A Study of Brittle Powder Compaction using a Combined Discrete Finite Element Approach

BY

I. PARKER

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Department of Mechanical Engineering
University of Cape Town
Rondebosch 7701
South Africa

December 1999
DECLARATION

This is to certify that the results, calculations and other work presented in this thesis are essentially my own, and that no part of it has been submitted for a degree at this or any other university.

Signed by candidate
Signature removed
I. Parker
December 1999
Acknowledgements

I wish to express my thanks to the following people for their guidance during my studies:

- My supervisors, Dr G.P. Mitchell and T.S. Pretorius.
- My colleagues at CERECAM.
- My wife, family and friends.

I would also like to thank De Beers Industrial Diamond Division for their financial support.
Synopsis

The Discrete Element Method (DEM) is a collection of numerical techniques based on a discontinuum idealisation of a physical system. Discrete Element Methods have been successfully applied to the simulation of the dynamic behaviour of granular media. Using the Finite Element Method to model the behaviour of each discrete element has resulted in the Combined Finite Discrete Element Method. This combination allows researchers to investigate the behaviour of assemblies of particles in which each particle need not behave in an entirely elastic manner. More importantly, discrete crack constitutive models may now be applied to each discrete element.

The purpose of this thesis is to simulate the macroscopic behaviour of an assembly of brittle particles, using the Combined Finite Discrete Element Method. Brittle behaviour is incorporated via the use of a Mode I and Mode II brittle failure constitutive models. The Mode I and Mode II failure models used are the Rankine and Tresca failure models, respectively. The development of a pre-processor to generate the initial configuration of the particle assembly also formed a major component of the thesis. Algorithms to fill a plane with randomly shaped polygons were developed and implemented for the pre-processor.

The cold compaction process is idealised as the compaction of a granular medium, in which each particle is a deformable polygon. Two algorithms are proposed and implemented to generate randomly shaped polygons. A further algorithm was developed to fill a simple bounding polygon with smaller polygons.

A commercial CFDEM code, ELFEN\Explicit, was used to investigate the effect of particle constitutive model and particle interaction on the macroscopic behaviour of the granular assembly. The built-in linear elastic and Mode I Rankine failure models were used in the initial simulations while a Tresca failure model was implemented to investigate the effect of a Mode II failure model.
# TABLE OF CONTENTS

**ACKNOWLEDGEMENTS** ......................................................................................... III

**SYNOPSIS** ........................................................................................................ IV

**TABLE OF CONTENTS** ....................................................................................... V

**LIST OF FIGURES** ............................................................................................ VII

**LIST OF TABLES** ............................................................................................... IX

1. **INTRODUCTION** ............................................................................................ 1

2. **COMPACTION OF BRITTLE MATERIALS** ....................................................... 4
   2.1 **POWDER FORMING PROCESS** ................................................................. 4
   2.2 **CONCEPTUAL MODEL OF BRITTLE MATERIAL COLD COMPACATION** ... 7
   2.3 **NUMERICAL MODELING OF COLD COMPACION** .................................. 10
      2.3.1 **The Continuum Approach** ............................................................... 10
      2.3.2 **The Discrete Approach** ................................................................. 12

3. **THE COMBINED FINITE DISCRETE ELEMENT METHOD** ......................... 14
   3.1 **GOVERNING EQUATIONS** .................................................................... 15
   3.2 **CONTACT DETECTION ALGORITHMS** .............................................. 17
      3.2.1 **Exhaustive Search** ......................................................................... 17
      3.2.2 **Grid Subdivision Search** ............................................................... 17
      3.2.3 **Binary Space Partition Search** .................................................... 18
      3.2.4 **Body-based Cell Search** ............................................................... 18
   3.3 **CONTACT INTERACTION** .................................................................... 19
      3.3.1 **Boundary Representation** ............................................................ 19
      3.3.2 **Contact Interaction** ................................................................. 20
   3.4 **FRACUTURE ALGORITHMS FOR DISCRETE ELEMENTS** .................... 22

4. **BRITTLE FAILURE MODELS** .................................................................... 24
   4.1 **STRAIN SOFTENING AND LOCALISATION** ......................................... 24
   4.2 **FRACTURE ENERGY BASED FAILURE MODELS** .................................. 26
   4.3 **RANKINE FAILURE CRITERION** ......................................................... 29
   4.4 **TRESCA FAILURE CRITERION** ......................................................... 31
   4.5 **MOHR-COULOMB FAILURE CRITERION** ........................................... 33

5. **IMPLEMENTATION OF SHEAR FAILURE MODEL** ..................................... 34
   5.1 **SELECTION CRITERIA** ......................................................................... 34
   5.2 **DERIVATION OF TRESCA FAILURE MODEL** ..................................... 35
      5.2.1 **Alternative Failure Function** .......................................................... 35
      5.2.2 **Derivation of Flow Vector** .............................................................. 36
      5.2.3 **Local Softening Law** ................................................................. 39
      5.2.4 **Stress Update Procedure** ............................................................. 40
   5.3 **IMPLEMENTATION** .............................................................................. 43
   5.4 **VERIFICATION OF TRESCA FAILURE MODEL** ................................. 45
5.4.1 Direct Tension Test ................................................. 45
5.4.2 Uniaxial Compression Test ........................................ 49
5.4.3 The Brazilian Test ................................................ 53

6. PARTICLE MODEL DEVELOPMENT ..................................... 56
6.1 NOMENCLATURE ...................................................... 57
6.2 CONCEPTUAL MODEL OF A BRITTLE PARTICLE .................. 59
6.3 MODEL GENERATION PIPELINE .................................... 60
6.4 PARTICLE GENERATION ............................................. 61
   6.4.1 Circumferential Polygon Generation .......................... 62
   6.4.2 Perturbation Polygon Generation ............................... 64
   6.4.3 Comparison of polygon generation algorithms ................ 65
6.5 PARTICLE PLACEMENT ............................................ 67
   6.5.1 Circle Contact Detection Algorithm .......................... 68
   6.5.2 Neighbour List Spatial Sorting Algorithm .................... 68
6.6 PARTICLE DISCRETISATION ....................................... 72
6.7 POST PROCESSING ALGORITHMS ................................... 73
   6.7.1 Particle Boundary Extraction .................................. 73
   6.7.2 Calculation of Volume Porosity ................................ 74
   6.7.3 Extraction of Particle Size Distribution ...................... 75
6.8 COMPUTER IMPLEMENTATION ...................................... 76
   6.8.1 Geometric Abstract Data Types ............................... 76
   6.8.2 Utility Abstract Data Types ................................... 80

7. SIMULATION OF COLD COMPACTION OF CBN POWDER ............... 82
7.1 COMBINED FINITE DISCRETE ELEMENT MODEL ...................... 82
7.2 MATERIAL PARAMETERS .......................................... 85
7.3 RESULTS OF CONSTITUTIVE MODEL PARAMETRIC STUDY ............ 87
   7.3.1 Mechanical Response of Linear Elastic Particle Assembly .... 87
   7.3.2 Mechanical Response of Rankine Particle Assembly ........... 90
   7.3.3 Mechanical Response of Tresca Particle Assembly .......... 92
   7.3.4 Comparison of Particle Assembly response ................... 94
7.4 EFFECT OF INTER-PARTICLE CONTACT .............................. 96

8. CONCLUSIONS AND RECOMMENDATIONS ................................ 98

REFERENCES ........................................................................ 100

APPENDIX A: EULER BACKWARD INTEGRATION SCHEME .................. 103

APPENDIX B: MATERIAL MODEL UTILITY FUNCTION SOURCE CODE .......... 105

APPENDIX C: ELFEN/EXPLICIT USER-MATERIAL SOURCE CODE .......... 110

APPENDIX D: C++ CLASS DESCRIPTION .................................. 119
LIST OF FIGURES

Figure 2-1: Typical geometry of crushed diamond particles ........................................... 5
Figure 2-2: Conceptual model of brittle powder compaction ........................................ 8
Figure 2-3: Schematic representation of Cold Compaction ........................................... 9
Figure 2-4: Finite element model of powder component .............................................. 11
Figure 3-1: Polygonal boundary representation ..................................................... 19
Figure 3-2: Contact Element Geometry ........................................................................ 20
Figure 3-3: Triangle-Quadrilateral Remeshing ......................................................... 22
Figure 3-4: Triangle-Triangle Remeshing ..................................................................... 23
Figure 4-1: Tensile stress-strain behaviour of concrete .............................................. 24
Figure 4-2: Uniaxial compressive stress-strain behaviour of concrete ...................... 25
Figure 4-3: Rankine Failure Surface, π plane representation ....................................... 29
Figure 4-4: Tresca yield surface .................................................................................. 31
Figure 5-1: Direct Tensile Test: mesh M5, M2 and M1 ............................................... 46
Figure 5-2: Comparison of Force Displacement Curves for Tresca Failure Model: Direct Tension Test .......................................................... 46
Figure 5-3: Comparison of crack propagation in Direct Tensile Test Specimen using Tresca Failure Model .................................................. 47
Figure 5-4: Comparison of Force Displacement curves for Rankine and Tresca Failure Models: Direct Tension Test, mesh M1 .................................................. 48
Figure 5-5: Crack Propagation in Direct Tensile for Rankine Failure Model: Mesh M1 .................................................. 48
Figure 5-6: Fracture pattern in uniaxial compression specimen .................................. 50
Figure 5-7: Finite element mesh for uniaxial compression test .................................. 50
Figure 5-8: Stress-Displacement history for compression test ................................... 51
Figure 5-9: Crack propagation in uniaxial compression specimen .............................. 52
Figure 5-10: Brazilian Test ......................................................................................... 53
Figure 5-11: Brazilian Test: finite element mesh and boundary conditions ................ 54
Figure 5-12: Crack propagation in Brazilian Test using Rankine Failure Model ......... 55
Figure 6-1: Geometric primitives .............................................................................. 57
Figure 6-2: Generic Cumulative Weight Plot .............................................................. 61
Figure 6-3: Schematic representation of Random Polygon Generation Algorithm 1.63 .................................................................................. 63
Figure 6-4: Schematic representation of polygon vertex randomisation ..................... 64
Figure 6-5: Results of Algorithm I Polygon Generation .............................................. 66
Figure 6-6: Results of Algorithm II Polygon Generation ............................................ 66
Figure 6-7: Neighbour List Sorting Algorithm ........................................................... 69
Figure 6-8: Determination of optimal bounding box .................................................. 70
Figure 6-9: Particle Boundary Representation ............................................................ 74
Figure 6-10: Finite Element Mesh ............................................................................. 78
Figure 6-11: Triangular Element data model .............................................................. 79
Figure 6-12: Continuous function discretisation ....................................................... 80
Figure 7-1: 103 Particle assembly ............................................................................... 83
Figure 7-2: Combined Finite Discrete Element mesh for particle assembly ............... 83
Figure 7-3: Macroscopic Stress-Strain Response of Elastic Assembly ...................... 88
Figure 7-4: Macroscopic Stress-Strain Response of Rankine Assembly .................... 90
Figure 7-5: Energy Response of Rankine Particle Assembly (μ=0.1) ......................... 91
Figure 7-6: Macroscopic Stress-Strain Response of Tresca Assembly ....................... 92
Figure 7-7: Cumulative Size Distribution ................................................................... 93
Figure 7-8: Sensitivity of Particle Assembly response to Constitutive Model .......... 94
Figure 7-9: Macroscopic response of Rankine Assembly........................................... 97
# LIST OF TABLES

Table 5-1: Material parameters for uniaxial test specimen ........................................ 45  
Table 5-2: Material parameters for concrete .................................................................. 49  
Table 7-1: Material parameters for CBN particles ...................................................... 86  
Table 7-2: Equivalent Stress in Rankine Assembly ..................................................... 97
1. Introduction

Sintered metal, hardmetal and ceramic components are extensively used in modern manufacturing processes. Examples of sintered components are wood, metal and rock cutting tools. The powder compaction phase of the process plays an important role in determining the final mechanical properties of the sintered component. Currently, most sintered components are designed by tapping into the design engineer's knowledge base and by trial and error. If the mechanisms involved in powder compaction were better understood, more efficient sintered components could be designed. Unfortunately, physically viewing the evolution of the powder material during compaction is not currently possible. Numerical simulation offers a convenient "numerical" laboratory for investigating powder compaction mechanisms.

The Finite Element Method (FEM) has been the main numerical tool used to gain an understanding of the macroscopic behaviour of powder compaction systems. Researchers have been able to accurately predict the porosity and density distributions within the compacted components. Implicit in the use of the FEM is the assumption that the powder can be treated as a continuum. Using continuum mechanics to describe the macroscopic behaviour of a granular material requires the use of complex constitutive models. These constitutive models are developed for specific classes of materials and often require extensive experimental data for calibration. However, no new information about the microscopic behaviour of the material can be extracted from the simulation results.

Cundill [13] has suggested that the use of a discontinuum based model would yield information about the microscopic mechanisms occurring within the granular material during compaction. The use of discontinuum models has already been successfully applied to such diverse fields as molecular dynamics and cosmology [18]. The Discrete Element Method (DEM) was developed to model systems for which the continuum assumption is invalid, i.e. the magnitude of the inter-particle distance is similar to the length scale of the simulation volume. Initially, the DEM was used to simulate jointed rock media and granular media. Current use of the DEM ranges from simulations of rock blasting to investigating the coherent structure in granular
materials. Improvements in the DEM and the availability of affordable powerful desktop computers have now made it possible to use this simulation technique to investigate systems containing large numbers of bodies.

The DEM has been used successfully to investigate the process of cold compaction [1] and hot pressing [20]. However, the initial investigations were based on the assumption that the individual powder particles are rigid. Brittle powder particles usually fracture during the cold compaction stage. Simple fracture models have been incorporated into the rigid discrete element simulation of granular media [19] with some success. Thus the need to incorporate more realistic fracture criteria into the simulation of powder compaction exists.

The development of the Combined Finite Discrete Element (CFDEM) has made it possible to incorporate constitutive models, developed for the FEM, into the DEM. Specifically, the use of softening plasticity failure models in the CFDEM provides a convenient method for the simulation of particle fracture. Thus, the CFDEM potentially offers an improved numerical tool for investigating brittle powder compaction.

The objectives of this thesis are:

- To develop and implement a particle generator that is capable of filling a general two-dimensional region with randomly shaped particles.
- To implement, and verify a simple fracture energy based softening plasticity failure model for use in a commercial CFDEM code.
- To investigate the effectiveness of using the CFDEM to simulate brittle powder cold compaction.
- To investigate the effect of the micro-mechanical constitutive law on the macroscopic response of the cold compaction simulation.
- To draw conclusions and recommendations based on the results of the study.

This thesis is structured in the following manner: Firstly, Chapter Two provides a brief overview of the powder forming process with an emphasis on brittle powder compaction. Current numerical techniques for simulating powder compaction are also
discussed briefly. Chapter Three is a review of the theory of the DEM, with emphasis on contact detection, contact interaction and solution strategies. Chapter Four contains a review of three brittle failure models and the relative advantages and disadvantages of each. The derivation, implementation and verification of a Tresca failure model with linear strain softening are contained in Chapter Five. Chapter Six deals with the development and implementation of the particle pre- and post-processors. The results of a Discrete Element simulation of the cold compaction of Cubic Boron Nitride (CBN) are presented in Chapter Seven. Finally, Chapter Eight ends with the conclusions and recommendations.
2. Compaction of Brittle Materials

This chapter provides an overview of brittle powder compaction, the conceptual model of brittle powder compaction and the numerical methods used to simulate brittle powder compaction.

2.1 Powder Forming Process

Sintered components are the result of subjecting a compacted powder to high heat, possibly with the application of a high pressure. The powder compaction process determines the porosity before sintering and the residual stresses within the particles. Both of these quantities affect the post sintering mechanical strength of the component. Numerous powder compaction techniques are available, but for this study only the cold compaction of brittle materials will be investigated. Examples of sintered brittle materials are Tungsten Carbide (WC), Cubic Boron Nitride (CBN) and Polycrystalline Diamond (PCD).

Cold compaction involves the densification, i.e. decreasing the porosity, of a powder without the addition of heat. The compaction occurs within a die, which is placed in either a mechanical or hydraulic press.

At a microscopic level, i.e. a particle level, the powder consists of particles whose shape ranges from spherical to angular. The process used to obtain the powder determines the particle shape and surface roughness. Brittle materials are generally reduced to powder form by milling and grinding, producing irregularly shaped particles. Figure 2-1 shows diamond particles that have been subjected to extensive milling.
In his study of the high pressure fracture of diamond particles, Bochechka [7] found that particle crushing, i.e. particle fracture, is dependent on the geometry and fracture toughness of the particles. His mathematical and experimental work showed that the probability of particle fracture increases when the sharp edge of one particle is in contact with the flat face of another.

Pre-compaction particle size and size distribution can play a significant role in determining the final porosity and size distribution of the compacted powder. Researchers investigating polycrystalline diamond [34] found that for a coarse powder the final porosity was not highly dependent on the initial average particle size. However, for a fine powder the final porosity is highly dependent on the initial average particle size. In general, for a given pressure the coarser particles crush more than the finer particles. The reason for the lower porosity obtained with coarser powders is due to the existence of smaller particles, formed by crushing, which fill the voids between the larger particles.

Experimental work on the cold compaction of CBN powders done by Andreev and Bezhenar [3] also revealed that the final porosity and particle size distribution was highly dependent on the initial particle size. They also found that their experimental
Compaction of Brittle Materials

results could be mathematically expressed by the Balshin model [8], which was
developed for covalent compounds. The Balshin model predicts the following
relationship between the porosity and the compaction pressure:

$$\pi = -\left(\frac{P}{A}\right)^n$$

where:

$\pi$ = volume percentage porosity

$P$ = pressure

$A$ = constant

$n$ = constant

For the CBN powder the coefficient $n$ was independent of the starting particle size
distribution whereas the coefficient $A$ was not.

A review of brittle powder compaction shows that the initial particle shape and
particle size distribution of the powder plays an important role in determining the
post cold compaction porosity and mechanical properties of the powder compact.
2.2 Conceptual Model of Brittle Material Cold Compaction

Based on the experimental and theoretical work, the cold compaction of a single component brittle material can be conceptualised as a process consisting of three distinct stages:

1. Particle Rearrangement.
2. Particle Crushing.
3. Void Filling by crushed particles.

The first stage involves the displacement of particles. During this stage, the particles coalesce under the influence of gravitational forces and the platen displacement. If present, electrostatic forces can significantly alter the particle motion. This stage is illustrated in Figure 2-2(a).

Once the particles are in contact with each other, the individual particles start to deform. This deformation results in stress fields being set up in the particles and if the stress level exceeds the strength of the material, particle crushing occurs. For brittle materials, crushing occurs due to brittle failure of the individual particles. Particle crushing is illustrated in Figure 2-2(b).

The final stage consists of the crushed particles occupying the voids formed between the larger particles. As the void volume decreases, the porosity of the particle system decreases. This void filling process is illustrated in Figure 2-2(c).
Figure 2-2: Conceptual model of brittle powder compaction

The critical parameters for the cold compaction stage are thus pressure, initial particle size, initial particle size distribution and some strength measure for the individual particles. The resultant output parameters are the volume percentage porosity, the average pore size and the final particle size distribution. A schematic representation of this process is shown in Figure 2-3.
Figure 2-3: Schematic representation of Cold Compaction
2.3 Numerical Modeling of Cold Compaction

2.3.1 The Continuum Approach

When the length scale of interest is very much larger than the inter-particle distance of a granular material, the material can be treated as a continuous medium. A simulation of the powder compaction system would thus involve the solution of a conventional solid mechanics problem. The solution has to satisfy the equations of motion, the compatibility conditions and the constitutive relations.

Using the continuum approach the powder compaction system is idealised in the following manner. Firstly, the powder is treated as a single continuous deformable domain. Secondly, the die is treated as a separate continuum. Finally, it is assumed that the two continuums cannot penetrate each other and that cold compaction is an isothermal and quasi-static process. Once a solution to the solid mechanics problem has been obtained, the stress, strain and apparent density fields can be extracted.

The Finite Element Method has been used extensively to find numerical solutions to the solid mechanics problem, i.e. the powder compaction system. Using this approach, the powder continuum is discretised with finite elements. The die is assumed to either rigid or deformable, and is discretised using appropriate finite elements. The contact and friction between the die and powder is normally modelled by using an interface element. Häggblad [16] investigated finite element simulations of the cold compaction of a hardmetal powder. Figure 2-4 (from [16]) illustrates a typical FE mesh used in the simulation of the compaction of a hardmetal component.

To characterise the macroscopic response of the material, a constitutive model has to be used. Ideally, this constitutive model should be based on the microscopic behaviour of the granular material. In practice this approach is generally not feasible and most constitutive models are developed based on experimental observation of the material of interest. Phenomenological plasticity constitutive models of the quadratic type have been developed to model metal powder
Compaction of Brittle Materials

compaction. These quadratic type constitutive models possess elliptical yield surfaces. Constitutive models developed for soil plasticity have also been used in the modelling of powder compaction. These models range from the simple Drucker-Prager or Coulomb models to the more advanced generalised Cap models [10].

Häggblad [16] provides an overview of some of the constitutive models currently being used in the simulation of cold compaction of hardmetal powders. Three approaches, arranged in ascending order of complexity, are utilised to derive the constitutive models, i.e. Cap plasticity, Multi-surface plasticity and Endochronic plasticity.

Figure 2-4: Finite element model of powder component
2.3.2 The Discrete Approach

To investigate the microscopic mechanisms occurring within a granular material, one has to treat the granular material as a discontinuum. The granular material is idealised as a collection of discrete particles, which are either rigid or deformable. If the individual particles are deformable, each particle is idealised as a continuum. Treating a media as a discontinuum provides a framework to investigate particle interaction and particle evolution. In the idealised model, each particle has to satisfy the equations of equilibrium or motion and is subjected to body and surface forces. Particle interaction occurs via inter-particle contact forces and, if present, electrostatic forces.

One popular discontinuum numerical method is the DEM that has been used in numerical investigations of the behaviour of granular materials. During the cold compaction stage, the powder material can be treated as a granular medium. The DEM allows for the finite displacements and rotations of discrete bodies and recognises new contacts automatically. Researchers have tended to use rigid circular or spherical discrete elements in their investigations of granular media [15]. DEM simulations of granular media using circular or spherical elements yield accurate results when compared with the physical results on ball bearings or glass disks. However, for soils the use of more complex shaped discrete elements has been suggested [35]. The use of elliptical and polygonal discrete elements has been found to yield more realistic results than the traditional circular-shaped discrete elements.

Polygonal discrete elements have until recently not been extensively used in the simulation of granular materials due to the computational expense associated with such elements. Issa and Nelson [19] investigated the behaviour of a granular material during a uniaxial strain test using 227 deformable polygonal discrete elements. They concluded that their model captured the essential behaviour of the granular material during loading and unloading. Tran and Nelson [36] developed an improved contact detection algorithm for polygonal discrete elements, and this allowed them to investigate the behaviour of 500 rigid polygonal discrete elements undergoing a uniaxial strain test. Similarly O’Connor [25] developed an improved
contact detection algorithm for use with two and three-dimensional polygonal discrete elements.

The use of the DEM specifically for the simulation of cold compaction has been limited to the use of rigid discrete elements. Aizawa et al [1], investigated the cold compaction of metal powder using rigid circular discrete elements. The inter-particle behaviour was modelled using a spring-dashpot model and was calibrated using a microscopic compression test. A random circle generator was used to generate the initial discrete element configuration. This collection of discrete elements satisfied a user specified particle size distribution. The numerical results were not compared to experimental data, but Aizawa et al [1] concluded that their approach yielded valid results.

Kenneth et al [20] investigated the consolidation of titanium powder by hot pressing. The discrete elements used were based on a collection of pin-jointed truss elements. The numerical results of the relative density evolution agreed closely with the observed experimental data.

It is important to note that it is not possible to include all the particles in the granular media in the discrete element simulation. As a first approximation, researchers [13] normally assume that no scale effect exists, i.e. the simulation results are independent of the number of particles included in the simulation.
3. The Combined Finite Discrete Element Method

As pointed out in Chapter 2, DEM simulation results can be improved by assuming that the elements are a deformable continuum. Initially, the deformable behaviour was governed by polynomial approximations of the strain field. Williams [39], used the concept of modal superposition to obtain simple deformation fields for each discrete element. Hua [13] used a linear approximation to the strain field to obtain deformable behaviour.

Barbosa and Ghaboussi [4] proposed that each deformable body in a system of particles be treated as a discrete unit. This discrete unit would then be idealised by a finite element model. The use of the finite element idealisation implies that any technique developed for traditional FEM can easily be incorporated into the DEM. In the following section the governing equations used in the Combined Discrete Finite Element Method (CFDEM) are discussed. This is followed by review of contact detection, contact force resolution and solution strategies.
3.1 Governing Equations

A deformable discrete element may undergo large translations and rotations with the possibility of finite strains. If finite strains are present, an appropriate formulation has to be used [4, 24]. For brittle and quasi-brittle materials, an infinitesimal strain formulation is sufficient. In the CFDEM each deformable discrete element is idealised as a finite element mesh and the equations of motion for the system of finite elements is:

\[ \dot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} = \mathbf{r}^{\text{ext}} - \mathbf{r}^{\text{int}} \]  \hspace{1cm} \text{Equation 3-1}

where:

\begin{align*}
\mathbf{M} & = \text{mass matrix} \\
\mathbf{C} & = \text{damping matrix} \\
\mathbf{r}^{\text{ext}} & = \text{external force vector} \\
\mathbf{r}^{\text{int}} & = \text{internal force vector}
\end{align*}

The external load vector includes contributions from contact stresses, external surface tractions and body forces. The internal force vector can be written in terms of element stresses as:

\[ \mathbf{r}^{\text{int}} = \sum_{k} \int_{V_k} \mathbf{B}_k^T \sigma_k dV_k \]  \hspace{1cm} \text{Equation 3-2}

where:

\begin{align*}
\mathbf{B}_k & = \text{element strain-nodal displacement matrix} \\
k & = \text{integer index running over all the elements in the system}
\end{align*}

The equations of motion are normally integrated using an explicit central difference integration scheme. Explicit integration schemes are conditionally stable, i.e. the integration time step must be less than some critical time step. For plane strain, the critical time step is defined as [30]:

15
The Combined Finite Discrete Element Method

\[ \Delta t_{\text{crit}} = L_c \sqrt{\frac{\rho (1 + \nu)(1 - 2\nu)}{E(1 - \nu)}} \]

Equation 3-3

where

- \( E \) = Young's Modulus
- \( \nu \) = Poisson’s Ratio
- \( L_c \) = characteristic element length
- \( \rho \) = material density

The equations of motion now become:

\[ \mathbf{M}_{n+1} \ddot{\mathbf{u}}_{n+1} + \mathbf{C}_{n+1} \dot{\mathbf{u}}_{n+1} = \left( \mathbf{r}^{\text{ext}} \right)_{n+1} - \left( \mathbf{r}^{\text{int}} \right)_{n+1} \]

Equation 3-4

where \( n+1 \) is the time step index.

The internal force vector is a function of the current stress state, which in turn is obtained from the constitutive model. In an explicit finite element system the incremental form of the constitutive law can be written as:

\[ \Delta \mathbf{\sigma}_{n+1} = \mathbf{D}\Delta \mathbf{\varepsilon}^e_{n+1} \]

Equation 3-5

where \( \Delta \mathbf{\sigma}_{n+1} \), \( \Delta \mathbf{\varepsilon}^e_{n+1} \) are the elastic stress and strain increments at time step \( n+1 \) and \( \mathbf{D} \) is the elastic constitutive matrix.

A review of the constitutive models considered for use in this study is presented in Chapter 4.
3.2 Contact Detection Algorithms

The process of geometric reasoning to determine whether or not two bodies occupy the same region of space is known as contact detection. Contact detection is the principal computational expense in the DEM. The cost of this depends on the number of discrete elements in the simulation and the complexity of the discrete element geometry. Improvements in contact detection algorithms have resulted in major improvements in the efficiency of discrete element codes [25]. A brief description of commonly used algorithms is presented below.

3.2.1 Exhaustive Search

The exhaustive search algorithm utilises no assumptions as to the spatial and temporal distribution of the particle system in the simulation volume. Contact checks are performed between every particle system for every time step. The exhaustive search algorithm will always resolve contact between bodies thus making it stable and reliable. However, for most practical problems this usually involves prohibitive computational effort.

3.2.2 Grid Subdivision Search

The grid subdivision search algorithm divides the simulation volume uniformly into rectangular cells each containing zero or more particles. Each cell has a buffer zone to account for particles moving between cells. Assuming that the particles cannot move between cells within a minimum number of time steps, exhaustive contact detection only has to be performed between particles in the same cell.

A major disadvantage of this algorithm is its sensitivity to the spatial distribution of the particles. If all the particles were located in one cell, the grid subdivision search would reduce to an exhaustive search.
3.2.3 Binary Space Partition Search

Binary space partition relies on partitioning the simulation volume with a series of cutting planes. These cutting planes, which are not necessarily parallel to the Cartesian axes, successively divide the simulation volume. A binary tree is used to store the location of the particles and only non-empty partitions are maintained in the tree. Within each partition, exhaustive searching is used for contact detection.

The binary space partition search performs best when the particles in the simulation volume do not move between the partitions. An adaptive binary space partition scheme can be adopted for dynamic systems. However, the additional computational expense involved in maintaining the binary space partition does not make this scheme suitable for simulations involving particles undergoing large displacements.

3.2.4 Body-based Cell Search

Instead of subdividing the simulation volume, an alternative approach is to centre the search around the particles. Each particle is enclosed by a cell centred about the particle’s centroid. This cell is a simple geometric shape, i.e. a circle or rectangle in a two dimensional system. All particles that lie within the object’s cell are placed in a neighbour list. Only particles in the neighbour list are subjected to an exhaustive contact search. The neighbour list does not have to be created at every time step.

The advantage of the body based cell search lies in its efficiency. This is due to the low computational expense of creating the neighbour list. Furthermore, the algorithm is not severely dependent on the spatial distribution of the particles. Finally, the computer implementation of the search algorithm does not require extensive computational “bookkeeping”.
3.3 Contact Interaction

In a multi-body system, and in the absence of body forces, the only path for stress transfer between bodies is via contact stresses. The accuracy of the contact force resolution depends on the body boundary representation and the contact law used.

3.3.1 Boundary Representation

With the advent of efficient contact detection algorithms, the use of polygonal discrete elements has gained popularity [25, 36]. The boundary \( \Gamma \) of a body is approximated by a finite number of straight line segments. The set of straight line segments constitute the boundary approximation \( \Gamma' \).

![Figure 3-1: Polygonal boundary representation](image)

The layer close to and interior to \( \Gamma' \) is referred to as the contact layer. A contact element is defined as being that part of the contact layer associated with one segment of the boundary. Since \( \Gamma' \) consists of straight line segments, the associated contact element is known as a line contact element. Each line contact element is defined by the nodes \( i \) and \( j \), and the constant thickness of the contact layer along the line segment is known as the field.
Figure 3-2: Contact Element Geometry

The discrete element boundary $\Gamma'$ is discretised into a collection of contact elements.

3.3.2 Contact Interaction

The three approaches that are commonly used to satisfy the contact constraints between interacting bodies are the Lagrange multiplier method, the least squares method and the penalty function method. Pande et al [28], provide a review of the first two methods as well as a commonly used linear contact interaction model. Most explicit discrete element codes utilise the penalty function formulation.

The most commonly used algorithm to determine contact force is described according to the following definitions:

- When a node is interior to the contact layer it is known as a contacting node.
- The contact element is then referred to as the contacting element.
- A contacting couple is defined as the combination of the contacting node and contacting element.

Each contacting element has two penalty parameters associated with it, i.e. $K$, the penetration penalty stiffness and $C_v$, the velocity penetration damping coefficient. The contact element field defines the upper limit of the penetration of a contacting node. If the contact node penetrates the contacting element by some distance $\delta$, with velocity $\delta'$ and $\delta \leq \text{field}$, then the contact force $f_c$ is calculated using:
\[ f = \frac{\delta}{\text{field}} K + \frac{\delta'}{\text{field}} C_v \]  

Equation 3-6

The contact force is assumed to act in a direction normal to the contacting element and pass through the contacting node. Krishnasamy et al [22] discuss the problems that can arise during the determination of the contact force direction.
3.4 Fracture Algorithms for Discrete Elements

Continuum formulations such as damage mechanics and fracture based softening plasticity attempt to model the post fracture behaviour of a material. Deformable discrete elements are capable of being separated into two or more elements when a critical stress state is reached. If partial failure occurs, only the element boundary will change.

Munjiza et al [24] assume that the finite element idealisation of the discrete element is valid until the load carrying capacity in the localisation zone is reduced to zero. Once the load carrying capacity is reduced to zero, a crack is assumed to open along the localisation zone. A failure model is required to detect the localisation zone. Munjiza [23] used a Rankine softening plasticity failure model to capture the localisation zone in rock blasting simulations. A description of this failure model is presented in Chapter 4.

If a finite element undergoes failure, at the integration point level, a new finite element mesh must be generated. This involves the subdivision of the original (or 'parent' element) into one or more 'child' elements. In two dimensions the remeshing can be achieved in one of two ways. The first method involves dividing the parent element along a plane passing through the centroid of the element. The orientation of the plane is determined by the failure model. This algorithm is illustrated in Figure 2-3.

![Figure 3-3: Triangle-Quadrilateral Remeshing](image)

Figure 3-3: Triangle-Quadrilateral Remeshing
The combined finite discrete element method

The major disadvantage of this algorithm is that both triangular and quadrilateral elements are generated during the remeshing procedure.

To ensure that only triangular child elements, from a triangular parent element, are generated the following remeshing procedure can be used. Firstly, the crack plane is assumed to pass through the element centroid. If the parent can be divided into two new triangular elements, the remeshing is complete. The crack plane is now translated along one of the axes until it intersects one node and one edge of the parent element. This ensures that two new triangular elements are generated. If the two new elements are smaller than a user's specified critical element size, the discretisation is discarded. This technique is illustrated in Figure 3-4. Finally, if none of the above two steps are successful, the crack plane is assumed to lie along one of the element edges and the crack propagates along the inter-element boundary.

Figure 3-4: Triangle-Triangle Remeshing
4. Brittle Failure Models

This chapter outlines various failure models that have been used to model brittle material behaviour.

4.1 Strain Softening and Localisation

Experimental tests on heterogeneous materials reveal that the test specimen neither yields nor fails at maximum load but that the load decreases gradually as the displacement increases. Figure 4-1 and Figure 4-2 (from [29]) illustrate typical stress-displacement curves for a concrete or rock specimen under uniaxial tension and uniaxial compression, respectively. These phenomena are known as strain softening and occur under uniaxial tension, uniaxial compression and triaxial stress states.

Figure 4-1: Tensile stress-strain behaviour of concrete
Figure 4-2: Uniaxial compressive stress-strain behaviour of concrete

Strain softening causes instabilities in material behaviour and results in bifurcation, i.e. the stress strain path is not uniquely defined. In brittle materials these instabilities manifest themselves as highly localised zones of intense straining caused by microcracking. The phenomenon whereby intense straining occurs within a narrow band is known as localisation. Crack propagation generally follows the coalescence of these bands of localisation.
4.2 Fracture energy based failure models

Researchers [6, 29] investigating localised fracture using the FEM have found that if the slope of the softening branch is treated as a material property, then the results are mesh dependent. To provide objective, i.e. mesh independent, results a nonlocal failure law has to be used. A nonlocal material law is one that is not valid pointwise in the continuum, but depends on one or more spatial quantities. Fracture energy based models were developed to provide objective results.

Hillerborg et al [38] proposed a two parameter fictitious crack model, with the parameters being tensile strength and the fracture energy release rate. If one considers a direct tension specimen in its post-peak region, the model assumes that elastic unloading occurs throughout the specimen except for an infinitely thin band of cracked material. The behaviour of the band of cracked material is assumed to be of Mode I type and is governed by a constitutive relation defined in terms of the normal stress, σ, and the crack width, δ. It is assumed that the crack has a stress transfer capability that reduces the stress as the crack opens. The stress is assumed to be zero once the crack width is equal to δ_u. The fracture energy release rate in tension is evaluated using

\[ G_I = \int_0^{\delta_u} \sigma \cdot d\delta \]  
\text{Equation 4-1}

The Mode I fracture energy release rate, in plane strain, can be calculated using [2]:

\[ G_I = \frac{K_1^2}{E} (1 - \nu^2) \]  
\text{Equation 4-2}

where \( K_1 \) is the fracture toughness.

For a triaxial compression specimen, the fracture energy release rate in shear, \( G_{fs} \), is approximated by the integral under the nominal shear stress - tangential displacement curve [38]. The fracture energy release rate in shear is typically an order of magnitude greater than the fracture energy release rate in tension [38]. Experimental results for the fracture energy release rate in shear are not easily obtainable.
Brittle Failure Models

William et al. [38] developed a composite damage model that was a smeared crack version of the fictitious crack model. They investigated composite fracture models for localised tensile fracture, localised shear slip and tensile and shear failure. Munjiza [23] investigated a localised tensile fracture model based on a strain softening discrete crack model.

To provide for mesh objectivity, the local softening modulus in the post peak region is a function of the material strength, fracture energy release rate and the element control length. The local softening slope in tension is defined as:

\[ H_f = -\frac{(\sigma_f)^2}{2 \cdot G_f} \cdot h \]  

Equation 4-3

where \(\sigma_f\) is the uniaxial tensile strength and \(h\) is the maximum element length in the direction normal to the localisation plane. Similarly, for mesh objectivity in shear the local softening modulus is defined as:

\[ H_{fs} = -\frac{(\tau_f)^2}{2 \cdot G_{fs}} \cdot h \]  

Equation 4-4

where \(\tau_f\) is the shear strength and \(h\) is once again the maximum element length in the direction normal to the crack plane.

For a plane triangular element, the element control length \(h\) is defined as the diameter of the circle with the same area as the element. The control length for the \(i^{th}\) element is thus defined as:

\[ h_i = 2 \cdot \sqrt{\frac{A_i}{\pi}} \]  

Equation 4-5

where \(A_i\) is the element area.
Brittle Failure Models

The implementation of these models implies that the softening modulus is different for each element in the mesh. This ensures that whatever the element size, a constant amount of energy is released per crack. A further consequence of Equation 4-3 and Equation 4-4 is that they set limits of the maximum element size to ensure that the tangent modulus does not become positive.

In the FEM, the fracture energy based failure model can be treated as a phenomenological model that is used to simulate the macroscopic phenomenon of strain softening. However, in the context of the current work, the fracture energy based failure model is being used to describe microscopic response of the individual particles in the brittle powder. The major reason for incorporating a fracture energy based failure model into the CFDEM is to provide a mechanism to track the evolution of the particles in the brittle powder during the cold compaction stage.
4.3 Rankine Failure Criterion

The Rankine failure criterion states that brittle failure occurs when the maximum principal stress at a point in the material reaches a value equal to the uniaxial tensile strength $\sigma_f$, regardless of the normal or shearing stresses that occur on planes through the point. The Rankine failure surface is defined by the following equations:

$$\sigma_1 = \sigma_f \quad \sigma_2 = \sigma_f \quad \sigma_3 = \sigma_f \quad \text{Equation 4-6}$$

where $\sigma_1$, $\sigma_2$, and $\sigma_3$ are the principal stresses.

These equations result in three planes perpendicular to the $\sigma_1$, $\sigma_2$, and $\sigma_3$, axes respectively. It is important to note that the tensile strength can also be a function of the equivalent plastic strain. Figure 4-3 illustrates the Rankine failure surface in the $\pi$-plane. For the case of isotropic softening, the failure surface uniformly tends to the origin as the material undergoes softening.

![Figure 4-3: Rankine Failure Surface, $\pi$ plane representation](image)

Figure 4-3: Rankine Failure Surface, $\pi$ plane representation
Brittle Failure Models

Willam et al [38] developed a Mode I smeared crack failure criterion based on the Rankine failure criterion, using a yield function of the form

$$F(\tilde{\sigma}, \kappa) = \sigma_i - \sigma_{\beta} (\kappa), \quad \sigma_{\beta} (\kappa) = \sigma_{\beta} (\tilde{\varepsilon}_f) = \sigma_{\beta} + H_1 \cdot \tilde{\varepsilon}_f$$

Equation 4-7

where $\kappa$ is a scalar state variable equal to the equivalent plastic strain. Unlike the Rankine failure criterion described above, Williams's [39] model only monitors the major principal stress $\sigma_1$, for the failure criterion. Bicanic [24] implemented a discrete crack version of the Williams model. In the discrete crack model, a crack is formed when the current tensile strength reaches zero. The crack lies in a plane perpendicular to the current maximum principal stress direction.

The Rankine softening plasticity model incorporated a bilinear local softening material law. This softening law is dependent on the fracture energy release rate and local control length. The model is assumed to be isotropic since the equivalent plastic strain is only monitored in the major principal direction.

The advantages of the Rankine failure criterion are as follows. Firstly, it provides a simple means of detecting tensile failure. Munjiza [23] successfully used the Rankine softening plasticity failure model in the simulation of rock blasting. Secondly, only two parameters, i.e. the tensile strength and the fracture energy release rate in tension are required to calibrate the model.
4.4 Tresca Failure Criterion

The Tresca yield criterion states that the material starts to yield when the shear stress at a point exceeds some critical value $Y$. The value of $Y$ is conventionally taken as the failure stress in simple tension. The Tresca failure criterion extends the concept of a yield surface to a failure surface. Mathematically the failure surface can be written as

$$\sigma_1 - \sigma_3 = Y(\kappa)$$  \hspace{1cm} \text{Equation 4-8}

As with the Rankine failure criterion, the strength parameter may be a function of the equivalent plastic strain. The Tresca failure criterion can be visualised as a hexagonal cylinder, with its centreline aligned with the hydrostatic line, in principal stress space. Figure 4-4 (from [27]) illustrates the Tresca failure surface in the principal stress space.

Figure 4-4: Tresca yield surface
Brittle Failure Models

If softening plasticity is incorporated into the failure criterion, failure is assumed to occur when the current shear strength is reduced to zero. In a discrete crack model, a crack is created when the material fails. For the two-dimensional case, this crack is orientated along one of two possible fracture planes. These fracture planes are defined by the angle $\varphi$, which is measured counter-clockwise relative to the positive global x-axis. The angle is defined as

$$\varphi = \theta \pm \frac{\pi}{4}$$  \hspace{1cm} \text{Equation 4-9}$$

where $\theta$ is the angle formed between the positive global x-axis and the first principal stress axis. The sign of the second term may be determined by using a pseudo random number generator. An alternative approach presented by Bazant and Cedolin [5] is to use an objective energy based criterion to select the direction of crack propagation. Their criterion states that the second order work

$$\delta^2 W = \frac{1}{2} \delta r \delta u$$ \hspace{1cm} \text{Equation 4-10}$$

should be minimal for a stable path where $\delta r$ and $\delta u$ are variations of the load and displacements, respectively.

The main advantage of the Tresca failure criterion is its simplicity and ease of calibration, i.e. only the tensile, compressive or shear strength of the material needs to be determined. If a softening plasticity model is incorporated into the basic Tresca failure criteria, then the Mode II fracture energy release rate needs to be determined as well.
4.5 Mohr-Coulomb Failure Criterion

The Mohr-Coulomb failure criterion is defined by:

$$|\tau| = c - \sigma_n \tan \phi$$  \hspace{1cm} \textbf{Equation 4-11}

where $\tau$ is the magnitude of the shearing stress, $\sigma_n$ is the normal stress, $c$ is the cohesion and $\phi$ is the angle of internal friction. The Mohr-Coulomb failure surface defines a conical surface in principal stress space whose cross section at any point is an irregular hexagon. The conical nature of the failure surface is a consequence of the fact that the failure depends on the hydrostatic stress.

If strain softening is incorporated into the failure criterion, the material is assumed to fail due to shear slip when the cohesion is reduced to zero. In the discrete crack model, the failure plane can lie along two possible planes. These planes are defined by:

$$\varphi = \theta \pm \left( \frac{\pi}{4} + \frac{\phi}{2} \right)$$  \hspace{1cm} \textbf{Equation 4-12}

Once again the direction of crack propagation may be determined by using a pseudo random number generator to determine the sign of the second term or by using Bazant and Cedolin's [5] criterion.

The Mohr-Coulomb failure criterion has been successfully used to simulate the failure of brittle materials such as rock and concrete. To calibrate the model, a series of tri-axial tests have to be performed. If an objective softening law is to be incorporated into the failure criterion, then the Mode II fracture energy release rate needs to be determined.
5. Implementation of Shear Failure Model

This chapter details the derivation of the Tresca softening plasticity failure model, its implementation and verification.

5.1 Selection Criteria

The selection of a failure model for use in the CFDEM simulation of brittle powder compaction is based on the following criteria:

1. model calibration
2. computational expense

Firstly, the ease of failure model calibration is important. Since the failure model is applied on a microscopic level the microscopic response of the material must be used to calibrate the failure model. This can be problematic since the particle size of brittle powders is measured in microns. At this scale the only reliable material test that can be performed is the splitting tensile test.

Secondly, the material model used must not be computationally expensive. The powder compaction simulation volume will typically contain more than 100 combined discrete finite elements. Since each discrete element contains a finite element mesh, the total number of elements in the system can be large. Ideally the failure model should not require many iterations to converge to a solution.

Based on the above two criteria, the Rankine and Tresca softening plasticity failure models were chosen for the powder compaction simulations. The Rankine model is incorporated into the commercial CFDEM code, ELFEN/Explicit [30], being used for the analysis. The Mohr-Coulomb model was not chosen because of a lack of microscopic triaxial compression test experimental data.
5.2 Derivation of Tresca Failure Model

The derivation of a rate independent Tresca softening plasticity failure model is given below. It is assumed that an isotropic linear elastic material model governs the intact material and an objective linear softening law governs the post fracture behaviour.

5.2.1 Alternative Failure Function

The method presented by Owen and Hinton [27] is used to derive a convenient form for the failure criterion. This method consists of the following substitution for the principal stresses

\[
\begin{pmatrix}
\sigma_1 \\
\sigma_2 \\
\sigma_3
\end{pmatrix} = \frac{2\sqrt{J_2}}{\sqrt{3}} \begin{pmatrix}
\sin\left(\theta + \frac{2\pi}{3}\right) \\
\sin(\theta) \\
\sin\left(\theta + \frac{4\pi}{3}\right)
\end{pmatrix} + \frac{J_1}{3} \begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
\]

Equation 5-1

where

\[\sigma_1 > \sigma_2 > \sigma_3\]

\[\theta = \frac{1}{3} \arcsin\left(\frac{3\sqrt{3} J_3}{2 J_2^{3/2}}\right), \quad -\frac{\pi}{6} \leq \theta \leq \frac{\pi}{6}\]

Equation 5-2

For plane strain problems the second and third deviatoric stress invariants become

\[J_2 = \frac{1}{2} \left(\sigma_x^2 + \sigma_y^2 + \sigma_z^2\right) + \tau_{xy}^2\]

\[J_3 = \sigma_z (\sigma_z^2 - J_2)\]

Equation 5-3
\[ F(J_2, \theta) = 2\sqrt{J_2} \cos(\theta) - Y_f(\kappa) \]  \hspace{1cm} \text{Equation 5-4}

The effective stress \( \sigma_{\text{eff}} \) is defined by

\[ \sigma_{\text{eff}} = 2\sqrt{J_2} \cos(\theta) \]  \hspace{1cm} \text{Equation 5-5}

### 5.2.2 Derivation of Flow Vector

For a general failure function \( F \) the flow vector is defined by

\[ \mathbf{a} = \frac{\partial F}{\partial \sigma} \]  \hspace{1cm} \text{Equation 5-6}

where for plane strain

\[ \sigma = \left( \sigma_x, \sigma_y, \tau_{xy}, \sigma_z \right)^T \]  \hspace{1cm} \text{Equation 5-7}

For a failure criterion of the form \( F = F\left(\sqrt{J_2}, \theta\right) \), the flow vector may be expressed as

\[ \mathbf{a}^T = \frac{\partial F}{\partial \left(\sqrt{J_2}\right)} \frac{\partial \left(\sqrt{J_2}\right)}{\partial \sigma} + \frac{\partial F}{\partial \theta} \frac{\partial \theta}{\partial \sigma} \]  \hspace{1cm} \text{Equation 5-8}
As presented in Owen and Hinton [27], the flow vector can be expressed in the following form

\[ \mathbf{a} = C_2 \mathbf{a}_2 + C_3 \mathbf{a}_3 \]  

Equation 5-9

where

\[ \mathbf{a}_2^T = \frac{\partial \left( \sqrt{J_2} \right)}{\partial \sigma} = \frac{1}{2\sqrt{J_2}} \left( s_x, s_y, 2\tau_{xy}, s_z \right) \]

\[ \mathbf{a}_3^T = \frac{\partial J_3}{\partial \sigma} = \left[ s_y s_z + \frac{J_2}{3} \right] \left[ s_x s_z + \frac{J_2}{3} \right] - 2s_z \tau_{xy} \left[ s_x s_y - \tau_{xy}^2 + \frac{J_2}{3} \right] \]

\[ C_2 = \left( \frac{\partial F}{\partial \left( \sqrt{J_2} \right)} - \frac{\tan(3\theta)}{\sqrt{J_2}} \frac{\partial F}{\partial \theta} \right) \]

\[ C_3 = \frac{-\sqrt{3}}{2 \cos(3\theta)} \left( J_2 \right)^{3/2} \frac{\partial F}{\partial \theta} \]

Equation 5-10

To determine the flow vector for the Tresca failure surface, only \( C_2 \) and \( C_3 \) need to be found. Differentiating Equation 5-4 with respect to \( (J_2)^{1/2} \) and \( \theta \) gives

\[ \frac{\partial F}{\partial \theta} = -2\sqrt{J_2} \sin(\theta) \]

\[ \frac{\partial F}{\partial \left( \sqrt{J_2} \right)} = 2\cos(\theta) \]

Equation 5-11
Implementation of Shear Failure Model

Substituting Equation 5-11 into Equation 5-10:

\[ C_2 = 2 \cos(\theta)(1 + \tan(\theta)\tan(3\theta)) \]

\[ C_3 = \frac{\sqrt{3} \sin(\theta)}{J_2 \cos(3\theta)} \]

Equation 5-12

The flow vector \( \mathbf{a} \) is not uniquely defined at the corners of the Tresca failure surface. When \( \theta = \pm 30^\circ \), the values of \( C_2 \) and \( C_3 \) become indeterminate thus making the direction of failure straining indeterminate. Owen and Hinton [27] suggest that the corners of the Tresca failure surface should be rounded, thus making the direction of failure straining unique. The assumption is that the flow vector at the corners is defined by the Von Mises failure surface that passes through the corner.

The Von Mises failure criterion is defined by

\[ F(J_2, \theta) = \sqrt{3J_2} - Y_f(\kappa) \]

Equation 5-13

From Equation 5-10 \( C_2 \) and \( C_3 \) are then defined by

\[ C_2 = \sqrt{3} \quad C_3 = 0 \]

Equation 5-14

For practical purposes, Equation 5-11 is used to evaluate the flow vector when \( |\theta| \leq 29^\circ \) and Equation 5-14 for all other values of \( \theta \).
5.2.3 Local Softening Law

It is assumed that the failure strength of the material is governed by a linear work softening law. For work softening the rate of change of the softening parameter \( d\kappa \) is equal to the rate of change of the dissipated energy in inelastic work, i.e.

\[
d\kappa = dW^{\text{inel}} = d\epsilon^f \sigma = d\epsilon^{ef} \sigma_{\text{eff}}
\]

Equation 5-15

where \( d\epsilon^{ef} \) is the incremental equivalent failure strain.

At increment \( n+1 \) the failure strength of the material is defined as:

\[
Y_f(n+1)(\epsilon_n^{ef}) = Y_f^0 + He_n^{ef}
\]

Equation 5-16

where

\[
\begin{align*}
Y_f(n+1) & \quad \text{is the failure strength at any increment } n+1. \\
Y_f^0 & \quad \text{is the initial failure strength} \\
\epsilon_n^{ef} & \quad \text{is the equivalent failure strain at the } n\text{-th increment} \\
H & \quad \text{is the local softening modulus}
\end{align*}
\]

To ensure mesh objectivity, the local softening modulus is a function of the fracture energy release rate in shear \( G_{fs} \), and the local control length \( h_i \). The local softening modulus for the \( i\)-th element is defined by

\[
H_i = \frac{\tau_f^2}{2G_{fs}} h_i
\]

Equation 5-17
5.2.4 Stress Update Procedure

The elastic plastic stress-strain law is integrated using an Euler backward integration scheme (see Appendix A). The Euler backward integration scheme is defined by

\[
\Delta \sigma_{n+1} = D(\Delta \varepsilon_{e,n+1}^f - \Delta \varepsilon_{n+1}^f)
\]

\[
\Delta \varepsilon_{e,n+1} = \Delta \lambda a_{n+1}
\]

\[
\Delta \varepsilon_{n+1}^f = \Delta \lambda
\]

\[
F_{n+1} = F(\sigma_{n+1}, \varepsilon_{n+1}^f) = 0
\]

Equation 5-18

The incremental plastic multiplier \(\Delta \lambda\) is obtained from the consistency condition. An elastic predictor, plastic corrector approach has been adopted in the stress update procedure. The elastic predictor and plastic corrector are defined by

\[
\Delta \sigma_{e,n+1} = D \Delta \varepsilon_{n+1}
\]

\[
\Delta \sigma_{n+1} = \Delta \sigma_{e,n+1} - \Delta \lambda D a_{n+1}
\]

Equation 5-19

where for the plane strain case the elasticity matrix \(D\) is defined by

\[
D = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix}
1 & \frac{\nu}{1-\nu} & 0 & \frac{\nu}{1-\nu} \\
\frac{\nu}{1-\nu} & 1 & 0 & \frac{\nu}{1-\nu} \\
0 & 0 & \frac{1-2\nu}{2(1-\nu)} & 0 \\
\frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 0 & 1
\end{bmatrix}
\]

Equation 5-20

where \(E\) and \(\nu\) are the Young's modulus and Poisson's ratio, respectively.
The incremental plastic multiplier is determined by making use of a Taylor series expansion around the elastic predictor state and the initial equivalent plastic strain, neglecting second and higher order terms. This yields

\[ F\left(\sigma_{n+1}^e, \varepsilon_{n+1}^{ef}\right) = F\left(\sigma_{n+1}^e - \Delta \lambda\mathbf{D}a_{n+1}^T, \varepsilon_{n+1}^{ef} + \Delta \lambda\right) = 0 \]

\[ \Rightarrow F\left(\sigma_{n+1}^e, \varepsilon_{n+1}^{ef}\right) + \left(\Delta \lambda \frac{\partial}{\partial \varepsilon_{n+1}^{ef}} - \Delta \lambda a_{n+1}^T \frac{\partial}{\partial \sigma_{n+1}}\right) F\left(\sigma_{n+1}^e, \varepsilon_{n+1}^{ef}\right) = 0 \]

\[ \Rightarrow F\left(\sigma_{n+1}^e, \varepsilon_{n+1}^{ef}\right) - \Delta \lambda a_{n+1}^T \mathbf{D}a_{n+1} + \Delta \lambda H = 0 \]

\[ \therefore \Delta \lambda = \frac{F\left(\sigma_{n+1}^e, \varepsilon_{n+1}^{ef}\right)}{a_{n+1}^T \mathbf{D}a_{n+1} - H} \quad \text{Equation 5-21} \]

where

\[ H = \frac{\partial}{\partial \varepsilon_{n+1}^{ef}} F\left(\sigma_{n+1}^e, \varepsilon_{n+1}^{ef}\right) \quad \text{Equation 5-22} \]

Once the incremental plastic multiplier has been obtained, the stress and equivalent plastic strain is updated, i.e.

\[ \sigma_{n+1} = \sigma_{n+1}^e - \Delta \lambda \mathbf{D}a_{n+1}^T \quad \text{Equation 5-23} \]

\[ \varepsilon_{n+1}^{ef} = \varepsilon_{n+1}^{ef} + \Delta \lambda \]

If this new stress state does not satisfy the failure criterion

\[ F\left(\sigma_{n+1}, \varepsilon_{n+1}^{ef}\right) = 0 \quad \text{Equation 5-24} \]

to within some tolerance, then the current stress state can be treated as a trial stress state, i.e. an elastic predictor. Equation 5-21 and Equation 5-23 are used to determine the new incremental plastic multiplier and stress state and equivalent failure strain, respectively. The iterative scheme continues until the current stress state lies on the failure surface.
If a failure surface is a linear function of stress and has a linear softening law, then the Taylor series expansion should ensure a return to the failure surface in one iteration [29]. However, due to the smoothing of the corners of the Tresca failure surface, more than one iteration may be required to ensure a return to the failure surface.

The elastic and inelastic energy, $W^e$ and $W^{inel}$ respectively, are updated via a trapezoidal integration scheme. The energy update scheme is defined by

$$W_{n+1}^e = W_n^e + \Delta W_{n+1}^e$$
$$W_{n+1}^{inel} = W_n^{inel} + \Delta W_{n+1}^{inel}$$

$$\Delta W_{n+1}^e = \frac{1}{2}(\sigma_{n+1} + \sigma_n)^T \Delta \varepsilon_{n+1} - \Delta W_{n+1}^{inel}$$  \hspace{1cm} \text{Equation 5-25}$$
$$\Delta W_{n+1}^{inel} = \frac{1}{2}(\sigma_{n+1} + \sigma_n)^T \Delta \varepsilon_{n+1}'$$
5.3 Implementation

The Tresca failure criterion with linear strain softening was incorporated into the commercial CFDEM code as a user material. The user material was implemented using the FORTRAN 77 and ANSI C programming languages. A listing of all the user material subroutines is provided in Appendix B and C.

ELFEN\Explicit is an explicit dynamic finite element code with support for combined finite discrete element simulations. The user defined constitutive model is defined by Equation 3-5.

The user material must update the current stress state, the current material state and the associated state variables. For two-dimensional continua a rotated stress configuration is used and the stress update is performed at the mid-point configuration.

Presently, ELFEN/Explicit is only capable of simulating the fracture of three noded constant strain finite elements using a plane strain formulation. The Tresca failure model was thus only implemented for this type of element.

The procedure proposed by Owen and Hinton [27] was adopted to provide a unique straining direction at the corners of the Tresca failure surface. The effective stress and flow vector is returned by the functions invariants and flowvector, respectively depending on the value of $\theta$. If $|\theta| \leq 29^\circ$ then the Tresca effective stress and flow vector is evaluated, otherwise the Von Mises effective stress and flow vector is evaluated.

Crack orientation is determined via the random direction approach outlined in Chapter 4. For the constant strain triangular finite element, ELFEN/Explicit uses the triangle-triangle remeshing algorithm, described in Chapter 3.4, to determine the fractured element geometry.
Implementation of Shear Failure Model

For each integration point, the following algorithm is used:

1. Evaluate elastic predictor: \( \sigma^{e}_{n+1} = \sigma_{n} + D \Delta \varepsilon_{n+1} \).
2. Zero iteration counter: \( i = 0 \).
3. Evaluate failure function: \( F_{f}^{i+1} = \sigma_{eff}^{i+1} - Y_{f}^{i} \).
4. IF \( F_{f}^{i+1} \leq 0 \) GOTO 13 ELSE GOTO 5.
5. Evaluate flow vector and plastic multiplier.
6. Update plastic strain and equivalent plastic strain:
   \[
   \varepsilon_{f}^{i+1} = \varepsilon_{n+1}^{i} + \Delta \lambda_{f}^{i+1} \varepsilon_{n+1}^{i+1}, \quad \varepsilon_{eff}^{i+1} = \varepsilon_{n+1}^{i} + \Delta \lambda_{f}^{i+1}
   \]
7. Update failure strength:
   \( Y_{f(n+1)}^{i+1} = Y_{n}^{i} - H \Delta \lambda_{f}^{i+1} \).
8. IF failure strength \( \leq 0 \) GOTO 9 ELSE GOTO 11.
9. Partition element and re-initialise failure strength, failure strain and equivalent failure strain:
   \( \varphi = \phi \pm \frac{\pi}{4}; \quad Y_{f(n+1)}^{i+1} = Y_{f}^{0}; \quad \varepsilon_{n+1}^{i+1} = 0; \quad \varepsilon_{f}^{i+1} = 0 \)
10. GOTO 13.
11. Increment counter: \( i = i + 1 \).
12. IF \( i > \) maximum iterations GOTO 13 ELSE GOTO 3.
13. Update energy and state variables.
5.4 Verification of Tresca failure model

The verification of the FORTRAN 77 failure model, as presented in Appendix C, was performed using three verification problems, i.e.

1. A direct tension test.
2. A plane strain compression test.
3. A Brazilian test.

5.4.1 Direct Tension Test

The first verification test is a plane strain simulation of a dynamic tension test. Both ends of the bar move in opposite directions with a constant velocity \( c = 0.01 \) m/s. The bar is \( l = 100 \) m long, \( h = 5 \) m high and has the following material properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>( E )</th>
<th>( v )</th>
<th>( \sigma_f )</th>
<th>( \tau_f )</th>
<th>( G_{f,1} )</th>
<th>( G_{f,II} )</th>
<th>( \rho )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>21 MPa</td>
<td>0.17</td>
<td>2 MPa</td>
<td>2.0 MPa</td>
<td>50 N/m</td>
<td>2500 N/m</td>
<td>2500 kg/m³</td>
</tr>
</tbody>
</table>

Table 5-1: Material parameters for uniaxial test specimen

Three different mesh sizes were used in the simulations, i.e. characteristic element lengths of five, two and one meter, respectively. For convenience the three meshes will be referred to as mesh M5, M2 and M1, respectively. Figure 5-1 illustrates the three finite element meshes used. The results of the latter mesh were compared with the Rankine failure criterion simulation of the same problem.
Figure 5-1: Direct Tensile Test: mesh M5, M2 and M1

For the Tresca failure model, the force displacement histories for the three meshes confirm that these results are not severely mesh dependent. Figure 5-2 clearly illustrates this trend.

![Graph showing force displacement for different meshes.]

Figure 5-2: Comparison of Force Displacement Curves for Tresca Failure Model: Direct Tension Test

From Figure 5-3 it can be seen that the location of the onset of fracture is similar for all three meshes. The inclination of the fracture plane, relative to the x-axis is roughly π/4 radians for meshes M1 and M5. However, the inclination of the fracture plane for mesh M2 is steeper than either mesh M1 or M5. This is thought to be caused by the triangle-triangle remeshing scheme. From Figure 5-2 it can be seen that the elements in mesh M2 are more elongated than the elements in meshes...
M1 and M5. As the triangular elements in a mesh become more elongated, the probability of fracture along the inter-element boundary increases. The triangle-triangle remeshing scheme is unable to partition the parent element along the failure plane determined from the Tresca failure model.

\[ T = 0.6 \text{s} \quad \text{h} = 5 \text{ m} \]

\[ T = 0.601 \text{s} \quad \text{h} = 2 \text{ m} \]

\[ T = 0.600 \text{s} \quad \text{h} = 1 \text{ m} \]

**Figure 5-3: Comparison of crack propagation in Direct Tensile Test Specimen using Tresca Failure Model**

A comparison of the force displacement history of the Rankine and Tresca failure models for mesh M1 reveals that the former behaves in a more brittle manner in the post peak load region. This trend is illustrated in Figure 5-4. This type of behaviour is expected since the Mode II fracture energy release rate is an order of magnitude greater than the Mode I fracture energy release rate. Thus, the magnitude of the softening slope will always be lower for the Tresca failure model.

The fracture plane of the Rankine failure model simulation is nearly perpendicular to the first principal stress direction. Cracks also occurred close to the endpoints of the specimen, but these were due to stress waves reflected from the endpoints.
Figure 5-4: Comparison of Force Displacement curves for Rankine and Tresca Failure Models: Direct Tension Test, mesh M1

Figure 5-5: Crack Propagation in Direct Tensile for Rankine Failure Model: Mesh M1
5.4.2 Uniaxial Compression Test

The uniaxial compression test is the standard test used to determine the uniaxial compression strength of a material. Figure 5-6 (from [37]) shows typical post failure crack patterns of concrete uniaxial compression specimens. It is assumed that the specimen is made of concrete, with a square cross section, 100 mm in length, and 200 mm high. Material data was obtained from the experimental work of van Vliet et al [37], on the uniaxial compressive fracture of concrete. The following material properties were used:

<table>
<thead>
<tr>
<th>Property</th>
<th>$E$ (GPa)</th>
<th>$v$</th>
<th>$\sigma_f$ (MPa)</th>
<th>$\tau_f$ (MPa)</th>
<th>$G_{t,1}$ (N/m)</th>
<th>$G_{t,II}$ (N/m)</th>
<th>$\rho$ (kg/m$^3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>21</td>
<td>0.17</td>
<td>40</td>
<td>20, 25</td>
<td>50</td>
<td>2500</td>
<td>2500</td>
</tr>
</tbody>
</table>

Table 5-2: Material parameters for concrete

The specimen is modelled in plane strain and because of its symmetric nature, only a quarter sector needs to be modelled. Since the depth of the specimen is not much larger than its length and width, it should be modelled as a plane stress problem. However, the commercial software used does not support the fracture of plane stress finite elements. The Tresca failure model was calibrated with the uniaxial compressive strength of the concrete.
Figure 5-6: Fracture pattern in uniaxial compression specimen

The finite element mesh of the model consists of 400 three-noded constant strain elements. As shown in Figure 5-7, the free end of the specimen is displaced linearly through a distance of 2 mm in 0.02s. The upper right hand corner node is constrained in the vertical direction, thus modelling a no-slip boundary.

Figure 5-7: Finite element mesh for uniaxial compression test
The computational compressive stress displacement history of the compression test specimen, depicted in Figure 5-8, shows the material softening behaviour in the pre-fracture regime. Figure 5-8 also clearly illustrates the dependence of the post-peak response on the shear strength of the material. A comparison of Figure 4-2 and Figure 5-8 as well as the experimental data presented by van Vliet et al [37], suggests that the post peak response is too ductile in nature.

![Stress-Displacement history for compression test](image)

**Figure 5-8: Stress-Displacement history for compression test**

A shear localisation band is initiated at the lower left-hand corner and progresses along a plane inclined at an angle of 45 degrees relative to the x-axis. The crack initiation occurs in the same lower left hand corner, but the crack propagates in a wide band parallel to the localisation band. Figure 5-9 illustrates the crack propagation through the specimen.
Figure 5-9: Crack propagation in uniaxial compression specimen
5.4.3 The Brazilian Test

The Brazilian Test, also known as the diametral compression test, is used to determine the tensile test of a brittle material. Figure 5-10 illustrates the basic configuration of the test and the typical post failure fracture pattern. In the simulation the disc has a radius of forty millimetres and is made of concrete. The material properties used are the same as those presented in Chapter 5.4.2. Once again the Tresca failure model is calibrated with the uniaxial compression strength of the material.

![Diagram of Brazilian Test]

(a) Basic configuration  (b) Typical Fracture Pattern

Figure 5-10: Brazilian Test

The disc is modelled in plane strain and since the problem is symmetric about the x and y axes, only a quarter of the disc is modelled. The finite element mesh consists of 1719 three-noded constant strain triangular elements, with the elements biased towards the vertical centreline. A controlled vertical displacement of 1 mm is applied, as illustrated in Figure 5-11.
Using the Rankine failure model, cracking is initiated along the vertical centreline of the specimen, and initially spreads in the vertical plane. Figure 5-12 shows the crack propagation at various stages of the simulation. In contrast, the Tresca failure model predicts that cracking is initiated close to the contact area between the specimen and the platen. This crack is initiated at an angle significantly different to that predicted by the Rankine model. Furthermore, due to the more “ductile” softening behaviour of the Tresca model, crack propagation will occur much later during the loading path. Since the softening behaviour of the Rankine model is more severe than the Tresca model, any cracks initiated in the Rankine model will propagate before those initiated in the Tresca model. Thus, for the Brazil specimen the dominant crack propagation behaviour is a tensile one. This implies that the use of the Tresca Failure model is not appropriate for this example. In order to capture the tensile crack propagation, the Rankine failure model needs to be incorporated into the Tresca Failure model. Alternatively, a Mohr-Coulomb model with a tension cut-off may be more appropriate. However, as stated before in Section 4.5, the material data to calibrate this type of model would be difficult to obtain.
Figure 5-12: Crack propagation in Brazilian Test using Rankine Failure Model
6. Particle Model Development

Unlike the FEM, the DEM requires the geometry and location of the discrete element assembly to be defined. The initial discrete element configuration is dependent on the initial porosity, the particle size distribution and the particle shape. For the CFDEM, the additional step of discrete element discretisation has to be considered. The generation of a valid assembly, i.e. one which satisfies the initial porosity, particle size distribution and particle shape constraints, of discrete and finite discrete elements is performed by a particle assembly pre-processor. It is important to note that the generation of a valid particle assembly is vital to the accuracy and quality of the discrete element simulation.

The development of rigid circular, elliptical and spherical particle pre-processors is well established [1]. In order to investigate the powder compaction process using the CFDEM efficiently, the creation of a valid particle assembly has to be automated. Furthermore, the microscopic particle behaviour has to be quantified using suitable macroscopic variables.

This chapter contains a discussion of the algorithms used in the development of a general two-dimensional polygonal particle generator. This is followed by an outline of the development of the particle post-processor. Finally, the computer implementation of the particle pre-processor is discussed.
6.1 Nomenclature

In the development that follows, a point is assumed to be a point in a two dimensional Cartesian space and is defined by a x and y co-ordinate. A node is assumed to be an indexed point, i.e. the point has a unique index associated with it, in a two dimensional Cartesian space. The term segment refers to a straight line in a two dimensional Cartesian space. A special case of a segment is a boundary segment that is assumed to be an exterior line segment of a polygon. The term polygon is applied to a two dimensional polygon unless otherwise stated. An n-sided polygon is a two dimensional polygon with n vertices, where \( \{n: n \in \mathbb{Z} \mid n \geq 3\}\). Finally, a circumcircle or more correctly a circumscribed circle, is the smallest circle that circumscribes all the vertices of an n-sided polygon. The circumcircle’s origin coincides with the centroid of the polygon it circumscribes. All these concepts are illustrated in Figure 6-1.

![Image of geometric primitives](image)

**Figure 6-1: Geometric primitives**

Whereas the FEM requires that the solution domain be discretised, the DEM only requires the generation of the discrete elements. The CFDEM requires the additional step of discretising the discrete element. A discrete element simulation of powder compaction requires a system of discrete elements to represent the discontinuum. The discrete element generation process can be viewed as the following problem:
Given any continuous two-dimensional region, fill this region with non-overlapping particles and then discretise the individual particles to create a system of combined finite discrete elements.

It is important that the geometry of the discrete element approximates that of the real powder particle as accurately as possible. Furthermore, the generated system of particles has to have the same size distribution as the real powder system. These two constraints are important since the particle shape and size distribution play a vital role in the cold compaction of a brittle powder. The remainder of this chapter deals with the idealisation of the brittle particle and the discrete element generation procedure.
6.2 Conceptual Model of a Brittle Particle

The brittle particle conceptual model utilised in the current work is based on the following assumptions:

1. The particle geometry is approximated by an n-sided polygon.
2. The particle is discretised into constant strain triangular elements.
3. A linear elastic isotropic constitutive model governs pre-failure deformation.
4. An objective linear softening plasticity model governs post-failure material behaviour.
5. The particle material and strength properties are strain rate and temperature independent.
6. Material and strength properties are independent of any scale effect.

The polygonal particle description was chosen as a result of the visual inspection of various brittle powders. Furthermore, the polygonal approximation can be discretised into finite elements exactly. The constant strain triangles were used in the finite element mesh since these were the only elements in the commercial code that supported fracturing.

In order to simplify the analysis of the powder compaction, the simplest possible material description, i.e. isotropy, strain rate and temperature independence, was chosen. Brittle materials do exhibit a marked scale effect since as the specimen size decreases, the probability of defects included in the specimen decreases. It has been assumed that the reduction in particle size is such that the scale effect is negligible.

Particle interaction is idealised using the Coulomb friction model since this model has been used successfully in rock joint interaction. Furthermore, the model is simple and computationally inexpensive.
6.3 Model Generation Pipeline

To generate the required system of particles for the CFDEM simulations a pre-processor was developed. The pre-processor had to satisfy the following criteria:

1. Generate polygonal particles with pseudo-random geometry.
2. Fill any planar area with non-overlapping polygons.
3. Discretise the resulting system of polygons.

The model generation can be viewed as a three-stage process, i.e. particle generation, particle placement and particle discretisation.
6.4 Particle Generation

The primary aim of this stage is to produce a randomly shaped n-sided polygon which belongs to some given particle size distribution. The particle size distribution is presented in the form of a cumulative weight plot as illustrated in Figure 6-2. By randomly choosing a number along the percentage finer axis, the relevant particle radius can be calculated. Any generated random polygon must have a circumscribed circle with this radius.

![Figure 6-2: Generic Cumulative Weight Plot](image)

Two algorithms were developed to generate the random polygon geometry and they are discussed in the following sections.
6.4.1 Circumferential Polygon Generation

In the first algorithm the polygon was generated by randomly placing the polygon vertices on the circumference of the circumscribed circle. Firstly, the number of polygon vertices is randomly selected between some user-defined limits. Secondly, the circumscribed circle is divided into \( n \) equal sectors, where \( n \) is the number of vertices. Thirdly, the \( n \) vertices are created, in a counter clockwise direction, by randomly selecting an angle in the current sector. Finally, the co-ordinates of the polygon vertex are calculated. The algorithm is outlined in the following pseudo code for a polygon with \( n \) vertices and which has a circumscribed circle centred at \((x_{\text{center}}, y_{\text{center}})\). It is assumed that the polygon vertex co-ordinates are stored in two one-dimensional arrays, i.e. \( x \) and \( y \).

\[
\begin{align*}
\text{n} & = \text{random(upper\_bound, lower\_bound)} \\
\Delta\theta & = 2\pi/\text{number\_of\_sides} \\
\text{for i = 1 to n} \\
& \{ \\
& \quad \theta = \text{random(0, } \Delta\theta) + (i+1)\Delta\theta \\
& \quad x(i) = \text{radius}\cdot\cos(\theta) + x_{\text{center}} \\
& \quad y(i) = \text{radius}\cdot\sin(\theta) + y_{\text{center}} \\
& \}
\end{align*}
\]

Figure 6-3 illustrates the generation of a polygon with four vertices. In the figure \( \Delta\theta_i (i = 1,2,3,4) \) is the incremental angle relative to the current sector.
Figure 6-3: Schematic representation of Random Polygon Generation Algorithm 1
6.4.2 Perturbation Polygon Generation

The second algorithm relies on the use of a master polygon list. Polygons in the master polygon list are representative of the general shape of the particles in the powder system. The polygon is generated by firstly randomly selecting a member of the master polygon list. The selected polygon is then subjected to a process of randomisation. Randomisation involves shifting each of the vertices of the polygon within some perturbation radius. Consider a local polar co-ordinate system centred on a vertex of the polygon. The vertex is now displaced by some randomly generated radial and angular component about the original local co-ordinate system.

![Diagram showing perturbation polygon generation](image)

Figure 6-4: Schematic representation of polygon vertex randomisation

The randomisation process is applied sequentially to each vertex in a counter clockwise fashion. Figure 6-4 illustrates the randomisation of a single vertex with a perturbation radius of $r_p$. Finally, the polygon is randomly rotated about its centroid and scaled to fit the given circumcircle.
6.4.3 Comparison of polygon generation algorithms

A comparison of the two methods revealed that Algorithm II produced polygons with a higher degree of randomness. Algorithm I polygons tended to have one boundary segment whose length was much larger than the remaining boundary segments of the polygon. These polygons tended to look like semicircles and are evident in Figure 6-5.

A further advantage of Algorithm II, compared to Algorithm I, is that the user has a higher degree of control on the average shape of the generated polygons. This is achieved by controlling the geometry of the polygons in the master polygon list as well as the magnitude of the perturbation radius. Due to these advantages it was decided to use Algorithm II in the pre-processor.
Figure 6-5: Results of Algorithm I Polygon Generation

Figure 6-6: Results of Algorithm II Polygon Generation
6.5 Particle Placement

Once the polygons have been generated they have to be randomly placed within the simulation volume subject to the constraint that they do not overlap one another. It is assumed that the simulation volume (area) is approximated by an n-sided polygon referred to as the boundary polygon. The polygon overlap test could be performed by testing if the vertex of any existing polygon resides inside the generated polygon. This test becomes computationally expensive as the number of polygons in the system increases. More efficient strategies exist [25], but due to time constraints no attempt was made to investigate these algorithms.

The primary aim of the particle placement stage is to insert non-overlapping polygons into the simulation volume, or in the two-dimensional case the simulation area. It is assumed that the simulation area is approximated by an n-sided polygon referred to as the boundary polygon. The boundary polygon has been scaled by some factor of less than one to prevent particle-boundary overlap. Particle placement can be broken down into a three-stage process, i.e.

1. Random Point Generation.
2. Particle Generation.
3. Contact Detection.

The first stage is accomplished by using a pseudo random number generator to generate a random point inside the bounding box of the boundary polygon. The crossing test [17] is then used to determine if the random point is inside the boundary polygon. If the random point fails the crossing test, a new random point is generated and the test is performed once again. The polygon generation stage is discussed in Chapter 6.4. Before the new polygon is inserted inside the boundary polygon, a contact detection test is performed to ensure that the new polygon does not overlap any of the existing polygons. If the new polygon does not pass the contact detection test it is rejected and a new random point is generated. The entire process is repeated until the desired number of points is generated.
6.5.1 Circle Contact Detection Algorithm

Contact detection between simple geometrical objects, e.g. rectangles or circles, is computationally less expensive than using polygonal contact detection. By embedding each polygon within a simple geometric object, the cost of contact detection is reduced. The simplest bounding object for a polygon is a circle. Thus instead of performing the contact detection tests between the polygons themselves, the tests are performed between the polygon’s circumcircles.

The contact detection algorithm for two circles is simple and invariant of the orientation of the circles. Two circles are said to be in contact if the sum of their radii is greater than the distance between their centres. Consider two circles centred at $\mathbf{A}$ and $\mathbf{B}$, with radii $r_\mathbf{A}$ and $r_\mathbf{B}$, respectively. The two circles will only be in contact if the following inequality is not satisfied:

$$|\mathbf{AB}| - (r_\mathbf{A} + r_\mathbf{B}) \geq 0$$  \hspace{1cm} \text{Equation 6-1}

Thus, whenever a new circumcircle is inserted inside the boundary polygon, Equation 6-1 has to be satisfied for all existing circumcircles. As the number of circumcircles inside the boundary polygon increases, so too does the computational expense. To improve the efficiency of the placement algorithm, the contact detection test should only be performed between circumcircles that are spatially close to each other.

6.5.2 Neighbour List Spatial Sorting Algorithm

A neighbour list sorting algorithm is used to determine which circumcircles are spatially close to each other. The neighbour list is created by determining which circumcircles lie within a bounding object centred at the point of insertion of the new circumcircle. A rectangular bounding object, i.e. a bounding box, is used in the current implementation of the neighbour list algorithm. If a circumcircle’s centroid is interior to the bounding box, the circumcircle is flagged as a neighbour. Consider three circles centred at points $\mathbf{P}^0$, $\mathbf{P}^1$, and $\mathbf{P}^2$, respectively. Assume that the
bounding box's centroid is at $P^0$ and has corner co-ordinates $(x_{\text{min}}, y_{\text{min}})$ and $(x_{\text{max}}, y_{\text{max}})$ as illustrated in Figure 6-7.

![Diagram showing circles with bounding box](image)

Figure 6-7: Neighbour List Sorting Algorithm

To determine if the two remaining circles are neighbours, the following inequalities have to be satisfied:

\[
\begin{align*}
P_x^i & \geq x_{\text{min}} \\
P_y^i & \geq y_{\text{min}} \\
P_x^i & \leq x_{\text{max}} \\
P_y^i & \leq y_{\text{max}}
\end{align*}
\]

Equation 6-2

for $i = 1, 2$

If there are $n$ circles inside the boundary polygon, then the index $i$ runs from 1 to $n$ and the test has to be performed whenever a new circle is inserted.

For a static system of circles, the size of the bounding box is calculated in the following manner: The largest circle has a radius of $r_{\text{max}}$, which is determined from the particle size distribution. Consider the worst case scenario, i.e. all the circles in
the system have a radius of $r_{\text{max}}$. Assume that point $P^0$ is the centre of both the bounding box and a circle of radius $r_{\text{max}}$. Now consider a set of circles, with radii $r_{\text{max}}$, that are tangential to the $0^{th}$ circle. The centres of this set of circles, $P^i$, form a locus of points that define the largest bounding box. The locus of points form a circle with radius $4 \times r_{\text{max}}$, thus implying that the largest bounding box is a square with sides of length $4r_{\text{max}}$.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure6-8.png}
\caption{Determination of optimal bounding box}
\end{figure}

Once the neighbour list has been created, the normal circle contact detection test is used. The computational advantages of the neighbour list sorting algorithm are only fully realised when the contact detection test has to be performed for a system
containing a large number of circles. Ideally, the size of the bounding box should be a function of the current particle size distribution. As the particles are compacted, the particles fracture thus resulting in a decrease in the upper particle size. However, to simplify the development of the pre-processor a static bounding box was used.
6.6 Particle Discretisation

The final stage of the particle model generation is the discretisation of each generated polygon into a finite element mesh. A shareware mesh generator, Triangle, [31] based on the principle of Delauney triangulation [26] is used to discretise the polygon into triangular finite elements. The mesh generator requires the following information:

1. The boundary representation (b-rep) of the polygon.
2. The minimum angle in any triangle in the discretisation.
3. The maximum area of any triangle in the discretisation.

The b-rep consists of a list of all the boundary nodes of the polygon. To ensure that the angles of the triangular elements are not small the minimum angle in any triangle is limited to some user-specified value. A minimum angle of 20 degrees is normally sufficient to ensure that no elongated triangular elements are generated. Mesh density is controlled by specifying the maximum area of any triangular element in the mesh. The minimum angle constraint has a higher priority than the maximum area constraint.

The polygon b-rep is written to a file, which is in turn read by the mesh generator. To ensure no elongated triangles are generated, an angle of 30 degrees is used for the minimum angle constraint while the area constraint is dependent on the total area of the polygon. Triangle is called from the pre-processor whenever a valid polygon has been generated.
6.7 Post Processing Algorithms

The CFDEM simulations involve large numbers of particles and the results cannot be easily interpreted using standard finite element post-processors. For this study only limited post processing was required. However, due to the nature of Object Orientated Programming (OOP), the post-processor can be extended to perform more complex data processing. The following section describes the algorithms used to extract the volume porosity and particle size distribution from the simulation results.

6.7.1 Particle Boundary Extraction

The first post-processing stage consists of determining the current b-rep of all the discrete elements in the simulation volume. ELFEN/Explicit stores the discrete element nodal information in two files. The first file defines the nodal geometry and topology as well as the contact element topology. The second file contains all the nodal simulation results, e.g. nodal displacement and nodal force, associated with the nodes defined in the first file.

The particle boundary is extracted from the contact element topology in the following manner. Firstly, the first node of the first contact element is stored. The remaining nodes, and the associated contact element topology, is read into memory. When the next node in the contact element topology list is identical to the first node read in, the particle boundary has been defined.
Figure 6-9: Particle Boundary Representation

The discrete element b-rep is stored in a polygon data structure, which is described in Chapter 6.8.

6.7.2 Calculation of Volume Porosity

The volume porosity of a porous medium is defined as:

$$\pi_v = \frac{V_{\text{total}} - V_{\text{particles}}}{V_{\text{total}}}$$

Equation 6-3

where $V_{\text{total}}$ is the total system volume and $V_{\text{particle}}$ is the total volume of the particles. Since the systems dealt with in this study are two dimensional, the volume porosity is defined as:

$$\pi_v = \frac{A_{\text{total}} - A_{\text{particles}}}{A_{\text{total}}}$$

Equation 6-4

where $A_{\text{total}}$ is the total system area and $A_{\text{particle}}$ is the total area of the particles. The total particle area is determined by summing the area of all the particles in the system.
6.7.3 Extraction of Particle Size Distribution

The creation of the particle size distribution of a granular media involves sorting all the particles according to some equivalent size. If the particles were circular in shape, the equivalent particle size would be the radius of the circle. For arbitrarily shaped particles an equivalent particle size needs to be determined. Three approaches were considered in determining the equivalent particle size:

1. mass sizing
2. mesh sizing
3. circumcircle sizing

Mass sizing involves determining the radius of the circle with the same area as the particle. In the mesh sizing approach, the size of the rectilinear opening through which the particle will just pass is assumed to be the equivalent size. Finally, the circumcircle approach is based on the assumption that the equivalent size is equal to the radius of the circumcircle associated with the particle.

Once the particles have been sorted, the particle size distribution can be determined. This distribution can be either discrete or piece-wise linear. The discrete distribution is obtained by sorting the equivalent particle sizes into a series of bins and counting the number of particles in each bin. A piece-wise distribution is obtained by sorting the equivalent particle sizes in descending order and normalising the list by the maximum equivalent particle size.

In the current post-processor, the equivalent particle size is obtained via the circumcircle approach. This approach is used because the polygonal particle representation makes this the easiest to implement. The equivalent particle sizes are used to obtain a piece-wise linear particle size distribution.
6.8 **Computer Implementation**

In order to develop flexible and extensible pre- and post-processors for the CFDEM simulations, all software was developed using the OOP technique [33]. OOP relies on the creation of an abstract data type (ADT) or class, which defines a data structure, with built-in functions that operate on the data. Thus both the data and the functions that operate on the data are grouped into one unit.

The classes developed for the pre- and post-processor divided into two groups, i.e. geometric and utility classes. The geometric classes are used to represent the geometry and topology of the individual particles as well as their combined finite discrete element representation. All other classes not related to geometric abstractions are collectively known as utility classes. A detailed description of the various classes and their respective member functions can be found in Appendix D.

6.8.1 **Geometric Abstract Data Types**

The primary purpose of developing geometric classes is to provide a convenient and extensible means of storing and manipulating the data defining a two dimensional polygon and its associated finite element mesh. Functions for data manipulation, known as member functions, were developed specifically for use in the random polygon generation algorithm. However, due to the nature of OOP, the classes can easily be extended for use in more general applications.

To provide the functionality needed to create a random polygon generator the following classes were developed:

1. Point2d
2. Node2d
3. Circle
4. Trifem
5. Segment2d
6. Polygon2d
7. Particle2d
These classes are described below.

To define a point in two-dimensional space, the Point2d class was developed. The Point2d class can also used as a two dimensional vector class. C++ permits operators to be redefined for various classes, and this procedure is known as operator overloading. For the Point2d class the basic mathematical operators, i.e. +, -, * and /, are overloaded to allow vector addition, vector subtraction and scalar vector multiplication and division. Thus the addition of two vectors \( \mathbf{a} \) and \( \mathbf{b} \), using the Point2d class would be:

\[
\text{Point2d } a, b, c \\
c = a + b
\]

instead of:

\[
\begin{align*}
\text{double } &a[2], b[2], c[2] \\
\text{int } &i \\
\text{for(}i=0,i<2,i++\text{)} \\
c[i] &= a[i]+b[i]
\end{align*}
\]

This overloading mechanism allows for the development of compact and easily readable code.

The second ADT implemented is that of a node, which as defined previously is an indexed two-dimensional point. A Node2d object consists of a unique integer index and a Point2d object to store its location. This data is only assessable via the class’s member functions. While this increases the data retrieval time, it ensures that the data is not inadvertently corrupted.

The contact detection algorithm used in the polygon generation algorithm relies on the concept of a circumcircle. For this purpose a circle ADT was developed and implemented in the Circle class. The Circle class contains a unique integer index, the radius of the circle and a Point2d object to store the co-ordinates of the circle
centre. Once again, the data can only be accessed via the use of the Circle member functions.

The only finite element used in this study is the three-noded constant strain triangular element and consequently only a three-noded finite element ADT was implemented. This finite element is defined by the location and connectivity of the three nodes defining the corners of the triangle. If only one element was to be used it would be feasible to store the location of each node in the element. Finite element meshes generally consist of a large number of elements, thus making this approach inefficient.

A more efficient approach is to use pointers to point to the appropriate nodes. The triangular element data structure thus consists of three Node2d pointers, thus allowing all the nodes in the finite element mesh to be stored in a single Node2d array. Figure 6-10 illustrates a finite element mesh consisting of four triangular elements.

![Finite Element Mesh](image)

**Figure 6-10: Finite Element Mesh**

The relevant element node pointer relationship is illustrated in Figure 6-11. This ADT is implemented in the Trifem class. The class contains a unique integer index and three pointers to the Node2d objects defining the vertices of the element.
The line segment abstraction is similar to that of the triangular element. Two pointers are used to point to the location in memory, of the nodal co-ordinates of the start and end-point of the line segment.

The Polygon2d class is the implementation of the two dimensional polygon ADT. The geometry and topology of the polygon is stored in an array of Node2d objects. Contained within each Polygon2d object is a unique integer index, the number of vertices, the polygon centroid, the polygon circumcircle radius, the minimum boundary segment length, a pointer to the furthest node from the polygon centroid and the polygon area. By storing this information, the speed of manipulating polygon objects is increased.

The particle abstract data type is implemented in the Particle2d class. This class defines the geometry and topology of the combined finite discrete element mesh of the particle. Data is stored in the following manner: Firstly all the nodes in the particle are stored in an array of Node2d objects. The finite element mesh and boundary topology of the particle is stored in an array of Trifem and Segment2d objects respectively. Finally, the particle index, the number of elements, the number of nodes and the number of boundary segments are stored in integer variables.
6.8.2 Utility Abstract Data Types

The utility classes developed are:
1. CirList
2. DiscreteFunction
3. RandomNumber

The CirList class is the implementation of a single linked list and is used to create lists of Circle objects during particle generation. Storing data in a list, as opposed to an array, decreases the computational expense of dynamically adding and removing data from the list. The CirList class consists of an integer index, a pointer to a Circle object and a pointer to the next Circle object in the list. Member functions are implemented to create, delete and retrieve the Circle objects in the list.

To approximate a continuous function of one variable, the DiscreteFunction class was developed. The continuous function is approximated in a piece-wise linear fashion. A continuous function is thus approximated by a finite set of points, with a linear variation between the points, as is illustrated in Figure 6-12. The DiscreteFunction class consists of an array of Point2d objects, as well as the range and domain of the function. Member functions are implemented to access, store and manipulate the data.

![Figure 6-12: Continuous function discretisation](image)
The particle pre-processor relies heavily on pseudo-random numbers to generate and fill the simulation volume with randomly shaped particles. In order to provide a convenient mechanism for generating pseudo-random numbers, the RandomNumber class was implemented. The RandomNumber class provides a framework for isolating the pseudo-random number generation algorithm from the rest of the code. Member functions are implemented to initialise and return pseudo-random numbers.
7. Simulation of Cold Compaction of CBN Powder

The objective of the development of the particle pre-processor and Tresca failure model is to assess the feasibility of using the CFDEM in the simulation of cold compaction of brittle powders and to investigate the effect of the microscopic constitutive model on the macroscopic material behaviour. In this chapter, the numerical results of a cold compaction simulation are presented and analysed. Firstly, the combined finite discrete element model is discussed. This is followed by a discussion of the various material parameters used to calibrate the Rankine and Tresca failure models. Finally, the results of both constitutive models are presented and compared.

7.1 Combined Finite Discrete Element Model

The CFDE simulation consists of the uniaxial compression of a 103-polygon assembly in a rigid container, with initial internal width and height of 0.6 mm. The initial area porosity of the particle assembly is 0.697. It is assumed that a plane strain stress state occurs within each particle. Since ELFEN/Explicit does not allow translating rigid bodies and discrete elements in the same analysis, the container is meshed using three-noded constant strain triangular elements. The particles are assumed to have a linear particle size distribution, with no particle being smaller than 63 µm and larger than 80 µm in diameter. A total of 1405 constant strain triangular finite elements are used in the particle CFDE mesh. Due to limited computational resources, a coarse mesh was used to decrease the simulation time. The particle assembly and combined finite discrete element mesh is illustrated in Figure 7-1 and Figure 7-2, respectively.
Figure 7-1: 103 Particle assembly

Figure 7-2: Combined Finite Discrete Element mesh for particle assembly
The left and right vertical container and bottom horizontal walls are constrained in both directions. The upper horizontal wall is constrained from moving in the horizontal direction. Displacement controlled vertical loading is applied to the upper edge of the upper horizontal wall. The upper wall moves downwards at a constant velocity of 100 mm/s for 0.001 s. Vertical reaction forces are monitored at each node along the upper edge of the horizontal wall, and are summed to obtain the force displacement history. In addition to the vertical displacement, the particles are subjected to a gravitational acceleration of 9.8 ms\(^{-2}\) in the negative \(y\) direction.

The macroscopic behaviour of the particle assembly is monitored via the force-displacement history. From the force-displacement history, the equivalent stress \((\sigma_{eq})\) and strain \((\varepsilon_{eq})\) histories are calculated using the following equations:

\[
\sigma_{eq} = \frac{F}{A}
\]

_\textit{Equation 7-1}_

where:

\(F\) = total reaction force acting on platen

\(A\) = total area of platen

Similarly, the equivalent strain for the particle assembly is defined as:

\[
\varepsilon_{eq} = \frac{H - H_0}{H_0}
\]

_\textit{Equation 7-2}_

where:

\(H\) = distance between platens

\(H_0\) = initial distance between platens


7.2 Material Parameters

The CBN particles are approximated as an isotropic linear elastic material, before the peak load has been reached, with Young’s modulus of 680GPa and Poisson’s ratio of 0.22. To increase the size of the time increment, the density of the CBN particles has been increased by a factor of 51. Mass scaling can be used since the cold compaction process is essentially quasi-static. The container is also treated as an isotropic linear elastic material, with Young’s modulus of 650 GPa and Poisson’s ratio of 0.3. A lower stiffness has been assigned to the container to ensure that the elements in the particles, rather than the elements in the container determine the critical time step. The lower stiffness of the container is offset by the fact that all the container nodes are either fully constrained, or are subjected to an applied displacement.

The post failure material behaviour of all the particles in a given particle assembly is either governed by a Rankine or a Tresca softening plasticity failure model. The uniaxial tensile strength of the CBN particles is assumed to be 1000 MPa while the fracture energy release rate for Mode I failure is approximated using Equation 4-2. The fracture toughness is taken as 5 MNm$^{-3/2}$ and the resulting value for the Mode I fracture energy release rate is 35 N/m. All of the above material properties were obtained from Brookes [9]. The shear strength of the CBN particles is assumed to be half that of the uniaxial tensile strength.

Experimental values for the Mode II fracture energy release rate of CBN could not be obtained. The value was estimated by using data presented by Willam et al [38]. Willam et al [38] state that, for concrete, the Mode I and Mode II fracture energy release rates are 44 N/m and 2452 N/m, respectively. The ratio of the Mode I and II fracture energy release rates was used to estimate a Mode II fracture energy release rate of 1950 N/m. To decrease the amount of failure strain before fracture, this value was decreased to 1750 N/m.
The friction coefficient for inter-particle and particle-wall contact is assumed to be identical. Based on available information, the value of the friction coefficient ($\mu$) was assumed to be 0.1. To evaluate the response of the particle assembly to a change in the particle interaction, a parametric study was undertaken. The purpose of this study was to investigate the sensitivity of the macroscopic response of the particle assembly on the friction coefficient.
7.3 Results of Constitutive Model Parametric Study

This parametric study was undertaken to investigate the effect of the particle constitutive model on the macroscopic response of the particle assembly. Three constitutive models were investigated, i.e. linear elastic, linear elastic with the Rankine failure criterion and linear elastic with the Tresca failure criterion. For simplicity, the particle assembly with the linear elastic particles will be referred to as the linear elastic particle assembly. Similarly, the remaining two particle assemblies will be referred to as the Rankine and Tresca particle assemblies.

The initial geometric configuration of the particle assembly was identical for all constitutive models investigated. The inter-particle friction coefficient was assumed to be 0.1 in all cases. All other assumptions used have been outlined in the preceding section.

7.3.1 Mechanical Response of Linear Elastic Particle Assembly

The macroscopic stress-strain response of the particle assembly is shown in Figure 7-3, and from this it can be seen that significant stresses are only developed after 70% of the loading cycle is complete. During this stage of the loading cycle, the particle assembly is still porous and hence the particles are free to move. Since the particles are not constrained by their neighbouring particles, no stresses are transmitted across the particle boundaries, and hence these particles are essentially stress-free. The particle assembly is in a state of rigid body motion. During the last 30% of the loading cycle, the particles are sufficiently close together to constrain each other, and transmit stresses. This is reflected on a macroscopic level by the increase of equivalent stress as a function of equivalent strain. The minor fluctuations in the equivalent stress-strain relation can be attributed to particle rearrangement.

Rigid body motion during the initial stage of the loading cycle was observed in all of the particle assemblies that were investigated. Since significant stresses only developed in the particle assembly during the last 30% of the loading cycle, the current approach is computationally expensive. The reason for this inefficiency is
the high initial porosity of the particle assembly as generated by the current pre-processor. For the initial stage of the loading cycle, rigid polygonal discrete elements would have been sufficient to capture the behaviour of the particle assembly and would result in a significant improvement in simulation time. Another alternative would be to develop a pre-processor paradigm that reduces the initial porosity of the particle assembly.

![Graph of Equivalent Stress vs. Equivalent Strain](image)

**Figure 7-3: Macroscopic Stress-Strain Response of Elastic Assembly**

During the loading cycle, the particles are displaced towards each other due to the downward motion of the platen as well as the gravitational loading. Once the particles are in contact with each other, certain groups of particles formed coherent structures, which are capable of distributing load across the entire structure. The load is distributed across the structure via the interaction of the boundary segments of the particles that are in contact with each other. The boundary segments in turn distribute loads via the frictional contact between them. For convenience, these coherent structures within the particle assembly will be referred to as “bridges”.

When a vertical “bridge” forms in the particle assembly, the vertical motion of the individual particles in the “bridge” becomes constrained. On the macroscopic level,
the stiffness of the particle assembly increases as a result of the formation of these “bridges”. The geometric configuration and the frictional forces between the individual particles determine the integrity of the “bridge”. If the frictional forces between the particles in the “bridge” are insufficient to constrain any horizontal motion, a particle could be ejected from the “bridge”. A further requirement for the collapse of the “bridge” would be a free region of space for the ejected particle to move into. Once the particle is ejected from the “bridge”, it will no longer be a continuous structure and hence it cannot sustain any load.

The stresses in the particle assembly are generated due to the frictional contact between the individual particles. When a “bridge” collapses, the particles in that structure are subjected to a reduced loading state, resulting in the recovery of any elastic strain and a reduced stress state. This behaviour manifests itself on a macroscopic level as rapid fluctuations in the force-displacement, and hence the stress-strain response of the particle assembly.

If the frictional forces generated between the particles are insufficient to constrain a particle, and the geometric configuration of the surrounding particles contains a large enough void, the particle will be ejected. Note that this situation can also arise if the particle is allowed to fracture. Once the particle has been ejected, the “bridge” is no longer a continuous structure and will collapse. This collapse is associated with a rapid decrease in the stiffness of the particle assembly and the phenomenon manifests itself on the macroscopic scale as fluctuations in the force displacement history, and hence the stress-strain history, of the particle assembly.
7.3.2 Mechanical Response of Rankine Particle Assembly

The incorporation of the Rankine failure criterion into the particle constitutive model has a significant effect on the macroscopic response of the particle assembly, as can be seen from Figure 7-4. On a macroscopic level, the equivalent stress does not continuously increase as a function of the equivalent strain. From Figure 7-4 it can be seen that the equivalent stress decreases suddenly to zero at an equivalent strain of approximately 0.00144, and only starts to increase again at a later stage.

![Figure 7-4: Macroscopic Stress-Strain Response of Rankine Assembly](image)

This behaviour can be explained as follows: Firstly, the stress in the particles increases linearly until the yield strength of the particle material has been reached. Inelastic deformation occurs in the particles resulting in the decay of the particle strength until the particles eventually fail. If these particles form part of a “bridge”, then the “bridge” will collapse when any of the particles fail, and hence on a macroscopic level a sudden decrease of the equivalent stress will be observed. Associated with the collapse of a “bridge” is a sudden increase in the kinetic energy of the assembly.
Energy is required to propagate a crack through a particle, and from Figure 7-5 it can be seen that the amount of energy required for crack propagation (inelastic energy) exceeds the energy required for elastic deformation. Since energy can now also be dissipated via crack propagation, the equivalent stress at the end of the simulation has decreased from approximately 4300 MPa (linear elastic assembly) to 1500 MPa for the Rankine particle assembly.

![Energy Response of Rankine Particle Assembly](image)

**Figure 7-5: Energy Response of Rankine Particle Assembly (μ=0.1)**

From Figure 7-7 it can be seen that the initial and final particle size distributions are significantly different. The incorporation of the Rankine failure criterion has resulted in the formation of smaller particles via failure of the original larger particles. Unfortunately, no experimental data was available to test the validity of the numerical results.
7.3.3 Mechanical Response of Tresca Particle Assembly

From Figure 7-6 it is clear that the macroscopic response of the Tresca particle assembly is substantially different from either the linear elastic or Rankine particle systems. The first major difference is the relatively low peak equivalent stress at the end of the simulation, i.e. an equivalent stress of approximately 540 MPa. Secondly, the relationship between equivalent stress and strain is much smoother than the equivalent relationships for the two previous particle assemblies. This suggests that particle rearrangement, either via slipping or failure, is not as prevalent as it was in the previous two particle assemblies.

![Graph showing stress-strain relationship](image)

**Figure 7-6: Macroscopic Stress-Strain Response of Tresca Assembly**

The yield strength in the Tresca failure model is the shear strength of the material, as opposed to the tensile strength in the Rankine failure model. From Table 7-1 it can be seen that the shear strength of the material is less than the tensile strength. The post-failure stress in a particle will always be lower than the yield strength of the material, and hence the Tresca particles will always experience a lower stress state than the Rankine particles.
As outlined in Sections 5.4.1 and 5.4.2 the softening slope of a particle undergoing fracture will always be lower for a Tresca failure model, relative to a Rankine failure model. This implies that more inelastic strain is required in the Tresca failure model (relative to the Rankine failure model) to cause total material failure. During the compaction of the Tresca assembly, insufficient inelastic strain is generated to result in the massive failure of the individual particles. However, the individual particles do undergo significantly more inelastic strain than their counterparts in the Rankine assembly. The macroscopic response of the Tresca assembly is thus more ductile in nature, relative to the Rankine assembly.

![Figure 7-7: Cumulative Size Distribution](image)

A comparison of the initial and final particle size distributions in Figure 7-7 reveals that no significant change has occurred. The major difference occurs at the lower end of the scale, where it is evident that a small amount of particle failure has occurred. This indicates that large scale failure of the particles does not occur in the Tresca assembly as it does in the Rankine assembly. This is consistent with the observation that the response of the particle assembly is too "ductile" when using the Tresca failure model.
7.3.4 Comparison of Particle Assembly response

The results presented above clearly indicate that the macroscopic response of the particle assembly is highly sensitive to the choice of constitutive model for the individual particles. In the case of the linear elastic material model, the equivalent stress-strain response is initially linear, with non-linear behaviour becoming more evident at higher strain levels. The deviations from the linear response are a result of particle slipping events. However, when a failure model is incorporated, the response becomes more complicated.

![Graph showing stress-strain response](image)

**Figure 7-8: Sensitivity of Particle Assembly response to Constitutive Model**

The incorporation of the Rankine failure model restricts the magnitude of the equivalent stress of the particle assembly. Initially, the equivalent stress-strain response is essentially linear. However, when the strength of the assembly has been reached, a sudden drop of the equivalent stress is observed. This behaviour is typical of brittle materials. The response of the Tresca assembly is significantly different from that of the Rankine assembly.

In the Tresca assembly the transition from linear to non-linear behaviour is smooth as opposed to the Rankine assembly. Furthermore, the peak equivalent stress is
approximately a third that of the Rankine assembly. This is due to increased amount of inelastic deformation that occurs in the individual particles. The response of the particle assembly is more like that of a ductile material than that of a brittle material. From this it appears that the nature of particle failure model, i.e. Mode I or II, is reflected in the macroscopic response of the particle assembly. Based on this observation, it is apparent that the Tresca failure model developed is unsuitable for modelling brittle particle assemblies. As suggested in Section 5.4.3 it may be more appropriate to incorporate a tension cut-off surface into the Tresca failure model.

The cumulative particle size distribution at the end of the compaction process (Figure 7-7) indicates that significantly more particle failure occurred in the Rankine assembly. As observed before, relative to the Rankine failure model, more energy is required to fully open a crack when using the Tresca failure model. From Figure 7-7 it is evident that the more than 90% of the particles did not undergo any failure during the compaction process. This further supports the observation that a tension cut-off surface should be incorporated into the Tresca failure model.
7.4 Effect of Inter-particle Contact

In this exercise the effect of the inter-particle contact on the macroscopic response of the particle assembly was investigated. The friction coefficient $\mu$ was assumed to be 0.1, 0.2 or 0.3. The post failure behaviour of the particles was described using the Rankine failure model, with all material parameters being defined in Chapter 7.2.

The Rankine failure criterion was chosen for this parametric study for the following reasons. Firstly, the linear elastic constitutive model is unsuitable because it cannot account for the particle evolution during the compaction process. Since no particle failure occurs, the resultant stress-strain relationship is unrealistic. Secondly, the Tresca failure model was not considered because the response of the individual particles, and hence the entire assembly it is too ductile.

Two phenomena are evident when analysing the equivalent stress-strain relationships of the Rankine particle assemblies, as shown in Figure 7-9. In the context of this problem, the peak stress is assumed to be the maximum equivalent stress reached in the particle assembly before the onset of large scale particle failure. The stress relaxation is assumed to be the variation in the equivalent stress from the peak stress to minimum equivalent stress reached after large scale particle failure occurs. Firstly, as the friction coefficient increases, the magnitude of the stress relaxation decreases. When $\mu = 0.1$, the maximum drop in the stress magnitude was approximately 1700 MPa, while for the remaining two cases the drop was 600 and 300 MPa respectively. However, as the friction coefficient increases, the fluctuation in the stress-strain relationship increases. This implies that as the friction coefficient increases, the number of particle rearrangement events, due to either slipping of particle failure, increases.

The second feature of the stress-strain relationships is that as the friction coefficient increases the peak magnitude of the equivalent stress decreases. The peak stress during the simulation was 1750, 1700 and 1275 for friction coefficients of 0.1, 0.2 and 0.3 respectively.
Table 7-2: Equivalent Stress in Rankine Assembly

Based on these observations of the simulated particle assembly, the following hypothesis has been developed. As the friction coefficient increases, the frictional sliding forces between the particles in the assembly increases. The particles are more constrained as the friction coefficient increases, and hence the stiffness of the particle assembly must increase as well. The implication of the increased stiffness of the particle assembly is higher stresses in the particles. However, due to the inclusion of the Rankine failure model, the higher stresses in the particles imply that more of these particles will fail. Since the number of particle failure events will increase, the peak force and hence peak equivalent stress should decrease. Thus, for a constant displacement, the peak force magnitude, and hence peak stress magnitude, should decrease as the friction coefficient increases.

Figure 7-9: Macroscopic response of Rankine Assembly
8. Conclusions and Recommendations

Based on the results of the particle assembly simulations discussed in the previous chapter, the following conclusions and recommendations can be drawn:

1. The pre-processor that was used in this investigation is not optimal for the generation of the initial configuration of the particle assembly. Due to the paradigm used, the porosity of the generated particle assembly is too high. Two recommendations can be made to decrease the porosity of the initial particle assembly. Firstly, the pre-processor paradigm should be modified to reduce the porosity of the generated particle assembly. Secondly, the output of the current pre-processor could be used as the input in a rigid discrete element code. During the initial stage of the compaction process, the deformation of the particles can be ignored.

2. The macroscopic response of the particle assembly is highly sensitive to the constitutive relationship used to describe the material failure behaviour of the particles in the assembly. It was found that the Tresca failure model was not suitable to simulate the particle evolution of brittle powder compaction. The Rankine failure model appears to be better suited, however, the validity of this model needs to be experimentally verified.

3. It was found that the particle interaction affected the macroscopic response of the particle assembly to a lesser extent. The numerical simulations indicate that as the frictional forces increase, the peak equivalent stress in the particle assembly decreases. It is recommended that, if possible, a laboratory experiment should be performed to verify the existence of this trend.
4. Finally, based on the results presented, it is believed that the CFDEM method is feasible for the simulation of brittle powder compaction. However, further experimental work on brittle powder compaction needs to be performed in order to qualify the validity of the current simulations.
References


References


Appendix A: Euler Backward Integration Scheme

The total strain increment at a material point is assumed to consist of elastic and failure components [27], i.e.

\[ \dot{\varepsilon} = \dot{\varepsilon}^e + \dot{\varepsilon}^f \]  \hspace{1cm} (A 1)

where \( \dot{\varepsilon} \) is the total strain rate vector \( \begin{pmatrix} \varepsilon_{xx} & \varepsilon_{yy} & \varepsilon_{zz} & 2\varepsilon_{xy} \end{pmatrix}^T \) for plane strain

\( \dot{\varepsilon}^e \) is the elastic strain rate

\( \dot{\varepsilon}^f \) is the failure strain rate

The stress at a material point is assumed to depend entirely on the elastic strain and may be calculated from Equation 3-5, i.e.

\[ \dot{\sigma} = D\dot{\varepsilon}^e = D(\dot{\varepsilon} - \dot{\varepsilon}^f) \]  \hspace{1cm} (A 2)

The relationship between the failure strain component and the stress increment can be derived by assuming that the failure strain increment is proportional to the stress gradient of the failure potential [25], i.e.

\[ \dot{\varepsilon}^f = \dot{\lambda} \frac{\partial Q}{\partial \sigma} \]  \hspace{1cm} (A 3)

where \( \dot{\varepsilon}^f \) is the failure strain rate

\( \dot{\lambda} \) is the plastic multiplier

\( Q \) is the plastic potential

\( \sigma \) is the stress vector

When the plastic potential is identical to the failure surface \( F \), Equation A 3 is termed an associated flow rule, and becomes:
Appendix A: Euler Backward Integration Scheme

\[ \dot{\varepsilon}^f = \dot{\lambda} \frac{\partial F}{\partial \sigma} \]  

A 4

For a work hardening material, Pankaj [29] defines the incremental equivalent failure strain as:

\[ d\varepsilon^{\sigma} = -\dot{\lambda} \frac{\partial F}{\partial Y} \]  

A 5

For the Tresca failure model, the failure surface is defined by Equation 4-8 and thus the incremental equivalent failure strain becomes:

\[ d\varepsilon^{\sigma} = \dot{\lambda} \]  

A 6

Using an Euler backward integration scheme, Equations A 2, A 5 and A 6 become:

\[ \Delta \sigma_{n+1} = D \left( \Delta \varepsilon_{n+1} - \Delta \varepsilon_{n+1}^f \right) \]

\[ \Delta \varepsilon_{n+1}^f = \Delta \lambda \left( \frac{\partial F}{\partial \sigma} \right)_{n+1} = \Delta \lambda \sigma_{n+1} \]

\[ \Delta \varepsilon_{n+1}^{\sigma} = \Delta \lambda \]  

A 7
Appendix B: Material Model Utility Function Source Code

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define M_Pi 3.14159265359

/*-----------------------------------------------*/
/* Prototypes */
/*-----------------------------------------------*/

void invariants( const double *Stress, const int *flag,
                 double *I1, double *J2, double *theta,
                 double *StressEff, double *StressDev );

void flowvector( const double *StressDev, const double *StressEff,
                 const int *flag, const double *theta, const double *I1,
                 const double *J2, double *A );

/*-----------------------------------------------*/
/* Function: invariants */
/* Purpose: Calculates the stress invariants and effective stress for a
           Tresca and a Rankine Yield function */
/* Written by: Imraan Parker */
/* Last Modified: 25/09/96 */
/* Parameter List: */
/* Stress - plane strain stress vector */
/* flag - material model flag, 1=Tresca, 2=Rankine */
/* I1 - first stress invariant */
/* J2 - second deviatoric invariant */
/* theta - */
/* StressEff - effective stress */
/* StressDev - deviatoric stress */
/* */
/*-----------------------------------------------*/

void invariants( const double *Stress, const int *flag, double *I1, double *J2,
                 double *theta,
                 double *StressEff, double *StressDev )
{
    double StressMean, J3, rootJ2;

    const double Angle = 29.0/180.0*M_Pi;
    const double SQRT3 = sqrt(3.0);
const double PI23 = 2.0/3.0*M_Pi;

/* calculate 1st stress invariant */
*I1 = Stress[0] + Stress[1] + Stress[3];

StressMean = *I1/3.0;

/* calculate deviatoric stress vector */
StressDev[0] = Stress[0] - StressMean;
StressDev[2] = Stress[2];

/* calculate 2nd deviatoric stress invariant */

/* calculate 3rd deviatoric stress invariant */

rootJ2= sqrt(*J2);
*StressEff = rootJ2;

if((*StressEff)!=0.0) {
    if((*J2!=0.0)&&(J3!=0.0)) *theta = -3.0*SQRT3*J3/(2.0*(J2)*rootJ2);
    else *theta = 0.0;
    if(*theta>1.0) *theta = 1.0;
    if(*theta<-1.0) *theta = -1.0;
} else
    *theta = 0;

/* calculate theta */
*theta = asin(*theta)/3.0;

/* select appropriate effective stress measure */
switch(*flag) {

    /* Tresca */
    case 1:
        if( fabs(*theta) < Angle ) {
            *StressEff *= 2.0*cos(*theta);
        }
        else {
            *StressEff *= SQRT3;
        }
        break;

    /* Rankine */
    case 2:

if( (*theta) < Angle )
    *StressEff = 2.0*rootJ2/SQRT3*sin(*theta+PI/3) + StressMean;
else
    *StressEff = rootJ2/SQRT3 + StressMean;
bread;
Appendix B: Material Model Utility Function Source Code

/*--------------------------------------------------------------------------------*/
/* Function: flowvector */
/* Purpose: Evaluates the flow vector for both Tresca and Rankine */
/* Yield Surfaces. */
/* Written by: Imraan Parker */
/* Last Updated: 25/09/96 */
/* Parameter List: */
/* StressDev - deviatoric stress */
/* StressEff - effective stress */
/* flag - material model flag, 1 = Tresca, 2 = Rankine */
/* theta - */
/* I1 - first stress invariant */
/* J2 - second deviatoric invariant */
/* A - flow vector */
/*--------------------------------------------------------------------------------*/

void flowvector( const double *StressDev, const double *StressEff,
        const int *flag, const double *theta, const double *I1,
        const double *J2, double *A )
{

double c1, c2, c3;

const double t = *theta;
const double A2Factor = 0.5/sqrt(*J2);
const double J2third = (*J2)/3.0;
const double Angle = 29.0/180.0*M_Pi;
const double SQRT3 = sqrt(3.0);
const double PI23 = 2.0/3.0*M_Pi;

switch(*flag) {
    case 1: /* Tresca Yield surface */
        c1 = 0;
        if( fabs(t) < Angle ) { /* not singularity */
            c2 = 2.0*(cos(t) + sin(t)*tan(3.0*t));
            c3 = SQRT3*sin(t)/( (*J2)*cos(3.0*t) );
        } else {
            c2 = SQRT3;
            c3 = 0.0;
        }
        break;
    case 2: /* Rankine Yield Surface */
        c1 = 1.0/3.0;
        if( t < Angle ) { /* not singularity */
            c2 = 2.0/SQRT3*sin(t+PI23) + tan(3*t)/SQRT3*( cos(t) + SQRT3*sin(t) );
            c3 = ( cos(t) + SQRT3*sin(t) )/( 2.0*(J2)*cos(3*t) );
        } else {
            c2 = 1/SQRT3;
            c3 = 0.0;
        }
}

108
break;
}

/* calculate flow vector */
Appendix C: ELFEN/Explicit User-Material Source Code

C @(#) Module:<usemat.f>  Version:1.12  Date:11/18/96
SUBROUTINE USEMAT
1( AREAI ,DEACTV ,DIRECT ,DSTRAN ,ELTEMP ,
 2 ENELAS ,ENINEL ,ETHKG0 ,FRSTAT ,NEL ,
 3 PRPELA ,PRPELT ,PRPUSR ,PRPUST ,ROTMAT ,
 4 STATEV ,STRES ,STVNAME ,TMPELA ,TMPUSR ,
 5 ADAMP ,DENS ,POISS ,WAVES ,YOUNG ,
 6 DTIME ,ILAYER ,IGROUP ,IELTYP ,IUSTYP ,
 7 MATNO ,NTMELA ,NTMUSR ,NTBELA ,NTBUSR ,
 8 NDCOS ,NELPRC ,NFSTAT ,NPRELA ,NPRUSR ,
 9 NROTM ,NROTT ,NSTATD ,NSTRE ,NSTRAN ,
O NSTTYP ,THMSTN ,TIME ,IFCUT ,TMPPRP )
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
CHARACTER
1 NAME*6 ,STVNAME*8
LOGICAL
1 DEACTV ,THMSTN ,TMPPRP ,IFCUT
DIMENSION
1 AREAI(NELPRC) ,DEACTV(NELPRC),
2 DIRECT(NDCOS,NELPRC) ,DSTRAN(NELPRC,NSTATD),
3 ELTEMP(NELPRC) ,ENELAS(NELPRC) ,
4 ENINEL(NELPRC) ,ETHKG0(NELPRC) ,
5 FRSTAT(NFSTAT,NELPRC) ,NEL(NELPRC)
6 ROTMAT(NRROTM,NRROTM,NELPRC),STATEV(NSTATD,NELPRC),
7 STRES(NELPRC,NSTATD) ,STVNAME(NSTATD),
8 PRPELA(NPRELA) ,PRPUSR(NPRUSR),
9 PRPELT(NPRELA,NELPRC) ,PRPUST(NPRUSR,NELPRC),
O TMPELA(NTMELA,NTBELA) ,TMPUSR(NTMUSR,NTBUSR),
1 ADAMP(NELPRC) ,DENS(NELPRC) ,
2 POISS(NELPRC) ,WAVES(NELPRC) ,
3 YOUNG(NELPRC)
DATA NAME/*USEMAT*/

C**********************************************************
C Main user subroutine for explicit user defined materials
C*HISTORY
C Name   Date       Comment
C M.E.Honnor   April 1995
C M.Vaz Jr. March 1996 Added temperature dependent properties
C I. Parker   Oct. 1996 Added Tresca linear softening failure model
C*EXTERNAL
C*Arrays
C AREA   - Initial element area
C=DEACTV - Logical flag to deactivate element, once a element is removed
C by setting this flag to .TRUE. it cannot be reactivated
C=DIRECT - Material direction cosines for orthotropic materials
C DSTRAN - Strain increments rotated to midpoint configuration
Appendix C: ELFEN\Explicit User-Material Source Code

C NSTTYP = 1 3D Solid ~ Exx,Eyy,Ezz,2Exy,2Eyz,2Ezx
C 2 Plane Stress ~ Exx,Eyy,2Exy,=Ezz
C 3 Plane Strain ~ Exx,Eyy,2Exy
C 4 Axisymmetric ~ Exx,Eyy,2Exy,Ezz
C 5 Thick Shell ~ Exx,Eyy,2Exy,2Exz,2Eyz,=Ezz
C Note: For plane stress and thick shells update the
C ---- thickness strain increment DSTRAN(=Ezz) so that it
C ---- can be used for the thickness update
C ELTEMP - Gauss point temperatures
C=ENELAS - Strain energy (Previous timestep values on entry)
C=ENINTEL - Inelastic work (Previous timestep values on entry)
C ETHG01 - Initial Gauss point thickness strains due to thermal loading
C=FRSTAT - Fracture state indicators
C .1 - Fracture fail factor, set to 1.0 for fracture
C .2 - Fracture fail angle anticlockwise from X axis in radians
C NEL - Element numbers
C PRPELA - Elastic material properties
C PRPELT - Elastic material properties at current temperature
C PRPUSR - User defined material properties
C PRPUS1T - User defined material properties at current temperature
C ROTMAT - Rotation matrix or spin terms
C 3D ~ NROTT = 3, NROTM = 1
C == ROTMAT(1,1,IELPRC) = Wxx(IELPRC)
C ROTMAT(2,1,IELPRC) = Wyy(IELPRC)
C ROTMAT(3,1,IELPRC) = Wzz(IELPRC)
C Wxx, Wyy and Wzz ~ Spin terms
C 2D ~ Plane and shells NROTT = 4, NROTM = 2
C == ROTMAT(1,1,IELPRC) = QC11(IELPRC)
C ROTMAT(2,1,IELPRC) = QC12(IELPRC)
C ROTMAT(3,1,IELPRC) = QC21(IELPRC)
C ROTMAT(4,1,IELPRC) = QC22(IELPRC)
C ROTMAT(1,2,IELPRC) = Q111(IELPRC)
C ROTMAT(2,2,IELPRC) = Q12(IELPRC)
C ROTMAT(3,2,IELPRC) = Q21(IELPRC)
C ROTMAT(4,2,IELPRC) = Q22(IELPRC)
C QC ~ Current rotation matrix
C Q1 ~ Initial rotation matrix
C=STATEV - State variables (Previous timestep values on entry)
C=STRES - Current stress components
C Previous timestep values rotated to midpoint configuration on
C entry. Returned values will be rotated back from the current
C rotated configuration after returning from this routine.
C NSTTYP = 1 ~ Sxx,Syy,Szz,Sxy,Syz,Szx
C 2 ~ Sxx,Syy,Sxy
C 3 ~ Sxx,Syy,Sxy,Szz
C 4 ~ Sxx,Syy,Sxy,Szz
C 5 ~ Sxx,Syy,Sxy,Szx,Syz
C STVNAME - State variable names
C TMPELA - Elastic material properties temperature function
C TMPUSR - User defined material properties temperature function
C =ADAMP - Rayleigh damping factor
C =DENS - Current density
C =POISS - Poisson's ratio
C =WAVES - Wave speed
C =YOUNG - Young's modulus
C Variables
C DTIME - Time step length
C ILAYER - Layer for shells
C IGROUP - Group number
C IELTYP - Elastic material type
C IUSTYP - User defined material type
C MATNO - Material number
C NTBELA - Number of elastic properties plus 1
C NTBUSR - Number of user material properties plus 1
C NTMELA - Dimension of elastic material properties temperature function
C NTMUSR - Dimension of user defined material properties temperature
C NDCOS - Number of direction cosines required to define material
directions
C ~ Angle only ( plane and shells )
C ~ All nine direction cosines ( 3D solid )
C NELPRC - Number of elements being processed, maximum 128. Each
group
C is split into 128 element sub-groups before processing.
C NFSTAT - Number of fracture state indicators (=2)
C NPRULA - Number of elastic properties
C NPRUSR - Number of user defined material properties
C NROTM - Number of rotation terms
C NROTT - Dimension of rotation term
C NSTATD - Number of state variables
C NSTRE - Number of stress components
C NSTRAN - Number of strain components
C NSTTYP - Stress type number 1 ~ 3D Solid
C 2 ~ Plane Stress
C 3 ~ Plane Strain
C 4 ~ Axisymmetric
C 5 ~ Thick shell
C NTMELA - Number of elastic material properties temperature function
C NTMUSR - Number of user defined material properties temperature
C function
C THMSTN - Thermal strain flag
C TIME - Current time
C (c) Copyright 1995, Rockfield Software Limited, Swansea, UK
C*******************************************************
C
C Local Variables:
C StresO - Stress values from previous time step.
C StresN - Updated stress values at end of current time step.
Appendix C: ELMENExplicit User-Material Source Code

C SDev - Deviatoric stress vector.
C StrmP - Plastic Strain vector.
C DStrnN - Current total strain increment.
C DStrnP - Current increment in plastic strain.
C A - Flow vector.
C EffSts - Effective Stress.
C EffStr - Equivalent "Plastic" Strain.
C CurYS- Current Yield Strength.
C Lambda - Plastic Multiplier.
C DLamb - Iterative Plastic multiplier.
C E - Young's Modulus.
C nu - Poisson's Ratio.
C SigmaT - Tensile Yield Stress.
C SigmaS - Shear Strength.
C Gc - Fracture Energy Release Rate. (Mode II for Tresca Model).
C Area - Element Area.
C h - Characteristic crack length.
C Angle - Angle subtended between x-axis and 1st principal stress direction.
C Dir - Random multiplier(-1/1) used to determine crack angle for Tresca Failure Model.
C D1,D2,G- Components of Elastic Constitutive Matrix.
C SftMod - Softening Modulus.
C I1 - First stress invariant.
C J2 - 2nd invariant of deviatoric stress tensor.
C theta - alternative stress invariant.
C We - Total elastic work.
C Wp - Total plastic work.
C DWe - Incremental Elastic Work.
C DWp - Incremental Plastic Work.
C rho - Density.
C Damp - Rayleigh Damping Coeff.
C WavSp - Wave Speed.
C
C******************************************************************************

DOUBLE PRECISION StresN(4), StresO(4), DStrnN(4), DStrnP(4),
+ StrmP(4), A(4), SDev(4)
DOUBLE PRECISION EffSts, EffStr, CurYS, Lambda, E, nu, SigmaT,
+ SigmaS, Gc, Area, DLamb, h, Angle, Dir
DOUBLE PRECISION D1, D2, G, atDa, F, SftMod, I1, J2, theta
DOUBLE PRECISION We, Wp, DWe, DWp
DOUBLE PRECISION PI, TOL
DOUBLE PRECISION rho, Damp, WavSp

PARAMETER ( PI = 3.14159265359, TOL = 0.001 )

INTEGER I, J, K, FLAG

113
C Set up plane strain constants
   E = PRPELA(1)
   nu = PRPELA(4)
   D1 = E*(1d0-nu)/((1d0+nu)*(1d0-2d0*nu))
   D2 = D1*nu/(1d0-nu)
   G = E/(2d0*(1d0+nu))

C Calculate material constants
* Density
   rho = PRPELA(13)
* Dam
   Damp = PRPELA(14)
* Wave Speed
   WavSp = SQRT(E*(1d0-nu)/((rho*(1d0+nu)*(1d0-2d0*nu))))
* Failure properties
   SigmaT = PRPUSR(1)
   Gc = PRPUSR(2)
   SigmaS = PRPUSR(3)

C Tresca Material Model
   IF(IUSTYP.EQ.101) THEN
      FLAG = 1
   C Rankine Material Model
   ELSE IF(IUSTYP.EQ.102) THEN
      FLAG = 2
   ELSE
      FLAG = -1
   END IF

   IF( (IUSTYP.EQ.101).OR.(IUSTYP.EQ.102) ) THEN

   IF(NSTTYP.EQ.3) THEN

      DO 1000 I = 1, NELPRC

   C First time step
   IF(TIME.LT.1d-12) THEN
      STATEV(3,I) = 0d0
      STATEV(4,I) = 0d0
      STATEV(5,I) = 0d0
      STATEV(6,I) = 0d0
      STATEV(7,I) = SigmaT
   END IF

   DO J = 1, 4
      StresO(J) = STRES(I,J)
      DStrnN(J) = DSTRAN(I,J)
      DStrnP(J) = STATEV(J+2,I)
   ENDDO
EffSts = STATEV(1,I)
EffStr = STATEV(2,I)
CurYS = STATEV(7,I)

Area = AREAI(I)

C Store old energy terms
  We = ENELAS(I)
  Wp = ENINEL(I)

C Calculate Softening Modulus based on material model
  h = 2d0*SQRAT(Area/PI)

IF(IUSTYP.EQ.101) THEN
  C--- Tresca
  SftMod = SigmaS*SigmaS/(2d0*Gc)*h
  ELSE IF(IUSTYP.EQ.102) THEN
  SftMod = SigmaT*SigmaT/(2d0*Gc)*h
  END IF

C Calculate Elastic Predictor
  StresN(1) = StresO(1) + D1*DStrnN(1) + D2*DStrnN(2)
  + D2*DStrnN(4)
  StresN(2) = StresO(2) + D2*DStrnN(1) + D1*DStrnN(2)
  + D2*DStrnN(4)
  StresN(3) = StresO(3) + G*DStrnN(3)
  StresN(4) = StresO(4) + D2*(DStrnN(1) + DStrnN(2))
  + D1*DStrnN(4)

C Check if F.LT.0

  Lambda = 0
  DLambda = 0
  DO J = 1, 4
    DStrnP(J) = 0d0
  END DO

  DO 200 J = 1, 10

C--- Calculate Stress Invariants & equivalent stress
    call Invariants(StresN,FLAG,1,J2,theta,EffSts,SDev)

C--- Evaluate Yield Function
    F = EffSts - CurYS

C--- Check if stress point is inside failure surface
IF(F.LE.0d0) GOTO 250

C--- Check if stress point is outside failure surface
       IF(F.GT.TOL) THEN
           call flowvector(SDev,EffSts,FLAG,theta,l1,J2,A)

           atDa = A(1)*( D1*A(1) + D2*( A(2)+A(4) ) ) +
                 A(2)*( D1*A(2) + D2*( A(1)+A(4) ) ) +
                 G*A(3)*A(3)
                 +
                 A(4)*( D1*A(4) + D2*( A(1)+A(2) ) )

           DLamb = F/( atDa - SftMod )
           IF(DLamb.LT.0d0) DLamb = 0d0
           lambda = lambda + DLamb

       * Update Incremental and Total Plastic Strain
           DO K = 1, 4
               DStrnP(K) = DLamb*A(K)
               StrnP(K) = StrnP(K) + DStrnP(K)
           END DO

       * Relax Elastic Predictor
           Stresn(1) = Stresn(1) - ( D1*DStrnP(1) + D2*DStrnP(2)
               + D2*DStrnP(4) )
           Stresn(2) = Stresn(2) - ( D2*DStrnP(1) + D1*DStrnP(2)
               + D2*DStrnP(4) )
           Stresn(3) = Stresn(3) - G*DStrnP(3)
           Stresn(4) = Stresn(4) - ( D1*DStrnP(4) + D2*DStrnP(1)
               + D2*DStrnP(2) )

       * Calculate Equivalent Plastic Strain
           EffStr = EffStr + DLamb

       * Calculate Current Yield Strength
           CurYS = CurYS - SftMod*DLamb

       * Calculate direction of 1st Principal Stress relative to x-axis
           Angle = Stresn(1) - Stresn(2)
           IF(Angle.EQ.0d0) Angle = 0.1d-20
           Angle = 0.5d0*ATAN(2d0*Stresn(3)/Angle)

       * Reset Fracture State indicator if material has failed in previous
        * time step.
           IF(FRSTAT(1,1).GT.1d0) FRSTAT(1,1) = 0d0

           FRSTAT(1,1) = 1d0 - CurYS/SigmaT
           FRSTAT(2,1) = Angle

           IF(FLAG.EQ.1) THEN

       C Choose Random Direction
CALL RandDir(Dir)
FRSTAT(2,I) = Angle + Dir*PI/4d0
ELSE IF(FLAG.EQ.2) THEN
FRSTAT(2,I) = Angle + PI/2d0
END IF

* Check if material has failed
IF(CurYS.LT.TOL) THEN
EffStr = 0d0
CurYS = SigmaT
DO K = 1, 4
StrnP(K) = 0d0
END DO
FRSTAT(1,I) = 1d0
END IF

ELSE IF( (F.GT.-TOL).AND.(F.LT.TOL) ) THEN
GOTO 250
END IF

200 CONTINUE
250 CONTINUE
C Store number of iterations required for convergence to
C failure surface.
STATEV(9,I) = J

C Calculate energy terms
DWe = 0d0
DWp = 0d0
DO J = 1, 4
DWp = DWp + 0.5*(StresN(J)+StresO(J))*DStrnP(J)
DWe = DWe + 0.5*(StresN(J)+StresO(J))*DStrnN(J)
END DO

DWe = DWe - DWp

C Update energy terms
We = We + DWe
Wp = Wp + DWp

C Update global variables
DO J = 1, 4
STRES(I,J) = StresN(J)
STATEV(J+2,I) = StrnP(J)
END DO
STATEV(1,I) = EffSts
STATEV(2,I) = EffStr
STATEV(7,I) = CurYS
STATEV(8,I) = F
Appendix C: ELFEN\Explicit User-Material Source Code

C Store Energy Terms
   ENELAS(I) = We
   ENINEL(I) = Wp
C Store Material Properties
   ADAMP(I) = Damp
   DENS(I) = rho
   POISS(I) = nu
   YOUNG(I) = E
   WAVES(I) = WavSp

1000 CONTINUE

END IF

ELSE
   CALL ERRDAT(IUSTYP,R0,",",1,"I")
   CALL WRTER("A0319E",NAME,1)
   CALL TRLERR
END IF
RETURN

END
Appendix D: C++ Class Description

This appendix contains a brief description of the most important abstract data types (classes) developed for use in both the pre- and post-processors. The member functions associated with the various classes are also outlined.
Circle

NAME
Circle - circle abstraction

DESCRIPTION
Circle is used to represent a circle in 2D Cartesian space. It handles all functions related to the index and position of the Circle.

PRIVATE DATA MEMBERS

int circle_number
Circle index.

Point2d centroid
Centroid of circle.

double radius
Radius of circle.

SUMMARY
Circle(void)
Circle(int number, Point2d centre, double radius)
Constructor methods used to declare and initialise instances of the Circle class.

void AddNumber(int number)
void AddCenter(Point2d centre)
void AddRadius(double radius)
void AddCircle(int number, Point2d centre, double radius)
The above methods allow the user to modify the index, centre and radius an instance of the Circle class individually or simultaneously.

int GetNumber(void)
Method returns the Circle number.

double GetX(void)
double GetY(void)
Point2d GetXY(void)
These methods retrieve the centre co-ordinates of the Circle instance. GetX() and GetY() retrieves the x and y components respectively whereas GetXY() returns a 2d vector.

double GetRadius(void)
Member function returns the radius of the Circle instance.
DiscreteFunction

NAME
DiscreteFunction - discrete scalar function of one variable

DESCRIPTION
The DiscreteFunction class represents a function of one variable sampled at discrete points. DiscreteFunction stores the discrete sample co-ordinates in an array of Point2d objects and uses linear interpolation to calculate the function value between data points. A member function is provided to read the data series from a file.

PRIVATE DATA MEMBERS
   int number_of_points
       Number of points in the data series.

   Point2d *point
       Pointer to an array of Point2d objects containing the co-ordinates of the data series.

   double min_x, max_x, min_y, max_y
       Limits of the data series.

SUMMARY
DiscreteFunction(void)
DiscreteFunction(int num_points)
DiscreteFunction(int num_points, Point2d *data)
   DiscreteFunction constructors.

int GetNumPoints(void)
   Returns number of data points.

void FindLimits(void)
   Private member function, calculates the x and y limits of the DiscreteFunction.

double GetMinX(void)
double GetMaxX(void)
double GetMinY(void)
double GetMaxY(void)
   Returns limits of DiscreteFunction.
int LinearInterpolate(double &interp_val, double y)

Uses linear interpolation to calculate the x value given the corresponding y value. Function returns 0 if successful, else 1 if unsuccessful.

int ReadFile(char *inputFileName)

Reads in DiscreteFunction data from a file. The filename is stored in a character array and the function returns 0 if operation was successful else 1 if the file could not be opened or 2 if the data is not in the proper format.
Appendix D: C++ Class Description

Node2d

NAME
Node2d - Node in 2d Cartesian space.

DESCRIPTION
Node2d abstracts an indexed point in 2d Cartesian space. The indexed point consists of a node number and a Point2d object containing the co-ordinates. Classes Particle2d and Polygon2d have access to the private data of the Node2d class.

PRIVATE DATA MEMBERS
int node_number
Node index.

Point2d node_coord
Vector containing x and y co-ordinates of the Node2d object.

SUMMARY
Node2d(void)
Node2d(int node_num, double xcoord, double ycoord)
Node2d(int node_num, Point2d coord)
Node2d constructors.

void AssignNodeNumber(int node_num)
Assign node number.

void AssignPoint(double xcoord, double ycoord)
void AssignPoint(Point2d &coord)
Assign co-ordinates to node.

doubleGetX(void)
double GetY(void)
Point2d GetXY(void)
Functions return the co-ordinates of the Node2d object.

void TranslateNode(Point2d &transvector)
Translates the Node2d object.

void Write(void)
Prints the values of the private data members of the Node2d object to the standard output.
Particle2d

NAME
Particle2d - Combined Finite Discrete Element mesh object.

DESCRIPTION
Particle2d contains information about the geometry and topology of a combined discrete finite element mesh. The geometry and topology of the mesh is stored in arrays of Node2d, Triangfem and Segment2d objects. Member functions are provided to store and access the private data members. Note that the function WriteGlobalElfenFile is a friend function of Particle2d.

PRIVATE DATA MEMBERS

int particle_number
   Particle index.

int number_of_elements
   Number of triangular elements, abstracted by the Triangfem class, in the Particle2d object.

int number_of_nodes
   Number of nodes, abstracted by the Node2d class, in the Particle2d object.

int number_of_boundary_segments
   Number of boundary segments, abstracted by the Segment2d class, in the Particle2d object.

Node2d *particle_nodes
   Pointer to the start of an array of Node2d objects defining the geometry of the Particle2d object.

Triangfem *particle_elements
   Pointer to the start of an array of Triangfem objects defining the topology of the Finite Element mesh of the Particle2d object.

Segment2d *particle_boundary_segments
   Pointer to the start of an array of Segment2d objects defining the boundary of the Particle2d object.
SUMMARY

Particle2d(void)
Particle2d(int particle_num)
~Particle2d(void)
    Particle2d constructors and destructor.

void AssignParticleNumber(int particle_num)
    Member function stores particle index.

void AssignNode(int nodeptr, int node_num, double x, double y)
    Member function stores nodal information into the private Node2d array.

void AssignElement(int elementptr, int element_num, int *nlist)
    Member function stores element information into the private Triangfem array.

void AssignBoundarySegment(int segmentptr, int segment_num, int nstart, int nend)
    Member function stores boundary segment information into the private Segment2d array.

int CreateParticleNodes(int particle_number)
int CreateParticleElements(int particle_elements)
int CreateParticleBoundarySegments(int num_bsegs)
    Allocates memory for the Node2d, Triangfem and Segment2d arrays describing the geometry and topology of the Particle2d object. Returns 0 if successful, else 1 if there is insufficient memory.

Node2d *FindNodePointer(int node_index)
    Member function returns a pointer to a Node2d object whose node number matches the node_index. Function returns a NULL pointer if no match could be found.

int NumberNodes(void)
int NumberElements(void)
int NumberBoundarySegments(void)
    Member functions returns the number of nodes, elements and boundary segments in the Particle2d object respectively.

int ParticleNumber(void)
    Member function returns the particle index.
void ReadFile( unsigned int offset, ifstream &node_stream, ifstream &element_stream, ifstream &bseg_stream)
    ReadFile reads data from the input streams supplied in the parameter list. The user must ensure that the files have been opened properly.

void WriteBoundarySegmentList(void)
void WriteElementList(void)
void WriteNodeList(void)
    These member functions dump the internal data associated with the Segment2d, Triangfem and Node2d arrays to the standard output.

void WriteElfenFile(ofstream &elfen_stream)
    WriteElfenFile converts the internal data of the Particle2d object into a format readable by the combined Finite Discrete element program Elfen. The user must ensure that the output file has been properly opened.
Appendix D: C++ Class Description

Point2d

NAME
Point2d - Two dimensional vector.

DESCRIPTION
Point2d describes a point in 2d Cartesian space. The addition, subtraction, multiplication and division operators have been overloaded to perform vector addition, vector subtraction, scalar multiplication and scalar division respectively.

PUBLIC DATA MEMBERS
double x, y
Co-ordinates of the point in Cartesian space.

SUMMARY
Point2d(void)
Point2d(double xcoord, ycoord)
Point2d constructors.

void Assign(double xcoord, ycoord)
Assign co-ordinates to Point2d object.

void Write(void)
Prints the values of the public data members to the standard output.
Appendix D: C++ Class Description

Polygon2d

NAME
Polygon2d - Polygon in 2D Cartesian space.

DESCRIPTION
Polygon2d contains information about the geometry and topology of a two dimensional polygon. Member functions are provided to translate, rotate, scale, create random polygons and calculate the polygon's area and centre of mass.

PROTECTED DATA MEMBERS

int polygon_number
   Polygon index.

int number_of_sides
   Number of segments on the boundary of the polygon.

double circum_circle_radius
   Radius of the circumscribed circle of the current Polygon2d object.

double minimum_segment_length
   Length of the smallest boundary segment.

double polygon_area
   Area of the current Polygon2d object.

Point2d centroid
   Co-ordinates of the polygon's centroid.

Node2d *furthest_node
   Pointer to the furthest node from the polygon centroid.

Node2d *polygon_nodes
   Pointer to the start of an array of Node2d objects defining the geometry and topology of the polygon.
SUMMARY

Polygon2d(void)
~Polygon2d(void)
  Polygon2d constructor and destructor.

double AreaOfPolygon(void)
  Member function calculates the area of the Polygon2d object.

void AssignPolygon(const int pnum, Node2d *nodeArray, const int numSides)
  AssignPolygon stores the polygon information in the private data members.

void AssignPolygonNumber(int polygon_number)
  Assigns polygon index.

int CentroidOfPolygon(void)
  Calculates the centre of mass of the polygon. Function returns 0 if successful, 1 otherwise.

int CirCumCircle(void)
  Calculates the circumscribed circle of the polygon. Also stores a pointer to the furthest node from the centroid of the polygon.

void CreateRandomPolygon(Circle BoundingCircle, int min_sides, int max_sides)
  Creates a randomly shaped polygon whose vertices lie on the circumference of the circle defined by BoundingCircle. The number of vertices of the polygon is chosen to lie randomly between min_sides and max_sides.

void FindMinSegmentLength(void)
  Finds the boundary segment with the smallest length and stores this value in the private data member minimum_segment_length.

double GetArea(void)
  Returns the polygon area.

Point2d GetCentroid(void)
  Returns the centroid co-ordinates.

double GetCirCumCircleRadius(void)
  Returns the circumscribed circle radius.

Point2d GetFurthestNodeCoords(void)
  Returns the co-ordinates to the furthest node from the polygon centroid.

int GetPoint(int vertex_number, Point2d &vertex_coord)
Appendix D: C++ Class Description

Co-ordinates of the polygon vertex with index vertex_number are stored in the Point2d object vertex_coord.

**double GetMinSegmentLength(void)**

Returns the minimum boundary segment length.

**int IsInsidePolygon(Point2d test_point)**

Function determines if test_point is inside the current polygon. Returns 0 if true, else 1 if false.

**Int NumberOfSides(void)**

Returns the number of sides of the polygon.

**int PolygonNumber(void)**

Returns the polygon index.

**void RandomisePolygon(Polygon2d &parent, double pert_radius, Circle &bound_circle)**

Function randomises the polygon parent by a perturbation radius of magnitude pert_radius. The resulting polygon is first randomly rotated and then scaled to fit inside the circle defined by bound_circle. The user must ensure that a valid perturbation radius is supplied.

**void RotatePolygon(double theta)**

Rotates the polygon about its centroid by an angle of theta radians.

**int ScalePolygon(double scale_factor)**

Scales the current polygon about its centroid by the factor scale_factor. Function returns 0 if successful, else 1 if not.

**void Write(void)**

**void Write(char *fname)**

These member functions dump the values of the protected data members to the standard output or the file with name fname.
RandomNumber

NAME
RandomNumber - Pseudo-Random number class

DESCRIPTION
RandomNumber is a pseudo-random number generator. It provides methods for generating pseudo-random numbers between upper and lower bounds. The seed for the pseudo-random number generator has to be set via the friend function SetSeed().

PRIVATE DATA MEMBERS
unsigned int generate_index
Positive integer, stores the number of times the pseudo-random number generator has been called.

double randomnumber
Pseudo-random number.

SUMMARY
RandomNumber(void)
RandomNumber constructor.

double GetRandom(void)
Returns double precision floating point value between 0 and 1.

double GetRandom(double lbound, double ubound)
Returns double precision floating point value between the lower and upper bounds.

unsigned int GetIndex(void)
Returns the current value of the seed.
Appendix D: C++ Class Description

Segment2d

NAME
Segment2d - Two dimensional straight line segment.

DESCRIPTION
Segment2d describes a straight line segment in 2D Cartesian space. The Segment2d object consists of a segment index and two Node2d objects. Class Particle2d has access to the private data members of this class.

PRIVATE DATA MEMBERS

int segment_number
Segment index.

int is_boundary_segment
Boundary flag set to 0 if segment is a boundary segment, else set to 1.

Node2d *node_start, *node_end
Pointers to the Node2d objects defining the segment.

SUMMARY

Segment2d(void)
Segment2d(int seg_num, Node2d node_start, Node2d node_end)
Segment2d constructors.

int IsBSeg(void)
Function returns 0, i.e. TRUE, if Segment2d object is a boundary segment, else returns 1 if not.

int GetSegNum(void)
Returns the segment index.

Node2d *GetStartNode(void)
Node2d *GetEndNode(void)
Returns pointers to Node2d objects defining the start and endpoints of the line segment.
Appendix D: C++ Class Description

Triangfem

NAME
Triangfem - Three noded triangular finite element.

DESCRIPTION
Triangfem is a data object that contains the geometry and topology of a three
noded triangular element. The data object contains an element number and a
three component vector of pointers to Node2d objects. The vector of pointers
defines both the geometry and topology of the element. Particle2d is a friend
class, i.e. any Particle2d instance has full access to the Triangfem internal data.

PRIVATE DATA MEMBERS

   int element_number
       Element index.

   Node2d *node[3]
       Array of pointers to Node2d objects that define the three nodes of the
       triangular element.

SUMMARY

   Triangfem(void)
Triangfem(int element_num, Node2d *node[3])
       Triangfem constructors.

   int GetElementNumber(void)
       Returns the element index.

   Node2d *GetNode(int node_index)
       Returns a pointer to the Node2d object defining the node node_index of
       the triangular element. The user must ensure that node_index is an
       integer between 1 and 3.