:][:,bk]
    # Kku = BigK[bk,:][:,bu]
    # Kk = BigK[np.ix_(slicek[:],slicek[:])]
    # Kku = BigK[np.ix_(slicek[:],sliceu[:])]
    Pp = BigP[bk]

    return Pp,bk,bu,slicek,sliceu,Xvals


def PointLoads(Loadset,BigP): #Loadset = [NodeNum, DofNum, LoadVal]

    # Pel = np.zeros([18])

    for ld in Loadset:
        node = ld[0]
        dof = ld[1]
        loadval = ld[2]
        BigP[3*node+dof] += loadval

    return BigP


def Assembly(elnum,Kel,Pel,BigK,BigP,Conn,nodes,dofn):
```

```
    rows = np.zeros([nodes*dofn])
    cols = np.zeros([nodes*dofn])
    for i in range(nodes):
        ii = i*dofn
        row = Conn[i]*dofn
        rows[ii:ii+dofn] = np.arange(row,row+dofn)
    for j in range(nodes):
        jj = j*dofn
        col = Conn[j]*dofn
        cols[jj:jj+dofn] = np.arange(col,col+dofn)
            # BigK[row:row+dofn, col:col+dofn] += Kel[ii:ii+dofn, jj:jj+dofn]

        BigP[row:row+dofn] += Pel[ii:ii+dofn]

    BigK[np.ix_(rows[:],cols[:])] += Kel[:, :]
    return BigK,BigP

def Autoload(L,dof,Load):
    Loadset = []
    Load = Load/len(L)
    for i,node in enumerate(L):
        Loadset.append([node,dof,Load])
    return Loadset

def Autodisp(L,dof,disp):
    Constset = []
    for i,node in enumerate(L):
        Constset.append([node,dof,disp])
    return Constset


def ElemStrainGP(Xout,Conn,NodeCoords,elnum,dofn,nodes,eldisp,D,Dinv, errF): #eldisp
    from WedgePost.py
    Tngp = 6
    gpt,wgp,Tngp = GaussPts(Tngp)
    StrainGP = np.zeros([Tngp,6])  #6 is no. of strain components
    StressGP = np.zeros([Tngp,6])
    # ExtrapMat = np.zeros([Tngp,Tngp]) # 'A' Matrix for GP -> node extrap
    for i in range(Tngp):
        gp = gpt[i,:]
        N,dNg,dNh,dNr = shapefuns(gp[0],gp[1],gp[2])
        detJ,dNx,dNy,dNz,J,Jinv = Jacobian(elnum, NodeCoords, N, dNg, dNh, dNr, errF)
        B = Elmat(elnum,NodeCoords,N,dNx,dNy,dNz,nodes,dofn)
        epsilon = B @ eldisp[:,None]
        sigma = D @ epsilon
        StrainGP[i,:] = epsilon[:,0]
        StressGP[i,:] = sigma[:,0]
        # ExtrapMat[i,:] = N

    return StrainGP,StressGP #,ExtrapMat

def Extrapolation():
    Tngp = 6
    MainNodes = 6
    gpt,wgp,Tngp = GaussPts(Tngp)
    Extrapmat = np.zeros([Tngp,MainNodes])
    for i in range(Tngp):
        gp = gpt[i,:]
        N,dNg,dNh,dNr = shapefuns(gp[0],gp[1],gp[2])
        Extrapmat[i,:] = N
    Extrapmatinv = la.inv(Extrapmat)
    return Extrapmat, Extrapmatinv
```

```
def GPtoNode(StrainGP,StressGP,StrainNode,StressNode,NodeRep,Extrapmatinv,Conn,elnum)
    :
    epsilon = Extrapmatinv @ StrainGP
    sigma = Extrapmatinv @ StressGP
    for i,node in enumerate(Conn):
        NodeRep[node] += 1
        StrainNode[node,:] += epsilon[i,:]
        StressNode[node,:] += sigma[i,:]

    return NodeRep,StrainNode,StressNode

def GaussCoords(gpt,NodeCoords,elnum,Conn,eldisp,GPList,GPDisp,Tngp,nodes):
    dispstack = np.zeros([nodes,3])
    dispstack[:,0] = eldisp[0::3]
    dispstack[:,1] = eldisp[1::3]
    dispstack[:,2] = eldisp[2::3]
    for i,gp in enumerate(gpt):
        Xgp = Mapping(NodeCoords,gp[0],gp[1],gp[2])
        Ugp = Mapping(dispstack,gp[0],gp[1],gp[2])
        GPList[elnum*Tngp+i,:] = Xgp
        GPDisp[elnum*Tngp+i,:] = Ugp
    return GPList,GPDisp

def CalcRFvector(BigK,bu,bk,Xout):
    Kuk = BigK[bu,:][:,bk]
    Kuu = BigK[bu,:][:,bu]
    RF = (Kuk @ Xout[bk]) + (Kuu @ Xout[bu])

    return RF

def NetForce(BigP,Facenodes,dof,dofn):
    force = 0
    for i in Facenodes:
        force += BigP[i*dofn+dof]
    return force

def AvgDisp(Xout,Facenodes,dof,dofn,bk):
    avgdisp = 0
    count = 0
    for i in Facenodes:
        if bk[i*dofn+dof]:
            avgdisp += Xout[i*dofn+dof]
            count += 1

    return avgdisp/count

#-------------------------------------------------------------------------------#
'''-----------------------------------------------
                QUADRATIC
   -----------------------------------------------'''


import InternalMesher_Multicell_Quadratic as mesh



global Failures
global Successes

def validation_solver(Cells,nIntElem,fibreRadius,kf,Dims,Matproplist,Clampfaces,
    Loadfaces,Loadaxes,PrescribedDisps,findEL,perturbCheck,rhomaxval):
    postProcessing = 1
    Failures = []
```

```
Successes = []


Coords,Elemconn,Elemtag,Faces,jobname,Dims,Cells = mesh.InternalMesh(1,Cells,
nIntElem,fibreRadius,kf,Dims,perturbCheck,rhomaxval)

eltype = 26
dofn = 3
nodes = 15

Nel = Elemconn.shape[0]
# Coords = np.array([[0, 0, 0],
#                    [1e-0, 0, 0],
#                    [0.5e-0, 0.866e-0, 0],
#                    [0, 0, 1],
#                    [1e-0, 0, 1],
#                    [0.5e-0, 0.866e-0, 1],
#                    [0, 0, 2],
#                    [1e-0, 0, 2],
#                    [0.5e-0, 0.866e-0, 2],
#                    [0.5e-0, 0, 0],  #midnodes
#                    [0.75e-0, 0.433e-0, 0],
#                    [0.25e-0, 0.433e-0, 0],
#                    [0.5e-0, 0, 2],
#                    [0.75e-0, 0.433e-0, 2],
#                    [0.25e-0, 0.433e-0, 2]])

errF = 0

Tnode = Coords.shape[0]
NDOF = Tnode*dofn
stifchk = 0
# E = 68e9;nu=0.25

NodesClamp = []
for clampface in Clampfaces:
    NodesClamp.extend(Faces[clampface])

Loadset = []
Constset = []


'''------------------------------------------------
               QUADRATIC
   ------------------------------------------------'''


loadforce = 0
loaddisp = 1
prescribedDisp = PrescribedDisps[0] #0.00012

if (loadforce):
    Loadset = Autoload(Faces[0],0,700000.0)

if (loaddisp):
    for nconst,loadface in enumerate(Loadfaces):
        Constset.extend(Autodisp(Faces[loadface],Loadaxes[nconst],PrescribedDisps
[nconst])) #0.0026417791763021104


BigK = lil_matrix((NDOF,NDOF))
BigP = np.zeros([NDOF])
Xout = np.zeros([NDOF])
```

```
checkt = time()
for elnum in range(Nel):
    D = Material(Matproplist,Elemtag[elnum])
    Dinv = la.inv(D)
    NodeCoords,Conn = Elemnodes(elnum, Coords, Elemconn, nodes)
    Kel,Pel = GaussLoop(0,NodeCoords,D,nodes,dofn)

    BigK,BigP = Assembly(elnum, Kel, Pel, BigK, BigP, Conn, nodes, dofn)
    if (elnum%100 == 0):
        print(f"{time()-checkt :.7f} {elnum}")
        checkt = time()
BigP = PointLoads(Loadset,BigP)
# Kk,Pp,bk,bu,Kku,Xvals = TrimBC(BigK,BigP,NodesClamp,Constset,NDOF,dofn)
Pp,bk,bu,slicek,sliceu,Xvals = TrimBC(BigP,NodesClamp,Constset,NDOF,dofn)

# Adding constraint force vector to actual.
Pconst = BigK[np.ix_(slicek,sliceu)] @ Xvals[bu]
print(f"{time()-checkt :.7f} BigK slice + Imposed disp done")
checkt = time()
Pp = Pp - Pconst
# print(f"{time()-checkt :.7f} Reaction forces done")
# checkt = time()

# Disp = la.inv(Kk) @ Pp
Disp = spsolve(BigK[np.ix_(slicek,slicek)].tocsr(), Pp)
print(f"{time()-checkt :.7f} BigK slice + spsolve done")
checkt = time()

Xout[bu] = Xvals[bu]
Xout[bk] = Disp
print(np.amax(abs(Xout)))

# RF = CalcRFvector(BigK, bu, bk, Xout)
# BigP[bu] = RF

RF = (BigK[np.ix_(sliceu,slicek)] @ Xout[bk]) + (BigK[np.ix_(sliceu,sliceu)] @
Xout[bu])
BigP[bu] = RF
print(f"{time()-checkt :.7f} BigK slice + RF done")
checkt = time()

################ POST PROCESSING ##################
if (postProcessing):
    gpt,wgp,Tngp = GaussPts(6)

    import WedgePost as post
    scale = 1.0e1
    ptdat = 1

    newCoords = post.Update_Position(Xout, Coords, Nel, Tnode, dofn, scale)

    StrainNode = np.zeros([Tnode,6])
    StressNode = np.zeros([Tnode,6])
    StrainGPs = np.zeros([6*Nel,6])
    StressGPs = np.zeros([6*Nel,6])
    GPList = np.zeros([6*Nel,3])
    GPDisp = np.zeros([6*Nel,3])
    NodeRep = np.zeros([Tnode])
    maxs=0
    for elnum in range(Nel):
        D = Material(Matproplist,Elemtag[elnum])
        Dinv = la.inv(D)
        NodeCoords,Conn = Elemnodes(elnum, Coords, Elemconn, nodes)
```

```
            eldisp = post.ElemDisp(Xout, Conn, nodes, dofn)
            GPList,GPDisp = GaussCoords(gpt, NodeCoords, elnum, Conn, eldisp, GPList,
     GPDisp, Tngp,nodes)
            StrainGP,StressGP = ElemStrainGP(Xout, Conn, NodeCoords, elnum, dofn,
    nodes, eldisp, D, Dinv, 0)
            StrainGPs[elnum*6:(elnum+1)*6,:] = StrainGP
            StressGPs[elnum*6:(elnum+1)*6,:] = StressGP

            if np.amax(StressGP)>maxs:
                maxs = np.amax(StressGP)
        '''-----------------------------------------------
                        QUADRATIC
           -----------------------------------------------'''

        post.vtk_output_format(jobname, Coords, Tnode, Elemconn, Nel, nodes, eltype,
    Xout, StrainNode, StressNode, ptdat)

        gpvtk = 1
        if gpvtk:
            post.vtk_outputgauss_format(jobname, GPList, Tngp*Nel, nodes, eltype,
    GPDisp, StrainGPs, StressGPs, ptdat)

        post.vtk_output_format(jobname+'_disp', newCoords, Tnode, Elemconn, Nel,
    nodes, eltype, Xout, StrainNode, StressNode, ptdat)

################ Verifying NASA Paper ##################

    if findEL[0]:   # findEL11
        netFaceForce = NetForce(BigP,Faces[0],1,dofn)

        areaFace = Dims[0]*Dims[2]
        EL11 = (netFaceForce * Dims[1]) / (areaFace * prescribedDisp)

        avgFaceDisp = AvgDisp(Xout, Faces[3], 2, dofn,bk) - AvgDisp(Xout, Faces[2],
    2, dofn,bk)
        # areaFace = Dims[0]*Dims[1]
        VL12 = (avgFaceDisp / Dims[2]) / (prescribedDisp / Dims[1])

        print(f"EL11 = {EL11} \nVL12 = {VL12}")
        return(-EL11,-VL12)
    if findEL[1]:  # findEL22
        netFaceForce = NetForce(BigP,Faces[2],2,dofn)

        areaFace = Dims[0]*Dims[1]
        EL22 = (netFaceForce * Dims[2]) / (areaFace * prescribedDisp)

        avgFaceDisp = AvgDisp(Xout, Faces[5], 0, dofn,bk) - AvgDisp(Xout, Faces[4],
    0, dofn,bk)
        VL23 = (avgFaceDisp / Dims[0]) / (prescribedDisp / Dims[2])

        print(f"EL22 = {EL22} \nVL23 = {VL23}")
        return(-EL22,-VL23)
    if findEL[2]:  # findEL33
        netFaceForce = NetForce(BigP,Faces[5],1,dofn)

        areaFace = Dims[1] * Dims[2]
        GL12 = (netFaceForce/areaFace) / (prescribedDisp/Dims[2])
        print(f"GL12 = {GL12}")
        return(GL12,0)
    if findEL[3]:
        netFaceForce = NetForce(BigP,Faces[5],2,dofn)
        areaFace = Dims[1] * Dims[2]
        GL23 = (netFaceForce/areaFace) / (prescribedDisp/Dims[2])
```

```
        print(f"GL23 = {GL23}")
        return(GL23,0)

# NEED TO RETURN (0,0) if not running EFFECTIVE MAT PROP (NASA) simulations
```

Listing 3: Quadratic Wedge Element Solver for the Unit Cell

D Appendix : InternalMesher_Multicell_Quadratic.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 24 12:23:06 2021

@author: aniru

Multi Cell Quadratic Internal mesher

Requires a seeded global boundary node list and centroid list (including quad)
Outputs a global node coordinate list with a global connectivity matrix

"""
import ExternalMesher_Multicell_Quadratic as exmesh
import Perturbation_Random as perturb
import numpy as np
from numpy import linalg as la

#### FUNCTIONS

def splitlayers(LayerConn,Centroids,module):
    L1 = LayerConn[module,:]
    L2 = LayerConn[module+1,:]
    L1c = Centroids[module]
    L2c = Centroids[module+1]
    L1 = np.concatenate((L1, np.array([L1[0]])),axis=0)
    L2 = np.concatenate((L2, np.array([L2[0]])),axis=0)
    return L1, L2, L1c, L2c

def distance(A,B):
    return la.norm(A-B)

def section(A,B,r):
    D = A-B
    R = r*D + B
    return R

def fibrenodes(BoundaryNodeCoords,L0,L0c,tLayerNodes,fibreRadius):
    INodeCoords = np.zeros([tLayerNodes,3])
    nodesadded = 0
    for n in range(tLayerNodes):
        Node = BoundaryNodeCoords[L0[n],:]
        Cent = L0c[:]
        dist = distance(Node,Cent)
        ratio = fibreRadius/dist
        NewNode = section(Node,Cent,ratio)
        INodeCoords[nodesadded,:] = NewNode
        nodesadded += 1
    return INodeCoords, nodesadded
```

```
def quadnodes1(nLayers,tLayerNodes,Centroids,CentroidListLin,seed0,pointer):
    global GlobalNodeCoords
    count = 0
    Coords = np.zeros([nLayers*tLayerNodes,3])
    for layer in range (nLayers):
        centre = CentroidListLin[Centroids[layer],:]
        for j in range (tLayerNodes):
            fnode = GlobalNodeCoords[seed0+count,:]
            irnode = section(fnode,centre,0.5)
            Coords[count,:] = irnode
            count += 1
    irnodes = count
    return(Coords,irnodes)

def quadnodes2(nLayers,tLayerNodes,Centroids,CentroidListLin,seed0,pointer,
    fibreRadius):
    global GlobalNodeCoords
    count = 0
    Coords = np.zeros([nLayers*tLayerNodes,3])
    for layer in range (nLayers):
        centre = CentroidListLin[Centroids[layer],:]
        for j in range (tLayerNodes):
            if (j != tLayerNodes-1):
                fnode1 = GlobalNodeCoords[seed0+count,:]
                fnode2 = GlobalNodeCoords[seed0+1+count,:]
            else:
                fnode1 = GlobalNodeCoords[seed0+count,:]
                fnode2 = GlobalNodeCoords[seed0+1+count-tLayerNodes,:]

            fnodem = section(fnode1,fnode2,0.5)
            ornode = section(fnodem,centre,fibreRadius/distance(fnodem,centre))
            Coords[count,:] = ornode
            count += 1
    ornodes = count
    return(Coords,ornodes)

def quadnodes3(nLayers,tLayerNodes,LayerConn,seed0,pointer):
    global GlobalNodeCoords
    count = 0
    Coords = np.zeros([nLayers*tLayerNodes,3])
    for layer in range (nLayers):
        L0 = LayerConn[layer,:]
        for j in range(tLayerNodes):
            mnode1 = GlobalNodeCoords[seed0+count,:]
            mnode2 = GlobalNodeCoords[L0[j],:]
            pmnode = section(mnode1,mnode2,0.5)
            Coords[count,:] = pmnode
            count += 1
    pmnodes = count
    return(Coords,pmnodes)

def quadnodes4(nLayers,tLayerNodes,LayerConn,seed0,pointer):
    global GlobalNodeCoords
    count = 0
    Coords = np.zeros([nLayers*tLayerNodes,3])
    for layer in range(nLayers):
        L0 = LayerConn[layer,:]
        for j in range(tLayerNodes):
            if (j != tLayerNodes-1):
                sq0 = np.array([L0[j], L0[j+1], seed0+j+1+layer*tLayerNodes, seed0+j+
    layer*tLayerNodes], dtype = int)
            else:
                sq0 = np.array([L0[j], L0[0], seed0+0+layer*tLayerNodes, seed0+j+
```

```
    layer*tLayerNodes], dtype = int)

            if ((distance(GlobalNodeCoords[sq0[0]],GlobalNodeCoords[sq0[2]]))>(
    distance(GlobalNodeCoords[sq0[1]],GlobalNodeCoords[sq0[3]]))):
                sl = [0,2] #split line
            else:
                sl = [1,3]
            mnode1 = GlobalNodeCoords[sq0[sl[0]]]
            mnode2 = GlobalNodeCoords[sq0[sl[1]]]
            smnode = section(mnode1,mnode2,0.5)
            Coords[count,:] = smnode
            count += 1
    smnodes = count
    return(Coords,smnodes)

def quadnodes5(nLayers,tLayerNodes,seed0,pointer):
    global GlobalNodeCoords
    count = 0
    Coords = np.zeros([nLayers*tLayerNodes,3])
    for layer in range(nLayers):
        for i in range(tLayerNodes):
            fnode1 = GlobalNodeCoords[seed0+count,:]
            fnode2 = GlobalNodeCoords[seed0+tLayerNodes+count,:]
            fmnode = section(fnode1,fnode2,0.5)
            Coords[count,:] = fmnode
            count += 1
    fmnodes = count
    return(Coords,fmnodes)


def updateGlobal(Array):
    global GlobalNodeCoords, gPos
    length = Array.shape[0]
    GlobalNodeCoords[gPos:gPos+length,:] = Array
    gPos += length

def updateGlobalEl(Array,n,tag):
    global GlobalElemConn, GlobalElemTag, gEl, gEq
    l = Array.shape[0]
    if (n==6):
        GlobalElemConn[gEl:gEl+l,0:6] = Array
        GlobalElemTag[gEl:gEl+l] = tag
        gEl = gEl + l
    else:
        GlobalElemConn[gEq:gEq+l,6:15] = Array
        gEq = gEq + l

def findFaceNodes(axis,val):
    global GlobalNodeCoords
    global gPos
    l = []
    for i in range(gPos):
        if (GlobalNodeCoords[i,axis] == val):
            l.append(i)
    return l



##### INPUTs
def InternalMesh(quadElems,Cells,nIntElem,fibreRadius,kf,Dims,perturbCheck,rhomaxval)
    :
    global GlobalNodeCoords
    global GlobalElemConn
```

```
global GlobalElemTag
global gPos
global gEl
global gEq

# Cells,[nL,nB,nH],fibreRadius,kf,Dims = inp.inputs()
nL,nB,nH = nIntElem
# fibreRadius = 0.0056/2 #0.8
# kf = 0.622

nCellX,nCellY,nCellZ = Cells

# nCellX = 1
# nCellY = 1
# nCellZ = 1
nCells = nCellX*nCellY*nCellZ

# x_G = squareside*nCellX #2
# y_G = 0.01#10
# z_G = squareside*nCellZ #2

x_G,y_G,z_G = Dims

L = y_G/nCellY #y
B = x_G/nCellX  #x
H = z_G/nCellZ  #z

# No. of wedges along dimensions
# nL = 20
# nB = 2
# nH = 2

vfF = (np.pi*fibreRadius**2)/(B*H)

BoundaryNodesLin,BoundaryNodesQuad,BoundaryNodesQuad2,BoundaryNodesTotal,
BConnQuad, BConnLin, \
BConnT,BConnQuad2,CentroidListLin,CentroidListQuad,CentroidConnLin,
CentroidConnQuad \
    = exmesh.ExternalMeshing_Quad(x_G, y_G, z_G, nCellX, nCellY, nCellZ, L, B, H,
 nL, nB, nH)


GlobalNodeCoords = np.zeros([30000,3])
GlobalElemConn = np.zeros([10000,15],dtype = int)
GlobalElemTag = np.zeros([10000],dtype=int)
gPos = 0
gEl = 0
gEq = 0

tBNodesLin = BoundaryNodesLin.shape[0]
tBNodesQuad = BoundaryNodesQuad.shape[0]
tBNodesQuad2 = BoundaryNodesQuad2.shape[0]
tCentroidsLin = CentroidListLin.shape[0]
tCentroidsQuad = CentroidListQuad.shape[0]


if (perturbCheck):
    CentroidListLin = perturb.fibrePerturb(tCentroidsLin, CentroidListLin, vfF,
fibreRadius, [L,B,H],rhomaxval)
    if (quadElems):
        CentroidListQuad = perturb.fibrePerturb(tCentroidsQuad,CentroidListQuad,
vfF,fibreRadius,[L,B,H],rhomaxval)
```

```
if (quadElems):
    tBoundNodes = tBNodesLin + tBNodesQuad + tBNodesQuad2
else:
    tBoundNodes = tBNodesLin


updateGlobal(BoundaryNodesLin)

if (quadElems):
    updateGlobal(BoundaryNodesQuad)
    updateGlobal(BoundaryNodesQuad2)

updateGlobal(CentroidListLin)

if (quadElems):
    updateGlobal(CentroidListQuad)

Faces = []
##################### START OF INTERIOR NODES ###################################
intstart = gPos
seedC = intstart


for cell in range(nCells): #nCells

    LayerConn = BConnLin[:,:,cell]
    tLayerNodes = LayerConn.shape[1]
    Centroids = CentroidConnLin[cell,:]
    QCentroids = CentroidConnQuad[cell,:]
    LayerConnQ = BConnQuad[:,:,cell]
    LayerConnQ2 = BConnQuad2[:,:,cell]
    # Make fibre Nodes
    for layer in range(nL+1):

        L0 = LayerConn[layer,:]
        L0c = CentroidListLin[Centroids[layer]]
        INodeCoords, nodesadded = fibrenodes(BoundaryNodesLin,L0,L0c,tLayerNodes,
fibreRadius)
        updateGlobal(INodeCoords)

    # Connectivity for fibre wedges

    ElConnFLin = np.zeros([tLayerNodes*nL,6], dtype=int)
    ElConnMLin = np.zeros([tLayerNodes*nL*2,6], dtype=int)

    for module in range(nL):

        #L1,L2,L1c,L2c = splitlayers(LayerConn,Centroids,module)
        for elem in range(tLayerNodes):
            conn = np.zeros([6],dtype=int)

            if (elem != tLayerNodes-1):
                conn[0] = Centroids[module] + tBoundNodes
                conn[2] = seedC + elem + module*tLayerNodes
                conn[1] = seedC + elem + 1 + module*tLayerNodes
                conn[3] = Centroids[module+1] + tBoundNodes
                conn[5] = seedC + elem + (module+1)*tLayerNodes
                conn[4] = seedC + elem + 1 + (module+1)*tLayerNodes
            else:
                conn[0] = Centroids[module] + tBoundNodes
                conn[2] = seedC + elem + module*tLayerNodes
                conn[1] = seedC + 0 + module*tLayerNodes
                conn[3] = Centroids[module+1] + tBoundNodes
```

```
                conn[5] = seedC + elem + (module+1)*tLayerNodes
                conn[4] = seedC + 0 + (module+1)*tLayerNodes

            ElConnFLin[module*tLayerNodes+elem,:] = conn
    updateGlobalEl(ElConnFLin,6,1)
    #seedC = gPos  ############## IMPORTANT, NEEDED FOR NEXT CELL
##################


    for module in range(nL):

        L1,L2,L1c,L2c = splitlayers(LayerConn,Centroids,module)

        for elem in range(tLayerNodes):
            if (elem != tLayerNodes-1):
                sq1 = np.array([L1[elem], L1[elem+1], seedC+elem+1+module*
tLayerNodes, seedC+elem+module*tLayerNodes], dtype = int)
                sq2 = np.array([L2[elem], L2[elem+1], seedC+elem+(module+1)*
tLayerNodes+1, seedC+elem+(module+1)*tLayerNodes], dtype = int)
            else:
                sq1 = np.array([L1[elem], L1[elem+1], seedC+0+module*tLayerNodes,
 seedC+elem+module*tLayerNodes], dtype = int)
                sq2 = np.array([L2[elem], L2[elem+1], seedC+0+(module+1)*
tLayerNodes, seedC+elem+(module+1)*tLayerNodes], dtype = int)
            # brickconn = np.concatenate((sq1,sq2),axis=0)
            if ((distance(GlobalNodeCoords[sq1[0]],GlobalNodeCoords[sq1[2]]))>(
distance(GlobalNodeCoords[sq1[1]],GlobalNodeCoords[sq1[3]]))):
                sl = [0,2] #split line
                sc = [1,3] #split corner nodes (not shared with split element)
            else:
                sl = [1,3]
                sc = [2,0]

            conn1 = np.zeros([6],dtype=int)
            conn2 = np.zeros([6],dtype=int)

            conn1[0] = sq1[sc[0]]
            conn1[1] = sq1[sl[0]]
            conn1[2] = sq1[sl[1]]
            conn1[3] = sq2[sc[0]]
            conn1[4] = sq2[sl[0]]
            conn1[5] = sq2[sl[1]]

            conn2[0] = sq1[sc[1]]
            conn2[1] = sq1[sl[1]]
            conn2[2] = sq1[sl[0]]
            conn2[3] = sq2[sc[1]]
            conn2[4] = sq2[sl[1]]
            conn2[5] = sq2[sl[0]]

            ElConnMLin[module*(2*tLayerNodes)+ 2*elem,:] = conn1
            ElConnMLin[module*(2*tLayerNodes)+ 2*elem + 1,:] = conn2
    updateGlobalEl(ElConnMLin,6,0)

    if (quadElems):
        #hari om#
        QList = []
        QList.append(gPos)
        # INNER RING NODES 0-1
        TempCoords,nPos = quadnodes1(nL+1,tLayerNodes,Centroids,CentroidListLin,
seedC,0)
        updateGlobal(TempCoords)
        QList.append(gPos)
```

```
        # OUTER RING NODES 1-2
        TempCoords,nPos = quadnodes2(nL+1,tLayerNodes,Centroids,CentroidListLin,
seedC,0,fibreRadius)
        updateGlobal(TempCoords)
        QList.append(gPos)
        #PRIMARY MATRIX NODES 2-3
        TempCoords,nPos = quadnodes3(nL+1, tLayerNodes, LayerConn, seedC, 0)
        updateGlobal(TempCoords)
        QList.append(gPos)
        #SPLIT MATRIX NODES 3-4
        TempCoords,nPos = quadnodes4(nL+1, tLayerNodes, LayerConn, seedC, 0)
        updateGlobal(TempCoords)
        QList.append(gPos)
        #FIBRE MID NODES 4-5
        TempCoords,nPos = quadnodes5(nL, tLayerNodes, seedC, 0)
        updateGlobal(TempCoords)
        QList.append(gPos)

        #PLOT the coords
        from matplotlib import pyplot as plt
        # fig = plt.figure()
        # ax = fig.add_subplot(projection='3d')

        # ax.scatter(GlobalNodeCoords[QList[0]:QList[1],0],GlobalNodeCoords[QList
[0]:QList[1],1],GlobalNodeCoords[QList[0]:QList[1],2])
        # ax.scatter(GlobalNodeCoords[QList[1]:QList[2],0],GlobalNodeCoords[QList
[1]:QList[2],1],GlobalNodeCoords[QList[1]:QList[2],2])
        # ax.scatter(GlobalNodeCoords[QList[2]:QList[3],0],GlobalNodeCoords[QList
[2]:QList[3],1],GlobalNodeCoords[QList[2]:QList[3],2])
        # ax.scatter(GlobalNodeCoords[QList[3]:QList[4],0],GlobalNodeCoords[QList
[3]:QList[4],1],GlobalNodeCoords[QList[3]:QList[4],2])
        # ax.scatter(GlobalNodeCoords[QList[4]:QList[5],0],GlobalNodeCoords[QList
[4]:QList[5],1],GlobalNodeCoords[QList[4]:QList[5],2])

        ######## QUADRATIC CONNECTIVITY ##############
        ElConnFQuad = np.zeros([tLayerNodes*nL,9], dtype=int)
        ElConnMQuad = np.zeros([tLayerNodes*nL*2,9], dtype=int)

            ## FIBRE QUADRATIC CONNECTIVITY

        for module in range (nL):
            for elem in range (tLayerNodes):
                elnum = module*tLayerNodes+elem
                qconn = np.zeros([9],dtype=int)

                if (elem != tLayerNodes-1):

                    qconn[2] = QList[0] + elnum
                    qconn[1] = QList[1] + elnum
                    qconn[0] = QList[0] + elnum + 1

                    qconn[5] = QList[0] + tLayerNodes + elnum
                    qconn[4] = QList[1] + tLayerNodes + elnum
                    qconn[3] = QList[0] + tLayerNodes + elnum + 1

                    qconn[6] = QCentroids[module] + tBoundNodes
                    qconn[8] = QList[4] + elnum
                    qconn[7] = QList[4] + elnum + 1
                else:

                    qconn[2] = QList[0] + elnum
                    qconn[1] = QList[1] + elnum
                    qconn[0] = QList[0] + elnum + 1 - tLayerNodes
```

```
                    qconn[5] = QList[0] + tLayerNodes + elnum
                    qconn[4] = QList[1] + tLayerNodes + elnum
                    qconn[3] = QList[0] + tLayerNodes + elnum + 1 - tLayerNodes

                    qconn[6] = QCentroids[module] + tBoundNodes
                    qconn[8] = QList[4] + elnum
                    qconn[7] = QList[4] + elnum + 1 - tLayerNodes

                ElConnFQuad[elnum,:] = qconn

        updateGlobalEl(ElConnFQuad, 9, 1)


        for module in range(nL):

            L1,L2,L1c,L2c = splitlayers(LayerConn,Centroids,module)
            L1Q = LayerConnQ[module,:]
            L1Q2 = LayerConnQ2[module,:]
            L2Q2 = LayerConnQ2[module+1,:]
            for elem in range(tLayerNodes):
                elnum = module*tLayerNodes+elem
                if (elem != tLayerNodes-1):
                    sq1 = np.array([L1[elem], L1[elem+1], seedC+elem+1+module*
tLayerNodes, seedC+elem+module*tLayerNodes], dtype = int)
                    sq2 = np.array([L2[elem], L2[elem+1], seedC+elem+(module+1)*
tLayerNodes+1, seedC+elem+(module+1)*tLayerNodes], dtype = int)
                    qsq1 = np.array([QList[2]+elnum, L1Q2[elem], QList[2]+elnum
+1, QList[1]+elnum, QList[3]+elnum],dtype=int)
                    qsq2 =  np.array([QList[2]+elnum+tLayerNodes, L2Q2[elem],
QList[2]+elnum+tLayerNodes+1, QList[1]+elnum+tLayerNodes, QList[3]+elnum+
tLayerNodes],dtype=int)
                    qsqm = np.array([L1Q[elem], L1Q[elem+1], QList[4]+elnum+1,
QList[4]+elnum],dtype=int)
                else:
                    sq1 = np.array([L1[elem], L1[elem+1], seedC+0+module*
tLayerNodes, seedC+elem+module*tLayerNodes], dtype = int)
                    sq2 = np.array([L2[elem], L2[elem+1], seedC+0+(module+1)*
tLayerNodes, seedC+elem+(module+1)*tLayerNodes], dtype = int)
                    qsq1 = np.array([QList[2]+elnum, L1Q2[elem], QList[2]+elnum
+1-tLayerNodes, QList[1]+elnum, QList[3]+elnum],dtype=int)
                    qsq2 = np.array([QList[2]+elnum+tLayerNodes, L2Q2[elem],
QList[2]+elnum+tLayerNodes+1-tLayerNodes, QList[1]+elnum+tLayerNodes, QList[3]+
elnum+tLayerNodes],dtype=int)
                    qsqm =np.array([L1Q[elem], L1Q[elem+1-tLayerNodes], QList[4]+
elnum+1-tLayerNodes, QList[4]+elnum],dtype=int)
                # brickconn = np.concatenate((sq1,sq2),axis=0)
                if ((distance(GlobalNodeCoords[sq1[0]],GlobalNodeCoords[sq1[2]]))
>(distance(GlobalNodeCoords[sq1[1]],GlobalNodeCoords[sq1[3]]))):
                    sl = [0,2] #split line
                    sc = [1,3] #split corner nodes (not shared with split element
)
                    qsa = [1,4,2]
                    qsb = [3,4,0]
                else:
                    sl = [1,3]
                    sc = [2,0]
                    qsa = [2,4,3]
                    qsb = [0,4,1]

                qconn1 = np.zeros([9],dtype=int)
                qconn2 = np.zeros([9],dtype=int)
                # QelConnMatrix[module*(2*tLayerNodes)+ 2*elem,0:6] =
```

```
ElConnMatrix[module*(2*tLayerNodes)+ 2*elem,:]
                # QelConnMatrix[module*(2*tLayerNodes)+ 2*elem + 1,0:6] =
ElConnMatrix[module*(2*tLayerNodes)+ 2*elem + 1,:]

                qconn1[0] = qsq1[qsa[0]]
                qconn1[1] = qsq1[qsa[1]]
                qconn1[2] = qsq1[qsa[2]]
                qconn1[3] = qsq2[qsa[0]]
                qconn1[4] = qsq2[qsa[1]]
                qconn1[5] = qsq2[qsa[2]]
                qconn1[6] = qsqm[sc[0]]
                qconn1[7] = qsqm[sl[0]]
                qconn1[8] = qsqm[sl[1]]

                qconn2[0] = qsq1[qsb[0]]
                qconn2[1] = qsq1[qsb[1]]
                qconn2[2] = qsq1[qsb[2]]
                qconn2[3] = qsq2[qsb[0]]
                qconn2[4] = qsq2[qsb[1]]
                qconn2[5] = qsq2[qsb[2]]
                qconn2[6] = qsqm[sc[1]]
                qconn2[7] = qsqm[sl[1]]
                qconn2[8] = qsqm[sl[0]]


                ElConnMQuad[module*(2*tLayerNodes)+ 2*elem,:] = qconn1
                ElConnMQuad[module*(2*tLayerNodes)+ 2*elem + 1,:] = qconn2

        updateGlobalEl(ElConnMQuad, 9, 0)

    seedC = gPos
    # gEq = gEl


import WedgePost as post

if (quadElems):

    jobname = f"UnitCellFiles/ReportMC{perturbCheck}_UnitCell_{nCellX}x{nCellZ}x{
nCellY}_Complete_Quad_{int(nB)}x{int(nH)}x{int(nL)}"
    #f"UnitCellFiles/WavyCompression{perturbCheck}_UnitCell_{nCellX}x{nCellZ}x{
nCellY}_Complete_Quad_{int(nB)}x{int(nH)}x{int(nL)}"
    nodes_per_elem = 15
    eltype = 26
    Xout = np.zeros(6)
    StrainNode = 0
    StressNode = 0
    ptdat = 0

    C = GlobalNodeCoords[0:gPos,:]
    E = GlobalElemConn[0:gEl,0:15]
    Etag = GlobalElemTag[0:gEl]
    post.vtk_output_format(jobname+'_meshQ', C, C.shape[0], E, E.shape[0],
nodes_per_elem, eltype, Xout, StrainNode, StressNode, ptdat)

else:

    jobname = f"UnitCellFiles/UnitCell_{nCellX}x{nCellZ}x{nCellY}_Complete_Lin_{
int(nB)}x{int(nH)}x{int(nL)}"

    nodes_per_elem = 6
    eltype = 13
    Xout = np.zeros(6)
```

```
        StrainNode = 0
        StressNode = 0
        ptdat = 0

        C = GlobalNodeCoords[0:gPos,:]
        E = GlobalElemConn[0:gEl,0:nodes_per_elem]
        Etag = GlobalElemTag[0:gEl]
        post.vtk_output_format(jobname+'_meshL', C, C.shape[0], E, E.shape[0],
    nodes_per_elem, eltype, Xout, StrainNode, StressNode, ptdat)
    # Faces along Length
    Faces.append(findFaceNodes(1, 0))
    Faces.append(findFaceNodes(1, y_G))
    # Faces along Height
    Faces.append(findFaceNodes(2, 0))
    Faces.append(findFaceNodes(2, z_G))
    # Faces along Width
    Faces.append(findFaceNodes(0, 0))
    Faces.append(findFaceNodes(0, x_G))
    return (C,E,Etag,Faces,jobname,Dims,Cells)
```

Listing 4: Internal Meshing Code with Quadratic Wedge Element for the Unit Cell

E Appendix : ExternalMesher_Multicell_Quadratic.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 21 21:39:52 2021

@author: aniru

Multi-element External node mesher

"""

import numpy as np
from numpy import linalg as la

## DOMAIN:
# x_G = 6
# y_G = 10
# z_G = 6

# nCellX = 3
# nCellY = 1
# nCellZ = 3

# L = y_G/nCellY #y
# B = x_G/nCellX  #x
# H = z_G/nCellZ  #z

# # No. of wedges along dimensions
# nL = 10
# nB = 3
# nH = 3

def ExternalMeshing_Quad(x_G,y_G,z_G,nCellX,nCellY,nCellZ,L,B,H,nL,nB,nH):

    tNodesX = nB * nCellX + 1
    tNodesZ = nH * nCellZ + 1
    tNodesY = nL * nCellY + 1
```

```
Origin = np.array([0.,0.,0.])

tBNodesface = (tNodesX)*(nCellZ+1) + (nCellX + 1)*(nH-1)*nCellZ
tBNodesLinear = tBNodesface*(nL*nCellY + 1)
tBNodesQuadratic = tBNodesface*(nL*nCellY)
tBNodesQ2faceA = (nB*nCellX)*(nCellZ+1)
tBNodesQ2faceB = (nH*nCellZ)*(nCellX+1)

print(tBNodesQ2faceA,tBNodesQ2faceB)
tBNodesQuadratic2 = (tBNodesQ2faceA + tBNodesQ2faceB)*(nL*nCellY+1)  #tBNodesface
 + (nCellX + 1)*nCellZ
print(tBNodesQuadratic2)
tBNodes1 = tBNodesLinear + tBNodesQuadratic
tBNodes2 = tBNodes1 + tBNodesQuadratic2

BoundaryNodesLin = np.zeros([tBNodesLinear,3])
BoundaryNodesQuad = np.zeros([tBNodesQuadratic,3])
BoundaryNodesQuad2 = np.zeros([tBNodesQuadratic2,3])
BoundaryNodesTotal = np.zeros([tBNodes2,3])

countL = 0
for nY in range(tNodesY):
    y = Origin[1] + (L/nL)*nY

    for nZ in range(tNodesZ):
        if (nZ%nH == 0):
            tNodesXr = tNodesX
        else:
            tNodesXr = nCellX + 1
        z = Origin[2] + (H/nH)*nZ

        for nX in range (tNodesXr):
            x = Origin[0] + (x_G/(tNodesXr-1))*nX
            BoundaryNodesLin[countL,:] = np.array([x,y,z])
            countL += 1

countQ = 0
for nY in range(tNodesY-1):
    y = Origin[1] + (L/nL)*nY + (L/nL)*0.5

    for nZ in range(tNodesZ):
        if (nZ%nH == 0):
            tNodesXr = tNodesX
        else:
            tNodesXr = nCellX + 1
        z = Origin[2] + (H/nH)*nZ

        for nX in range (tNodesXr):
            x = Origin[0] + (x_G/(tNodesXr-1))*nX
            BoundaryNodesQuad[countQ,:] = np.array([x,y,z])
            countQ += 1

countQ2 = 0
for nY in range(tNodesY):
    y = Origin[1] + (L/nL)*nY

    for nZ in range(nCellZ + 1):
        tNodesXr = tNodesX-1
        z = Origin[2] + (nZ) * H
        xoff = B/nB * 0.5
        xlen = B/nB
        # else:
```

```
        #     tNodesXr = nCellX + 1
        #     z = Origin[2] + (H/nH)*nZ - H/nH*0.5
        #     xoff = 0
        #     xlen = B

        for nX in range (tNodesXr):
            x = Origin[0] + xlen*nX + xoff
            BoundaryNodesQuad2[countQ2,:] = np.array([x,y,z])
            countQ2 += 1

sbnodeswitch = countQ2
print(sbnodeswitch)
for nY in range(tNodesY):
    y = Origin[1] + (L/nL)*nY

    for nZ in range(tNodesZ-1):
        tNodesXr = nCellX + 1
        z = Origin[2] + (H/nH)*nZ + H/nH*0.5
        xoff = 0
        xlen = B

        for nX in range (tNodesXr):
            x = Origin[0] + xlen*nX + xoff
            BoundaryNodesQuad2[countQ2,:] = np.array([x,y,z])
            countQ2 += 1

print(countQ2)
############## ELEM CONN ############

nCell = nCellX * nCellY * nCellZ

nLayersLin = nL + 1
nLayersQuad = nL
nLayersT = nLayersLin + nLayersQuad
tLayerNodes = 2*(nB) + 2*(nH)

BConnLin = np.zeros([nLayersLin,tLayerNodes,nCell],dtype=int)
BConnQuad = np.zeros([nLayersQuad,tLayerNodes,nCell],dtype=int)
BConnQuad2 = np.zeros([nLayersLin,tLayerNodes,nCell],dtype=int)
BConnT = np.zeros([nLayersT,tLayerNodes,nCell],dtype=int)

cellnum = 0
for elY in range(nCellY):

    for elZ in range(nCellZ):

        for elX in range(nCellX):

            for layer in range(nLayersLin):

                conn = np.zeros([tLayerNodes],dtype = int)
                count = 0
                seed0 = tBNodesface*(elY*nL+layer) + ((nH-1)*(nCellX+1)+tNodesX)*
elZ + (nB)*elX
                seed1 = seed0 + tNodesX - (elX*(nB-1))
                seed2 = tBNodesface*(elY*nL+layer) + ((nH-1)*(nCellX+1)+tNodesX)
*(elZ+1) + (nB)*elX
                for i in range(nB+1):
                    conn[i] = seed0 + i
                count += nB+1

                for i in range(nH-1):
                    conn[count+i] = seed1 + (nCellX+1)*i + 1
```

```
                count += nH-1

                for i in range(nB+1):
                    conn[count+i] = seed2 + nB - i
                count += nB+1

                for i in range(nH-1):
                    conn[count+i] = seed1 + (nCellX+1)*(nH-2-i)
                count += nH-1

                BConnLin[layer,:,cellnum] = conn

            cellnum += 1


cellnum = 0
for elY in range(nCellY):

    for elZ in range(nCellZ):

        for elX in range(nCellX):

            for layer in range(nLayersQuad):

                conn = np.zeros([tLayerNodes],dtype = int)
                count = 0
                seed0 = tBNodesLinear + tBNodesface*(elY*nL+layer) + ((nH-1)*(
nCellX+1)+tNodesX)*elZ + (nB)*elX
                seed1 = seed0 + tNodesX - (elX*(nB-1))
                seed2 = tBNodesLinear + tBNodesface*(elY*nL+layer) + ((nH-1)*(
nCellX+1)+tNodesX)*(elZ+1) + (nB)*elX
                for i in range(nB+1):
                    conn[i] = seed0 + i
                count += nB+1

                for i in range(nH-1):
                    conn[count+i] = seed1 + (nCellX+1)*i + 1
                count += nH-1

                for i in range(nB+1):
                    conn[count+i] = seed2 + nB - i
                count += nB+1

                for i in range(nH-1):
                    conn[count+i] = seed1 + (nCellX+1)*(nH-2-i)
                count += nH-1

                BConnQuad[layer,:,cellnum] = conn

            cellnum += 1

cellnum = 0
for elY in range(nCellY):

    for elZ in range(nCellZ):

        for elX in range(nCellX):

            for layer in range(nLayersLin):

                conn = np.zeros([tLayerNodes],dtype = int)
                count = 0
                seed0 = tBNodes1 + (elY*nL+layer)*(tBNodesQ2faceA) + (elZ*nB*
```

```
nCellX) + elX*nB
                seed3 = tBNodes1 + sbnodeswitch + (elY*nL+layer)*(tBNodesQ2faceB)
 + elZ*nH*(nCellX+1) + elX
                seed2 = seed0 + nCellX*nB
                seed1 = seed3 + 1
                for i in range(nB):
                    conn[i] = seed0 + i
                count += nB

                for i in range(nH):
                    conn[count+i] = seed1 + (nCellX+1)*i
                count += nH

                for i in range(nB):
                    conn[count+i] = seed2 + nB-1 - i
                count += nB

                for i in range(nH):
                    conn[count+i] = seed3 + (nCellX+1)*(nH-1-i)
                count += nH

                BConnQuad2[layer,:,cellnum] = conn
            cellnum += 1

nCentroidsLin = (nCellY*(nL) + 1)*(nCellX*nCellZ)
print(nCentroidsLin)
nCentroidsQuad = (nCellY*(nL))*(nCellX*nCellZ)
CentroidListLin = np.zeros([nCentroidsLin,3])
CentroidListQuad = np.zeros([nCentroidsQuad,3])

countc = 0
for elZ in range(nCellZ):
    z = H*0.5 + elZ*H + Origin[2]
    for elX in range(nCellX):
        x = B*0.5 + elX*B + Origin[0]
        for elY in range(nCellY):
            y0 = L*elY + Origin[1]
            if (elY == nCellY-1):
                nLr = nL+1
            else:
                nLr = nL
            for layer in range(nLr):
                y = y0 + (L/nL)*layer
                CentroidListLin[countc,:] = [x,y,z]
                countc+=1

countc = 0
for elZ in range(nCellZ):
    z = H*0.5 + elZ*H + Origin[2]
    for elX in range(nCellX):
        x = B*0.5 + elX*B + Origin[0]
        for elY in range(nCellY):
            y0 = L*elY + Origin[1]
            if (elY == nCellY-1):
                nLr = nL
            else:
                nLr = nL
            for layer in range(nLr):
                y = y0 + (L/nL)*(layer + 0.5)
                CentroidListQuad[countc,:] = [x,y,z]
                countc+=1

CentroidConnLin = np.zeros([nCell,nL+1],dtype = int)
```

```
    CentroidConnQuad = np.zeros([nCell,nL],dtype = int)
    CentroidConn = np.zeros([nCell,2*nL+1],dtype = int)

    cellnum = 0
    for elY in range(nCellY):
        seed0 = elY*(nL)
        for elZ in range(nCellZ):
            seed1 = seed0 + elZ*(nCellX*(nCellY*nL+1))
            for elX in range(nCellX):
                seed2 = seed1 + elX*(nCellY*nL+1)
                for layer in range(nL+1):
                    CentroidConnLin[cellnum,layer] = seed2 + layer
                cellnum+=1
    cellnum = 0
    for elY in range(nCellY):
        seed0 = elY*(nL) + nCentroidsLin
        for elZ in range(nCellZ):
            seed1 = seed0 + elZ*(nCellX*(nCellY*nL))
            for elX in range(nCellX):
                seed2 = seed1 + elX*(nCellY*nL)
                for layer in range(nL):
                    CentroidConnQuad[cellnum,layer] = seed2 + layer
                cellnum+=1


    # from matplotlib import pyplot as plt
    # fig = plt.figure()
    # ax = fig.add_subplot(projection='3d')

    # ax.scatter(BoundaryNodesLin[:,0],BoundaryNodesLin[:,1],BoundaryNodesLin[:,2])
    # ax.scatter(BoundaryNodesQuad[:,0],BoundaryNodesQuad[:,1],BoundaryNodesQuad
    [:,2])
    return (BoundaryNodesLin,BoundaryNodesQuad,BoundaryNodesQuad2,BoundaryNodesTotal,
    BConnQuad,BConnLin,BConnT,BConnQuad2,CentroidListLin,CentroidListQuad,
    CentroidConnLin,CentroidConnQuad)



# from matplotlib import pyplot as plt
# fig = plt.figure()
# ax = fig.add_subplot(projection='3d')

# ax.scatter(BoundaryNodesLin[:,0],BoundaryNodesLin[:,1],BoundaryNodesLin[:,2])
# ax.scatter(BoundaryNodesQuad[:,0],BoundaryNodesQuad[:,1],BoundaryNodesQuad[:,2])
```

Listing 5: External Meshing Code with Quadratic Wedge Element for the Unit Cell

F Appendix : Perturbation_Random.py

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 18 11:59:16 2022

@author: aniru

Basic Perturbation Method prepared from Catalanotti and Sebaey
"""
import numpy as np
import math
```

```
from math import factorial as fact
import matplotlib
from matplotlib import pyplot as plt

def bezier(t,n,Ps):
    B = np.zeros([3])

    for i in range(1,n+1):

        coeff = (fact(n)/(fact(i)*fact(n-i))) * (1-t)**(n-i) * (t)**i
        B += coeff * Ps[i-1]

    return B

def Controlpts(n):

    Ps = np.zeros([n,3])
    Ps[1,0] = 1
    Ps[2,0] = 2
    Ps[1,1] = 2

    Ts = np.arange(0,1.05,0.05)

    Curve = np.zeros([Ts.shape[0],3])

    for pt,t in enumerate(Ts):
        Curve[pt,:] = bezier(t,n,Ps)

    plt.plot(Curve[:,0],Curve[:,1])
    plt.plot(Ps[:,0],Ps[:,1])

def fibrePerturb(n,Points,fvf,fibreRadius,Dims,rhomaxval):
        for i in range(n):
            P = Points[i,:]
            uk = stochasticDisp(n, Points, fvf, fibreRadius, Dims,rhomaxval)
            P[0] += uk[1]
            P[2] += uk[2]
            P[1] += uk[0]
            Points[i,:] = P
        return Points

def stochasticDisp(n,Points,fvf,fibreRadius,Dims,rhomaxval):
        uk = np.zeros([3])
        rand1 = np.random.ranf()
        thetak = 2*np.pi*rand1

        lambdak = np.random.ranf()
        rhok = lambdak*rhomaxk(0, n, Points,rhomaxval)

        uk[0] = 0
        uk[1] = rhok*np.cos(thetak)
        uk[2] = rhok*np.sin(thetak)

        return uk

def rhomaxk(Dims,n,Points,rhomaxval):

    rhomaxk = rhomaxval-0.00001
    return rhomaxk
```

Listing 6: Basic Perturbation code for fibre centroids

G Appendix : WedgePost.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 13 19:42:52 2021

@author: aniru
VTK FILE FORMAT
"""

import numpy as np
np.set_printoptions(linewidth=np.inf)

jobname = 'Wedge_Linear_1'
def vtk_output_format(jobname,Points,nPoints,Cells,nCells,nodes,eltype,Xout,
    StrainNode,StressNode,ptdat):


    outputfile = jobname + '_output.vtk'

    with open(outputfile,'w') as output:

        # write header
        output.write('# vtk DataFile Version 3.1\n')
        output.write('Wedge Element Output File\n')
        output.write('ASCII\n')
        output.write('DATASET UNSTRUCTURED_GRID\n')

        #writing points
        # output.write('POINTS '+ str(nPoints) +' double\n')
        output.write(f"POINTS {nPoints} DOUBLE\n")

        for pt in Points:
            output.write(f"{pt[0]} {pt[1]} {pt[2]}\n")

        #writing Elems
        output.write("\n")
        output.write(f"CELLS {nCells} {nCells*nodes + nCells}\n")
        for el in Cells:
            output.write(f"{nodes}")
            for i in el:
                output.write(f" {i}")
            output.write("\n")

        #writing Cell type
        output.write("\n")
        output.write(f"CELL_TYPES {nCells}\n")
        for i in range(nCells):
            output.write(f"{eltype}\n")

        #write Point data
        if ptdat:
            output.write("\n")
            output.write(f"POINT_DATA {nPoints}\n")
            #write Displacement Vector
            output.write(f"VECTORS displacement DOUBLE\n")
            for i in range(nPoints):
                output.write(f"{Xout[i*3]} {Xout[i*3+1]} {Xout[i*3+2]}\n")

        #write FIELDS WOOOHOO
        if ptdat:
            output.write("\n")
```

```
            output.write(f"FIELD FieldData {2}\n")
            #Strain Time
            output.write(f"Strain {6} {nPoints} DOUBLE\n")
            for i in range(nPoints):
                p = StrainNode[i,:]
                output.write(f"{p[0]} {p[1]} {p[2]} {p[3]} {p[4]} {p[5]}\n")
            #Stress Time
            output.write(f"Stress {6} {nPoints} DOUBLE\n")
            for i in range(nPoints):
                p = StressNode[i,:]
                output.write(f"{p[0]} {p[1]} {p[2]} {p[3]} {p[4]} {p[5]}\n")


        output.close()

def vtk_outputgauss_format(jobname,Points,nPoints,nodes,eltype,GPDisp,StrainGPs,
    StressGPs,ptdat):

    outputfile = jobname + '_Gauss_output.vtk'

    with open(outputfile,'w') as output:

        # write header
        output.write('# vtk DataFile Version 3.1\n')
        output.write('Wedge Element Gauss Point Output File\n')
        output.write('ASCII\n')
        output.write('DATASET POLYDATA\n')

        #writing points
        # output.write('POINTS '+ str(nPoints) +' double\n')
        output.write(f"POINTS {nPoints} DOUBLE\n")

        for pt in Points:
            output.write(f"{pt[0]} {pt[1]} {pt[2]}\n")

        if ptdat:
            output.write("\n")
            output.write(f"POINT_DATA {nPoints}\n")
            #write Displacement Vector
            output.write(f"VECTORS displacement DOUBLE\n")
            for i in range(nPoints):
                output.write(f"{GPDisp[i,0]} {GPDisp[i,1]} {GPDisp[i,2]}\n")

        #write FIELDS WOOOHOO
        if ptdat:
            output.write("\n")
            output.write(f"FIELD FieldData {2}\n")
            #Strain Time
            output.write(f"Strain {6} {nPoints} DOUBLE\n")
            for i in range(nPoints):
                p = StrainGPs[i,:]
                output.write(f"{p[0]} {p[1]} {p[2]} {p[3]} {p[4]} {p[5]}\n")
            #Stress Time
            output.write(f"Stress {6} {nPoints} DOUBLE\n")
            for i in range(nPoints):
                p = StressGPs[i,:]
                output.write(f"{p[0]} {p[1]} {p[2]} {p[3]} {p[4]} {p[5]}\n")

        output.close()
```

```
def Update_Position(Xout,Coords,Nel,Tnode,dofn,scale):
    newCoords = np.zeros_like(Coords)

    for i in range(Tnode):
        for j in range(dofn):
            # print(i,j)
            newCoords[i,j] = Coords[i,j] + scale*Xout[i*3 + j]


    return newCoords

def ElemDisp(Xout,Conn,nodes,dofn):
    eldisp = np.zeros([nodes*dofn])
    for i,nd in enumerate(Conn):
        eldisp[i*dofn:(i+1)*dofn] = Xout[nd*dofn:(nd+1)*dofn]

    return eldisp
```

Listing 7: Post-processing and VTK output codes