Development and Implementation of a Combined Discrete and Finite Element Multibody Dynamics Simulation Environment

by
Petros Komodromos

Diploma of Civil Engineering, University of Patras, Greece
Master of Science in Civil and Environmental Engineering, MIT

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Information Technology

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2001

Author .....
Department of Civil and Environmental Engineering
June 27, 2001

Certified by .....
John R. Williams
Associate Professor, Department of Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies
Department of Civil and Environmental Engineering

Development and Implementation of a Combined Discrete and Finite Element Multibody Dynamics Simulation Environment

by

Petros Komodromos

Submitted to the Department of Civil and Environmental Engineering on June 27, 2001, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Information Technology

Abstract

Some engineering applications and physical phenomena involve multiple bodies that undergo large displacements involving collisions between the bodies. Considering the difficulties and cost associated when conducting physical experiments of such systems, there is a demand for numerical simulation capabilities. The discrete element methods (DEM) are numerical techniques that have been specifically developed to facilitate simulations of distinct bodies that interact with each other through contact forces. In DEM the simulated bodies are typically assumed to be infinitely rigid. However, there are multibody systems for which it is useful to take into account the deformability of the simulated bodies.

The objective of this research is to incorporate deformability in DEM, enabling the evaluation of the stress and strain distributions within simulated bodies during simulation. In order to achieve this goal, an Updated Lagrangian (UL) Finite Element (FE) formulation and an explicit time integration scheme have been employed together with some simplifiying assumptions to linearize this highly nonlinear contact problem and obtain solutions with realistic computational cost.

An object-oriented extendable computational tool has been built specifically to allow us to simulate multiple distinct bodies that interact through contact forces allowing selected bodies to be deformable. Database technology has also been utilized in order to efficiently handle the huge amounts of computed results.

Thesis Supervisor: John R. Williams
Title: Associate Professor of Civil and Environmental Engineering

Acknowledgments

First, I would like to thank my thesis advisor, Prof. John Williams, and my professors and doctoral committee members, Prof. Jerome Connor and Kevin Amaratunga, for all the valuable advise, guidance and help they have provided to me throughout my graduate studies at MIT. I could not have made it without their help and support.

I am also grateful to many professors during my graduate studies, both here at MIT and at the University of California at Berkeley, for the valuable knowledge and education they have offered me. I am particularly grateful to Prof. George Kocur for his support and the insightful discussions we had, and to Prof. Bathe for his valuable advise. I am also extremely grateful to my professors during my undergraduate studies at the University of Patras in Greece, in particular to Prof. Dimitris Beskos, Michalis Fardis and Stavros Anagnostopoulos.

Next, I would like to thank my many-many good friends that I have been fortunate to have here at MIT and Boston, who have made my life much better and happier. Although I would like to thank everyone of them individually, this is impossible in these few lines. Therefore, I have to limit it to few who had the most impact on my academic and personal life.

First, I would like to thank my very close friends Debjit Das, Mathew Kurian, and Ben Cook, whose help has been the most critical during my studies and life at MIT. I am extremely thankful to them for everything they have done for me, especially for providing the most invaluable advise not only regarding my Ph.D. work, but also my personal life. I am also very grateful to all my friends here at the Civil Engineering Department, in particular to Bharath, Sugata, Sudarshan, JP, George (Constantinides), Noelle, Emilio, Eric, Kiran, Siva, Sameer, Achilleas, Sofoklis, Dimitris, Chang, Nadine, Yijun, Emma, Moon-Seo, Sang Jyun, Fuxin, Julio, Jingsong and Constantinos.

There are no words to sufficiently thank my buddy, Joannie McCusker, for everything she has done for me. She has always been a main source of strength and support for most students of the Information Technology group. Also, I am very grateful to Anthee, Elaine, Stephanie, Jessie, Cynthia and Maria for their support, help and care, making my student life much smoother.

I would also like to express my gratitude and thanks to many good friends outside the CEE Department, especially my dear Greek and Serbian friends, who have made my life in Boston much more colorful. Many-many thanks to my very dear friend Eleni, who has been a source of strength, reminding me always, in the most noble way, our beloved country, Cyprus. Many-many thanks to my dear friends Peggy, Rada, Thalia, Sophia, and Constantinos (Boussios) for the great company and friendship they have offered me. I have been so fortunate to have so many good Greek friends through the Hellenic Students Association of MIT that it is practically impossible to thank them all individually. Thanks to all of them for everything!

Last but not least, I would like to thank my family and friends in Cyprus and Greece for believing in and supporting me. It is indeed a blessing to have them all emotionally so close to me. Their unconditional love, emotional support and encouragement have always been my driving force in pursuing my goals and fulfilling my dreams. I would like to specifically thank my grandparents, George and Chrisanthia; my sister Marina, who has been next to me during all hard times I went through; the sweetest and most lovable girl I have ever met, Marina Cocconi, and her parents; my best friends Georges, Zenon, Andreas, George, Marios, Stefanos, Katia, Michalis and Nondas; my sister Christina; and all my cousins, uncles and aunts, especially Elena, Alexis, Georgio, Maria, Michalis, Dora, Lefteris, Lia, Yianna, Jason, Irene, Metaxas, and Herodotos. Finally, it is difficult to find words to express my love and gratitude to my parents, Androulla and Yiannis, whose love, encouragement, and understanding have always been a constant source of motivation and strength for me. Without their endless support and courage this achievement would not have been possible. A lifetime is too short to pay off all the sacrifices they have done to enable my studies and goals.

Αφιερωμενο, με ολη μου την αγαπη, στους γονεις μου.

Dedicated with all my love to my parents

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In many physical phenomena and engineering applications there are systems with large numbers of distinct bodies that undergo large displacements and rotations involving contacts between the bodies of the system. Examples range from particulate granular materials, such as soil, to multibody systems, such as masonry structures. Such discontinuous systems are characterized by the contacts that occur between the moving distinct bodies of the system and the freedom of the individual bodies to move in space under the action of the forces that are exerted on them.

A major category of discontinuous systems is granular materials, which can be found in many applications. In engineering practice, the macroscopic behavior of granular, or particulate in general, materials is defined assuming that the material is continuous. Bulk material properties and empirical rules based on observations and experiments at the macroscopic level are employed. Continuous idealizations of the behavior of granular materials are used, assuming that local perturbations at the particle level have negligible effects at the scale of length of interest. Essentially, the continuity assumption allows characterization of the macroscopic behavior of the system using bulk material properties by averaging the interactions that occur at the particle level. This assumption may be justified in some cases by considering that the microstructure length scale is much smaller than the

dimensions of the problem under investigation and assuming that local perturbations at the particle level have negligible effects at the scale of interest.

However, macroscopic behavior depends not only on bulk material properties but also upon the properties and geometric information of the constituent bodies, such as the size and shape of the particles, their distribution, packing density, interparticle friction, and particle-to-particle interaction laws. In essence, the macroscopic behavior of particulate materials depends on the properties of individual particles and their interactions. For granular materials, such as sand and gravel, the deformation and load transfer are mainly due to rearrangements of the particles and interparticle contacts, respectively.

Therefore, it is useful to study particulate systems at the particle level, and not as a continuum, to better understand the macroscopic behavior and how it is affected by particle-level characteristics. Besides the need for more accurate modeling of particulate systems, the understanding of the relationship between microscopic parameters and macroscopic behavior is essential for the development of more rational constitutive models for granular materials. Physical experiments at the particle level are almost impossible to be performed, since any instrument installations change the physical problem characteristics. Therefore, computational tools that allow numerical simulations of such discontinuous systems are required.

Discrete element methods (DEM) research is motivated and driven by the need to provide numerical means to simulate and study particulate and multibody systems (Cundall [5], [6] and [7], and Williams et al [23]). The DEM provide alternative numerical means to simulate multiple interacting bodies undergoing large motions in order to observe and record detailed interactions occurring at the particulate level. Although DEM have been introduced as a tool to study granular soils and rock masses, they can also be used to model and study many other physical problems that involve large numbers of moving and

colliding distinct bodies. Numerical simulations of multibody systems are useful, not only for understanding the behavior of particulate systems, but also for making engineering decisions that are necessary for the design and analysis of systems that process and transport particulate systems.

In general, the DEM assume that the simulated bodies are infinitely rigid, which is a reasonable assumption for many particulate systems. For example, the deformation of granular materials is mainly due to the rearrangement of the particles (Rege [26]) and not the particle deformations, and the stress distribution within a particle is not of interest.

However, there are multibody systems, such as masonry structures, for which it is desirable to be able to obtain the stress and strain distributions within the bodies as they collide with each other. The research presented in this thesis has been motivated and driven by the need to provide numerical means to simulate and study multibody systems that may be deformable, allowing the evaluation of the stresses and strains within the simulated bodies. In order to enable the consideration of deformability of simulated bodies and the evaluation of their stresses and strains distributions, an Updated Lagrangian (UL) Finite Element (FE) formulation and an explicit time integration scheme have been employed together with some simplifying assumptions to linearize this highly nonlinear contact problem and obtain solutions with realistic computational cost.

An extendable, object-oriented and portable computational tool has been built to enable numerical simulations of multiple distinct bodies that interact through contact forces while allowing selected bodies to be deformable. Since huge quantities of results are computed, database technology has also been utilized in order to efficiently store, search and manipulate them.

1.2 Discrete Element Methods

1.2.1 Description

The discrete element methods (DEM) are numerical techniques specifically developed to simulate multiple distinct bodies that interact through contact forces (Cundall [5], [6] and [7], and Williams et al [23]). The main capability of DEM is that they allow the simulation of assemblages of multiple unrestrained distinct bodies that interact through contacts. Each discrete element, i.e. body, has distinct boundaries, which physically separate it from each other element in the analysis, and interacts with other bodies only with contact forces whenever the bodies are identified to collide with each other.

A major characteristic of the DEM is that they allow fast and efficient automatic recognitions of new contacts as well as complete detachments of bodies, which were previously in touch, during simulations. Since the greatest difficulty in simulations of large number of interacting bodies is the contact detection and the determination of the contact forces, the DEM give priorities to these aspects and make some simplifying assumptions so as to make the problem tractable.

The numbers of distinct bodies that are required to be used in such simulations range from several bodies, in the case of multibody systems, to several thousands, or even millions, in the case of granular systems. The use of a large numbers of bodies is often required when simulating a particulate system so as to have a representative sample of the physical problem. In order to be able to model systems with large numbers of interacting, through contact forces, discrete bodies, emphasis is put on the most computationally intensive part of the problem is the contact detection.

A typical assumption of DEM is that the constituent bodies, or particles, are infinitely rigid, which substantially reduces the computational cost. This assumption is made considering either that the deformations of each body are negligible under the expected load-

ings, or, in the case of particulate systems, that the overall strains of particulate materials are mainly due to the relative sliding and packing between particles and much less due to deformation of individual particles. In the case of particulate systems, such as granular soil, the individual particles are very stiff relative to the mineral skeleton of granular materials, which is usually deformable.

In addition, multibody systems are typically modeled in DEM using a very simple geometric representation of each distinct body, such as spheres, to facilitate fast contact detection algorithms. With these, and several other assumptions, close to real-time interactive simulations can be performed with the current computing capabilities.

Other assumptions include simplified representations of contact effects, which are used, often in the form of idealized springs, to decouple the problem. When two bodies come in contact, forces must be applied to the bodies to push them apart. Two different approaches are used: the hard contact approach and soft contact approach. The decision of which contact representation to use depends on the nature of the problem and the quantities of interest.

When the hard contact approach is used, no interpenetration is allowed while displacement compatibility in the normal directions as well as equilibrium and constitutive law must be satisfied. No contact springs coefficients need to be defined, which is a major problem in the soft contact approach. The collisions are usually assumed to be very brief and they are typically modeled as instantaneous exchanges of momentum. However, this approach is computationally very costly since iterations may be required, and it is difficult to ensure displacement compatibility at all contacts while satisfying equilibrium, constitutive laws, and conserving momentum and energy. In addition, considering the bodies to be infinitely rigid while prohibiting contact overlapping is somehow unrealistic because in reality the colliding bodies have some local deformations at the contact locations.

According to the soft contact approach, which is used in this thesis, a finite normal stiffness is assumed to exist and, in consideration of the actual deformability at the vicinity of the contact, some overlapping of the bodies in contact is allowed. In physical problems overlapping and interpenetrations of bodies do not take place, but surface deformations instead allow these relative movements. The magnitude of the contact forces is assumed to start from zero when the bodies first come in contact and increases as the bodies interpenetrate each other up to a maximum value and then starts decreasing and eventually becomes equal to zero when the bodies detach from each other.

1.2.2 Advantages and limitations

Although standard numerical methods are available to model discontinuities to some extent they are not specifically developed to solve, with reasonable computational cost, assemblages of many interacting discrete bodies. In DEM emphasis is placed on the efficient object representation and fast contact detection. Appropriate simplified assumptions are made to allow simulation of many discrete bodies with sufficient geometric details. However, for numerical simulations of systems with very large numbers of bodies, the main limitation is still the computational cost for contact detection. That cost increases with the number of discrete bodies and the complexity of the geometry of each individual body.

Most discrete element (DE) algorithms make some simplifying assumptions, typically, assuming infinitely rigid bodies with simple geometric shapes, such as spheres, and simple representations of contact effects, to reduce the high computational cost. Although in some cases these assumptions are reasonable and justified by the need to make the problem tractable, there are certain problems for which it would be useful if more accurate mathematical models are used. This thesis addresses the cases in which the deformability of the individual bodies needs to be taken into account. The consideration of the deform-

ability of simulated bodies is achieved by employing a large-displacement finite element methods (FEM) formulation, which is efficiently combined with the DEM.

1.2.3 Classes of DEM

The discrete element methods (DEM) have been introduced in the late '60s early '70s as tools to simulate granular materials (Cundall [5], [6] and [7], and Williams et al [38]). There has been a great interest in these numerical techniques in the past decade, especially during the last few years. Substantial research has been done in the area of contact detection algorithms leading to significant reductions of the required computational time. There are many different classes of DE formulations and computer programs. The main classes of DE formulations are the following:

• **Distinct (or discrete) Element Programs:** Soft contacts are used, while the bodies may be rigid or deformable, (Cundall [5], Williams et al [38]). This is the main DE formulation and the one that is used in this thesis. Direct integration methods, such as explicit numerical schemes, may be used to numerically solve the equations of motion.

• **Modal Methods:** Modal decomposition is employed in these methods to solve the equations of motion (Williams and Pentland [39]). The rigid body motion and strain-displacement equations for each distinct body may be decoupled. Then, the deformability of each discrete body can be expressed in terms of its eigenvectors that correspond to the non-zero frequency eigenmodes. Depending on the desired accuracy the number of included modal contributions may be selected properly. The zero frequency eigenmodes result in rigid body motion, i.e. the motion is decoupled into a rigid body motion and a relative motion that results in internal body deformations. These methods can be used to consider the deformability of discrete bodies in loosely packed systems.

• **Discontinuous Deformation Analysis:** In this approach contacts are considered rigid while bodies may be assumed infinitely rigid or considered deformable (Shi [29] and [30]). Iterations are required to ensure that no interpenetrations between distinct bodies occur. These methods use implicit integration algorithms and need large time steps to be

able to simulate large numbers of interacting bodies. However, large time steps may result in missing contacts or overlaps that may occur over the time step. In addition, in the case of external load with high frequencies, the representation of the externally applied load may be very inaccurate.

• **Momentum Exchange Methods:** These are the simplest methods in which both contacts and bodies are assumed rigid. No penetration is allowed and the collisions are assumed instantaneous. Motion is determined by momentum exchanged between two contacting bodies during an instantaneous collision.

Numerical methods to model materials at finer level than that considered by DEM have been developed and used, such as the cellular automata and the lattice gas methods. These methods can simulate materials at a microscopic, or even molecular, level.

1.3 Applications

Although DEM have been initially developed to simulate fractured rock masses and granular soils, they can be used for many other applications that involve discontinuous, or, in general, multibody systems that undergo large displacements and rotations as well as collisions among the moving discrete bodies. Numerical simulations of multibody systems with graphical visualization capabilities can provide valuable information and insight of the behavior of such systems avoiding actual physical experiments, or costly manufacturing and testing of prototypes. The following are some of the many potential applications for which DEM can be used for numerical simulations.

1.3.1 Granular materials

A granular material is a collection of a large number of distinct particles that are not connected with each other. Such particulate materials are very common in many different areas of nature and engineering. Since there is no generally accepted theory for granular materials, DEM can be used to study them at the particle level subject to external excita-

tions, such as oscillations, providing very useful information about their behavior.

A major category of granular materials is that of granular soils, in geomechanics, whose macroscopic behavior is heavily influenced by interactions between the constituent particles, and particle-level properties, such as size, shape and distribution of individual particles. DEM simulations allow the identification of the effects of microscopic properties to the overall macroscopic behavior, such as the effect of the particle shape on the macroscopic behavior. Numerical simulations can also reveal phenomena that occur at the particle level, but are difficult to be captured in actual physical experiments without disturbing the samples (e.g. Rege [26]). More rational macroscopic constitutive behavior of granular soils may be developed, based on the findings of numerical simulations at the particle level, than those based on continuity assumptions. In addition, DEM may be used to simulate and observe physical phenomena at the particle level, such as the failure process in a slope stability analysis.

DEM can also be used to study the behavior of other granular materials. The manufacturing and transportation of granular materials, such as powders in manufacturing industries, cereal grains in food industries, and tablets in pharmaceutical industries may be simulated with DEM. Phenomena such as the segregation, which often occurs during vibrations of granular materials can also be studied with DEM. Segregation is observed when mixtures containing different size particles are subjected to vibrations, which usually result in rising of large particles and falling of the small particles at the bottom. In addition, flow of granular materials in industrial applications, e.g. through oscillating hoppers, which mainly depends on contacts between the constituent particles, can also be modeled using DEM. Wave propagation through granular materials, which sometimes are used as shock absorbers to isolate sensitive equipment, can also be studied using DEM.

Finally, a defense-related application is the study of a projectile penetration of a missile into granular materials, e.g. sands. A similar application is the simulation of a footing of a structure under dynamic loading. In particular, discrete elements can be employed to model the soil below the footing enabling the simulation of the penetration of the footing in the soil due to dynamic loadings, such as earthquake excitations.

1.3.2 Masonry structures

Analysis tools for masonry structures are very important for the maintenance and restoration of historic structures. The dynamic analysis of masonry structures is a rather challenging problem due to the discontinuities of these structures. Masonry structures, such as those built from stones and bricks, are typically analyzed with very crude empirical rules using estimated static loads to take into account seismic effects. Considering the increasing interest in studying old masonry structures, DEM analysis can capture the discrete behavior of these structures providing a better understanding of their response and their potential failure mechanisms under earthquake excitations. Old masonry buildings were, typically, been built without any earthquake resistant design. In addition, the excessive weight of masonry structures due to over dimensioned walls and high material densities, results in high seismic loads. Finally, non-proper design, e.g. eccentricities and deterioration of the quality of the materials (stone, bricks and mortars) make these structures very vulnerable to earthquake excitations. Therefore, a better understanding of the dynamic behavior of these structures is very important. These structures exhibit a highly nonlinear behavior when excited to earthquake excitations which is very difficult or even impossible to capture with classical methods. Typically, limit analysis is used to perform stability analysis using equivalent static loads.

Although some FEM (using smeared cracks or special "gap elements"), finite difference methods (FDM), and boundary element methods (BEM) models have been used for

simulations of these discontinuous structures, certain difficulties arise mainly due to the continuity approach of these methods. These methods are ineffective in modeling many interacting bodies, especially for dynamic analysis, since they are based on continuity assumptions. The "smeared cracking technique", and the "gap elements" or the "no-tension contact elements" that have been used in FEM have limited capabilities and the analysis often becomes unstable. For quasi-static problems the "limit analysis" has been used for stability studies, but it is limited only for static problems and does not provide any information for the collapse process.

However, equivalent static forces cannot adequately represent the earthquake excitations and the resulting dynamic response of the masonry structures. DEM provide alternative numerical means to simulate masonry structures in order to obtain their dynamic response and their failure mechanisms under extreme events, without the need of building costly laboratory tests or using very simplified static analysis. The dynamic response of these structures involves rocking and impact as well as sliding of the brick or stone blocks which cannot be modeled by any other method.

The structure can be modeled as a system of many distinct bodies put together in the same way that the structure had been physically constructed, and allowed to interact through contact stresses with their adjacent bodies. Each distinct body can be modeled as a single rigid, or deformable, discrete element, while element interactions are modeled using idealized contact and cohesive springs. DEM can be used to more realistically simulate blocks, such as bricks or stones, separated by joints, as well as joint-mortar using cohesion bonds. The opening and closure of joints, crack formations, sliding and rocking can be modeled with the DEM, which is almost impossible with other standard numerical methods. Phenomena such as cracking and separation of parts of the structure into various

smaller parts that vibrate in different ways due to different frequencies, which result in further destruction, can be studied using the DEM.

It is not only masonry buildings that can be analyzed using DEM, but also masonry bridges as well as any kind of masonry structure. DEM can allow the dynamic simulation of these structures under earthquake excitations allowing more accurate and realistic analysis and, in case of failure, the observation and recording of the initiation and progress of the collapse pattern.

The dynamic response of such structures can be easily obtained for many different earthquake excitations that can be applied to the foundations of the structure, providing valuable information for the assessment of the seismic vulnerability of such structures. For problems that are dominated by in-plane behavior, 2D DEM can be used. To consider the flexibility of the individual blocks plane stress or plane strain FEM can be combined with 2D DEM, which is exactly the problem that is addressed in this thesis.

DEM can also used to perform quasi-static numerical simulations of masonry structures subjected to slowly applied lateral loads to get an estimation of the static lateral forces that cause structural collapse. Although limit analysis may be used for static loadings to determine the collapse static load, using DEM more realistic and detail representation of a discontinuous structure can be made. The DEM results may be compared with "limit analysis" results to verify the DEM model and the selected mechanical properties.

In addition to the large displacements and efficient contact effect representations of DEM, the methods also allow a realistic modeling of the mortar joints using "cohesion" bonds combined with certain strength and failure criteria. The degradation of mortars can also be modeled by changing their mechanical properties according to certain criteria during the simulation. Using flexible bodies, it is also possible to consider the stones' or bricks' deformability or even strength allowing fracturing to occur. Finally, it may be pos-

sible to take into account existing reinforcement by incorporating some special structural elements in the simulation combined with failure criteria.

1.3.3 Rock masses

DEM can also be used for simulations of rock formations, such as those studied in rock mechanics. Dynamic stability analysis to consider seismic excitations and other dynamic loadings can be performed using DEM instead of the currently used equivalent static analysis. Then, the dynamic response, including the slip and separation of rocks at points of contact, as well as the failure pattern can be simulated. If the DEM tool has deformable bodies then the deformations and stress distribution of the individual rock bodies can also be computed.

1.3.4 Fragmentation and blasting

One of the first applications of DEM was to study the impact effects of sea-ice flow on offshore structures (Williams et al [40]) and in general the fracture process of ice around the structure. DE formulations have been developed (e.g. Munjiza [17]) that allow the discrete bodies to undergo progressive fracturing resulting in automatic generation of more in number and smaller in size bodies to model the fragmentation process. Fracture criteria need to be defined so as to determine the fracturing of distinct bodies into more than one new distinct bodies. Fragmentation phenomena, which are studied in fracture mechanics, can be simulated using DEM.

Another area where DEM may be useful to perform numerical simulations is that of explosions, blasting in mines (Taylor and Preece [29]), and controlled demolitions of structures. It is very interesting to be able to model explosive fragmentation of an assemblage of bodies, which fracture and break into multiple bodies, and do particle tracking studies. For this kind of problems the distinct bodies have not only large displacements

and rotations, but also large strains and consequently non-linear material behavior, due to the very high load pressures, making the problem much more difficult.

1.3.5 Realistic multibody animations

DEM may also be used to create more realistic multibody animations used in computer graphics, virtual reality, video games and cartoons by incorporating mechanics into the animations. In recent years there is an increased interest in developing realistic physical animations in computer aided design and virtual reality environments, e.g. for educational and training simulations. More realistic multibody animations and more meaningful numerical simulations can be performed by combining engineering-oriented numerical techniques with computer graphics algorithms supplemented with contact detection algorithms.

1.3.6 Multibody dynamic and mechanical systems

There are practical applications that involve both mechanical systems, and large numbers of particles interacting together. Simulations of such systems may be possible by incorporating in DEM simple mechanical system formulations, which allows optimization of the design product prior to the actual manufacturing.

1.4 Literature Review

A detailed literature review of DEM and related procedures is provided by O'Connor [21]. Here, the research done and the contributions made specifically in the Intelligent Engineering Systems Laboratory (IESL) at MIT are briefly described.

Rege [26] developed a computational materials laboratory, named Modeling Interacting Engineering Systems (MIMES), which is a user-friendly two-dimensional discrete element program. MIMES can be used as a simulation environment to perform numerical experiments of planar problems with bodies of arbitrary shapes. Using MIMES, Rege was

able to study granular material behavior in 2D from a particle perspective and to capture detail information of individual particles during simulations. In particular, Rege performed a series of biaxial compression tests in order to study the effect of different microscopic parameters, such as particle size, shape, and distribution, as well as confining stress and loading rates. These tests indicated the presence of coherent, vortex-like, structures, named circulation cells, which may influence the global failure of the specimen. These coherent structures, which essentially are groups of particles that instantaneously translate and rotate as rigid bodies, grow forming eventually failure bands.

O'Connor [21] developed an efficient object representation and contact detection algorithms. In particular, O'Connor developed a scheme, named Discrete Function Representation (DFR), to model complex 3D geometries and facilitate fast contact detection algorithms. He also used parallel computing to distribute the computational work over a number of processors.

Klosek [14] extended the 2D DE program developed by Rege incorporating fluid flow using Finite Elements Methods (FEM) to approximately model the fluid and calculate fluid pressures. Klosek also developed numerical techniques to calculate the fluid forces on arbitrary shape bodies.

Chiou [3] used the Discrete Function Representation (DFR), which was originally developed by O'Connor [21], to simulate large numbers of 3D infinitely rigid bodies. He implemented a hashtable-based spatial reasoning algorithm for similar size bodies to further reduce the computational time for contact detection. He also enhanced the original sequential contact detection algorithm to enable parallel processing.

Cook [4] has studied the fluid flow through granular material using an extended version of MIMES, in which he incorporated the lattice-Boltzmann numerical method in order to be able to direct simulate solid-fluid systems at the grain level. In particular, Cook

has formulated and implemented an accurate, efficient, and robust modeling capability for the direct simulation of solid-fluid systems. Cook used a highly efficient numerical scheme based on the discrete-element (DEM) and the lattice-Boltzmann (LB) methods to solve the coupled equations of motion governing both the fluid phase and the individual particles comprising the solid phase. He used the coupled method, which he had incorporated into MIMES, for simulation and analysis of two-dimensional solid-fluid physics demonstrating its accuracy and robustness over a wide range of dynamical regimes. He was able to reproduce in simulations various fundamental phenomena, including drafting-kissing-tumbling interactions between settling particles, and the saltating transport regime of bed erosion.

1.5 Thesis Objectives

In DEM the constituent bodies of a system are, typically, assumed infinitely rigid in order to reduce the computational cost. However, there are multibody systems for which it would be useful to be able to consider the deformability of selected individual bodies in the numerical simulations. The incorporation of flexibility of individual bodies in the simulation would allow the calculation of the stress distributions as well as the resulting strains in the simulated bodies, which would be very useful in some applications. This is the main issue that is addressed by this thesis.

The deformability of the individual bodies is taken into account using a large-displacements finite element (FE) formulation coupled with the discrete element procedures. The additional computational cost, which is a critical implication of the incorporation of FE into a DE simulation tool, has been carefully addressed taking into account the available computational resources and making proper simplifying assumptions.

A spatial reasoning approach is used in this thesis to reduce the cost of the contact detection and allow simulation of systems with large numbers of bodies of different sizes moving within a non-limited simulation volume. An object representation scheme, which directly extendable in 3D, is used to represent the simulated bodies.

The goal of this thesis and research is to develop and implement simplified efficient numerical procedures that can simulate multibody systems taking into account the deformability and the distribution of stresses within the individual bodies. The following issues are addressed in the thesis:

- development of an efficient contact detection algorithm
- simulation of contact interaction
- efficient modeling of deformable discrete elements
- utilization of object-oriented programming and database technology

In addition, although the actual implementation of the software due to lack of computing resources is limited to the 2D case, the implementation has been done in an extendable program that can be easily enhanced and used for 3D simulations. In particular, all the architecture of the program, such as computer graphics, vectors, methods, have been developed as 3D.

1.6 Thesis Outline

In this chapter, an introduction to the problem that this thesis addresses is presented. First, the motivation for the development and use of discrete element methods is discussed, followed by a general description and some applications of these methods. Then, the research work that has been previously conducted by our group here at MIT, is presented. Finally, the thesis objectives and expected contribution are briefly stated.

Chapter 2 discusses the object representation and modeling, which is selected to enable an efficient contact detection procedure. The contact detection algorithms and procedures are presented in Chapter 3. After a brief introduction to contact detection methods and related issues the selected procedure is discussed in detail. In particular, the spatial reasoning and the contact resolution phases are presented. The chapter concludes with a discussion about the contact forces that are applied during collisions and contacts between the simulated bodies.

Chapter 4 presents the underlying physics of the numerically simulated problem. In particular, the physical laws that govern the motion of the simulated bodies and the simplified assumptions in order to efficiently take into account the contact effects are described.

The major part of this research work is the incorporation of the deformability of the individual bodies, which is discussed in Chapter 5. In particular, a large-displacements/ small strains finite element formulation is coupled with a discrete element procedure. Some simplifications and assumptions were necessary in order to reduce the complexity of the problem and be able to realistically execute numerical simulations with multiple bodies considering the currently available computational resources.

Since the computational requirements are usually very high, it is desirable to use a simple and efficient mesh generation procedure that facilitates fast FE meshing and analysis. This issue is discussed in Chapter 6, which starts with an introduction to grid generation techniques, followed by a description of the main categories of grids. Then, the selected mesh generator, in particular an algebraic mesh generator that generates structure grids, is described in detail

The software design and implementation is presented in Chapter 7, beginning with an overview of the design. The benefits of object-oriented programming are stressed, followed by a description of the structure and the components of the simulation tool that has

been developed. Then, a description of the technology that has been used to develop the software is briefly described. A paragraph of this chapter discusses how database technology can be utilized to efficiently manage the input data and output results.

Chapter 8 provides applications of the software that has been developed, which is named DAFES (Discrete And Finite Element Simulator). Simple verification examples that demonstrate simple phenomena, such as the exchange of momentum, the conservation and dissipation of energy, and wave propagation are presented.

Finally, Chapter 9 provides a summary of the work that has been done, followed by conclusions that have been made. Finally, this chapter concludes with some remarks and suggestions regarding future work.

In addition, three appendices are provided at the end of the thesis with supplementary material. In particular, Appendix A presents the graphical user interface of DAFES, Appendix B discusses how large strains can be taken into account using FEM in a potential extension of this work towards that direction, and Appendix C presents the source code that has been used to compare the performance of C++ and Java.

Chapter 2

Object Representation and Modeling

2.1 Introduction

The selection of a suitable geometric object representation is very important for the simulations under consideration, since it should facilitate both the computationally expensive contact detection and the Finite Element (FE) meshing and analysis. The contact detection process, which is used to identify the bodies that are in contact during simulation, is typically the major computational bottleneck in simulations of multiple infinitely rigid bodies.

Many object representation schemes have been developed and extensively used in computational geometry and computer graphics, such as constructive solid geometry (GSC), boundary representations (B-Rep.), and methods using explicit, implicit and parametric surface representations. GSC objects are described using intersections, unions or subtractions of simpler objects. Implicit, explicit and parametric surfaces are defined by mathematical functions. B-Rep explicitly lists boundary features of objects, such as faces, loops, edges and vertices.

However, many of these representation methods are inefficient either for contact detection, or for FE meshing purposes. Therefore, the object representation should be selected based on its suitability for efficient contact detection checks between the simulated bodies and FE meshing. A compromise may be necessary as the most suitable representation for contact detection may be very inefficient for FE meshing and analysis.

In particular, the geometric object representation scheme should facilitate the contact resolution phase of the contact detection. As it is explained in detail in the following chapter the contact detection that has been selected consists of two phases, the spatial reasoning and the contact resolution. Since for the spatial reasoning module bounding boxes are used, the selection of the object representation should be based on its suitability to facilitate the contact resolution module. The latter checks whether a pair of bodies that are selected by the spatial reasoning are indeed in contact and in that case determines the contact geometry. The contact geometry is used together with the history of the motion of the bodies to compute the contact forces that need to be applied to the bodies in contact.

2.2 Review of previous work

The superquadratic geometric representation and the Discrete Function Representation (DFR), which was originally developed by O'Connor [21], are presented here as an example of an effective representation that has been used in DEM. Although it has not been used in the coupled DE/FE simulation environment that has been developed, it can be used in the future to extent the program in considering bodies with a general shape.

The currently developed software implements the classes and methods that refer to the simple geometric shapes that have been selected, in particular rigid and deformable rectangular bodies. However, the class hierarchy as shown in Figure 2.1, allows easy extensions of the program to consider any other shape as long as the associated methods that are required for the simulation modules, such as the contact resolution, are implemented.

In this research effort, the DFR has not been utilized as the bodies that had been implemented were of simple geometrical shape. However, the developed program is extendable and the DFR is considered to be one of the most promising representation schemes to rep-

resent bodies of more general shape. Therefore, it is presented here for completeness, although more detail information is presented in O'Connor [21].

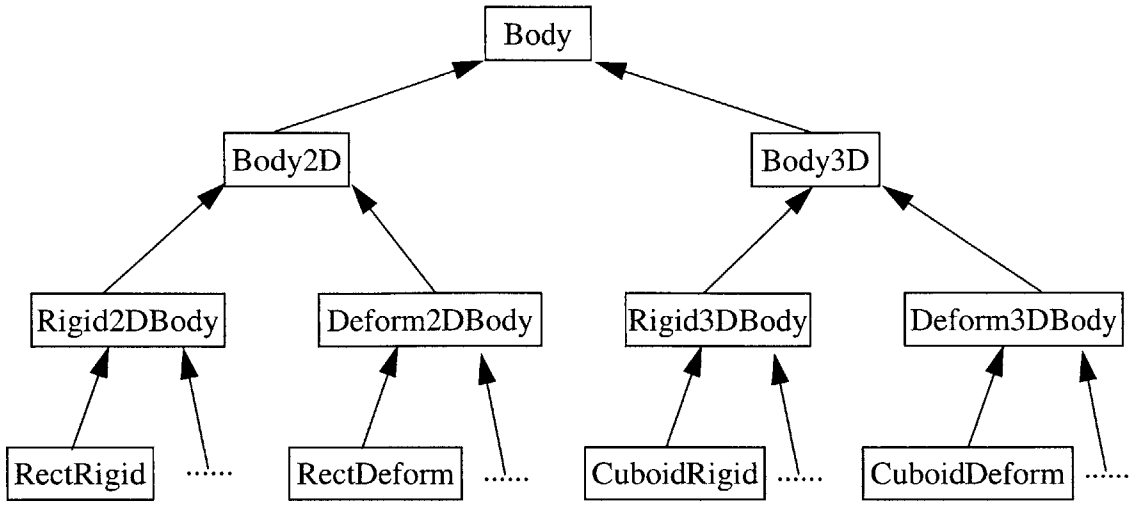


Figure 2.1: Body-related class hierarchy.

2.2.1 Superquadratic geometric representation

The superquadratic representation is an implicit function that can be used to describe the geometry of the simulated bodies. An implicit function, such as the superquadratic, facilitates fast inside/outside tests, since it is easy to determine whether a point lies inside or outside the boundary of the object that is expressed by the function. In addition, the majority of solid geometric objects can be sufficiently well represented by a superquadratic function.

A superquadratic function is an implicit analytical expression that geometrically defines the boundary of a two dimensional (2D) body. Its general, 2D equation has the following form:

$$F(x, y) = \left|\frac{x}{C_x}\right|^{p_x} + \left|\frac{y}{C_y}\right|^{p_y} - 1.0 = 0.0 \tag{2.1}$$

where the coefficients C_x and C_y specify the lengths of the principal axes of the geometric object, and, the exponents p_x and p_y control the shape of the surface.

For example, the family of superquadratics with $P_x = P_y = 2$ corresponds to ellipsoids, while the special case of equal coefficients, e.g. $C_x = C_y$, corresponds to a circle.

Similarly, the three dimensional (3D) superquadratic function can be used to describe the boundary of a 3D geometric object, and its general equation is:

$$F(x, y, z) = \left|\frac{x}{C_x}\right|^{p_x} + \left|\frac{y}{C_y}\right|^{p_y} + \left|\frac{z}{C_z}\right|^{p_z} - 1.0 = 0.0 \tag{2.2}$$

where the coefficients C_x, C_y, and C_z specify the principal axes of the geometric object, and, the exponents p_x, p_y, and p_z control the shape of the surface.

A point can easily be tested whether it lies inside, outside or on the boundary surface of a geometric object. The following cases should be considered:

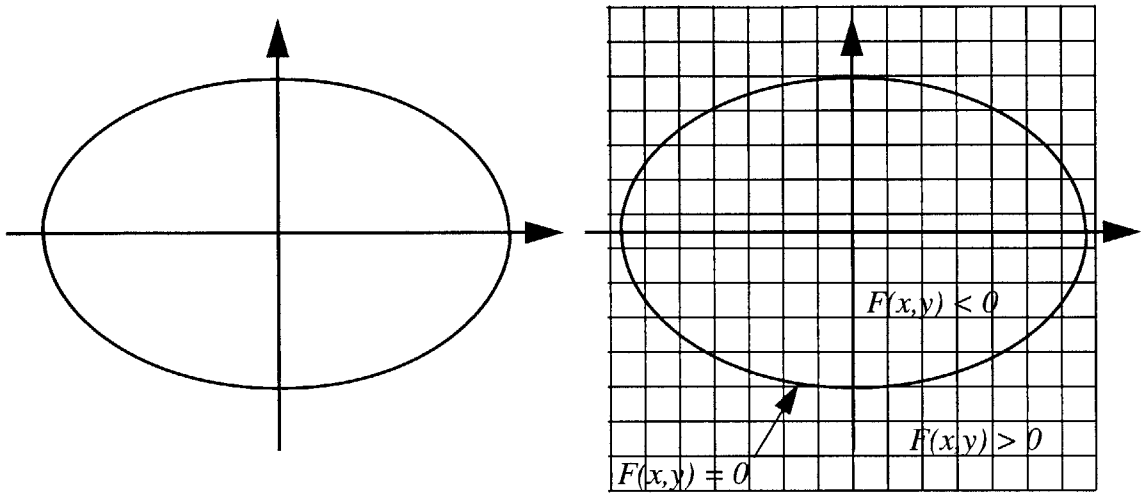
- if $F(x, y, z) > 0.0$, the point (x, y, z) is located outside the object's surface
- if $F(x, y, z) = 0.0$, the point (x, y, z) is located on the surface of the object
- if $F(x, y, z) < 0.0$, the point (x, y, z) is located inside the object's surface

2.2.2 Discrete function representation (DFR)

The Discrete Function Representation (DFR), which was originally developed by O'connor [21], can be used to represent arbitrary geometric objects. It facilitates a very fast contact detection procedure that requires only $O(M)$ computations to perform contact resolution between two bodies that may be in contact.

In particular, a grid of a certain resolution is imposed slicing the object's volume to cubical voxels. Values are, then, assigned at each intersection of the grid using the equation that describes the object's geometry, e.g. the superquadratic equation.

The following two dimensional example demonstrates the use of DFR to represent the geometry of a 2D object. In order to use DFR to represent a 2D object, such as that in Figure 2.2.a, a 2D grid is used to slice the body into a set of squares, as it is shown in Figure 2.2.b. At the intersection points of the grid values of the function that describes the object's geometry are computed and stored to represent the boundary of the object.



a. Object to be represented with DFR b. Use of a 2D grid for DFR

Figure 2.2: Example of 2D DFR.

The boundary of the object can be approximated using the values at the intersections of the grid to compute with interpolation the intersection points of the object's boundary with the squares of the grid.

The boundary of the object is treated as a pair of single-value functions, $y = f_1(x)$ and $y = f_2(x)$. Values of these functions are sampled at a certain uniform resolution dx, as shown in Figure 2.3, using the local coordinate system, and stored.

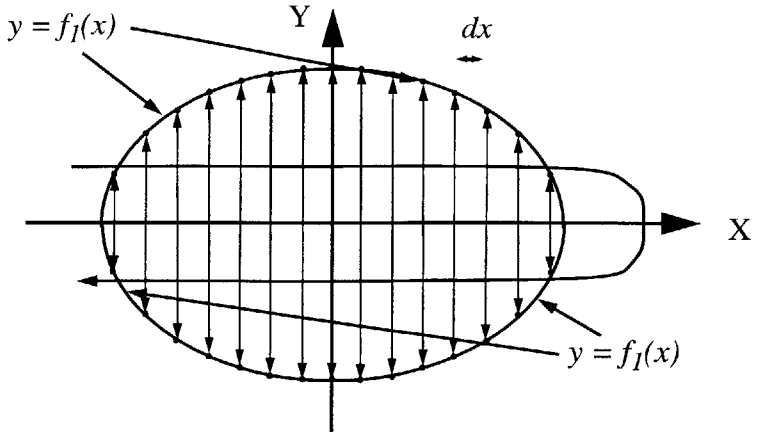


Figure 2.3: Sampling of values for 2D DFR.

The values from the uniformly sampled boundary points are then stored in an array using a direct mapping between the x-coordinate and the storage space, which enables fast retrieval of values, Figure 2.4.

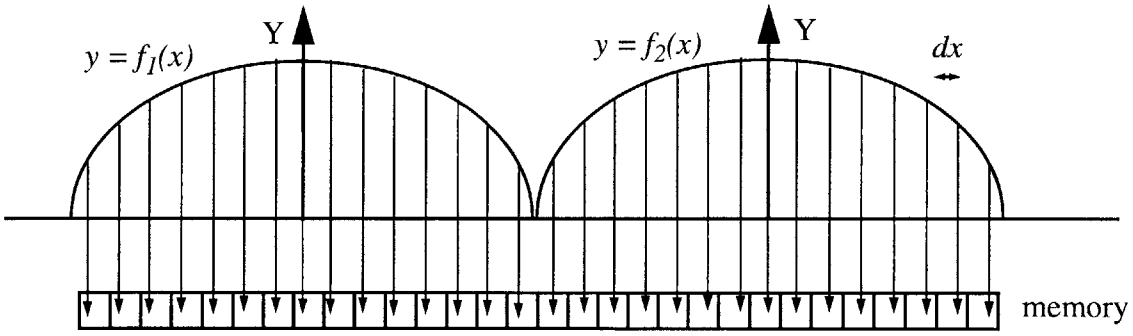


Figure 2.4: Physical-Storage mapping of sampled values.

Having an implicit function, $f(x, y, z) = 0$, a point (x_i, y_i, z_i) corresponds to a value $f(x_i, y_i, z_i)$ of the function. The values of the function can be considered as samples of a scalar potential field that represents the volume of the shape that is defined by the implicit function. This approach is similar to the technique that is used in the Magnetic Resonance Imaging (MRI), where the scalar values measure the material density.

The 3D DFR data structure provides an efficient way of enclosing a simulated body with a set of cells, which is called discrete bounding hull (DBH), that intersect the 3D body. The sampling grid is used as the local coordinate system for each body with unit the cell size.

2.3 Selected Geometric Representation

In general, most DEM use spherical objects as they provide simplicity and efficiency during contact detection. The typical use of spherical objects is also justified by the common encounter of such shapes in simulations of very large numbers of discrete bodies. However, in simulations where there is interest for the stress and strain distributions within individual simulated bodies and the number of bodies is relatively not very large, the most typical shape is polyhedral. In particular, rectangular and box shape bodies are very common in 2D and 3D problems, respectively. Rectangular bodies are the ones selected to be implemented using a essentially a B-Rep representation scheme.

2.3.1 Polyhedral representations: cuboids and rectangular shapes

Polyhedral representations are composed of polygons that have certain relationships and constraints to one another forming solid objects. For example a rectangular shape object in 2D and a cuboidal-shape object in 3D can be represented by their bounding edges and faces, respectively. The objects are described, according to the B-rep., in terms of their bounding entities, such as faces, edges and vertices.

The bounding relations of the cube are based on the six faces, each of which is bounded by four edges, and each of the latter is bounded by two vertices. Similarly, a rectangular object is bounded by the four edges, each of which is bounded by the corresponding vertices.

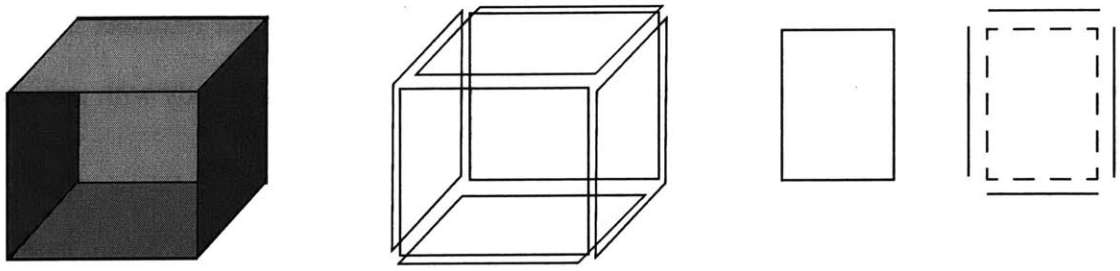


Figure 2.5: Boundaries of a cuboidal and a rectangle.

Geometrical information provides a complete specification of the shape of the object. Some characteristics of the geometrical as well as the topological (such as adjacency relationships) information can be utilized to make the contact detection and resolution more efficient.

Taking into account the type of analysis that is aimed, rectangular shapes have been used in the actual software implementation, although the software is designed in a way that is directly extendable to 3D polyhedral representations. The rectangular, in 2D, and cuboidal, in 3D, shapes are very common in engineering practice. In addition, both shapes represented as B-Rep are very suitable for FE analysis, although they are relatively difficult to be used in DE analysis, in particular during the contact resolution phase of the contact detection. However, as the focus of this research is the coupling of DE and FE analysis using this kind of shapes is a reasonable compromise.

A naive contact detection procedure requires the consideration of all combinations of features, i.e. faces, edges and corners, of each body with every feature of every other object. However, in reality at a specific time instant only a very small number of bodies can be in contact taking into account adjacency relations among the simulated bodies. This issue is considered in Chapter 3 in order to reduce the number of required computations during contact detection.

Chapter 3

Contact Detection and Forces

3.1 Introduction

The major computational bottleneck in simulations of multibody systems is usually the contact detection, since it is the most computationally demanding process (Williams [39], O' Connor [21]). Therefore, the selection of an efficient contact detection algorithm is very critical to reduce the computational cost of a simulation, especially when dealing with very large numbers of bodies. A naive way to perform contact detection is an exhaustive checking of each body against all others without taking advantage of the available information about the shape and spatial distribution of the bodies. This process is computationally very expensive when the number of bodies, N, is large, since it requires an order $O(N^2)$ checks. The cost of an individual pairwise contact detection and resolution depends on the geometry of the bodies under consideration. However, even in the simple case of contact detection between spheres, the $O(N^2)$ required checks may be computationally too expensive when dealing with very large numbers of bodies. Only if an efficient and fast contact detection algorithm is used, sufficiently large number of bodies, or particles, can be simulated. This chapter describes ways that can be used to reduce the computational cost of the contact detection process.

The computational time that is required for contact detection can be substantially reduced if a spatial reasoning is used to avoid an exhaustive check of all possible pairs, which require $O(N^2)$ checks. The spatial reasoning identifies the bodies that may be in

contact avoiding unnecessary pairwise contact detection checks between bodies that cannot be in contact. During the pairwise contact detection, which follows the spatial reasoning, only adjacent bodies that may be in contact are considered. This approach saves a significant amount of computational time reducing the computational cost below $O(N^2)$.

Several methods have been developed to reduce the required operations for the contact detection. These methods can be grouped into two categories, the grid subdivision and the spatial sorting methods. With a grid subdivision algorithm the simulation volume is subdivided into cells using either a uniform or an adaptive grid. For each discrete body the overlapping cells are identified and a pointer is used in each of them to keep track of the cells that each body occupies. Each cell maintains a list of the bodies that fall within or across its boundaries allowing fast determination of possible contacts for each body with its adjacent bodies looking in the cells in its immediate vicinity. However, methods based on this idea have some certain restrictions and limitations. The other category of contact detection methods is based on spatial reasoning of the simulated bodies. In particular, the geometric boundaries of the bodies are used to sort them spatially in certain directions in the simulation space in order to reduce the contact detection checks by excluding bodies that are so far apart that cannot be in contact. This thesis uses the spatial sorting approach to allow simulation of systems with bodies of different sizes moving within a non-limited simulation volume.

3.2 Contact Detection Schemes

The development of efficient contact detection algorithms is useful not only in discrete element simulations but also in many other applications and areas that involve contact detection between parts of the system that is simulated. Examples include CAD/CAM systems, path planning in robotics and industrial processes, molecular dynamics, simulations

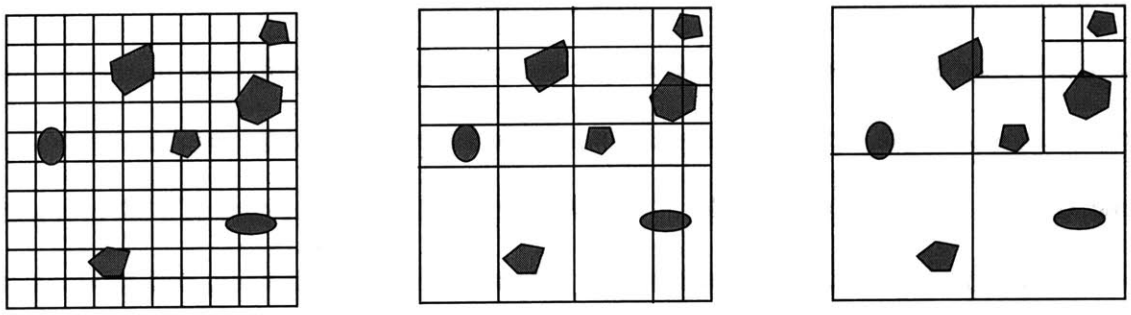
of astrophysics phenomena, computational geometry, physically-based computer graphics and virtual reality environments with collision detection capabilities. Several methods have been developed to reduce the required operations for contact detection.

Many procedures that have been developed and used for contact detection are based on a grid subdivision, also known as space, or cell, decomposition, where the simulation volume is subdivided into cells, using a uniform or, an adaptive, gridding. For each discrete body the overlapping cells are identified and a pointer is used in each of them to keep track of the cells that each body occupies. Therefore, each cell maintains a list of the bodies that fall within, or across, its boundaries. This allows fast determination of all possible contacts for each body with its adjacent bodies by considering all bodies that are marked in the cells at its immediate vicinity. The grid subdivision scheme subdivides the simulation volume, or area, into uniform size cells as shown in Figure 3.1.a for the case of a 2D problem. Each cell may enclose zero or more bodies. The contact detection using cell subdivision can be performed in $O(N)$ operations under certain conditions and restrictions.

However, methods based on the subdivision idea have some limitations. First, the simulation volume must have finite dimensions since there are finite memory resources to store the information about each cell. Therefore, bodies that happen to exit the simulation volume cannot be tracked. In addition, the method performs well only if the simulated system has uniform spatial distribution and the simulated bodies have similar sizes. If there are very small bodies the cell resolution may be so small that the memory requirements may be prohibitively high. The latter limitation may be addressed with an adaptive cell subdivision, as shown in Figure 3.1.b for the 2D case. With this technique the simulation volume is subdivided by planes, or lines, for the 3D and 2D cases, respectively, parallel to the Cartesian axes, in a way that there is the same number of bodies on either side of the

cutting plane, or line. However, adaptive cell subdivision has an additional overhead to maintain information about the adaptive subdivisions and the multiple cell dimensions.

Other procedures that use octrees for 3D problems, or quadtrees for 2D problems (Figure 3.1.c), have been developed and used (Samet [33]). These methods subdivide the simulation volume in cells, which are stored in a tree structure. Only the cells that contain bodies are maintained in the tree structure. However, there is a significant overhead of rebuilding the associated trees at each timestep due to the dynamic nature of multibody dynamics simulations.



a. Grid subdivision b. Adaptive subdivision c. Quadtree subdivision

Figure 3.1: 2D space subdivision schemes.

Considering the above mentioned disadvantages, in particular the substantial memory requirements and overhead to manage and update the cell array at each step, and the usual numbers and types of deformable bodies that need to be simulated in a combined DE/FE analysis, a procedure that uses a spatial reasoning (O'Connor [21]) prior to the actual contact detection resolution is preferred.

3.3 Selected Contact Detection Procedure

The contact detection scheme that has been developed and implemented consists of two phases, the spatial reasoning and the pairwise checks phases. These two phases are illustrated in Figure 3.2.

In particular, a spatial reasoning phase is used to exclude all bodies that cannot be in contact, using a spatial sorting of the bodies, which is based on their bounds in a selected direction, and a spatial searching to identify candidate pairs of bodies that may be in contact. Essentially, the spatial reasoning phase identifies bodies that are good candidates to be in contact. Having identified pairs of bodies that may be in contact, pairwise checks are used to verify the contact between the two bodies and determine the contact geometry during the contact resolution phase.

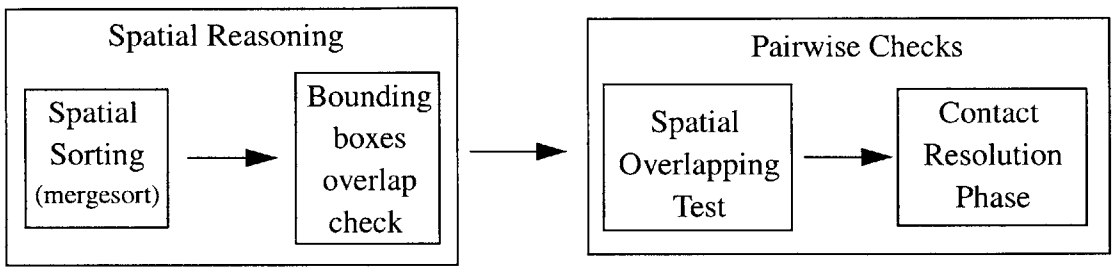


Figure 3.2: Contact detection phases.

The contact resolution phase makes detail contact checks for each selected pair of bodies and determines whether the bodies under consideration are indeed in contact. Whenever the bodies are found to be in contact, the overlapping region is identified and used for the calculation of the contact forces that are applied during the formulation of the equations of motion. The contact resolution phase consists of the pairwise contact verification and resolution phase, as shown in Figure 3.2.

The contact detection procedure must be performed at discrete time instances with a sufficiently small time interval so as to avoid missing any contacts. The time interval should be kept relatively very small to the speeds of the bodies and the dimensions of the problem.

3.4 Spatial Reasoning Phase

The spatial reasoning phase procedure that has been developed and implemented to identify bodies that may be in contact in order to avoid an exhaustive contact detection of every individual body against every other body, consists of a spatial sorting followed by a spatial searching in a selected Cartesian direction. The spatial reasoning reduces the computational cost from $O(N^2)$ to $O(N \cdot lgN))$, which is the order of computations corresponding to the mergesort algorithm that is used for the spatial sorting.

The spatial sorting reduces the number of pairs of bodies that need to be considered for further investigation. The bounding boxes tests reduce even further the number of pairs of bodies that may be in contact. However, there is an overhead associated with the bounding boxes tests, in particular the need to maintain in a data structure the bounding box of each body and the computing cost of sorting of the bounding boxes in each direction.

3.4.1 Spatial sorting: based on bounding boxes

During the spatial sorting phase a mergesort algorithm is used to sort the simulated bodies in a selected direction based on the bounds of the bodies in that direction. The direction that is used for the spatial reasoning can be selected either manually by the user, or automatically by the program. The automatic selection takes into account the spatial dimensions of the system that is simulated and selects the direction that corresponds to the largest size of the bounding box that contains all bodies of the system. With this selection it is more likely to have smaller lists with pairs of bodies that may be in contact during the spatial reasoning phase.

In particular, bodies are sorted at the selected cartesian direction using the mergesort algorithm, which has an $O(N \cdot lgN)$ worst case running time. The sorting is based on the bounds of the body in that direction. Bounding boxes that have edges parallel to the three cartesian axes are determined to facilitate the spatial reasoning phase. Therefore, the sort-

ing of the bodies in the selected direction is done using the lower bound of the bounding box of each body. The sorting process results in a sorted array of references to the simulated bodies, which correspond to the selected cartesian axis.

The cost of sorting is $O(N \cdot lgN)$ in the worst case while in most cases it is much lower since the bodies are almost sorted from the previous step which defers only a very small time increment. Insertion sort, which has been embedded into the mergesort algorithm, is used whenever the size of the set to be sorted is so small that insertion sort is faster than mergesort due to the smaller constants of its computational cost. Although insertion sort has a worst case computational time of $O(N^2)$, the fact that the bodies are almost sorted from one time step to the next results in computational time that is even better than the $O(N \cdot lgN)$ which corresponds the worst case for mergesort. In particular, the computational time for insertion sort for sets that are almost sorted, especially those of small size, that do not require many interchanges approaches from above $O(N)$ computational time. In addition, the storage requirements for the sorting is $O(N)$ since the mergesort algorithm sorts in place without any extra storage requirements.

3.4.2 Spatial searching: identification of potential pairs of bodies

Having sorted the bodies in the selected direction using their bounding boxes, a linear search while identifying bodies that may be in contact is performed. In particular, for each body, all bodies that have minimum bound that is greater than the minimum bound and less than the maximum bound of the body under consideration, are identified. This selection is performed fast by stepping forward in the sorted array of references until a body with a minimum bound that is larger than the maximum bound of the body under consideration is found. For all the bodies from the beginning off the forward stepping until such body is found checks in the other two cartesian directions are performed to exclude the bodies that cannot be in contact.

Therefore, the determination of the range in the sorted arrays that extends between the minimum and the maximum of each body under consideration can be done with a constant number of iterations, which on average is a small number. The whole procedure requires computations of order $O(N \cdot lgN)$ for the spatial sorting, and an order $O(N)$ number of operations to check the projection of the bounds of the selected bodies in the other direction.

3.4.3 Example of 2D spatial reasoning

For simplicity a 2D example is considered to demonstrate the spatial reasoning procedure in one direction. The following figure, which shows bodies in the X-Y plane, demonstrates the spatial sorting and selection to identify bodies that may be in contact. In this case the X-direction has been selected for sorting. First, a sorting based on the minimum boundary of each body in the X direction is performed.

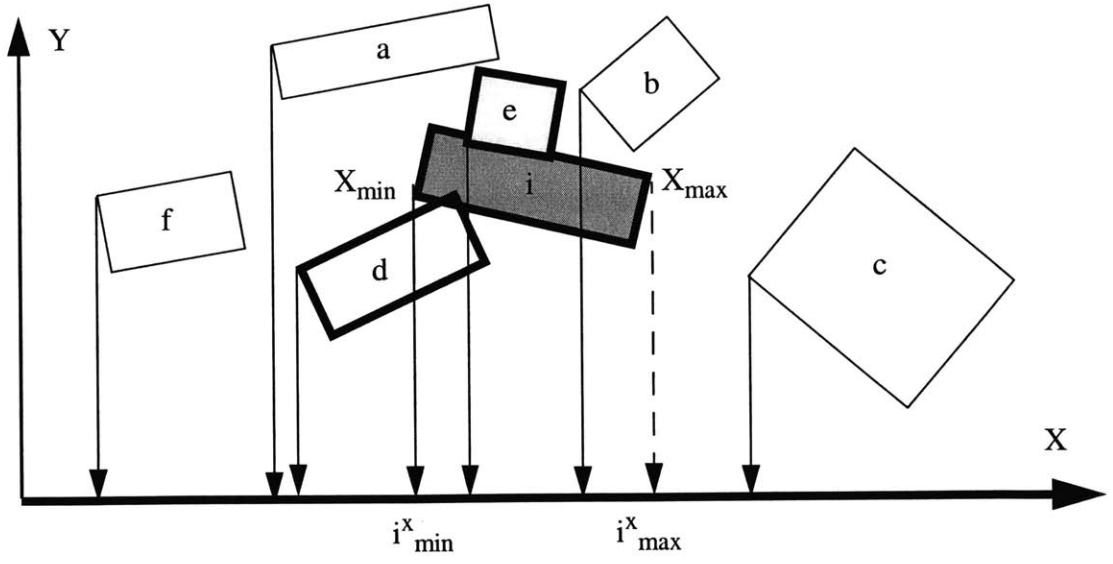


Figure 3.3: Bodies in the 2D X-Y plane.

The results of the sorting is a set of references to the simulated bodies with the sequence that is presented in the following table. The sorting algorithm that has been implemented tool uses mergesort with embedded insertion sort.

Body	f	a	d	i	e	b	c

Table 3.1: Ordering of bodies in X-direction based on their minimum bounds.

Having sorted the bodies in the X-direction using their minimum bound in the X-direction, the spatial search identifies the bodies that may be in contact taking into account the extent of the body under consideration. Each body will be considered in sequence following the sorted order.

In this case, assuming that the body i is under consideration, the algorithm selects the range of bodies, in the sorted bodies that may be in contact with i. The selection of a body is done by comparing its minimum bound whether it lies within the extents of the projection of the body under consideration, body i, on the X-axis. Starting from the references of body i each subsequent body is considered as long as its minimum bound is less than the maximum bound of body i. As soon as a body is found to have a minimum bound larger than the maximum bound of body i the search stops.

Considering the above example, the spatial searching for body i would select bodies e and b, which are the ones that may overlap with i considering the X-direction. The projections of the bounding boxes of bodies a and d in the X-direction also overlap with that of body i, but the pairwise selection of those two pairs is done when bodies a and d, respectively, are considered. The indices i^x_{min} and i^x_{min} keep track of the, intermediate bodies, which in this case are b and e, that may be in contact with i.

The following figure shows the way bodies e and b are selected as candidates to be in contact with body i.

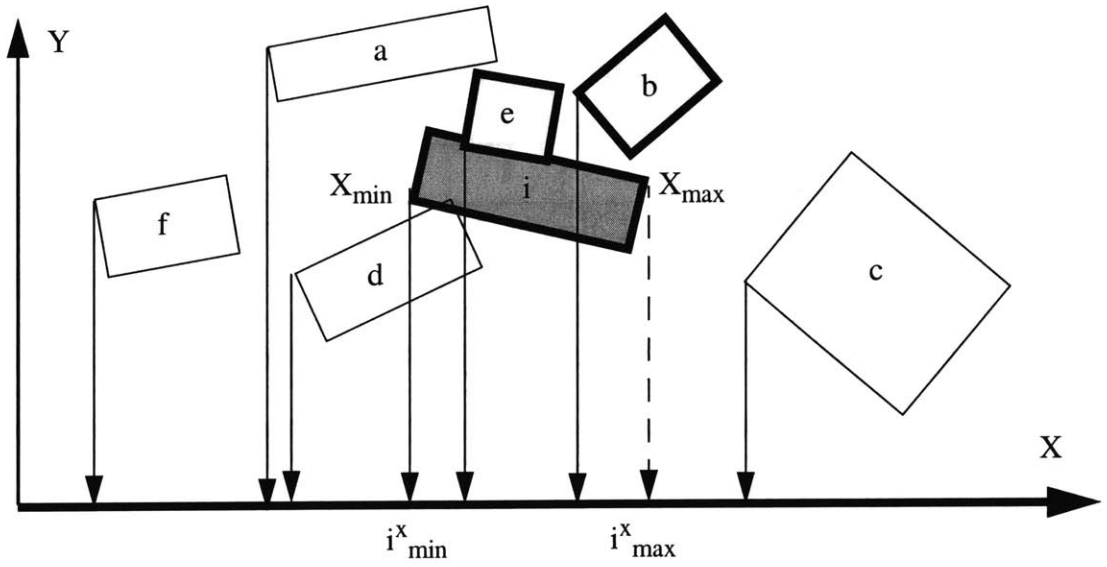


Figure 3.4: Spatial Sorting in the X-direction.

The following table shows the bodies that *e* and *b* are selected from the sorted array of references to bodies, since their lower bound of their projections to the X-axis lies within the bounds of the projection of body *i* on the X-axis.

Body	f	a	d	i	e	b	c

Table 3.2: Selection of bodies that may be in contact with body *i*.

Then, the projections of the bounding boxes of the selected bodies on the other direction axis, i.e. the Y-direction here, have to be checked against the projection of the body under consideration. In this case, the minimum and maximum projections of bodies *b* and *e* on the Y-axis are compared with those of body *i*. This check will exclude body *b* since its lower bound in the Y-direction is larger than the maximum bound of body *i*, which means that bodies *i* and *b* cannot be in contact. The pair of bodies *i* and *e* is kept as a candidate pair to be in contact.

3.5 Pairwise Contact Detection Checks

Having selected candidate pairs of bodies that may be in contact pairwise contact detection checks are performed to verify that the selected bodies are indeed in contact and, in that case, determine the contact geometry. These two operations are performed by the spatial overlapping and the contact resolution phases, respectively, as explained in the following two paragraphs. For each body all potential candidates for contact are identified by the previously described phase, the spatial reasoning.

3.5.1 Spatial overlapping tests

The first phase of the pairwise checks tests whether the bodies are indeed in contact. For the case of rectangular bodies this is performed easily using the local coordinate system of each body to transform the vertices of the other body to its coordinate system and then test whether any of the vertices is located inside the boundaries of the body, which happen to coincide with its bounding box in local coordinates.

As it is shown in Figure 3.5 the vertices of body (2) are transformed in the local coordinate system of body (1), which is located in the center of the latter and aligned to its edges. Therefore, it is very efficient to check whether any of the vertices of body (2) are located within body (1), as it can be done with four inequality checks. If no vertex is found to be located within body (1), then the exact same procedure is performed to check whether any vertex of body (1) is located inside body (2).

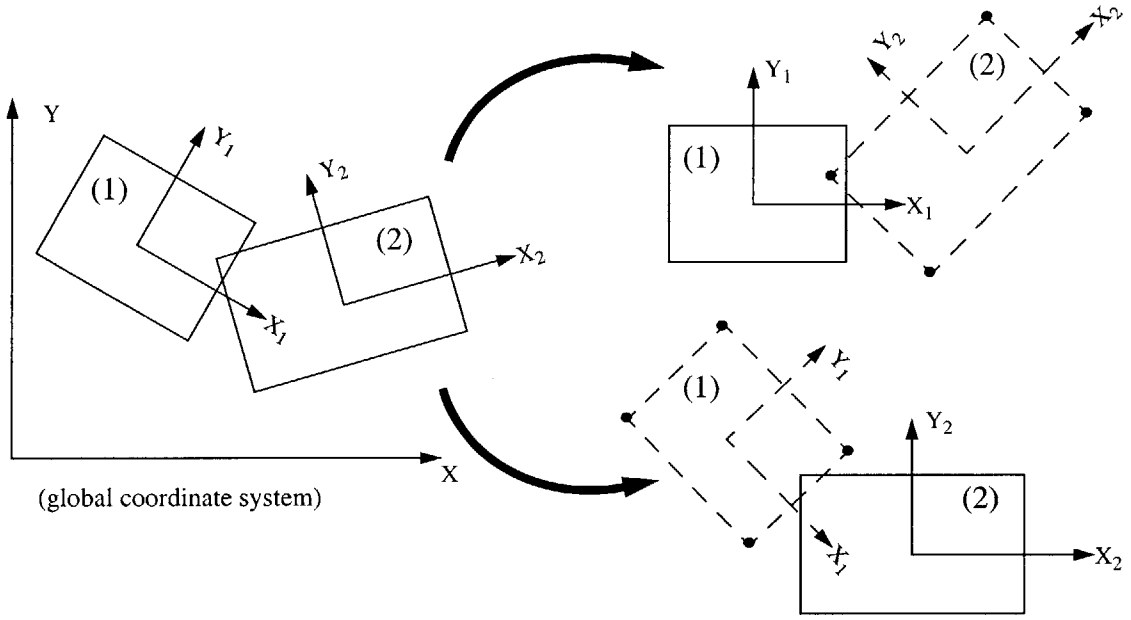


Figure 3.5: Spatial overlapping test between two rectangles.

As soon as any vertex of one body is found to be located within the other body then the two bodies are definitely in contact and the contact detection proceeds to the next phase, the contact resolution, which is described in the following paragraph.

In contrast, if after checking all vertices of the one body against the vertices of the other body and vice-versa, no vertex is found to be within the other body, the two bodies are not in contact and the contact detection between these two bodies terminates.

3.5.2 Contact resolution phase

The second phase of the pairwise checks is the contact resolution, which performs more detail checks between all pairs of bodies that are found to be in contact. The checks in this phase are done for each body using only a small number of adjacent bodies which are selected during the spatial overlapping tests, which in turn use the only the selected by spatial reasoning phase candidate for contact pairs.

3.5.2.1 Contact resolution using sampling points

The contact resolution phase has been developed and implemented specifically for rectangular bodies. Sampling points are used, as shown in Figure 3.6, to determine the overlapping region between two bodies in contact. As in the spatial overlapping tests, presented in paragraph 3.5.1, the local coordinate systems of the two bodies are used together with interpolation functions to determine which sampling points of the one body are located inside the other and vise-versa.

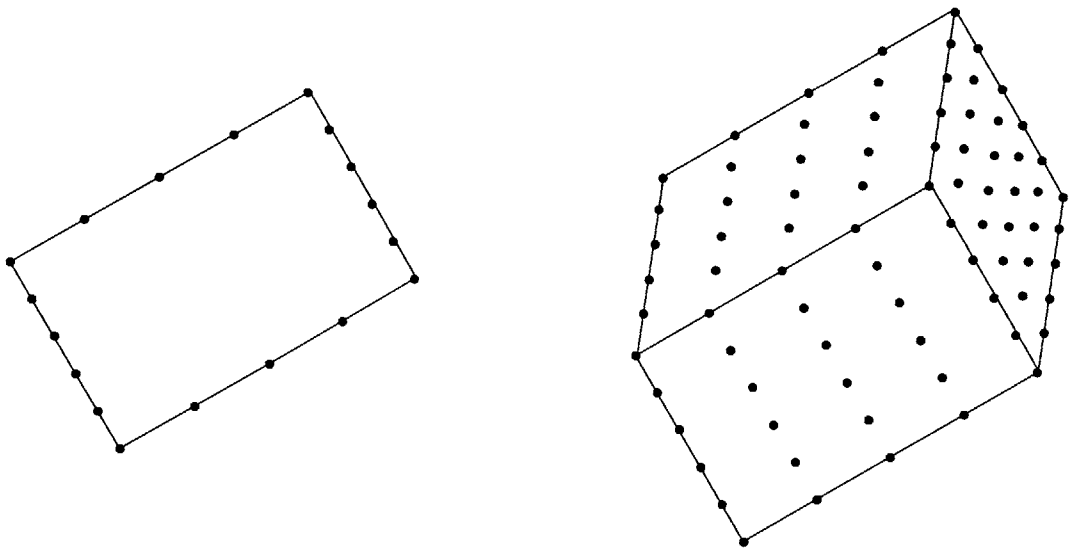


Figure 3.6: Sampling points used for contact resolution.

Whenever a sampling point of a body is found to be within another body a contact point is either formed, when the contact-point is first encountered, or updated when it is a preexisting contact point from previous time steps. At each time step a contact detection is performed and any previous contact points that are not any more within the body-in-contact cease to be contact points and are removed from the data structure that keeps track of the contact points between the two bodies in contact.

3.5.2.2 Normal vectors to contact points

In addition to the determination of the contact points for rectangular shape bodies, it is necessary to determine the normal and tangential vectors to the contact line in order to be able to apply the normal and tangential contact forces as well as to satisfy the Coulomb law of friction. Similarly, in 3D the normal and tangential planes to the contact plane must be determined. The details regarding the applied contact forces are discussed in the following chapter.

For a rectangular body the normal vector to a sampling point is defined as shown in Figure 3.7. All sampling points that are located at the corners of the rectangular object have normal vectors in directions that form 45^{o} angles with extensions of their adjacent edges. All other nodes have normal vectors that are normal to the corresponding edge.

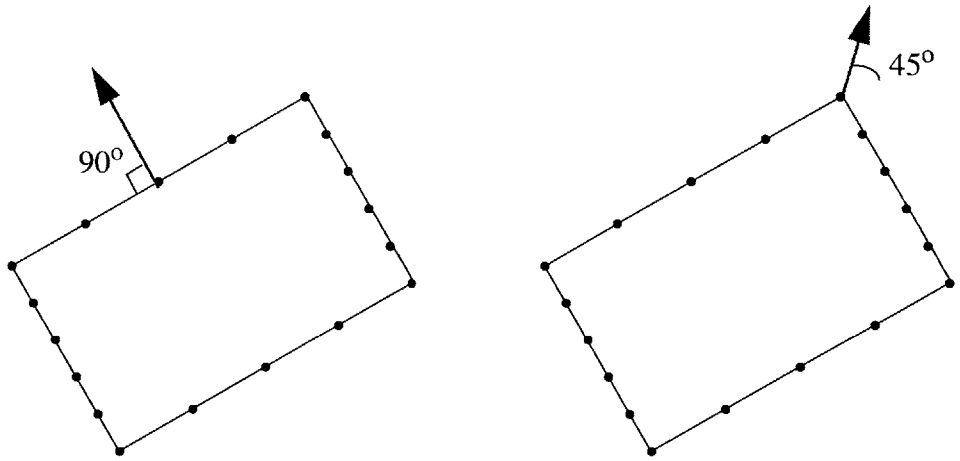


Figure 3.7: Normal vectors at sampling points of a rectangular body.

This definition can be extended for 3D cuboidal objects as shown in Figure 3.8. All sampling points that are located at the corners of the rectangular object have normal vectors in a direction that forms a 45^{o} angle with extensions of its adjacent edges. All sampling points that are located on edges have normal vectors in a direction that forms a 45o angles with extensions of their adjacent faces. All other nodes have normal vectors that are normal to the corresponding face.

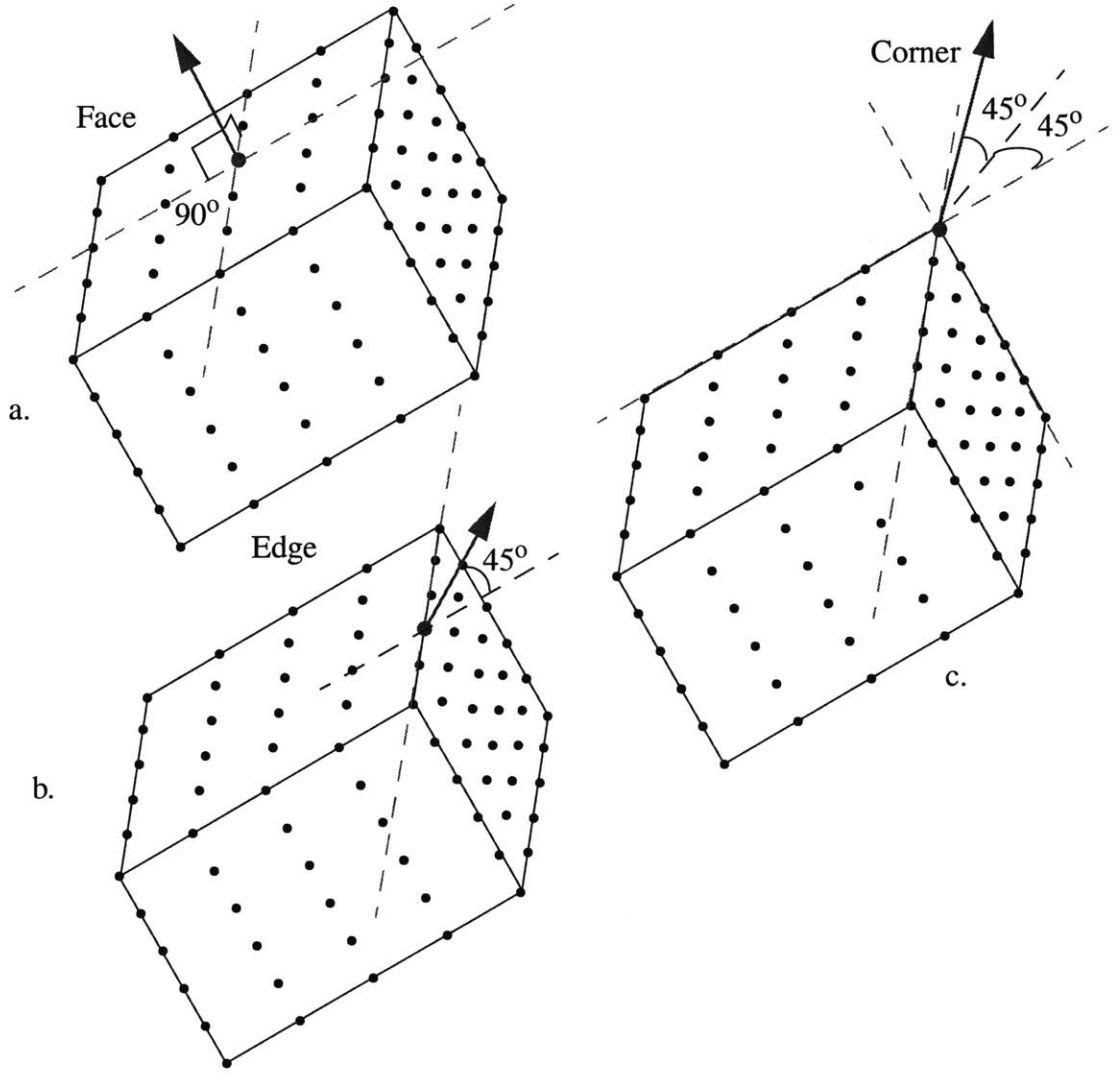


Figure 3.8: Normal vectors at sampling points of a cuboidal body.

3.5.2.3 Normal vectors to contact points

The normal vectors of all sampling points of a body that are qualified to be contact points are taken into account to determine the normal vector to the contact regarding the body under consideration. The normal vector to the contact of a body in contact is determined by summing the normal vectors of all contact points of the body involved in the specific contact and then normalizing the vector. In special cases, such as a contact of an edge of a rectangular body with another body the contribution of the corner sampling point is ignored in order to avoid discrepancy of the normal to the edge.

3.5.2.4 Contact plane

The contact plane must be determined in order to be able to define the normal and tangential directions and use the corresponding normal and tangential relative velocity components and contact coefficients, K_n and K_s, respectively. The contact plane is the plane which is theoretically tangential to the contact points. For rectangular bodies the contact plane is a contact line. The orientation of the contact plane, or line, can be defined by a unit normal vector to that plane, or line, respectively. The normal vector is determined by the features of each of the two bodies that are in contact.

Depending on the points that are determined on a body to be within the area of the other body, we can have one of the two cases:

- a corner contact, which happens when only a corner point is found to be in contact with the other body;
- an edge contact, which happens when one, or more, points on an edge, other than the corner point are found to be in contact with the other body;

Then, having determined for both bodies the kind of contact that each of them experience, there the following three different combinations:

- edge-edge
- edge-corner
- corner-corner

In all three cases the average of the two normal vectors to the contact line, one for each body in contact, are added and then normalized to obtain the normal and tangential vectors to the contact. The latter two vectors are normalized and stored in the contact object where all information regarding the contact between the two bodies is stored. In the next chapter we will discuss how this information is used to compute, or update, the contact forces at each contact point.

Similarly, in 3D problems which involve cuboids, we can have one of the three cases, depending on the points that are determined on a body to be within the volume or area, of the other body, we can have one of the three cases:

- a corner contact, which happens when only a corner point is found to be in contact with the other body;
- an edge contact, which happens when one, or more, points on an edge, other than the corner point are found to be in contact with the other body;
- a face contact, which is the case when one, or more, points on a face that do not belong to any edge are found to be in contact.

Then, having determined for both bodies the kind of contact that each of them experience, there can be any of the following six different combinations:

- face-face
- face-edge
- face-corner
- edge-edge
- edge-corner
- corner-corner

Similar assumptions can be made in 3D for the averaging of the resultant normal and tangential vectors from each body in contact to the contact plane. The computed normal and tangential vectors are required in the calculation of the contact forces that should be applied in the normal and the two tangential directions.

Although the implemented software has been designed and built considering rectangular bodies, it has been carefully designed to allow extension to handle objects of any other shapes as long as the corresponding methods that are required to perform the contact resolution are provided. The spatial reasoning algorithm can be used for objects of any shape

since it uses bounding boxes, but in the contact resolution phase a body of a certain shape can be used only if its corresponding contact resolution function is provided. Currently only the contact resolution functions for simple primitives, rectangle-to-rectangle in particular, have been developed and implemented. Since an object-oriented programming language and design is used, these shape specific functions are polymorphic and, therefore, the program can easily be extended to allow to modelling of any other shape as long the contact resolution function for that specific shapes against all other shapes used in the program are provided.

3.6 Contact Data Structure

Having determined for first time a contact between two bodies, a contact object between the two bodies is constructed. Links to that object are stored in linked lists of contacts that each body preserve. The data structure that is used to store all information relevant to contacts is shown in Figure 3.9.

In particular, as soon as a new contact is detected between two bodies, a contact object is created where all the relevant information is stored, including the contact points of each body, which are stored in two linked lists of contact points, the normal and tangential vectors to the contact, the normal and tangential forces at the contact. Every contact object is then referenced by the two contact lists of the bodies in contact. When a contact is not any more valid it is removed from the corresponding contact lists. Contact lists have been selected and used as they facilitate easy addition and removal of new contacts, which happen very frequently in the simulations under consideration.

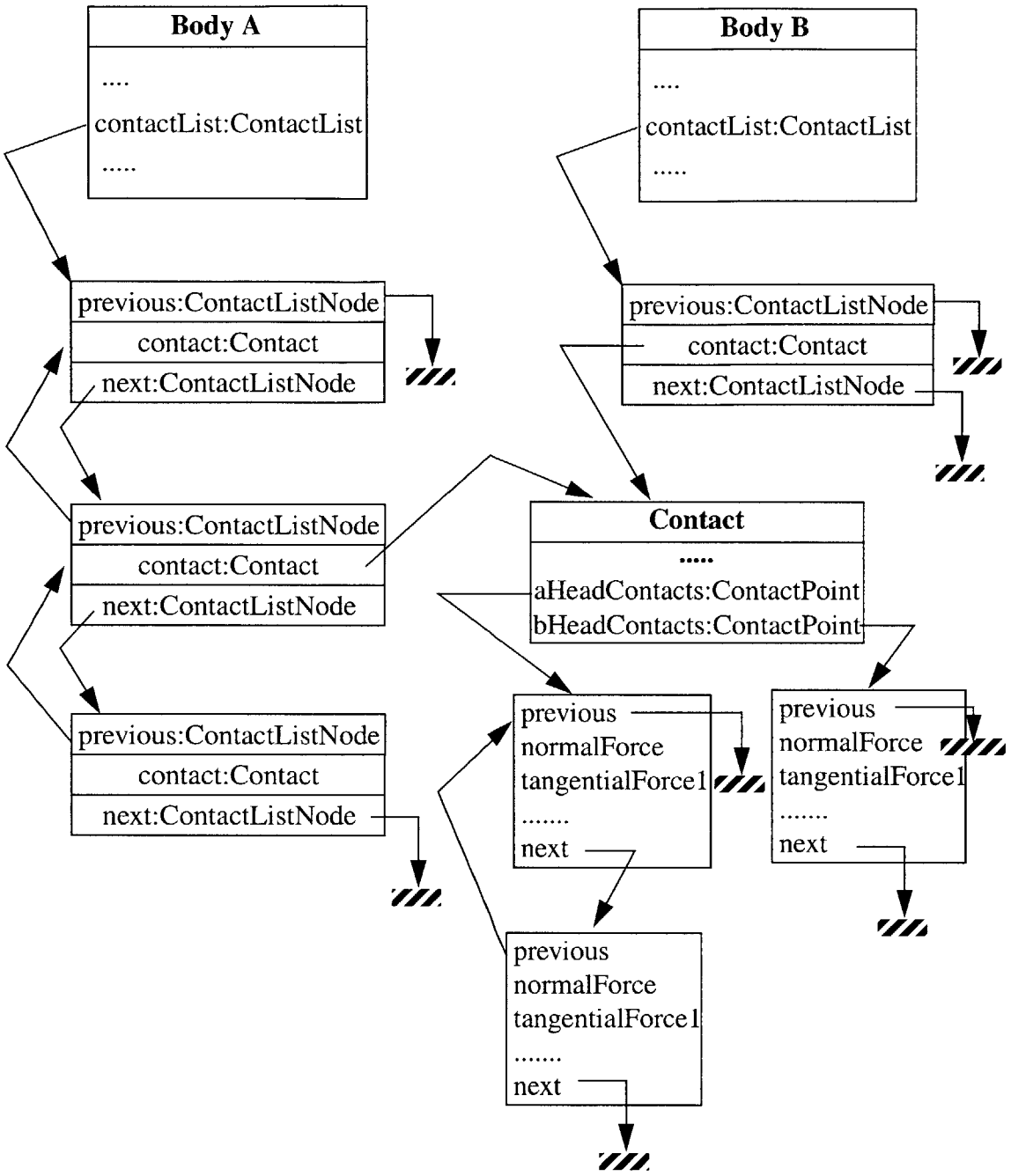


Figure 3.9: Contact storage data structure.

Chapter 4

Physics of the Problem

The individual bodies in discrete element (DE) simulations are, typically, considered to be infinitely rigid. The physics that are used in the simulations regarding contacts between any body that is simulated and the equations of motion of infinitely rigid bodies, i.e. not those directly related to the FE formulation, are described in this chapter.

The following chapter, Chapter 5, presents the methodology that is used to take into account the deformability of at least some simulated bodies using finite element (FE) procedures combined with DEM.

4.1 Introduction

The mathematical model that is used is based on certain assumptions that allow realistic solution of the problem taking into account the limited computing resources, while representing well enough the physical problem. The physical problem is idealized in a form that can be mathematically modeled, as shown in Figure 4.1. Then, the governing equations of the resulting mathematical model are solved providing results that are interpreted and judged in order to draw some conclusions about the physical problem.

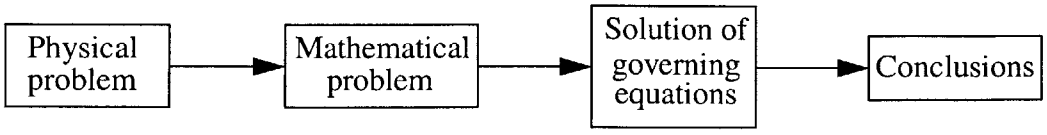


Figure 4.1: Pipeline of analyzing a physical problem through idealizations.

The choice of the mathematical model depends on the phenomena that are to be studied as well as to the provided computing resources. It was deemed necessary to make several assumptions and simplifications due to the complexity of the problem. In this case, the alternative to an attempt to have a more accurate answer would be a no answer. Therefore, the objective is to build a reliable, robust and efficient simulation tool with realistic and reasonable computational demands.

4.2 Contact Effects

4.2.1 Assumptions and simplifications

The contact interaction between colliding bodies is an extremely complicated phenomenon that it would be unrealistic to attempt to simulate without significant simplifications and assumptions. This phenomenon involves stress and strain distributions within the colliding bodies, thermal, acoustical and frictional dissipation of energy due to the contact, as well as plastic deformations. It is evident that with the current state of art in computing one can anticipate to solve only a simplified version of this problem after making certain assumptions in order to make its solution feasible.

The contact effects could also have been taken into account formally as unknowns coupled with the unknown displacements using methods like Langrange multiplier. However, this would have resulted in a huge system of coupled highly nonlinear equations that cannot be solved in reasonable computational time for large numbers of bodies. The formal way to solve contact problems, e.g. using FEM, is to keep the contact surface and forces as unknowns together with the unknown displacements and using Langrange multiplier methods to impose the constraints that characterize the contact problem resulting in a mixed FE formulation. However, such detail analysis which is highly nonlinear due to the

changing contact forces and the large displacements, would have problems to deal with large number of distinct bodies.

Replacing the unknown contact forces at each time step with an estimation based on the previous known positions of each body, allows the decoupling of the equations of motion which can be solved for each discrete body independently in order to compute its new positions. For each time step all external forces including the contact forces according to the current position of the bodies are used. In order to compute the new positions of the bodies, the new contacts are detected and the corresponding contact forces are evaluated to be used in the next time step.

4.2.2 Selected contact approach

The 'soft contact approach' (Goyal et al [12]), which allows the bodies to instantly overlap assuming localized deformability of the colliding bodies, has been used to model the contact effects. Each distinct body interacts with its adjacent bodies that is in contact with interparticle contact forces that are modeled using equivalent normal and tangential springs to simulate the contact effects and provide at each step estimations of the contact forces. The interactions between bodies may involve new contacts, renewed contacts, slippages and complete detachments from other bodies with which are in contact.

Contact forces need to be applied on colliding bodies to simulate the effects of contacts and collisions and prevent interpenetrations of bodies that collide during simulation. Several methods have been developed and used to simulate collisions and take into account contact effects. A penalty function approach is employed here, as it is computationally less demanding than other methods. Having determined the relative motion of bodies in contact during the contact detection procedure, the contact forces are computed, using a penalty approach. The contact forces are applied to the bodies of the simulated system together with the all surface and body forces. Then, a time integration numerical

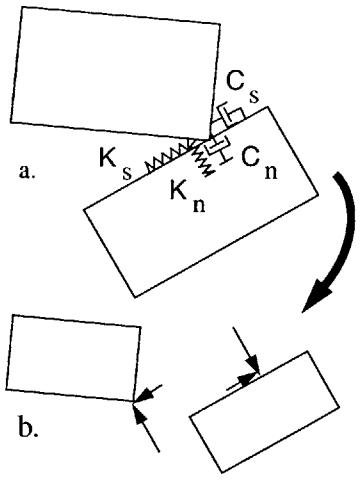
method, which is based on Newtonian mechanics, is used to determine the motion of the simulated bodies under the action of all forces.

At each point that is identified to be a contact point, a pair of equal and opposite contact forces are applied to the two bodies pushing them apart. The magnitude of the force is determined by the relative velocities of the two bodies, the stiffness coefficients and the time step.

4.2.3 Computation of contact force

Contact forces are computed from relative motion of the bodies in contact and the assumed interaction relations, i.e. spring stiffness. These forces are added to the applied external forces, such as the gravity forces. At each time step the contact forces in the normal and the two shear directions are computed using the corresponding stiffness and the relative velocities between the two bodies that collide. The relative velocity between two colliding bodies is determined using the normal and tangential vectors to the contact to which the velocities of the two bodies are projected in order to determine the relative velocities. The increments of the normal and tangential forces are computed, as shown in Figure 4.2, by the product of the relative velocity of the two bodies in the relevant direction, i.e. normal or tangential, times the time step, times the corresponding contact stiffness.

The multiplication of the relative velocity with the time step provides the increment or decrement of the bodies overlap at the contact point. When the bodies first come in contact normal and tangential contact forces are computed. In each subsequent step these normal and tangential forces are updated considering the change of the penetration depth. The bodies eventually are pushed apart due to the action of the contact forces and when there is no overlap between the two bodies the corresponding contact data structure is removed and no contact forces are applied between the two-previously in contact bodies.



Normal forces:

$$^{t+\Delta t}F_n^c = {}^tF_n^{ec} + v_n^{rel} \cdot \Delta t \cdot K_n + v_n^{rel} \cdot C_n$$

Tangential forces:

$$^{t+\Delta t}F_s^c = {}^tF_s^c + v_s^{rel} \cdot \Delta t \cdot K_s + v_s^{rel} \cdot C_s$$

Coulomb Friction: $|F^c{}_s| \leq |F^c{}_n \cdot \mu|$

Figure 4.2: Computation of contact forces.

The normal and tangential contact forces between two bodies in contact are stored at each time step with respect to the associated normal and tangential vectors to the contact, as that is evaluated from the contact detection taking into account all contact points. This is necessary in order to be able to correctly evaluate the contact forces when there is no change in the interpenetration between the two bodies in contact while the bodies rotate. In essence, the contact forces need to be computed and stored in a way that their magnitude is invariable to rigid body rotation of the pair of bodies that are in contact.

When damping is taken into account the increment of the contact forces consists of two parts the elastic and the damping force increments. Therefore, the normal and shear (i.e. the tangential to the contact) forces can be expressed in terms of the elastic (indicated by the *e* superscript) and the damping (indicated by the *d* superscript) force components.

$$\text{Normal force: } ^{t+\Delta t}F_n^c = {}^{t+\Delta t}F_n^{ec} + {}^{t+\Delta t}F_n^{dc} \tag{4.1}$$

$$\text{Shear force: } ^{t+\Delta t}F_s^c = {}^{t+\Delta t}F_s^{ec} + {}^{t+\Delta t}F_s^{dc} \tag{4.2}$$

The magnitudes of the elastic forces are accumulating during simulation.

$$^{t+\Delta t}F_n^{ec} = {}^{t}F_n^{ec} + v_n^{rel} \cdot \Delta t \cdot K_n \tag{4.3}$$

$$^{t+\Delta t}F_n^{ec} = {}^{t}F_s^{ec} + v_s^{rel} \cdot \Delta t \cdot K_s \tag{4.4}$$

In contrast, the damping forces that lead to energy dissipation are equal to:

$$^{t+\Delta t}F_n^{dc} = v_n^{rel} \cdot C_n \tag{4.5}$$

$$^{t+\Delta t}F_n^{dc} = v_s^{rel} \cdot C_s \tag{4.6}$$

where C_n and C_s are the damping coefficients in the normal and tangential direction and which are computed using a user-selected damping ratio x the mass of the bodies and their natural frequencies.

4.2.4 Contact stiffnesses, Coulomb friction and energy dissipation

The magnitudes of the contact forces are expressed as functions of the relative velocity of the colliding bodies, using the corresponding contact stiffnesses. For the computation of the contact forces the contact stiffness is approximately defined using the Hertzian theory. This theory can be used to derive the normal stiffness for the contact between two deformable spheres. Similar measures can be used for the contact stiffness in the other directions.

Considering tangential contact forces, i.e. perpendicular to the normal contact direction, Coulomb friction can be used to limit these shear forces below a certain magnitude taking into account the magnitude of the normal contact force and the coefficients of friction of the two bodies in contact.

Energy dissipation during simulations is considered using dashpots which are used in parallel with the contact springs to simulate viscous damping. The ratio of damping, ξ, is selected by the user and used to derive the damping coefficient from the equation:

$$\frac{c}{m} = 2\xi\omega \quad \Rightarrow \quad c = 2\xi\sqrt{km} \tag{4.7}$$

However, the selection of the damping ratio must be done very carefully because it may affect not only the numerical stability of the time integration scheme, but also the accuracy of the analysis due to overdamping of certain frequencies. Laboratory experiments of single blocks may be performed to obtain realistic values for the contact stiffnesses and damping coefficients and properly calibrate the DEM model.

4.2.5 Application of contact forces on infinitely rigid bodies

Having determined the normal and tangential contact forces at each contact point of a body the forces should be taken into account when forming the equations of motion in order to determine the response of the body. In the case of infinitely rigid bodies the contact forces from each contact point have to be transformed to the centroid of the body with respect the equations of motion are written. As shown schematically in Figure 4.3, the contact forces are evaluated at the centroid of each infinitely rigid body.

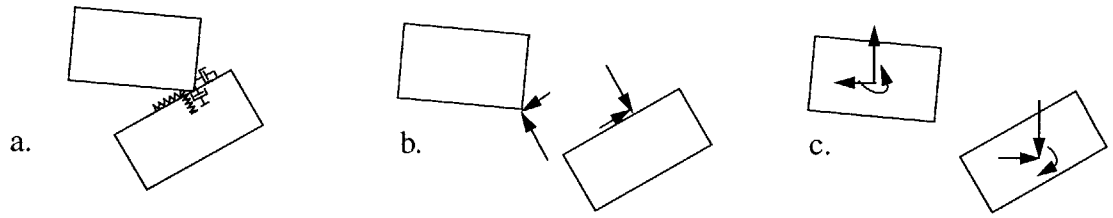


Figure 4.3: Computation of the contact forces resultants.

The computed contact forces are added to all other external force tractions and body forces. Then, the equations of motion for an infinitely rigid body are used to determine its motion as it is described in the following paragraph.

Regarding deformable bodies, as it is described to the next chapter the contact forces are transformed to equivalent nodal forces and the solution of the equations of motion is

performed at the nodal level. Therefore, there is no need to transform the contact forces to the centroid of a deformable body.

4.3 Equations of Motion

Each distinct body is, typically, subjected to gravity and contact forces. The motion of the discrete bodies is based on physical laws, such as the Newtonean mechanics, and can be computed using any of the many time stepping methods. The distinct bodies may undergo large displacements and rotations. An explicit time stepping integration scheme may be used to compute the motion of each body. Finally, the program must automatically and efficiently identify new contacts and complete detachments of bodies that were previously in contact.

Having automatically determine the bodies in contact, the contact forces are calculated, from the current positions and velocities of the bodies, and applied as external surface tractions for the next time step of the solution. The motion of the discrete bodies is governed by physical laws such as the Newton's laws. Then, the equations of motion are solved by time stepping integration methods computing the displacements and, therefore, the updated positions of all discrete bodies. Finally, new contacts between bodies are detected and complete detachments of bodies that were previously in contact are identified, and the corresponding forces are evaluated and used for the next time step.

In particular, after determining the contact and any other external or body forces, the motion of each individual body is computed using the central difference method (CDM), which is an explicit time stepping integration method. The CDM is based on the following approximations for the velocity and acceleration.

$$\dot{u}_{t+\Delta t/2} = \frac{u_{t+\Delta t} - u_t}{\Delta t} \quad \text{and} \quad \dot{u}_{t-\Delta t/2} = \frac{u_t - u_{t-\Delta t}}{\Delta t} \tag{4.8}$$

$$\ddot{u} = \frac{\dot{u}_{t+\Delta t/2} - \dot{u}_{t-\Delta t/2}}{\Delta t} = \frac{u_{t+\Delta t} - 2u_t + u_{t-\Delta t}}{\Delta t} \tag{4.9}$$

Considering the equation of motion in any translational degree of freedom an expression as the one below can be derived for the displacement at time $(t+\Delta t)$.

$$F_t = m \cdot \ddot{u}_t = m \cdot \frac{u_{t+\Delta t} + u_{t-\Delta t} - 2 \cdot u_t}{\Delta t^2} \tag{4.10}$$

$$u_{t+\Delta t} = \Delta t^2 \cdot F_t / m + 2 \cdot u_t - u_{t-\Delta t} \tag{4.11}$$

$$u_{t+\Delta t} = \Delta t^2 \cdot F_t / m + 2 \cdot u_t - u_{t-\Delta t} \tag{4.12}$$

After computing the displacements and rotations of all bodies for each time step their corresponding positions are determined. A new cycle of contact detection, contact resolution, application of forces and solution of equations of motion follows iteratively, based on these new positions, until the end of the simulation.

Chapter 5

Consideration of the Deformability of Simulated Bodies

5.1 Introduction

The aim of the research presented in this chapter is to enable the evaluation of the stress and strain distributions within simulated deformable bodies. In essence, the objective is to take into account the deformability of, at least some of, the simulated bodies, which is the main goal of the research presented in this thesis. The difficulties involved in this research effort are due to the nonlinearities of the problem regarding the contact interactions and the large displacements and rotations of the simulated bodies. Certain simplifications and assumptions are required in order to make the problem tractable for large numbers of simulated bodies.

The deformations of the bodies are assumed to be sufficiently small to permit a small strains analysis. Therefore, according to this assumption, a large-displacements and small-strains analysis can be used. The deformability of individual bodies is taken into account using a displacement-based (DB) updated-Lagrangian (UL) finite element (FE) formulation. An explicit time-integration scheme, specifically the central difference method (CDM), is employed to perform the numerical direct integration of the equations of motion during the dynamic analysis. The combination of the UL formulation with CDM provides very significant simplifications of the problem as it allows the direct integration of the equations of motion without a need for any iterative procedure for convergence, regardless of the large displacements and rotations.

In contrast, the total Lagrangian (TL) FE formulation, which should be used when the strains are large, as described in Paragraph 5.3.1, leads to a coupled system of nonlinear equations. The solution of the latter requires an iterative procedure for convergence at each time step during simulation, rendering the method non-practical to be used for simulations that involve large numbers of deformable bodies with the currently available computational resources.

In the next paragraphs of this section the traditional finite element approach for contact problems is briefly discussed followed by a short outline of the simplified finite element model and formulation that is employed. Then, Section 5.2 presents in more detail the latter. In particular, the relevant continuum mechanics equations, the corresponding FE matrices and isoparametric formulation are presented. The application of the contact forces, the numerical integration of the equations of motion, and issues related to the software implementation of this FE formulation are also discussed. The chapter concludes with a presentation of potential future extensions of this formulation. In particular, an approach based on the TL formulation to enable the consideration of large strains is discussed. Although the formulation that is implemented for this research is based on a small-strains assumption the last paragraph of this chapter and Appendix B briefly describe the approach that can be used in the general case of large strains for a potential extension of the current work.

5.1.1 Traditional finite element formulations for contact problems

The contact effects can be formally taken into account in very simple systems of bodies as unknowns coupled with the unknown displacements using methods like Langrange multipliers method. This is the formal way to solve contact problems using FEM, and it has been applied only in very simple, usually 2D, systems with very few numbers of bodies. The parts of the bodies that may come in contact, typically, have to be defined prior to the

actual simulation, while in the problems under consideration no prior knowledge of the upcoming contacts is available.

The problem is highly nonlinear not only because the simulated bodies may undergo large displacements and rotations, which result in geometric nonlinearities, but also because of the boundary conditions changes due to contact effects during simulations. The contact effects are, typically, taken into account in traditional FE formulations by keeping the contact surface and forces as unknowns coupled with the unknown displacements. Langrange multipliers methods are often used to impose the constraints that characterize the contact problem resulting in a mixed FE formulation with a system of highly nonlinear coupled equations. This huge system of coupled highly nonlinear equations can not be solved in reasonable computational time for large numbers of bodies. Considering the excessive computational requirements, due to the huge number of degrees-of-freedom (DOF), and the high nonlinearities of the coupled system of equations, it is unrealistic to solve problems with many interacting bodies using such classical contact FE approach. Therefore, some approximations to simplify the problem allowing a more efficient way to incorporate FEM in order to enable the simulation of deformable bodies with reasonable computational cost, have been sought.

5.1.2 Simplified finite element model and formulation

Considering the available computing resources and the extremely high complexity of the problem, it is evident that certain simplifications and assumptions are necessary in order to enable simulations of systems of multibody systems. Sometimes, obtaining a less accurate solution is preferable than having no solution.

In particular, the governing equations are decoupled by considering the contact effects through simplified contact springs and the overlapping of the bodies in contact, as it has

been discussed in the previous chapter. This assumption greatly simplifies the problem avoiding the need for solving huge systems of coupled nonlinear equations.

First, the contact forces using the positions of the simulated bodies from the previous time step are determined and used as external surface traction loads. The motion of each discrete body for a new time step is determined from the dynamic equilibrium equations. The equations of motion are solved by an explicit time step integration method computing the displacements as well as the deformations and stresses within each deformable body. Having computed the motion of each discrete body at each new time step, the positions of all discrete bodies are updated and a new contact detection process determines the new contacts and evaluates the corresponding contact forces, which are used in the next time step. In order to perform a FEA an efficient automatic mesh generation module is used, as it is described in detail in Chapter 6, on page 99.

5.2 Finite Element Formulation

The Updated Lagrangian (UL) FE formulation is selected to be used in order to take into account the large displacements and rotations. According to the UL formulation all quantities and variables are referred to the latest computed configuration, rather than the original configuration as in the Total Lagrangian (TL) formulation. Assuming that we know the solution up to time t, we want to determine the solution for time $t+\Delta t$, i.e. the displacements, strains and stresses and, in general, the state of the simulated bodies at the new time instance. This is achieved by employing the UL FE formulation together with the CDM for the time integration of the governing equations of motion that are formed at time t. A lumped diagonal mass matrix and a mass proportional damping matrix are used to decouple the resulting FE equations and further reduce the involved computational cost.

A dynamic analysis provides the nodal displacements under the action of all external forces including the contact forces. From the computed nodal displacements, the body deformations and internal stresses can be computed. For the time integration, the central difference method (CDM) is employed, although a strict stability criterion must be satisfied by its time step, as it substantially simplifies the solution when lumped diagonal mass and damping matrices are used. Then, the system of equations can be solved without matrix factorizations, only by doing simple multiplications. This allows the evaluation of the effective load at each nodal DOF on the element level considering only the elements associated with the specific node.

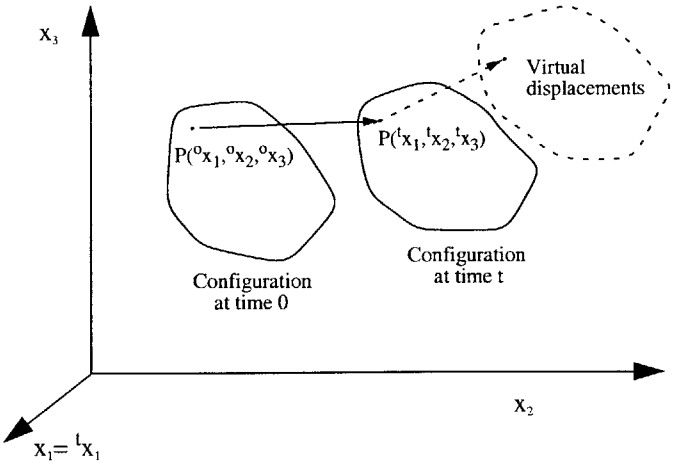
Therefore, no stiffness matrix of the complete element assemblage needs to be constructed and the solution can be carried out locally with very limited high speed memory requirements. This is very important for systems that are typically analyzed with DEM which may have large numbers of bodies. The shortcoming of an explicit method is that the method is conditionally stable and a sufficiently small time step is required to ensure stability. However, regardless of this requirement we need to use small time steps in order to avoid errors during the contact detection part and the calculation of the contact forces. In addition, decoupling the system of equations facilitates the use of parallel computing, which is necessary in order to be able to simulate very large numbers of bodies with complex shapes.

5.2.1 Continuum mechanics equations

According to the PVW: 'the internal virtual work, ${}^{t}\mathcal{f}$, is equal to the external virtual work, ${}^{t}\mathfrak{R}$, for any arbitrary virtual displacements that satisfy the essential, i.e. the displacement, boundary conditions":

$$ {}^{t}\mathcal{f} = {}^{t}\mathfrak{R} \tag{5.1} $$

The following figure shows a general body in its configuration at time *0* (original configuration) and at time *t*. Writing the equations of motion at time *t* the displacements at time $t+\Delta t$ can be computed using an explicit direct integration scheme and stepping forward in time. Figure 5.1 also shows the virtual displacements that can be applied on the configuration at time *t*. Note that the cartesian coordinate axes are stationary, i.e. ${}^{0}X_i = {}^{t}X_i$.



$${}^{t}x_i = {}^{0}x_i + {}^{t}u_i$$

$${}^{t}u_i = {}^{t}x_i - {}^{0}x_i$$

(2D: $i = 1, 2$)

(3D: $i = 1, 2, 3$)

Figure 5.1: Configurations of a moving body at times *0* and *t*.

The internal virtual work, i.e. the work due to internal stresses, can be computed as:

$${}^{t}f = \int_{{}^{t}V} {}^{t}\tau_{ij} \cdot \delta_t\varepsilon_{ij} \cdot {}^{t}dV \tag{5.2}$$

where: ${}^{t}\tau_{ij}$ are the Cauchy stress tensor components at time *t;*

$\delta_t\varepsilon_{ij}$ are the virtual strain tensor components corresponding to the virtual displacements δu_i, which refer to configuration at time *t;*

δu_i are virtual displacements, i.e. variations of the real displacements ${}^{t}u_i$;

${}^{t}x_i$ are the Cartesian coordinates of a point at time *t;*

${}^{t}V$ is the volume of the simulated body at time *t.*

The external virtual work ${}^{t}\Re$ at time t is equal to:

$$
\begin{aligned}
{}^{t}\Re = & \int_{{}^{t}V} {}^{t}f_i^B \cdot \delta u_i \cdot {}^{t}dV + \int_{{}^{t+\Delta t}S_f} {}^{t}f_i^S \cdot \delta^S u_i \cdot {}^{t}dS + \sum_i R_c^i \cdot \delta u_i \\
& - \int_{V} {}^{t}\rho_i \cdot {}^{t}\ddot{u}_i \cdot \delta u_i \cdot {}^{t}dV - \int_{{}^{t}V} {}^{t}\xi \cdot {}^{t}\dot{u}_i \cdot \delta u_i \cdot {}^{t}dV
\end{aligned}
\tag{5.3}
$$

where: ${}^{t}f_i^B$ are the body force components, not including inertia and damping, (force per unit volume);

${}^{t}f_i^S$ are the surface tractions (force per unit area);

R_c^i are concentrated forces;

${}^{t}V$ is the volume at time t;

${}^{t}S_f$ is the surface at time t, where surface tractions are applied;

${}^{t}\rho$ is the mass density

${}^{t}\xi$ is a damping property parameter

$\dot{u}_i$, and $\ddot{u}_i$ are velocity and acceleration components, respectively, at time t;

δu_i, $\delta\dot{u}_i$ and $\delta\ddot{u}_i$ are virtual displacement, velocity and acceleration components, respectively, imposed on configuration at time t.

The inertia and damping forces are formally computed using consistent mass and damping matrices, as shown by the following equations. The consistent mass and damping matrices can be constructed using the same approximations for accelerations and velocities as those that are used for displacements, requiring substantial computations.

${}^{t}\boldsymbol{F}_I = \int_{{}^{t}V} {}^{t}\rho \cdot {}^{t}\ddot{u}_i \cdot \delta u_i \cdot {}^{t}dV$: inertia forces, which resist any change of momentum

${}^{t}\boldsymbol{F}_D = \int_{{}^{t}V} {}^{t}\xi \cdot {}^{t}\dot{u}_i \cdot \delta u_i \cdot {}^{t}dV$: damping forces, resisting motion by dissipating energy

Substituting in the PVW the expressions for the internal and external virtual work we get the following equation.

$$\int_{{}^{t}V} {}^{t}\tau_{ij} \cdot \delta_{t}e_{ij} \cdot {}^{t}dV = \int_{{}^{t}V} {}^{t}f_i^B \cdot \delta u_i \cdot {}^{t}dV + \int_{{}^{t}S_f} {}^{t}f_i^S \cdot \delta^s u_i \cdot {}^{t}dS + \sum_i R_c^i \cdot \delta u_i - \int_{{}^{t}V} {}^{t}\rho_i \cdot {}^{t}\ddot{u}_i \cdot \delta u_i \cdot {}^{t}dV - \int_{{}^{t}V} {}^{t}\xi \cdot {}^{t}\dot{u}_i \cdot \delta u_i \cdot {}^{t}dV \tag{5.4}$$

Rearranging the terms, we get the familiar equations of dynamic equilibrium. These equations express the equilibrium and compatibility requirements of any general body at time t. In addition, assuming that the proper constitutive relations are used, the stress-strain law is also satisfied.

$$\int_{{}^{t}V} {}^{t}\rho \cdot {}^{t}\ddot{u}_i \cdot \delta u_i \cdot {}^{t}dV + \int_{{}^{t}V} {}^{t}\xi \cdot {}^{t}\dot{u}_i \cdot \delta u_i \cdot {}^{t}dV + \int_{{}^{t}V} {}^{t}\tau_{ij} \cdot \delta_{t}\varepsilon_{ij} \cdot {}^{t}dV = \int_{{}^{t}V} {}^{t}f_i^B \cdot \delta u_i \cdot {}^{t}dV + \int_{{}^{t}S_f} {}^{t}f_i^S \cdot \delta^s u_i \cdot {}^{t}dS + \sum_i R_c^i \cdot \delta u_i \tag{5.5}$$

Instead of consistent mass and damping matrices, using diagonal matrices significantly simplifies the solution as there is no need, as it is shown in the following paragraphs, to form any matrices, since the solution for each DOF can be done independently and very efficiently. Therefore, considering the substantial efficiency that is gained, a lumped mass matrix and a mass-proportional damping matrix, which are diagonal, are used instead of the consistent ones.

$$\delta \boldsymbol{U}^T \cdot \boldsymbol{M} \cdot \ddot{\boldsymbol{U}} + \delta \boldsymbol{U}^T \cdot \boldsymbol{C} \cdot \dot{\boldsymbol{U}} + \int_{{}^{t}V} {}^{t}\tau_{ij} \cdot \delta_{t}\varepsilon_{ij} \cdot {}^{t}dV = \int_{{}^{t}V} {}^{t}f_i^B \cdot \delta u_i \cdot {}^{t}dV + \int_{{}^{t}S_f} {}^{t}f_i^S \cdot \delta^s u_i \cdot {}^{t}dS + \sum_i R_c^i \cdot \delta u_i \tag{5.6}$$

where the first two terms, $\delta \boldsymbol{U}^T \cdot \boldsymbol{M} \cdot \ddot{\boldsymbol{U}}$ and $\delta \boldsymbol{U}^T \cdot \boldsymbol{C} \cdot \dot{\boldsymbol{U}}$, which correspond to the inertia and damping forces, are written directly in matrix form as the simpler lumped matrices are used instead of the FE consistent mass and damping matrices.

5.2.2 Finite element matrices

The expression of the PVW in Equation (5.6) can be expressed by the following equation, in matrix form, where the superscript t is dropped since all quantities in this paragraph refer at time t:

$$\delta \boldsymbol{U}^T \cdot \boldsymbol{M} \cdot \ddot{\boldsymbol{U}} + \delta \boldsymbol{U}^T \cdot \boldsymbol{C} \cdot \dot{\boldsymbol{U}} + \int_V \delta\varepsilon^T \cdot \tau \, dV = \tag{5.7}$$
$$\int_V \delta U \cdot F^B dV + \int_{S_F} \delta U_{S_F} \cdot F^{S_F} dS + \sum_i \delta U^i \cdot R_i^c$$

The vector $\delta \boldsymbol{U}^T$ contains the nodal virtual displacements. The vectors $\dot{\boldsymbol{U}}$ and $\ddot{\boldsymbol{U}}$ are vectors that contain the nodal velocities and accelerations at time t;

$\delta U = \left[\delta u \ \delta v \ \delta w\right]^T$ are the virtual displacements at a point (x,y,z);

$U = \left[u \ v \ w\right]^T$ are the displacements of a point (x,y,z) at time t;

$\delta\varepsilon = \left[\delta\varepsilon_{xx} \ \delta\varepsilon_{yy} \ \delta\varepsilon_{zz} \ \delta\gamma_{xy} \ \delta\gamma_{yz} \ \delta\gamma_{xz}\right]^T$ are the virtual strains;

$F^B = \left[F_x^B \ F_y^B \ F_z^B\right]^T$ are the body forces, excluding the inertia and damping forces, which are represented by the first two terms on the left-hand side of Equation (5.7).

Similarly, the F^{S_F}, and R_i^c, are the surface, and concentrate forces, respectively, and

$\tau = \left[\tau_{xx} \ \tau_{yy} \ \tau_{zz} \ \tau_{xy} \ \tau_{yz} \ \tau_{zz}\right]^T$ are the actual Cauchy stresses.

The above integrations must be performed over the deformed volume, V, and surface areas, S_F, of the body at time t, since the displacements are large, as shown in Figure 5.2.

The displacements $U(x, y, z) = \left[u \ v \ w\right]^T$ at time t are assumed known and the aim is to compute the unknown displacements at time $t+\Delta t$. Here, we consider the general 3D case, although the 2D plane-stress and plane-strain are actually implemented, as described

in a following paragraph. The more general 3D case is presented as the general case for the sake of generality.

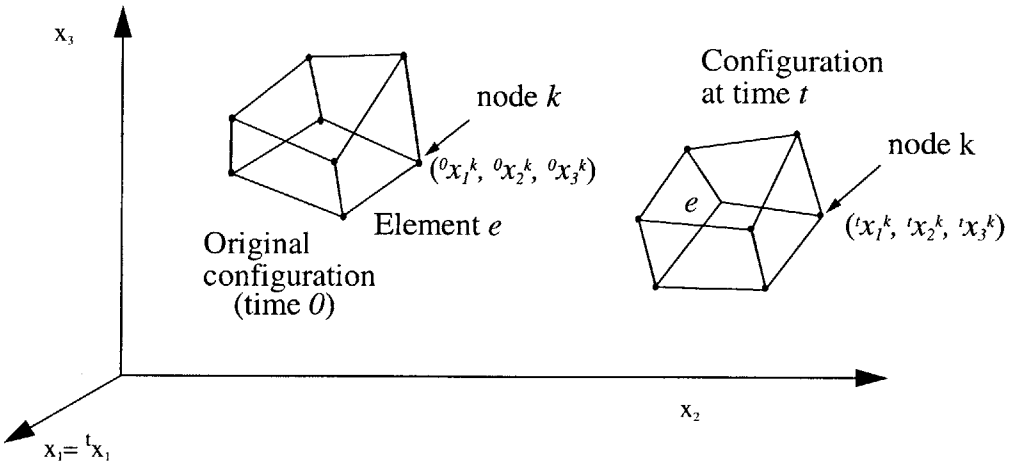


Figure 5.2: Configuration of body at times 0 and t.

A 3D body is discretized into an assemblage of finite elements that are interconnected at nodal points on the element boundaries. The displacements within an element, $\boldsymbol{U}^e$, are expressed in terms of interpolation functions, $\boldsymbol{H}^e$, and the nodal point displacements, $\boldsymbol{U}$.

$$\boldsymbol{U}^e = \begin{bmatrix} u^e(x, y, z) \\ v^e(x, y, z) \\ w^e(x, y, z) \end{bmatrix} = \boldsymbol{H}^e(x, y, z) \cdot \boldsymbol{U} \tag{5.8}$$

The displacements of a point inside an element depend only on the displacements of the nodal points of the particular element. Therefore, the displacement interpolation matrix $\boldsymbol{H}^e$, has nonzero elements only in the columns that correspond to the DOFs associated with the nodal points of that particular element. This fact can be utilized so as to reduce the storing and computing requirements.

The strains can be obtained by proper differentiation of the displacements in terms of a strain-displacement matrix, $\boldsymbol{B}^e$. However, since the displacements are large the usual rela-

tions for infinitesimal strains cannot be used. Instead, the Almansi strains should be used as they are the work-conjugate of the Cauchy stresses. The Almansi strains are given by the following equation:

$$ {}^{A}_{t}\varepsilon_{ij} = \frac{1}{2} \cdot ({}^{t}_{t}u_{i,j} + {}^{t}_{t}u_{j,i} - {}^{t}_{t}u_{k,i} \cdot {}^{t}_{t}u_{k,j}) $$

For example, the Almansi strain ${}^{A}_{t}\varepsilon_{xy}$ in a 3D problem is equal to:

$$ {}^{A}_{t}\varepsilon_{xy} = \frac{1}{2} \cdot \left(\frac{\partial u}{\partial {}^{t}y} + \frac{\partial v}{\partial {}^{t}x} - \frac{\partial u}{\partial {}^{t}x} \cdot \frac{\partial u}{\partial {}^{t}y} - \frac{\partial v}{\partial {}^{t}x} \cdot \frac{\partial v}{\partial {}^{t}y} - \frac{\partial w}{\partial {}^{t}x} \cdot \frac{\partial w}{\partial {}^{t}y} \right) $$

Therefore, the Almansi strains can be expressed in terms of nodal displacements.

$$ {}^{A}\varepsilon^{e} = {}^{A}_{t}\boldsymbol{B}^{e} \cdot \mathbf{U} \tag{5.9} $$

The same assumptions can be used for the virtual displacements and strains:

$$ \delta\boldsymbol{U}^{e} = \boldsymbol{H}^{e} \cdot \delta\boldsymbol{U} \tag{5.10} $$

$$ \delta {}^{A}\varepsilon^{e} = {}^{A}_{t}\boldsymbol{B}^{e} \cdot \delta\boldsymbol{U} \tag{5.11} $$

The stresses can then be computed using the material constitutive law as it is specified by the elasticity matrix, $\boldsymbol{C}^{e}$, which is the same as the one used for small displacements when the material is isotropic and linear elastic.

$$ \tau = \boldsymbol{C}^{e} \cdot {}^{A}\varepsilon^{e} + \tau_{i}^{e} \tag{5.12} $$

where τ_{i}^{e} are any initial stresses.

The stress-strain matrix, $\boldsymbol{C}^{e}$, for a 3D element has the following form. The stress-strain matrices, $\boldsymbol{C}^{e}$, for plane-stress and plane-strain are presented in the Paragraph 5.2.7, where the actually implemented FE formulation is described.

$$C^e = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} (1-\nu) & \nu & \nu & 0 & 0 & 0 \\ \nu & (1-\nu) & \nu & 0 & 0 & 0 \\ \nu & \nu & (1-\nu) & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{(1-2\nu)}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{(1-2\nu)}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{(1-2\nu)}{2} \end{bmatrix} \tag{5.13}$$

The PVW, expressed by Equation (5.7), can be written as a summation of integrals over the subdomains of the finite elements to which the body is discretized, as follows:

$$\delta U^T \cdot M \cdot \ddot{U} + \delta U^T \cdot C \cdot \dot{U} + \sum_e \int_{V^e} \delta^A \varepsilon^{e^T} \cdot \tau^e dV = \tag{5.14}$$
$$\sum_e \int_V \delta U^e \cdot F^{B_e} dV + \sum_e \int_{S_F} \delta U^e_{S_F} \cdot F^{S_F e} dS + \sum_i \delta U^i \cdot R_i^c$$

These integrations can be performed using whatever coordinate system is most convenient for each element. Again the inertia and damping terms have been added for completeness using the corresponding lumped matrices, which are directly assembled without any integration performed to determine them in a consistent way. Using the assumptions for the displacements the corresponding relations are substituted in the PVW:

$$\delta U^T \cdot M \cdot \ddot{U} + \delta U^T \cdot C \cdot \dot{U} + \sum_e \int_{V^e} ({}^A_t B^e \cdot \delta U)^T (C^e \cdot {}^A_t B^e \cdot U + \tau^e_i) dV = \tag{5.15}$$
$$\sum_e \int_V (H^e \cdot \delta U)^T F^{B_e} dV + \sum_e \int_{S_F} (H^e_{S_F} \cdot \delta U)^T F^{S_F e} dS + \sum_i \delta U^T \cdot R_i^c$$

Taking the virtual displacements out of the integrals, the PVW can be written as:

$$\delta U^T \left\{ M \cdot \ddot{U} + C \cdot \dot{U} + \sum_e \int_{V^e} {}^A_t B^{eT} C^e \cdot {}^A_t B^e dV \cdot U \right\} = \tag{5.16}$$
$$\delta U^T \left\{ \sum_e \int_{V^e} H^{eT} F^{B_e} dV + \sum_e \int_{S_F} H^{e\,T}_{S_F} F^{S_F e} dS + \sum_i R_i^c \right\}$$

Applying the PVW as many times as the number of DOFs, using each time virtual displacements that are all zero except one DOF at a time with a unit value we get the following equation, where an identity matrix in front of each side of the equation is canceled out:

$$\boldsymbol{M}\cdot\ddot{\boldsymbol{U}}+\boldsymbol{C}\cdot\dot{\boldsymbol{U}}+\sum_{e}\int_{V^e}{}_{t}^{A}\boldsymbol{B}^{eT}\boldsymbol{C}^{e}\cdot{}_{t}^{A}\boldsymbol{B}^{e}\,dV\cdot\boldsymbol{U}= \tag{5.17}$$
$$\sum_{e}\int_{V^e}\mathbf{H}^{eT}F^{B_e}dV+\sum_{e}\int_{S_F}\mathbf{H}_{S_F}^{e\ T}F^{S_F e}dS+\sum_{i}R_i^{c}$$

The third term on the LHS of the above equation, which represents the elastic forces, can be written as a vector of nodal point forces, due to internal stresses, at time t:

Vector of internal forces: ${}_{t}^{t}\boldsymbol{F}=\sum_{e}\int_{V^e}{}_{t}^{A}\boldsymbol{B}^{e}\tau\,dV$

where ${}^{t}\tau=\left[{}^{t}\tau_{11}\ {}^{t}\tau_{22}\ {}^{t}\tau_{33}\ {}^{t}\tau_{12}\ {}^{t}\tau_{23}\ {}^{t}\tau_{13}\right]^{T}$ is the Cauchy stresses vector.

Defining the body, surface, and concentrate forces vectors as

$$\text{Body forces: } \boldsymbol{R}_B=\sum_{e}\int_{V}\mathbf{H}^{eT}\boldsymbol{F}^{B_e}dV$$

$$\text{Surface forces: } \boldsymbol{R}_S=\sum_{e}\int_{S_F}\mathbf{H}_{S_F}^{e\ T}\boldsymbol{F}^{S_F e}dS$$

$$\text{Concentrate forces: } \boldsymbol{R}_C=\sum_{i}\boldsymbol{R}_i^{c}$$

The external forces, except the inertia and damping forces, can be expressed as:

$${}^{t}\boldsymbol{R}=\boldsymbol{R}_B+\boldsymbol{R}_S+\boldsymbol{R}_C$$

Then, the governing equations of dynamic equilibrium, i.e. the equations of motion, can be expressed in a more compact matrix form:

$$\boldsymbol{M}\cdot{}^{t}\ddot{\boldsymbol{U}}+\boldsymbol{C}\cdot{}^{t}\dot{\boldsymbol{U}}+{}_{t}^{t}\boldsymbol{F}={}^{t}\boldsymbol{R} \tag{5.18}$$

Although the direct stiffness method could be employed to construct the FE equations and matrices of the entire assemblage of elements, for memory efficiency purposes the equations of motion are decoupled and solved working directly at the nodal level consider-

ing only the interactions from the adjacent elements and avoiding the costly construction and manipulation of matrices. The decoupling of the equations is achieved using the UL FE formulation together with the CDM, while assuming diagonal mass and damping matrices.

The PVW is valid even when there are no essential boundary conditions (BCs), i.e. restrained DOF, which characterizes multibody unrestrained systems as the ones under consideration. The latter systems do not have, in general, essential BCs, but only natural BCs, i.e. prescribed boundary forces and moments due to contact effects. In these cases the derived stiffness matrix, $\boldsymbol{K}$, is singular and cannot be inverted to determine the resulting displacements, which is expected since the body is unsupported and unstable. Therefore, it is impossible to use a method, such an implicit direct integration scheme, that requires the inversion of the stiffness matrix, since it is singular, to compute the unknown displacements. Therefore, an explicit integration method should be used as it does not require the construction or inversion of the stiffness matrix.

Instead, the elastic nodal forces that correspond to the internal stresses and are equal to the product of the stiffness matrix with the nodal displacements must be computed.

$$ {}_t\boldsymbol{F} = \boldsymbol{K} \cdot \boldsymbol{U} = \sum_e \int_{V^e} {}_t^A\boldsymbol{B}^{e\ T} \tau \, dV \tag{5.19} $$

5.2.3 Application of contact forces

The contact forces are computed using the procedure described in Chapter 4 and applied as externally applied forces. At each time step, t, the contact detection determines the bodies in contact and then the contact forces are evaluated based on the positions, orientations and geometries of the simulated bodies. These forces are imposed as externally applied forces that are distributed according to the FE interpolation functions to the nodes of the corresponding to the contact point FE of the body. Then, the equations of motion are

solved and the displacements and new positions of the bodies at time $t+\Delta t$ are computed and used for the new contact detection.

Actually, simulating the contact effects with idealized contact springs that oppose overlapping of colliding bodies is a very similar approach to the penalty methods that are used in FEA to impose boundary conditions. In particular, an alternative way to impose prescribed displacements using traditional finite element formulations is, instead of partitioning the FE equations and removing the equations corresponding to the corresponding DOF prior to their solution, to use the penalty method. In particular, constraint equations, which physically correspond to very stiff springs, are added to the FE equations at the prescribed DOFs together with specific loads so as to lead to the prescribed displacements.

5.2.4 Dynamic analysis and numerical integration of equations of motion

The motion of each discrete body for a new time step is determined from the dynamic equilibrium equations. The equations of motion are solved by the central difference method (CDM) an explicit time step integration method, computing the displacements. An explicit integration scheme is selected because a discrete body typically is not fully constrained, in which case it has a singular stiffness matrix. Therefore, an implicit integration method cannot be used as the stiffness matrix cannot be inverted.

In addition, if the mass and damping matrices are selected properly the system of equations which govern the dynamic equilibrium of the body, using CDM, is decoupled and can be solved very efficiently. The solution is substantially simplified when diagonal mass and damping matrices are used, which is a reasonable assumption. Then, the system of equations can be solved without matrix factorizations, but only doing matrix multiplications. This allows the evaluation of the effective load at each nodal displacement on the element level considering only the elements associated with the specific node. Therefore, no stiffness matrix of the complete element assemblage need to be constructed and the

solution can be carried out locally with very limited high speed memory required. This is very important for systems that are typically analyzed with DEM which may have large numbers of bodies. The shortcoming of an explicit method is that the method is conditionally stable and a sufficiently small time step is required to ensure stability. However, a very small time steps is required anyway by the contact detection algorithm in order to avoid errors during the contact detection and the contact forces calculation.

According to Equation (5.18) the governing equations of motion for time t are:

$$^{t}\boldsymbol{M} \cdot {}^{t}\ddot{\boldsymbol{U}} + {}^{t}\boldsymbol{C} \cdot {}^{t}\dot{\boldsymbol{U}} + {}^{t}\boldsymbol{F} = {}^{t}\boldsymbol{R}$$

Using proper finite difference approximations for the velocities and accelerations, the solution for time $t+\Delta t$ can be obtained. The expressions for the velocities and accelerations can be derived by expressing the displacements ${}^{t-\Delta t}\boldsymbol{U}$ and ${}^{t+\Delta t}\boldsymbol{U}$ at times $t-\Delta t$ and $t+\Delta t$, respectively, using Taylor series about time t:

$${}^{t-\Delta t}\boldsymbol{U} = {}^{t}\boldsymbol{U} - \Delta t \cdot {}^{t}\dot{\boldsymbol{U}} + \frac{\Delta t^2}{2} \cdot {}^{t}\ddot{\boldsymbol{U}} - \frac{\Delta t^3}{6} \cdot {}^{t}\dddot{\boldsymbol{U}} + \ldots \tag{5.20}$$

$${}^{t+\Delta t}\boldsymbol{U} = {}^{t}\boldsymbol{U} + \Delta t \cdot {}^{t}\dot{\boldsymbol{U}} + \frac{\Delta t^2}{2} \cdot {}^{t}\ddot{\boldsymbol{U}} + \frac{\Delta t^3}{6} \cdot {}^{t}\dddot{\boldsymbol{U}} + \ldots \tag{5.21}$$

The expression for the velocities can be obtained by subtracting the first equation from the second, while the expression for the accelerations can be obtained by adding the two equations together. In both cases the higher order terms are neglected.

Approximation for velocities: ${}^{t}\dot{\boldsymbol{U}} = \frac{1}{2 \cdot \Delta t} \cdot ({}^{t+\Delta t}\boldsymbol{U} - {}^{t-\Delta t}\boldsymbol{U})$

Approximation for accelerations: ${}^{t}\ddot{\boldsymbol{U}} = \frac{1}{\Delta t^2} \cdot ({}^{t+\Delta t}\boldsymbol{U} - 2 \cdot {}^{t}\boldsymbol{U} + {}^{t-\Delta t}\boldsymbol{U})$

Substituting the above approximations into the governing equations we obtain an expression with the displacements at time $t+\Delta t$, the only unknowns.

$${}^{t}\boldsymbol{M} \cdot \frac{1}{\Delta t^2} \cdot ({}^{t+\Delta t}\boldsymbol{U} - 2 \cdot {}^{t}\boldsymbol{U} + {}^{t-\Delta t}\boldsymbol{U}) + {}^{t}\boldsymbol{C} \cdot \frac{1}{2 \cdot \Delta t} \cdot ({}^{t+\Delta t}\boldsymbol{U} - {}^{t-\Delta t}\boldsymbol{U}) + {}^{t}\boldsymbol{F} = {}^{t}\boldsymbol{R}$$

Solving for the unknown displacements at time $t+\Delta t$:

$$\left(\frac{1}{\Delta t^2}\cdot {}^{t}\boldsymbol{M}+\frac{1}{2\cdot\Delta t}\cdot {}^{t}\boldsymbol{C}\right)\cdot {}^{t+\Delta t}\boldsymbol{U} = {}^{t}\boldsymbol{R}-{}^{t}\boldsymbol{F}-\left(\frac{1}{\Delta t^2}\cdot {}^{t}\boldsymbol{M}-\frac{1}{2\cdot\Delta t}\cdot {}^{t}\boldsymbol{C}\right)\cdot {}^{t-\Delta t}\boldsymbol{U} \tag{5.22}$$

Then, selecting a diagonal mass matrix, by lumping the mass on the nodes, and a mass-proportional damping matrix, instead of using the consistent with the FE formulation matrices, the equations are decoupled and the displacement of each DOF can be computed as a simple fraction of two coefficients.

This time stepping method requires prior knowledge of the initial conditions ${}^{0}\boldsymbol{U}$ and ${}^{0}\dot{\boldsymbol{U}}$ at time 0. Then, the acceleration at time 0 can be determined using the equation of motion at that time:

$${}^{0}\ddot{\boldsymbol{U}} = {}^{0}\boldsymbol{R} = {}^{0}\boldsymbol{M}^{-1}\cdot({}^{0}\boldsymbol{R}-{}^{0}\boldsymbol{F}-{}^{0}\boldsymbol{C}\cdot{}^{0}\dot{\boldsymbol{U}})$$

Also a procedure to determine the displacements at time t-Δt, ${}^{-\Delta t}\boldsymbol{U}$, is needed. Knowing ${}^{0}\boldsymbol{U}$, ${}^{0}\dot{\boldsymbol{U}}$, and ${}^{0}\ddot{\boldsymbol{U}}$, we may use the approximate expressions derived previously to obtain an expression for the displacements at time t-Δt.

$${}^{0}\dot{\boldsymbol{U}} = \frac{1}{2\cdot\Delta t}\cdot({}^{\Delta t}\boldsymbol{U}-{}^{-\Delta t}\boldsymbol{U}) \Rightarrow {}^{-\Delta t}\boldsymbol{U} = 2\cdot\Delta t\cdot{}^{0}\dot{\boldsymbol{U}}+{}^{\Delta t}\boldsymbol{U}$$

$${}^{0}\ddot{\boldsymbol{U}} = \frac{1}{\Delta t^2}\cdot({}^{\Delta t}\boldsymbol{U}-2\cdot{}^{0}\boldsymbol{U}+{}^{-\Delta t}\boldsymbol{U}) \Rightarrow {}^{-\Delta t}\boldsymbol{U} = \Delta t^2\cdot{}^{0}\ddot{\boldsymbol{U}}-{}^{\Delta t}\boldsymbol{U}+2\cdot{}^{0}\boldsymbol{U}$$

$${}^{-\Delta t}\boldsymbol{U} = {}^{0}\boldsymbol{U}-\Delta t\cdot{}^{0}\dot{\boldsymbol{U}}+\frac{\Delta t^2}{2}\cdot{}^{0}\ddot{\boldsymbol{U}} \tag{5.23}$$

The new coordinates, which are needed not only for the contact detection but also for the FEA on the next time step, are obtained by adding to the current coordinates the computed displacements.

5.2.5 Isoparametric FE formulation

In order to obtain the FE formulation we need to employe interpolation functions to express the displacements of any point within a finite element e, in terms of the nodal

point displacements. The isoparametric FE formulations use the same interpolation functions for both the displacements and the coordinates within the element, i.e. relating the original with the natural coordinates of the element.

Coordinate interpolations: at time *0*: $^{0}x_i = \sum_{k=1}^{n} h_{k(r,s,t)} \cdot {}^{0}x_i^k$,

and, at time *t*: $^{t}x_i = \sum_{k=1}^{n} h_{k(r,s,t)} \cdot {}^{t}x_i^k$

Displacement interpolations: at time 0 : $^{0}u_i = \sum_{k=1}^{n} h_{k(r,s,t)} \cdot {}^{0}u_i^k$,

and, at time *t*: $^{t}u_i = \sum_{k=1}^{n} h_{k(r,s,t)} \cdot {}^{t}u_i^k$

where: n is the number of nodes per element (which is 8 for an 8-node brick element)

$i = 1, 2, 3$ is the direction of the stationary coordinate system XYZ

r, s, t are the axes of the natural coordinate system RST (with values from -1 to +1), and $h_i = h_i(r, s, t)$ are the interpolation functions. e.g. for an 8-node FE the following interpolations are used, which can be, systematically, determined by inspection.

$$h_1 = \frac{1}{8} \cdot (1 + r) \cdot (1 + s) \cdot (1 + t) \qquad h_2 = \frac{1}{8} \cdot (1 - r) \cdot (1 + s) \cdot (1 + t)$$

$$h_3 = \frac{1}{8} \cdot (1 + r) \cdot (1 - s) \cdot (1 + t) \qquad h_4 = \frac{1}{8} \cdot (1 - r) \cdot (1 - s) \cdot (1 + t)$$

$$h_5 = \frac{1}{8} \cdot (1 + r) \cdot (1 + s) \cdot (1 - t) \qquad h_6 = \frac{1}{8} \cdot (1 - r) \cdot (1 + s) \cdot (1 - t)$$

$$h_7 = \frac{1}{8} \cdot (1 + r) \cdot (1 - s) \cdot (1 - t) \qquad h_8 = \frac{1}{8} \cdot (1 - r) \cdot (1 - s) \cdot (1 - t)$$

The expressions in this paragraph refer to a general 3D FE, although the developed software implements plane-stress and plane-strain FE. The expressions for the latter elements which are provided in Paragraph 5.2.7.

The displacements of a point, within an element e, can be expressed in matrix form, as follows, in terms of the element nodal point displacements:

$$\mathbf{U}^{(e)}(x, y, z) = \mathbf{H}^{(e)}(r, s, t) \cdot \mathbf{U}(x, y, z) \tag{5.24}$$

where $\boldsymbol{H}^{(e)} = \boldsymbol{H}^{(e)}(r, s, t)$ is the interpolation matrix, (for the entire FE assemblage), and has the following form:

$$\boldsymbol{H}^{(e)} = \begin{bmatrix} 0 \;..\; h_1\; 0\; 0\; h_2\; 0\; 0\; h_3\; 0\; 0\; h_4\; 0\; 0\; h_5\; 0\; 0\; h_6\; 0\; 0\; h_7\; 0\; 0\; h_8\; 0\; 0 \;..\; 0 \\ 0 \;..\; 0\; h_1\; 0\; 0\; h_2\; 0\; 0\; h_3\; 0\; 0\; h_4\; 0\; 0\; h_5\; 0\; 0\; h_6\; 0\; 0\; h_7\; 0\; 0\; h_8\; 0 \;..\; 0 \\ 0 \;..\; 0\; 0\; h_1\; 0\; 0\; h_2\; 0\; 0\; h_3\; 0\; 0\; h_4\; 0\; 0\; h_5\; 0\; 0\; h_6\; 0\; 0\; h_7\; 0\; 0\; h_8 \;..\; 0 \end{bmatrix},$$

and, $\boldsymbol{U}$ are the nodal displacements: $\boldsymbol{U} = \begin{bmatrix} u_1\; u_2\; u_3\; \ldots\ldots\; u_{3 \cdot N} \end{bmatrix}$ (N is the total number of nodes).

The velocities and accelerations can also be interpolated differentiating the expression for the displacement interpolations. The same interpolation functions are also employed for the virtual displacements.

$$\delta \boldsymbol{U}(x, y, z) = \boldsymbol{H}(r, s, t) \cdot \delta \boldsymbol{U} \tag{5.25}$$

Differentiating properly the displacements, as they are expressed in terms of the interpolation functions, we can obtain expressions for the incremental strains. The strains are partial derivatives with respect to the local coordinate system, i.e. x, y, and z, of the displacements, which are expressed in terms of the interpolation functions in terms of natural coordinates, r, s, and t.

Therefore, the Jacobian, $\boldsymbol{J}$, that relates the derivatives with respect to the natural coordinates to the derivatives with respect to the local coordinates need top be established.

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{bmatrix} \tag{5.26}$$

The inverse relationships between the derivatives with respect to the local and natural coordinates need to be established, as well.

$$\begin{bmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \\ \frac{\partial}{\partial t} \end{bmatrix} = \boldsymbol{J} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} = \boldsymbol{J}^{-1} \begin{bmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \\ \frac{\partial}{\partial t} \end{bmatrix} \tag{5.27}$$

The Jacobian, can be inverted when all angles are less than 180^{o}, i.e. if there is a unique correspondence between the natural and local coordinate systems. This is always true for the kind of elements used in this work.

The integrations over the actual volumes of the elements can be replaced by integrations using the natural coordinates by changing the limits of integration from -1 to 1 and the differential volume by: $dV = dxdydz = |\boldsymbol{J}|drdsdt$.

5.2.6 Numerical integration

The integrations required in order to form the FE matrices are evaluated numerically. In particular, the Gauss quadrature, which is based on sampling the values of the functions to be integrated at certain points, is used. The positions and weights are selected in a way that maximizes the accuracy of the numerical integration. A simple integral is numerically evaluated as

$$\int_{x_a}^{x_b} f(x) = w_1 f(x_1) + w_2 f(x_2) + \ldots + w_N f(x_N) \tag{5.28}$$

The Gauss quadrature with N function evaluations evaluates accurately a polynomial of order $(2N-1)$. In contrast, Newton Codes, e.g. the Simpson's rule, with N function evaluations can evaluates accurately a polynomial of order $(N-1)$. The positions and weights for the Gauss quadrature are presented in the following table.

N	Position	Weight
1	0.0	2.2
2	$\pm 0.57735027 = \pm 1/\sqrt{3}$	1.0
3	± 0.77459667 0.00	0.555555555556 0.888888888889

Table 5.1: Sampling points and weights for Gauss Quadrature

This numerical integration method can be extended and applied in multidimensional integrals. In particular, two points in each direction can be used to accurately evaluate the integrals in order to form the FE matrices.

5.2.7 Implementation of the FE formulation

The FE formulation is implemented for the cases of plane-stress and plane-strain. However, the software has been designed and implemented according to an object-oriented paradigm enabling its easy extension to other types of FEA. Here, the specific FE matrices that correspond to the plane-stress and plane-strain are presented, assuming that we have an FE, as the one shown in Figure 5.3.

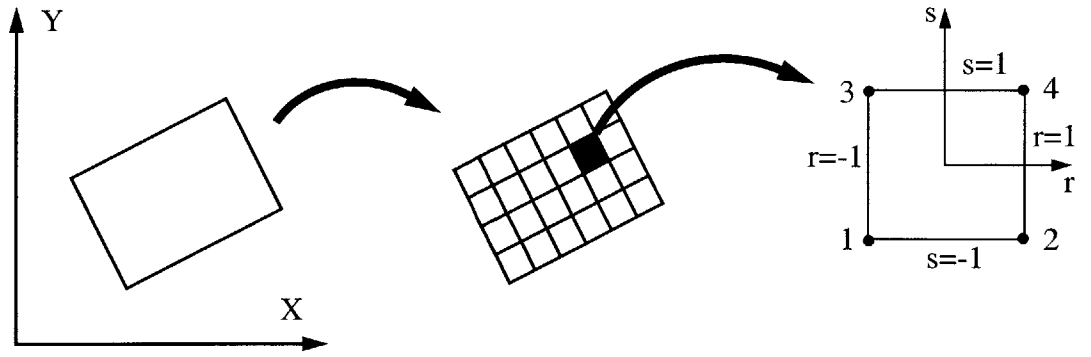


Figure 5.3: Isoparametric 2D FE.

The interpolation functions are presented below:

$$h_1 = \frac{1}{4} \cdot (1-r) \cdot (1-s) \qquad h_2 = \frac{1}{4} \cdot (1+r) \cdot (1-s) \tag{5.29}$$

$$h_3 = \frac{1}{4} \cdot (1-r) \cdot (1+s) \qquad h_4 = \frac{1}{4} \cdot (1+r) \cdot (1+s) \tag{5.30}$$

Then, the displacements within an element e can be expressed in terms of the nodal displacements.

$$\boldsymbol{U}^{(e)}(x, y) = \boldsymbol{H}^{(e)}(r, s) \cdot \boldsymbol{U}(x, y) \tag{5.31}$$

The interpolation matrix for an element e has the following form:

$$\boldsymbol{H}^{(e)} = \begin{bmatrix} h_1 & 0 & h_2 & 0 & h_3 & 0 & h_4 & 0 \\ 0 & h_1 & 0 & h_2 & 0 & h_3 & 0 & h_4 \end{bmatrix}, \tag{5.32}$$

The Almansi strains are provided by the following expressions:

$${}^{A}_{t}\varepsilon_{xx} = \frac{1}{2} \cdot \left(\frac{\partial u}{\partial {}^{t}x} + \frac{\partial u}{\partial {}^{t}x} - \frac{\partial u}{\partial {}^{t}x} \cdot \frac{\partial u}{\partial {}^{t}x} - \frac{\partial v}{\partial {}^{t}x} \cdot \frac{\partial v}{\partial {}^{t}x} \right)$$

$${}^{A}_{t}\varepsilon_{yy} = \frac{1}{2} \cdot \left(\frac{\partial v}{\partial {}^{t}y} + \frac{\partial v}{\partial {}^{t}y} - \frac{\partial u}{\partial {}^{t}y} \cdot \frac{\partial u}{\partial {}^{t}y} - \frac{\partial v}{\partial {}^{t}y} \cdot \frac{\partial v}{\partial {}^{t}y} \right)$$

$${}^{A}_{t}\varepsilon_{xy} = \frac{1}{2} \cdot \left(\frac{\partial u}{\partial {}^{t}y} + \frac{\partial v}{\partial {}^{t}x} - \frac{\partial u}{\partial {}^{t}x} \cdot \frac{\partial u}{\partial {}^{t}y} - \frac{\partial v}{\partial {}^{t}x} \cdot \frac{\partial v}{\partial {}^{t}y} \right)$$

Therefore, the Almansi strains can be expressed in terms of nodal displacements and the strain-displacement matrix.

$$ {}^{A}\varepsilon^{e} = {}^{A}_{t}\boldsymbol{B}^{e} \cdot \boldsymbol{U} \tag{5.33} $$

The strain-displacement matrix has the following form:

$$ {}^{A}_{t}\boldsymbol{B}^{e} = \begin{bmatrix} {}_{t}h_{1,1} & 0 & {}_{t}h_{2,1} & 0 & {}_{t}h_{3,1} & 0 & {}_{t}h_{4,1} & 0 \\ 0 & {}_{t}h_{1,2} & 0 & {}_{t}h_{2,2} & 0 & {}_{t}h_{3,2} & 0 & {}_{t}h_{4,2} \\ {}_{t}h_{1,2} & {}_{t}h_{1,1} & {}_{t}h_{2,2} & {}_{t}h_{2,1} & {}_{t}h_{3,2} & {}_{t}h_{3,1} & {}_{t}h_{4,2} & {}_{t}h_{4,1} \end{bmatrix} \tag{5.34} $$

The stress-strain, i.e. the elasticity, matrix is provided below for both cases:

$$ \text{Plane stress: } \boldsymbol{C}^{e} = \frac{E}{(1-\nu^{2})} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{5.35} $$

$$ \text{Plane strain: } \boldsymbol{C}^{e} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \tag{5.36} $$

Then, the stresses are expressed in terms of the Almansi strains.

$$ \tau = \begin{bmatrix} \tau_{xx} \\ \tau_{yy} \\ \tau_{xy} \end{bmatrix} = \boldsymbol{C}^{e} \cdot \begin{bmatrix} {}^{A}_{t}\varepsilon_{xx} \\ {}^{A}_{t}\varepsilon_{yy} \\ {}^{A}_{t}\varepsilon_{xy} \end{bmatrix} \tag{5.37} $$

The Jacobian matrix and its inverse for a 2D element have the following form.

$$ \boldsymbol{J} = \begin{bmatrix} \frac{\partial\, {}_{t}x}{\partial r} & \frac{\partial\, {}_{t}y}{\partial r} \\ \frac{\partial\, {}_{t}x}{\partial s} & \frac{\partial\, {}_{t}y}{\partial s} \end{bmatrix} \tag{5.38} $$

$$\boldsymbol{J}^{-1} = \frac{1}{|\boldsymbol{J}|} \cdot \begin{bmatrix} \frac{\partial\, {}^{t}y}{\partial s} & \frac{\partial\, {}^{t}y}{\partial r} \\ \frac{\partial\, {}^{t}x}{\partial s} & \frac{\partial\, {}^{t}x}{\partial r} \end{bmatrix} \tag{5.39}$$

The derivatives with respect to the x and y coordinates can be expressed in terms of the derivatives with respect to the r and s coordinates.

$$\begin{bmatrix} \frac{\partial}{\partial\, {}^{t}x} \\ \frac{\partial}{\partial\, {}^{t}y} \end{bmatrix} = \frac{1}{|\boldsymbol{J}|} \cdot \begin{bmatrix} \frac{\partial\, {}^{t}y}{\partial s} & \frac{\partial\, {}^{t}y}{\partial r} \\ \frac{\partial\, {}^{t}x}{\partial s} & \frac{\partial\, {}^{t}x}{\partial r} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial r} \\ \frac{\partial}{\partial s} \end{bmatrix} \tag{5.40}$$

The nodal forces due to internal stresses ${}^{t}_{t}\boldsymbol{F}$ at time t is computed using the above relations and a numerical integration to evaluate the following integral.

$${}^{t}_{t}\boldsymbol{F} = \sum_{e} \int_{V^e} {}_{t}\boldsymbol{B}^{e\ T}\, \tau\, dV = \sum_{e} {}^{t}_{t}\boldsymbol{F}^{e} \tag{5.41}$$

A 2x2 Gauss quadrature is used for the numerical integration. Therefore, for each element e, its contribution to the nodal forces ${}^{t}_{t}\boldsymbol{F}$ is evaluated by the sum of the product of the value of the quantity under the integral, ${}^{A}_{t}\boldsymbol{B}^{e\ T}\tau$, at the specific integration point, (r_i, s_j), times the corresponding weight W_{ij}.

$${}^{t}_{t}\boldsymbol{F}^{e} = \sum_{i=1}^{2} \sum_{j=1}^{2} {}_{t}\boldsymbol{B}^{e\ T}\tau \cdot |\boldsymbol{J}|\Big|_{\text{at } (r_i, s_j)} \cdot W_{ij} \tag{5.42}$$

At each time step t, the external nodal forces vector ${}^{t}\boldsymbol{R}$, including contact forces, is formed. In particular, the contact forces that are computed at the contact points, which are computed at the contact points, are converted to equivalent nodal forces that are added to any other external forces, such as gravity forces, that are also lumped to the nodes of the elements.

The equations of motion have the following form

$$ {}^{t}\boldsymbol{M}\cdot{}^{t}\ddot{\boldsymbol{U}}+{}^{t}\boldsymbol{C}\cdot{}^{t}\dot{\boldsymbol{U}}={}^{t}\boldsymbol{R}-{}^{t}_{i}\boldsymbol{F} \tag{5.43} $$

The solution is determined of the equations of motion is computed using the CDM, as described in Paragraph 5.2.4. The displacements at time $t+\Delta t$ are equal to:

$$ {}^{t+\Delta t}\boldsymbol{U}=\frac{1}{\left(\frac{1}{\Delta t^{2}}\cdot{}^{t}\boldsymbol{M}+\frac{1}{2\cdot\Delta t}\cdot{}^{t}\boldsymbol{C}\right)}\cdot\left\{{}^{t}\boldsymbol{R}-{}^{t}_{i}\boldsymbol{F}-\left(\frac{1}{\Delta t^{2}}\cdot{}^{t}\boldsymbol{M}-\frac{1}{2\cdot\Delta t}\cdot{}^{t}\boldsymbol{C}\right)\cdot{}^{t-\Delta t}\boldsymbol{U}\right\} \tag{5.44} $$

Some certain issues must be considered in order to achieve an efficient and robust implementation of this formulation and simulation model. Special attention is required for the use of memory in order to avoid swapping. Secondary storage memory access is several orders slower than the RAM and it is a very potential bottleneck during simulation.

5.3 Potential Future Extensions

There are many potential extension that can be considered for future projects in this research area. The main extensions are the consideration of large strains, which is a physical extension to this work, and the incorporation of fracturing and fragmentation capabilities, which has many practical applications.

5.3.1 Large strains formulation

An issue of concern is whether the deformations would be considered large so that they should be considered in the contact detection algorithms, or whether the initial geometry would be used.

When large strains are expected the small stains assumption is not valid. Then, a different formulation and a much more computationally intensive nonlinear FEA are required. The total Lagrangian formulation can be used to take into account the large strains as well as the large displacements. The exact description of this formulation and

procedure is provided in Bathe [1]. Appendix B provides a brief presentation of the total Lagrangian approach that can potentially be used to take into account the case of large strains in a future extension of this work is outlined for completeness.

5.3.2 Fracturing and fragmentation capabilities

Fracturing and fragmentation capabilities can be very useful for simulating many problems such as blasting. To incorporate progressive fracturing capabilities in the coupled DE/FE model, some fracturing criteria must be defined according to which a discrete body will be fractured when its stress state satisfies these criteria. An efficient remeshing algorithm is required for the two or more new discrete bodies that a previously single distinct body is fractured.

Chapter 6

Mesh Generation

6.1 Introduction

The stress and strain distributions within the colliding multibody systems can be expressed in terms of continuum mechanics partial differential equations that can be solved numerically using a finite element analysis (FEA) as it is described in the previous Chapter. FEA can determine approximate values of the desired solutions at certain points, called nodes, in the domain of interest. Then, values of the quantities of interest can be estimated using interpolation functions.

The first step in a FEA is the generation of an appropriate mesh by discretizing the domain under consideration. Mesh, or grid, generation is the process with which the physical domain of an engineering problem, is broken into smaller subdomains, by defining nodes and their connectivity within the volume of a body that specify the finite elements. A mesh provides the necessary framework in order to obtain a numerical solution of the engineering problem doing a FEA of the discretized system.

Although the implemented software uses a very simple mesher, a brief discussion regarding meshing is provided in this chapter for completeness as the implemented code can be extended to consider other more geometrically complex objects.

The individual elements of a FE mesh must be well shaped without either very small or obtuse angles, and conform to the boundary of the domain. The mesh should be fine enough in order to achieve sufficient accuracy, while the number of elements should be

limited to enable a FEA with a reasonable computational cost. A generated mesh should be conformal. The conformity property is satisfied if the union of all elements is the domain under consideration or a good approximation of it, the intersection of any two elements reduces to an empty set, a point, an edge, or a face, and, all elements have a non-empty interior.

It is desirable to have an automatic and robust mesher that is able to generate a mesh of sufficient quality with a reasonable computational cost. The latter increases with geometrical complexity. The desired quality of a mesh depends upon the purpose of the application of interest, the underlying physics, and the geometry of the body.

It is relatively easier to generate a mesh for a convex region, or volume. A region is convex if when it contains two points it also contains the entire line segment that joins the two points. In this thesis only convex bodies are considered.

There are two major classes of grids, the structured and the unstructured grids which are described below. Then, the approach that has been used is described after a brief discussion to justify its selection.

6.2 Structured Grids

A structured grid is a logically cubical array of nodes, for a 3D problem, with a regular and well defined relationship between each node and its neighbor nodes. The domain is subdivided into regularly ordered elements that conform to the surface boundaries. The major methods that can be used to generate a structured grid are grouped into elliptic and non-elliptic grid generators.

The main category of non-elliptic generators is the algebraic methods which are, typically, based on interpolation. The latter enables fast generation of a mesh compared with other methods and direct control over the grid point locations. Interpolation-based alge-

braic grid generation methods are known as transfinite interpolation (TFI). They are based on the mapping of a reference mesh, which corresponds to an elementary geometry, on the real domain using a transformation function.

In the case of a 3D cuboid, a two-unit size cube, which is used as an elementary geometry, is meshed by defining intermediate points on the edges in each direction and joining them. Then, using a transformation function that maps any point of the reference cube to the physical domain a mesh is generated. An isoparametric transformation, similar to the one that is used in the FEA, can be employed to generate the mesh of each body that is defined by eight nodes. Higher order polynomials can be used to enable the representation of more complicated and curved geometries.

Elliptic grid generation methods are based on the solution of boundary value problems for elliptic differential equations. They provide smoother grids but they are much slower since they computationally much more involved than the algebraic equations. Taking into account the computational demands, a non-elliptic generator, which is based on an algebraic method, is used for the mesh generation.

6.3 Unstructured Grids

In contrast, an unstructured grid has no logical or implied relationship between the nodes and the elements of the grid. Therefore, an explicit relationship of each element and node to its neighbors is necessary. An unstructured grid conforms easier to the geometric domain allowing significant element size variations and geometric flexibility. A 3D domain is, typically, subdivided into connected tetrahedral elements using triangulation. The most widely used approaches for generating an unstructured mesh are point insertion methods based on Delaunay triangulation, advancing front methods, and octree methods.

Delaunay triangulation in 2D is a network that ensures that circles enclosing three nodal points that form a triangle do not enclose any other points. It is based on the geometric concept that the perpendicular bisector of the line that join two points P_1 and P_2 subdivides the plane into two regions R_1 and R_2, as shown in Figure 6.1.a. The regions R_i can be assigned to the closest point than to any other point of a set of already defined points, resulting in a set of non-overlapping convex polygons called Voronoi regions, that cover the entire domain. The Voronoi diagram, which is formed by the perpendicular bisectors between adjacent nodes, subdivides the space where a set of nodes is defined into a set of adjacent polygons, where each polygon encloses a node. Connecting nodes that have common Voronoi boundaries a triangulation is formed, as it is shown in Figure 6.1.b.

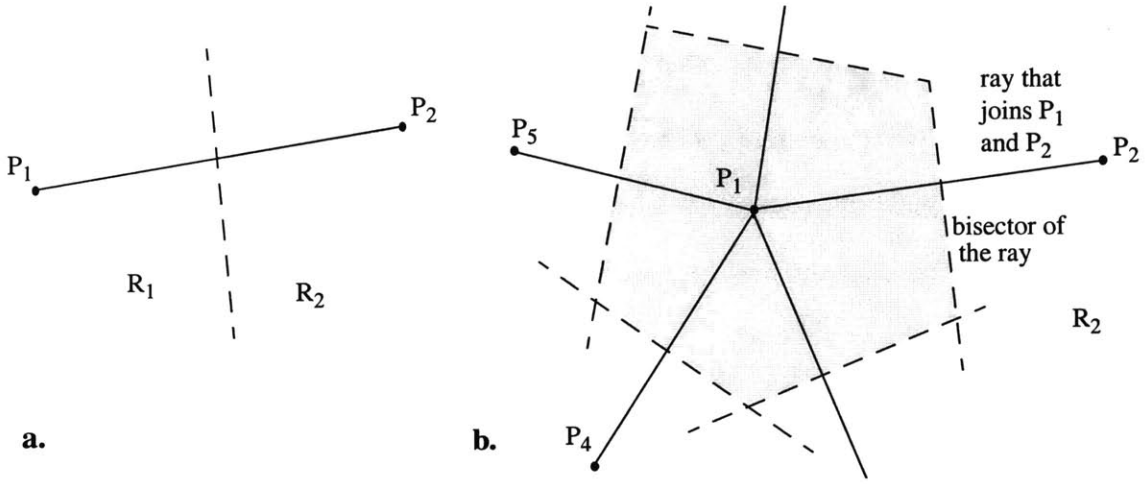


Figure 6.1: Voronoi diagrams and Delaunay triangulation in a plane.

Similarly, in 3D surfaces can be defined to be equidistant from two points forming the faces of Voronoi polyhedra. Connecting the point pairs that have common Voronoi faces a set of tetrahedra is formed that cover the convex hull of the set of points. The convex hull is the smallest possible polyhedron that completely encloses a set of points. Delaunay triangulation connects a set of existing nodes in a way that no node can be contained within the circumspshere of any tetrahedra of the mesh.

Delaunay and Voronoi methods enable the connectivity of already defined nodes. The nodes can be defined by iterative refinement of the initial triangulation which can be obtained by connecting the boundary nodes. Methods to iteratively refine a triangulation using a Delaunay triangulation usually add nodes either at element centroids, or along element edges.

The advancing front method starts with an initial triangulation along the faces of the domain and extends the meshes inward one layer at a time maintaining an active front until the entire domain is meshed. Having initially a set of faces of the triangulated boundary surface, a node is created from which an element is created. Repeating this process moving from the boundary inwards the mesh is extended until the advancing front is empty. The advancing front method allows the generation of elements that are aligned to certain direction and have certain aspect ratios.

Octree methods subdivide a 3D domain into elements, along the three Cartesian axes, using a recursive subdivision based on a spatial tree structure. For example a cube that encloses the domain volume is subdivided into eight cubes, which are then individually checked whether they sufficiently match the corresponding subdomains, and, whichever of them necessary further subdivided. The recursive subdivisions continue until a desired resolution is achieved.

6.4 Selected Mesh Generator

The mesh generator that is selected and implemented is described in the following paragraphs. In addition, its natural extension to 3D bodies is also presented.

6.4.1 Introduction

The selection of the mesh generation method depends on the usage of the mesh, the required accuracy, and the available computational resources. Since computational cost is

very important for our simulations the selection must put an emphasis into this factor even at the expense of accuracy since the latter is already limited by other assumptions that have been made. Structured meshes are simpler and can be used much more efficiently than unstructured meshes. A structured mesh has much less memory requirements since most of the mesh related data can be calculated rather than explicitly stored as it is required for an unstructured mesh.

The error in the FE solution is affected by the quality of the meshing. In general, the error is minimized when the grid is smooth and orthogonal. In addition, it should be finer at regions where the FE solution varies rapidly. Therefore, it is important to be able to generate a solution-adapted mesh in order to reduce the error. However, for the purpose of this thesis this issue has not been addressed because of high computational requirements and the reduced accuracy due to the assumptions and simplifications that have already been made to enable simulations of multibody colliding bodies. Future extensions may incorporate solution-adaptive mesh refining in order to improve the accuracy according to the available computing resources.

6.4.2 Mesh description

Although the developed software implements only rectangular bodies, the procedure for the mesh generation is identical with that of general quadrilateral bodies. Therefore, the description of the mesh generation in this paragraph refer to quadrilaterals for the sake of generality. A rectangular body is just the simplest form of a quadrilateral. Quadrilateral elements, and specifically rectangular elements, have been selected due to their superior performance and accuracy when compared to triangular shaped elements with the same number of degrees of freedom.

The area of a 2D quadrilateral body is subdivided into a set of quadrilateral elements using interpolation functions similar to the ones used in the FEA. The mapping that is

used for the isoparametric finite element formulation, as shown in Figure 6.4, is used to generate algebraically the FE mesh.

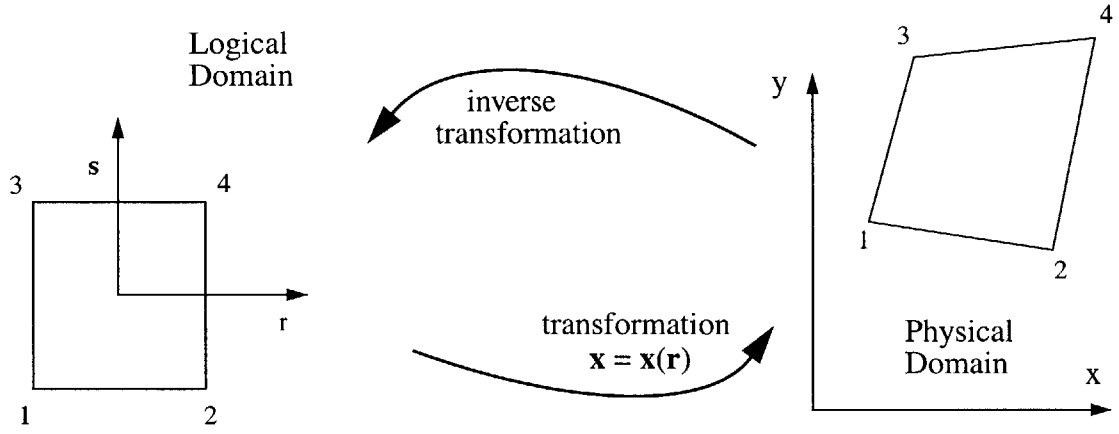


Figure 6.2: Logical and physical domain mappings.

A quadrilateral body can be meshed by mapping to it a meshed two-unit size rectangle using the coordinates of its four vertices. The two-unit size rectangle is meshed easily by varying the logical coordinate in each direction from -1 to +1 using a specific number of increments according to the desired number of nodes in that direction. The domain of mapping of the logical space is the coordinate directions r, and s.

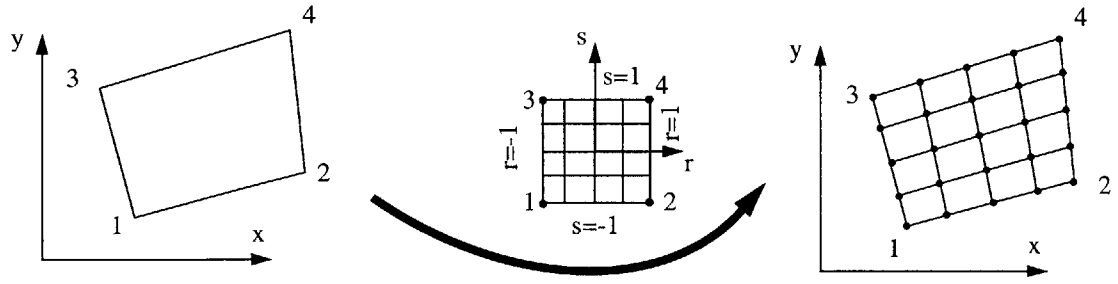


Figure 6.3: Meshing of a 2D element.

Having meshed the reference rectangle a transformation function can map the generated nodes to any physical domain that is defined by four nodes, generating a mesh. The Jacobian of the transformation must be non-singular so that the transformation has an inverse. The transformation functions are used to determine from the logical-space coordi-

nates, the coordinates X, and Y in the physical space are the same as the ones used for the FEA.

Choosing a uniform mesh in the logical domain and transforming it into the physical domain generates the desired mesh in the physical space. There should be one-to-one correspondence between the points in the logical and the physical domain. The coordinate map to transform from the logical to the physical domain has the following form.

$$\begin{matrix} x=x(r,s) \\ y=y(r,s) \end{matrix} \Rightarrow \boldsymbol{x} = \boldsymbol{x}(r, s) \tag{6.1}$$

The mapping functions between the natural and physical domain must be smooth. In particular, the map should be continuous and have continuous derivatives. The Jacobian, $\boldsymbol{J}$, should be non-singular to allow the inverse transformation, and, therefore, the determinant of the Jacobian should be positive.

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial}{\partial r}x & \frac{\partial}{\partial r}y \\ \frac{\partial}{\partial s}x & \frac{\partial}{\partial s}y \end{bmatrix} \tag{6.2}$$

The interpolation functions are used to determine the coordinates of the intermediate nodes from the vertices and the values of the logical coordinates, r and s, as follows:

$$x_i = \sum_{k=1}^{4} h_{k(r,s)} \cdot x_k^i \tag{6.3}$$

where:

$i=1,2$ is the direction of the stationary coordinate system XY

r, s are the axes of the logical coordinate system RS, which take values ranging from -1 to $+1$, and,

$h_i = h_i(r, s)$ are the interpolation functions for a 4-node quadrilateral:

$$h_1 = \frac{1}{4} \cdot (1-r) \cdot (1-s) \qquad h_2 = \frac{1}{4} \cdot (1+r) \cdot (1-s) \tag{6.4}$$

$$h_3 = \frac{1}{4} \cdot (1-r) \cdot (1+s) \qquad h_4 = \frac{1}{4} \cdot (1+r) \cdot (1+s) \tag{6.5}$$

6.4.3 Extension of the mesh generator to 3D

This procedure is easily extended to mesh a 3D cuboidal body into hexahedral elements and its surface into quadrilateral shapes. The hexahedral elements have superior performance and accuracy when compared to tetrahedral shaped elements with the same number of degrees of freedom.

Again, the mapping that is used for the isoparametric finite element formulation is used to generate algebraically the FE mesh for cuboidal bodies. Having the coordinates of the eight vertices that define the cuboidal body the mesh can be determined by mapping a meshed two-unit size cube. The latter is meshed easily by varying the logical coordinate in each direction from -1 to +1 using a specific number of increments according to the desired number of nodes in that direction. The domain of mapping of the logical space is the coordinate directions r, s, and t. The boundary of the reference cube that is used in the logical space E^3 consists of 8 vertices, 12 line segments, and 6 open faces.

Having meshed the reference cube a transformation function can map the generated nodes to any physical domain that is defined by eight nodes, generating a mesh. The Jacobian of the transformation must be non-singular so that the transformation has an inverse. The transformation functions are used to determine from the logical-space coordinates, the coordinates X, Y, and Z (or, X_1, X_2 and X_3) in the physical space as shown in Figure 6.4.

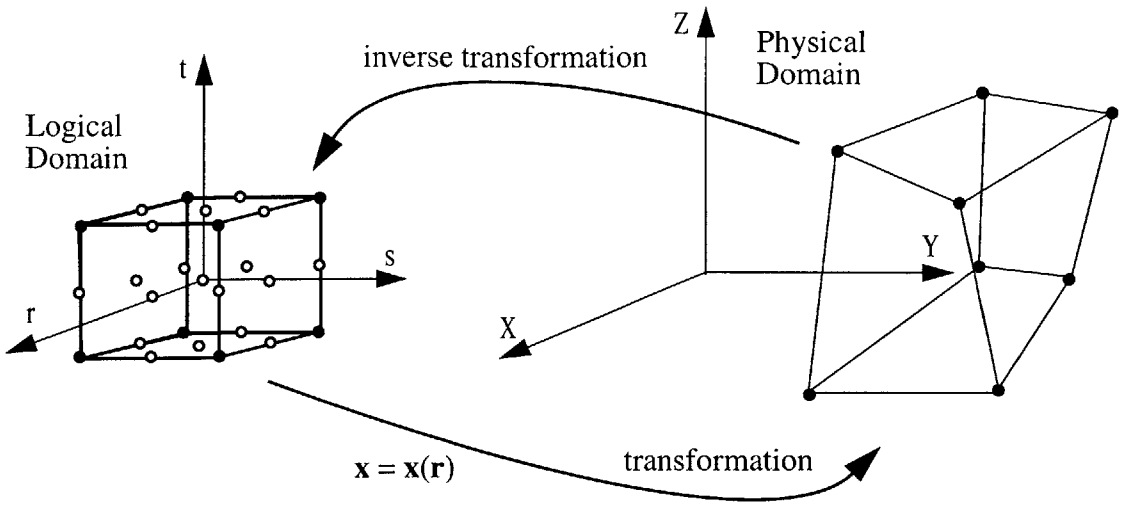


Figure 6.4: Logical and physical domain.

Choosing a uniform mesh in the logical domain and transforming it into the physical domain generates the desired mesh in the physical space. There should be one-to-one correspondence between the points in the logical and the physical domain. The coordinate map to transform from the logical to the physical domain has the following form.

$$\begin{aligned} &x=x(r,s,t) \\ &y=y(r,s,t) \\ &z=z(r,s,t) \end{aligned} \Rightarrow \boldsymbol{x} = \boldsymbol{x}(r, s, t) \tag{6.6}$$

The mapping functions between the natural and physical domain must be smooth. In particular, the map should be continuous and have continuous derivatives.

The Jacobian, $\boldsymbol{J}$, should be non-singular to allow the inverse transformation, and, therefore, the determinant of the Jacobian should be positive.

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial x}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial t} \\ \frac{\partial x}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} \tag{6.7}$$

Figure 6.5 presents the reference cube in the logical domain that is meshed using a desired refinement and then mapped to the physical domain to obtain the meshing of the actual cuboidal element.

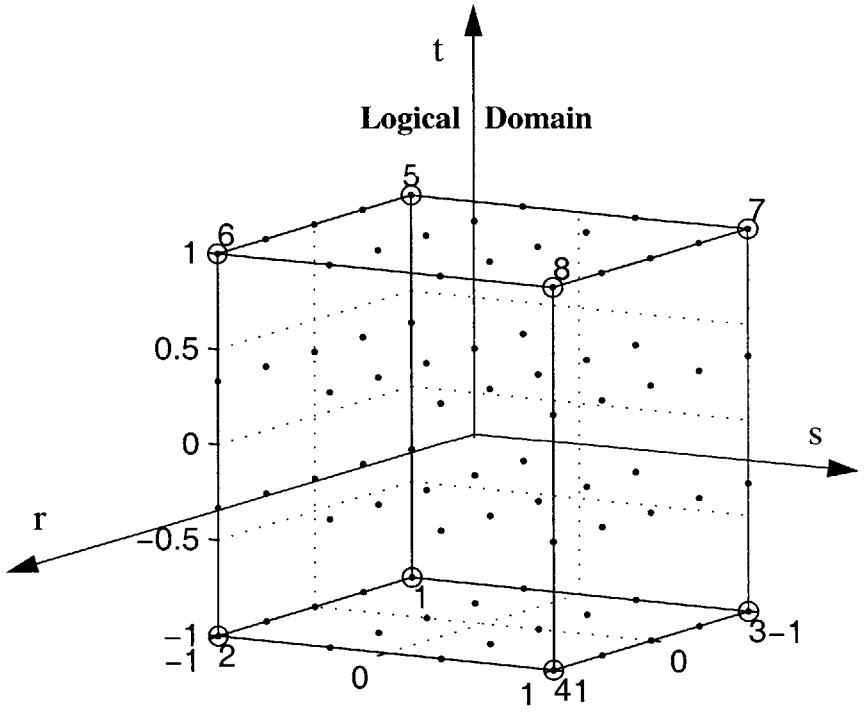


Figure 6.5: Meshed reference cube in logical domain.

A grid is first defined in the logical space and then mapped to the physical space using the corresponding mapping functions. The nodes in the logical space are generated by subdividing the cube into smaller identical rectangular cubes. In this particular example, the grid that is selected has five, four and four nodes in the r, s and, t directions, respectively, of the logical coordinate system.

Then, the specified nodes in the logical space must be mapped into the physical domain. The isoparametric transformation that is used for coordinate interpolations in FEA, is used to determine the coordinates of the intermediate nodes from the vertices and the values of the logical coordinates, r, s, and t, as follows:

$$x_i = \sum_{k=1}^{8} h_k(r,s,t) \cdot x_k^i \tag{6.8}$$

where:

i is the direction of the stationary coordinate system XYZ and takes the values *1*, *2*, and *3*, respectively

r, s, t are the axes of the logical coordinate system RST, which take values ranging from *-1* to *+1*, and,

$h_i = h_i(r, s, t)$ are the interpolation functions for an eight-node cube:

$$h_1 = \frac{1}{8} \cdot (1 + r) \cdot (1 + s) \cdot (1 + t) \qquad h_2 = \frac{1}{8} \cdot (1 - r) \cdot (1 + s) \cdot (1 + t)$$

$$h_3 = \frac{1}{8} \cdot (1 + r) \cdot (1 - s) \cdot (1 + t) \qquad h_4 = \frac{1}{8} \cdot (1 - r) \cdot (1 - s) \cdot (1 + t)$$

$$h_5 = \frac{1}{8} \cdot (1 + r) \cdot (1 + s) \cdot (1 - t) \qquad h_6 = \frac{1}{8} \cdot (1 - r) \cdot (1 + s) \cdot (1 - t)$$

$$h_7 = \frac{1}{8} \cdot (1 + r) \cdot (1 - s) \cdot (1 - t) \qquad h_8 = \frac{1}{8} \cdot (1 - r) \cdot (1 - s) \cdot (1 - t)$$

Using the above interpolation functions the nodal points of the reference cube in the logical domain are mapped into the physical domain defining the nodes as presented in the Figure 6.6.

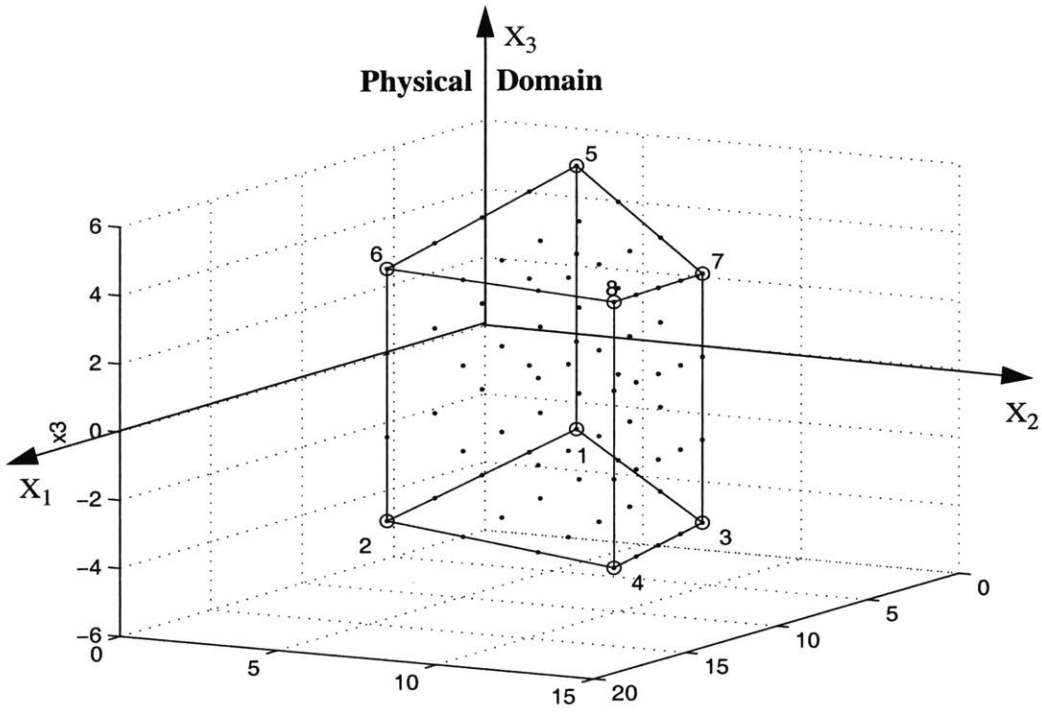


Figure 6.6: Meshed cuboidal element in the physical domain.

Chapter 7

Software Design and Implementation

7.1 Software Design Overview

An extensible and flexible computational tool for multibody simulations has been built using the Java programming language and the Java3D graphics application programming interface (API). Java technology we selected because it provides portability, architecture-neutrality, and multithreading. In addition, using a pure object oriented programming (OOP) language, like Java, more robust programs can be designed that allow easy future modifications and incorporations of new algorithms without the need of rebuilding the whole system.

In particular, for the computational part and the graphical user interface have been implemented using core Java language. For the graphical part the Java3D API has been used, while for the database access, which is used in both preprocessing and postprocessing, the Java Database Connectivity (JDBC) API has been employed. The Java2D API has also been used for the postprocessing to convey information of the results in the form of 2D graphs.

OOP is nowadays the dominating programming paradigm since it has certain advantages over the Procedural-Oriented Programming (POP). OOP enables information hiding, inheritance, encapsulation, dynamic memory allocation and polymorphism with which robustness, efficiency, extensibility, and modularity can be achieved.

7.2 Computational Considerations: Java *vs* C++

A decision had to be made concerning the language to be used. This is interrelated with the decision of which computer graphics library to use for the rendering of the simulated model. The latter is articulated in the next paragraph, "Computer Graphics Considerations" on page 120.

The languages considered to be used were C++ and Java. C++ programs are compiled into executable code, i.e. machine instructions, specifically for the underlying architecture and operating system. In contrast, Java source code is compiled into bytecode that is translated into machine instructions during execution by the Java Virtual Machine (JVM). Although this two-phase compilation-interpretation provides portability, it has some overhead on the computational performance.

However, the latest JVM allows just-in-time (JIT) compilation which achieves performance comparable to that of a compiled language. Although during compilation of Java source code into bytecode only very limited or no optimization is used, during execution profiling and recompilation enables run-time optimization. In particular, whenever a block of code is encountered for first time, it is compiled into machine code by the Java HotSpot performance engine and then executed. In addition to the JIT, blocks of frequently used code are profiled and recompiled for optimization purposes, using profiling data from the interpreted code. In contrast, using C++ any optimization is done during compilation based on hints that may be provided by the programer and some fixed rules. The run-time, i.e. dynamic, optimizations are more powerful since they can take advantage of the given conditions during the execution of the program, while static compilation can take advantage of only what the compiler can predict based on certain predefined rules.

Finally, the concept of virtual machines and run-time optimizations are rather recent advances that can be greatly be improved. The differences between Java and C++ regard-

ing performance are not very significant, as shown in the next example. Considering the expected further future improvements of the JVM and language this difference should decrease even more. Even within the last five years, since the birth of the Java language, there have been great improvements in its performance and capabilities. The initial JVM was interpreting and executing each instruction individually, which was as slow as interpreted languages. Later, the just-in-time (JIT) capabilities allowed the compilation of any block of code as soon as it is first encountered into machine code followed by its execution. During execution a Java program is built by the JVM incrementally as needed and not monolithically, according to the instructions of the corresponding bytecode. The latest versions of the JVM allow not only JIT compilation but also run-time, i.e. dynamic, optimization of frequently used blocks of code using relevant profiling data and taking advantage of the given conditions during execution.

A sorting algorithm has been used to compare the C++ and Java programming languages. The following figure shows the time required to sort an N-size array with random numbers, using the mergesort algorithm, implemented both in C++ and Java. A logarithmic scale was used for the axis of size, i.e. number N of elements.

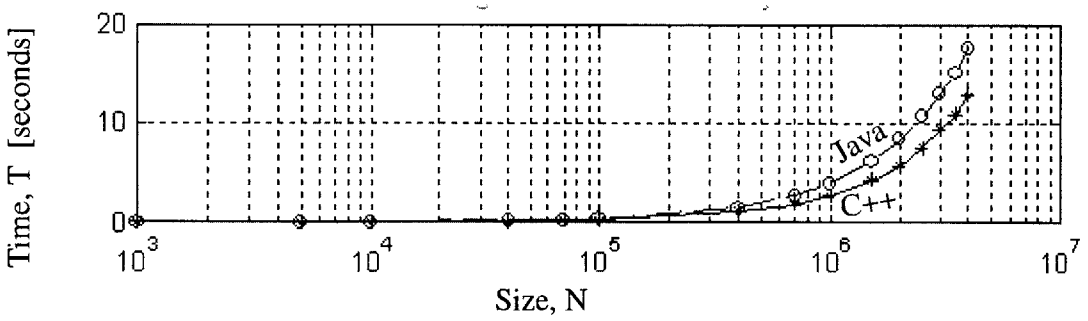


Figure 7.1: Time required to sort an N-size array with randomly generated elements, using the mergesort algorithm, implemented in C++ and Java.

Figure 7.1 shows that the performance of the Java implementation, although 35-50% slower, is comparable to the C++ implementation. The GNU GCC compiler has been used

for the C++ version with its optimization flag selected. Both programs were executed one a Sun ULTRA 10, 333MHz, workstation with a 128MB DRAM. The For each size, N, of the problem a set of executions were used to determine the statistical average times, T.

The following figure, Figure 7.2, shows clearly the difference between the C++ and Java implementations. The horizontal axis represents the size of the array, N, while the vertical axis represents the time, T, divided by the N.log(N). The time required by the Java program was 35 to 50% more than that required by the C++ implementation. The difference in performance using Java instead of C++ is less than the difference of performance that can be achieved by using a slightly faster machine. In addition, Java showed the same scalability with C++ without any significant variation of the relative performance with the size of the problem.

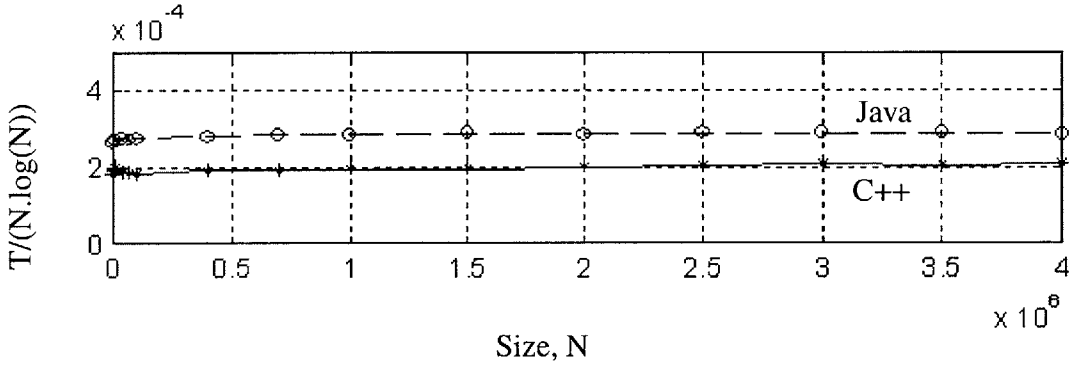


Figure 7.2: Performance Comparison of C++ and Java using a mergesort algorithm to sort an N-size array with randomly generated elements.

A second performance test that involves two dimensional arrays was performed to compare the computing efficiency of C++ and Java. In particular, two square arrays of size $N \times N$, having elements randomly selected numbers of double precision, were multiplied with each other resulting in an $N \times N$ array. For each size N the elements are set to random numbers and then multiplied five times taking as required time the average of the five

recorded times. The multiplication of the arrays requires an $O(N^3)$ computational time. The following figure, Figure 7.3 shows the required time for each implementation.

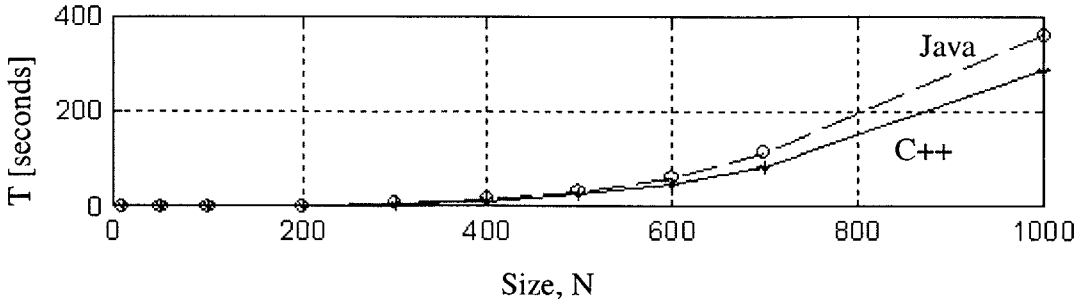


Figure 7.3: Time required to perform multiplication of two *NxN* size arrays with random double precision floating-point elements, using C++ and Java.

The above figure demonstrates that Java is not that much slower from C++. The same machine, Sun ULTRA 10, 333MHz, workstation with a 128MB DRAM, has been used. The C++ code has been compiled with the optimization flag on, while the just-in-time compiler of the JVM has been used.

Although C++ is still faster the difference is not that much to overweigh the significant advantages offered by the Java language and technology. In particular, the differences in performance between Java and C++ vary from 26% to 59% and does not vary significantly with the size N of the problem, as shown in Figure 7.4. The time T has been divided by the corresponding N^3, since the algorithm is $O(N^3)$ in order to be normalized.

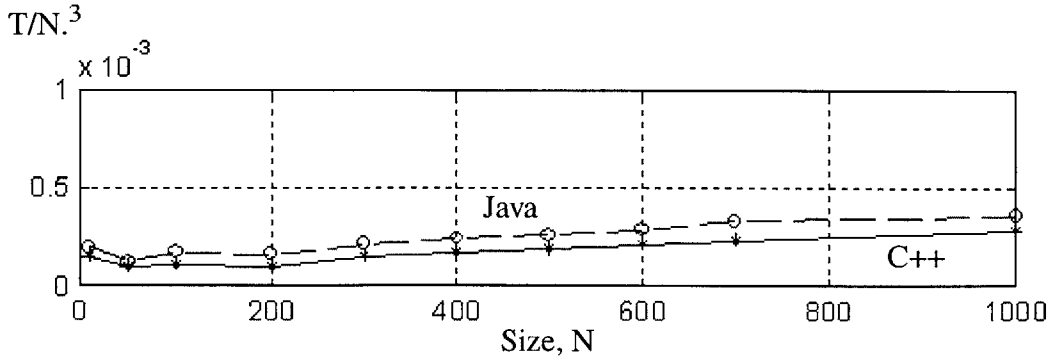


Figure 7.4: Performance Comparison of C++ and Java using multiplication of two NxN

size arrays with randomly generated elements. Time T is divided by the N^3.

Similar results have been obtained for both performance tests, i.e. the mergesort algorithm and the matrices multiplication, using Visual C++ and Visual J++. The relative difference was actually smaller, but it was preferred to present the results of using the GCC compiler for C++, which is part of the GNU project, and the JVM provided by Sun for Java, as these are not only according to the standards but also freely available.

A research report [37] that compared C, C++, Java and Ada against the Steelman requirements have found that the latter were satisfied at 53%, 68%, 72% and 93%, respectively, of the cases. The Steelman requirements [35] were set more than two decades ago after widely reviewed research in an effort to set requirements for general-purpose languages concerning efficiency and reliability and do not reflect very well the performance of real world object-oriented software. Java satisfies 72% of the Steelman requirements although the latter do not take into account object-orientation which is a strength of Java. Ada scores so high, 93%, because it was specifically designed to meet the Steelman requirements and does not have other very useful capabilities that are provided by C++ and Java.

In this research work certain measures were taken to achieve the highest possible performance with Java. However, there are only limited options that can be taken to achieve higher performance with Java. This is not necessarily bad, since it makes things much simpler and encourages the developer to focus in developing more efficient algorithms instead of playing with optimization options that are usually machine dependent. Although the handling of memory in Java is mostly performed automatically, there are certain issues that can be controlled by the programmer or the user in order to enhance performance. During execution of a Java application the user can specify the initial and maximum mem-

ory settings using the *-Xms* and *-Xmx* flags, instead of using the default values of 2MB and 64MB, respectively, used by the HotSpot VM.

In addition, explicit invocations of the garbage collector can be used whenever appropriate to minimize its impact on the behavior of the application. The invocations of the garbage collector can be monitored by using the flag *-verbose:gc*. The option *-XX:NewSize=200m* can be used to set the initial nursery size to avoid unnecessary invocations of the garbage collector. The HotSpot VM, as well as most other VMs, is, by default, generational, i.e. instead of collecting all the memory, it divides the memory into two or more generations and when the younger generation, or nursery, is nearly full it performs a partial garbage collection. This strategy takes advantage of the common situation when young objects are more likely to be eligible for garbage collection to avoid full garbage collections, which are very expensive in terms of performance. In general, generational garbage collectors achieve reduction of both the duration and the frequency of full garbage collections.

Finally, the HotSpot VM can run in incremental mode by using the *-Xincgc* flag during execution of the program. Incremental garbage collection divides the set of objects into small subsets which are processed incrementally, aiming to make smaller rather than larger pauses. However, although the pauses may be smaller the overall cost of the garbage collection tends to be higher, since incremental garbage collection does not run only when the memory is almost full, but whenever it sees an opportunity to run in the background. Therefore, the incremental garbage collection may be used when an application runs in an interactive mode where response time to the graphical user interface interaction should be guaranteed. Otherwise, the generational garbage collection should be preferred.

The HotSpot VM learns over time utilizing profiling data during execution to achieve optimizations and better performance of the application. However, it is not trivial to tune

the memory management of the HotSpot VM, for which run tests are required to provide insight in its interactions with the code.

7.3 Computer Graphics Considerations

7.3.1 Selection of a graphics library

A computer graphics library had to be selected for the rendering part of the simulation. The options included the Java3D API, the Open Inventor 3D toolkit, and the OpenGL graphics library. Java3D and Open Inventor are much higher level graphics libraries than OpenGL, on which their implementations are actually built. Java3D uses either OpenGL or Direct3D as its underlying rendering system. The decision of which graphics engine to use was based on several factors as explained below. The investigation of these factors has been carried out in parallel to the investigation regarding which programming language to use, C++ or Java. Naturally, both investigations influenced each other.

OpenGL is a set of graphic libraries implemented in C++. These libraries are optimized to run fast on computers with special graphic boards which are specifically designed to make matrix operations in parallel instead of sequentially on the CPU of the workstation.

Java3D and Open Inventor have inferior performance to that of OpenGL and limited access to the rendering pipeline details. Although Java3D and Open Inventor cannot achieve peak performance, portability and rapid development advantages may overweigh the slight performance penalty for many applications.

Considering that the focus of this research on the implementation of a simulation tool that can be easily extended rather than dealing with fine rendering details it was decided to use a higher rather than a lower level graphics library. Although use of OpenGL enables more efficient rendering with regards to performance during simulation the time required

to develop and maintain robust code using Open GL overweighs its potential benefits. The high level Java3D and Open Inventor allow rapid application development. Although OpenGL has very flexible low level control on rendering, it is not object-oriented, as it is written in C, and does not have a high level interface resulting in very inefficient development. In contrast, both Open Inventor and Java3D are high level and object-oriented graphics libraries that enable rapid development of software.

Since the Open Inventor and Java3D are both very high level graphics libraries and very similar, a decision which of the two is more appropriate, had to be made. The two options were evaluated based on a pilot DEM program that was developed using both Java3D and Open Inventor. In particular, a 3D discrete element program with infinitely rigid spheres and boxes has been developed using C++ and Open Inventor, Figure 7.5.

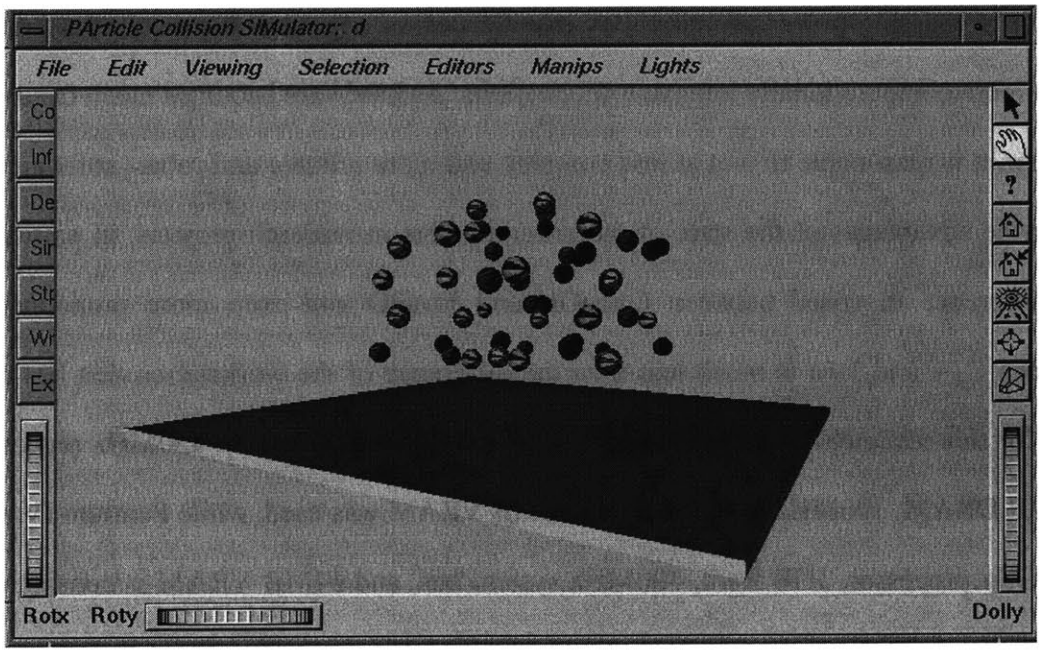


Figure 7.5: Particle Collision Simulator (PACSIM): a 3D DEM program, with infinitely rigid bodies, implemented using C++ and Open Inventor.

In particular, that program, named PACSIM (Particle Collision Simulator), was developed using the C++ programming language for the computational part of the simulations,

the Tool Command Language and the Toolkit library (Tcl/Tk) for the user interface, and the Open Inventor for the 3D computer graphics. The program has shown performance similar to an identical program implemented in Java3D. Therefore, since the two programs demonstrated similar performance, other factors, such as robustness and extensibility, were considered to select Java3D as the most appropriate graphics library.

First and foremost, Java3D, as any Java technology, is freely available to anyone, avoiding the licensing problems of other proprietary rendering libraries, such as Open Inventor. In addition, Java3D enables portability and cross-platform support when combined with other Java APIs, with which can be very well integrated and tightly coupled. Finally, programs written in Java can be very easily well documented using the *javadoc* automated document generation tool.

The processing power of computers experiences an exponential growth that permits much more computationally intensive applications. As hardware becomes much faster and cheaper, it is reasonable to aim at less complex and more reliable and robust software utilizing the advantages of the state-of-the-art computers in research projects. in particular, the difference in speed between OpenGL and Java3D and even more pronouncedly between C++ and Java is much less than the difference of the workstation that has been used for this research. In particular, I have used a Pentium II with a 450MHz processor, 128MB DRAM, 100MHz system bus, and 8MB VRAM was used, while Pentium IV, with a 1.5GHz processor, 1GB Ram, 400MHz system bus, and 64MB VRAM is considered a system that any serious programmer should have and use, nowadays. Since the computing power could easily be increased by more than 400-500%, it would be meaningless to prefer C++ over Java because of a 30-50% difference in performance, difference that becomes narrower day by day.

In essence, the speed depends much more heavily on the quality of the graphics hardware, which Java3D can take advantage, than on the 3D programming API that is selected to create the 3D graphics.

The decision to use Java3D was based on the software revolution of the recent few years that leads to a Java-unified environment and supports Sun's motto "The network is the Computer". Java3D, as any other 3D graphics library, requires sufficient CPU speed and memory in order to perform the underlying mathematical operations. The technology is available, thus, the selection of Java was evident. Software engineering should advance and evolve parallel to the advances of the hardware technologies.

7.4 Graphical User Interface Considerations

7.4.1 Selection of GUI framework

The decision of which graphical user interface framework to use dependent on the decisions of which language to use for the computationally intensive part and which computer graphics library to use for the 3D graphics. As those decisions leaned towards Java it was evident that either AWT (Abstract Window Toolkit), or Swing should have been used.

The AWT was provided by the initial version of Java to enable programmers develop GUI's in Java. Later, AWT provided the foundation on which the Java Foundation Classes (JFC) and Swing were built. The main difference between AWT and Swing is that the components of the latter are implemented without any native code. Because of this, Swing components are called lightweight, while AWT components, which use native code, are called heavyweight components. Lightweight components are drawn entirely using Java, while heavyweight components use native peers. Actually, AWT 1.1 also introduced some lightweight components, while earlier versions were based solely on heavyweight components. Swing components, in general, have much more capabilities than the AWT corre-

sponding ones. For instance some Swing components, such as labels and buttons can display icons, while the corresponding AWT components cannot. Swing provides a number of additional components that were not provided by AWT.

In this research the AWT was used initially as it was easier to couple it with Java3D as the latter uses a heavyweight component, in particular the Canvas3D which extends the heavyweight AWT class Canvas. The main problem is that there is one-to-one correspondence between heavyweight components and their window system peers, i.e. native OS window components. In contrast, a lightweight component expects to use the peer of its enclosing container since it does not have a peer. However, taking certain measures Swing can be used with Java3D as it was eventually decided to be used here. Swing and the precautions taken in order to couple it with Java3D are described in the Paragraph "Swing" on page 149.

7.5 Utilization of Database Technology

7.5.1 Motivation for using database technology

Advances in computational technology enable an increasing level of complex engineering analyses and simulations that often produce an enormous amount of data. In addition, an increasing demand for complex computational analyses and simulations puts new demands on the ability to provide computational support to efficiently manage all data produced in this work flow. Large amounts of data that are generated by modern engineering applications can be efficiently managed using database technology. Database technology is traditionally used for efficient management of data, of simpler form than data in engineering analysis, usually found in administrative applications. However, the continuing development of modern database technology makes efficient data management capabilities more applicable to complex engineering data.

Data management can be considerably improved in a database system compared to conventional applications. The DBMS provides a better accessibility to the data and a query language that can be used for retrieving and evaluating analysis results. The developed software generates huge amounts of output data that need to be efficiently managed. This is accomplished by integrating a database system in the simulation program and taking advantage of the database technology. The data can be searched and compared using an extensible query language that is provided by the database system and is, therefore, integrated into the application.

In particular, a relational database management system is used through the Java database connectivity (JDBC) application programming interface (API) that interfaces the application with a conventional relational database management system (RDBMS), as illustrated in Figure 7.6. The database connectivity accomplishes a high level of program and data independence making the system design very flexible and change tolerant. Since data is stored in the database, it is available from the query language that enables a variety of capabilities to compose queries for retrieving, combining, constraining, and transforming data. A drawback in this approach is the impedance mismatch between the relational data model and the object-oriented host language, the Java language, used for implementing the application.

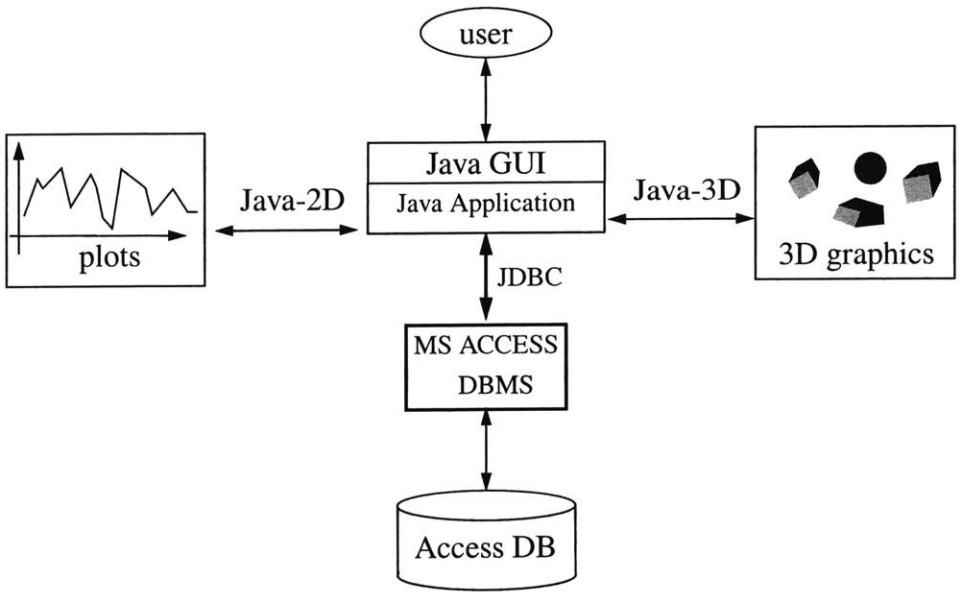


Figure 7.6: Architecture of DAFES

7.5.2 Potential future extension

These ideas can be further by directly embedding a main-memory (MM) DBMS within the application. In that case, the integration can be made through a Java native interface provided by the DBMS. An embedded DBMS avoids unnecessary data shipping between a client application and a database server application. Furthermore, using an object-oriented DBMS reduces the impedance mismatch and makes it possible to handle tailored data representations and operators.

The Active Mediator Object System (AMOS), which is an object-oriented database system that has a functional data model with a relationally complete object-oriented and extensible query language (QL), has been considered for future extensions. The data manager of AMOS-II is designed as a main-memory Database Management System (DBMS) in order to achieve good performance. It is optimized for efficient execution when the database fits in the main memory. The queries in AMOSQL are optimized before execu-

tion. The AMOS-II is implemented in C and currently runs only under a Windows operating system.

AMOS-II provides the traditional DBMS facilities such as storage, transactions, and recovery managers, and an object-oriented (OO) relationally-complete query language (QL), named AMOSQL. In addition, AMOS-II is a distributed mediator DBMS that allows multiple AMOS-II mediator servers to communicate with each other over the internet. Therefore, an application with an integrated AMOS-II can access data over the internet from multiple distributed data sources through their corresponding AMOS-II servers.

AMOS-II provides client/server and inter-database communication facilities. AMOS servers can communicate with each other over TCP/IP. External data sources, such as a relational database, can be accessed using the appropriate wrapper, which is an AMOS-II module with query processing and data translation facilities. AMOS-II provides a graphical user interface (GUI), named Goovi and implemented in Java, that enables multi-database browsing and integration.

Work that has been done with AMOS-II demonstrated that it can be embedded easily into DAFES. A combined DAFES-AMOS system can have database capabilities such as a storage manager, a data model, a database query language and processor, remote access and transactions to other data sources. In addition, the database supports, among others, concurrency, data exchange and transformation, data sharing and distribution, and interoperability. It has been shown, Kjell [22], in a similar database integration that the DBMS functionality can be supplied without any major performance cost. The following figure shows a potential architecture of a combined AMOS-DAFES.

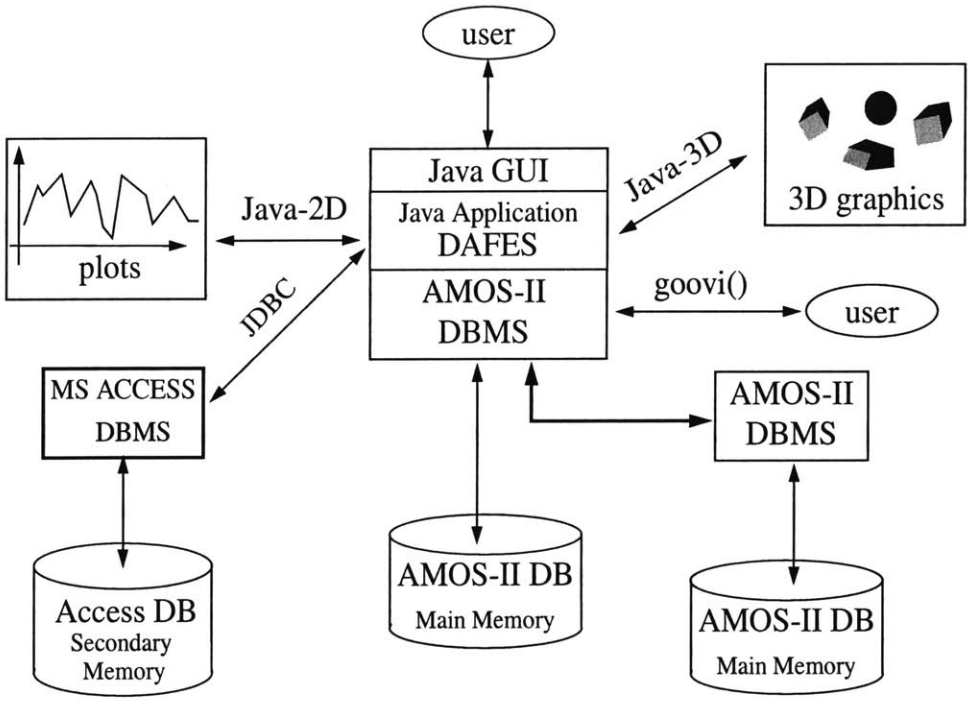


Figure 7.7: Potential future DAFES architecture combined with AMOS-II.

AMOS-II also provides an interface, named goovi(), for a menu-driven interaction with the database. In addition to the provided by AMOS object-oriented query language, named AMOSQL, new data types and operators can be added to AMOSQL using a programming language such as Java. Unlike traditional engineering applications, DAFES-AMOS stores the input and output data in a database from where they can be efficiently accessed and queried by the AMOS-II DBMS, which is part of the application.

7.6 Structure of the Software Application

The simulation tool that has been developed consist of several modules as shown in the following diagram, which provides the basic structure of the program:

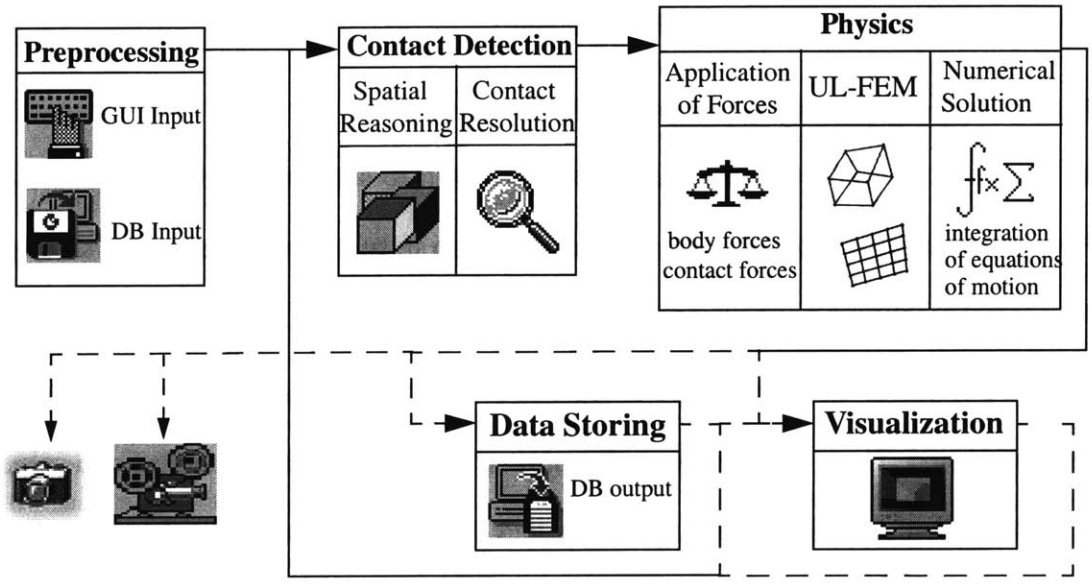


Figure 7.8: Simulation pipeline.

7.6.1 Preprocessor module

The preprocessor is used to obtain from the user using a graphical user interface values of simulation parameters and conditions, as well as specific information for the individual bodies of the system to be simulated. In particular, the geometric information, boundary conditions, and physical properties of each simulated body must be specified by the user. A friendly graphical user interface component has been built to allow easy and visually effective generation of the system to be simulated and parameters of the numerical simulation. The graphical user interface, which is used for both preprocessing and postprocessing, has been developed using the Swing API of Java.

The user can easily provide the geometry and layout of the simulated bodies, their boundary conditions, i.e. constraints specifically determined by the user, and their material properties. The user can specify or change simulation parameters, such as gravity, time

step for the numerical integration, and desired output. Support for automatic generation of 3D bodies has also been provided. Finally, automatic experiment setup capabilities have also been provided to enable easy generation of multiple bodies samples according to some bulk characteristics provided by the user.

7.6.2 Computationally intensive part

An efficient contact detection module has been built to support the 3D DE/FE program. The general idea of separating the contact detection in two phases the spatial sorting and the contact resolution has been followed.

The physics of the problem are addressed in three faces. first, the forces on each body are determined. Then, the equations of motion of the bodies are formulated and solved to determine their motion. Finally, the stress and strain distributions of selected bodies are computed.

An explicit integration scheme, e.g. central difference method, which is based on the equilibrium conditions at time t is very efficient, Although a strict stability criterion must be satisfied by the time step, an explicit integration algorithm simplifies the solution substantially when a lumped diagonal mass matrix is used while either no damping is considered or it is assumed mass proportional. Then, the system of equations can be solved without matrix factorizations, but only doing matrix multiplications. This allows to evaluate the effective load at each nodal displacement on the element level considering only the elements associated with the specific node. Therefore, no stiffness matrix of the complete element assemblage need to be constructed and the solution can be carried out locally with very limited high speed memory required. This is very important for systems that are typically analyzed with DEM which may have millions of particles.

The shortcoming of an explicit method is that the method is conditionally stable and a sufficiently small time step is required to ensure stability. However, in any way we need to

use small time steps to avoid errors in the contact detection part and the contact forces calculation.

Using instead an implicit method, i.e. Newmark method, would result in a huge system of coupled nonlinear equations which would be probably impossible to be solved in reasonable computational time.

The main loop will be an explicit time step integration scheme. Central difference time stepping method may be used to perform the time integration of the equations of motion. Using the contact forces from time step together with any other external load the displacements at time $(t+\Delta t)$ are determined and then the internal strains and stresses of each discrete body are calculated. finally, a new contact detection identifies the new contacts and determines the new contact forces that will be used for the next time step.

7.6.3 Postprocessor module

Analysis results of systems with multiple bodies require the support of powerful post-processor so as to be presented in an effective visual way. The Java3D API which is a graphic's library, similar to Open Inventor, has been used for the visual presentation of the results. Effective visual presentation of the changes of the simulated system as the simulation progress allows the capturing of animations of the particles which may reveal phenomena that occur at the particle level.

For complicated simulations the computations can be separated from the graphics in order to fully utilize the hardware resources by postponing the rendering of the simulated bodies. A color-coding scheme is used to represent different physical states, such as magnitude of stresses.

The output results are stored in a database from where they can be effectively accessed by the user using the graphical user-friendly interface. Storing the results in a database allows the manipulation of results and an output data analysis, which would be otherwise

very difficult considering the amount of information that results from such kind of simulations.The Java Database Connectivity (JDBC) has been used to access the database which is used to store the results of simulations. Finally, graphs capabilities using the Java2D API can be used to convey useful information to the user by plotting certain sets of values.

Furthermore, off-canvases are used to record snapshots of the simulations in order to have either individual snapshots in the form of JPEG files, or a set of sequential JPEG files that are put together constructing a movie of the simulation.

7.7 Software design

The Unified Modeling Language (UML) has been used to specify, visualize and document the software design. UML facilitates the design process and the development of class interfaces, while enhancing the reusability of the code and communication between programmers. It is suitable for an object-oriented programming language such as Java and C++. UML is based on earlier Object-Oriented Analysis and Design (OOAD) methods and has been standardize by the Object Management Group (OMG).

Considering the high complexity of the system and the limited development resources, good modeling and abstraction techniques are required to be employed in order to focus on relevant details, while neglecting others, since comprehension of such a system in its entirety would be unrealistic. UML enables the systematic use of abstraction through a set of different views of the system under development, such as use case, class, behavior, state chart, activity, interaction, sequence, collaboration, implementation, component, and deployment diagrams. Although UML provides may models and techniques only few of them are required and used in this project.

7.8 Software Implementation

7.8.1 Java language

After a very detail comparison of C++ and Java, taking into account the strengths and weaknesses of each language, the latter was selected as more appropriate to be used as it provides a more elegant software implementation. The performance concern of using Java instead of a compiled language, such as C++, has already been discussed in the Paragraph "Computational Considerations: Java vs C++" on page 114. Java was selected due to the advantages of Java, compared to C++, in terms of modularity, robustness, portability, security and superior memory management. Java is a pure object-oriented programming (OOP) language combining the best concepts and mechanisms of C++ and other programming languages, such as virtual functions and polymorphism, inheritance, function overloading, and exception handling, while removing mechanisms that complicate, in general needlessly, the software development.

Java is a pure object-oriented programming language that enables the development of very robust and dynamic, architecturally neutral and portable software with multithreading and security capabilities. In particular, Java has key object-oriented characteristics and mechanisms that allow efficient development of portable, robust and reusable programs with graphics and graphical user interfaces. Dynamic memory allocation, which is typical a source of mistakes and memory leaks in C and C++, is avoided since Java provides a garbage collector. The latter removes the responsibility of the programmer to release dynamically allocated memory from the heap using the *delete* operator. Furthermore, destructors are not required in Java, although clean up of objects before being deallocated can be done using finalizers. In addition, the complexity due to certain cases of multiple inheritance are avoided, since Java supports single inheritance, while achieving the effects of multiple inheritance without the associated problems through multiple interface inherit-

ance. Java enables more general class initialization, while the class initialization capabilities of C++ are very limited.

Java provides multithreading, which facilitates interactivity and concurrency of programs. A threat is a single sequence of steps executed one at a time, i.e. a sequential flow of control, running within a program and taking advantage of the resources allocated for that program and its environment. All threads of a multithreading Java program share the same data and system resources, running at the same time and performing different tasks. Since most computers have only one CPU, threads share the CPU with other threads. A thread can be created in Java either by subclassing the Thread class and overriding its run() method, or by providing a class that implements the *Runnable* interface, which requires the implementation of the run() method. Synchronization capabilities of the Java language allow a thread to lock an object in order to avoid data corruption. Multithreading enables a program to perform several tasks, i.e. to use more than one flow of control, concurrently. Concurrent programming is directly supported as threads of operations are allowed to run concurrently taking advantage of multiprocessor systems. In contrast, multithreading and concurrent programming can be accomplished in C++ using operating system calls and libraries, such as the Message Passing Interface (MPI) standard, respectively.

More checks and verification tests are performed with in Java by both the Java compiler and the JVM. Unlike C++, Java performs index checks for arrays avoiding problems with indices that are out of bounds. Security is also ensured by the bytecode verifications performed by the Java Runtime in order to determine whether there was any modification of the bytecode after ti was generated. Finally, Java provides access to compilation via the *java.lang.Compiler* package, which enables a Java program to generate code, and, then, compile, load and use it in a single session. Class information is accessible during execu-

tion in Java through the *Object* and *Class* classes, while such information is not provided in C++.

A very important characteristic of Java is portability, which is guaranteed at the byte-code level. In contrast, C++ supports portability at source level with certain limitations and depending heavily on the programmer. In order to achieve portability in Java, the compilation generates JVM instructions in the form of architecturally neutral bytecode, which are mapped to actual machine instructions at execution time. In addition, graphical user interfaces and graphic components are mapped only at execution time to the native window environment and graphics libraries. Access to system functions is provided in a platform independent way by the *java.lang.System*. Finally, any primitive data type is defined in an architectural neutral way with a predefined size of memory associated to it.

Software development is much faster and efficient with Java due to the excellent and consistent on-line documentation and certain mechanisms that avoid needless repetitive tasks. For example, there is not need for class declarations, as in C++, which not only require extra work, typically, by copying the header of the definitions of functions into the corresponding header files with fine modifications, but also are sources of errors. In addition, Java, does not require recompilation of subclasses because of changes made to superclasses, since linking occurs only at execution time. In contrast, recompilation is required in C++, in order to take into account any changes in size, which has an overhead on the compilation time. The Java compiler identifies and compiles the classes on which a program depends automatically, without the need of a makefile. Finally, In addition, documentation of Java software is automatically generated using the *javadoc* tool of the Java Development Kid (JDK). The *javadoc* parses the source code using certain comments in the code to create an up to date Hypertext Markup Language (HTML) document as an API reference for each class.

However, a useful mechanism of C++ that is missing from the Java language is that of operator overloading, which is used only by the + operator for string concatenation. Another limitation of Java is that primitive data types can be passed only by value and all other objects can be passed only by reference. Although, Java does not provide direct access to memory locations since there are no pointers avoids problems due to pointer arithmetic mistakes. Callbacks, which are implemented in C++ using pointers to functions, can instead be implemented using interfaces. Finally, Java does not allow parameters of methods to have default values, which is allowed in C++.

The Java platform consists essentially by the Java virtual machine (JVM), which takes care of the compilation and interpretation issues, e.g. portability, and by the Java Application Programming Interface (API), which provides a large collection of software components that can be directly used by a Java programmer. The Java API provides several classes that can be used to efficiently write programs with graphics content and graphical user interfaces. The latter can be achieved with C++ only by combining it with graphic libraries such as Open Inventor or OpenGL, and with toolkit libraries such as TCL and TK. Furthermore, Java facilitates the development of programs that deal with networking, security issues, databases, 3D graphics, and many other issues that a typical high level language, such as C++, does not provide.

In addition, Java programs can be executed in any machine irrespectively of its architecture and operating system. The portability of Java programs is based on the JVM and the intermediate compilation into bytecode. The bytecode can, then, be interpreted by the Java VM, which translates the bytecode instructions into machine instructions that your computer can understand and execute.

The software was implemented using the latest version of Java 2 platform, the Java 2 Standard Edition 1.3. The Java 2 platform offers enhanced performance through the Java

Hot Spot process, which performs just-in-time (JIT) compilation and profiling, and the Java Foundation Classes (JFC). The Java 2 standard Development Kit (SDK) provides Forte, an integrated development environment (IDE), which facilitates software development and debugging.

7.8.2 Java 3D API

Java3D is a general-purpose, platform-independent, object-oriented API for 3D-graphics that enables high-level development of Java applications and applets with 3D interactive rendering capabilities. With Java3D, 3D scenes can be built programmatically, or, alternatively, 3D content can be loaded from VRML or other external files. Java3D, as a part of the Java Media APIs, integrates well with the other Java technologies and APIs. For example, Java2D API can be used to plot selected results, while the Java Media Framework (JMF) API can be used to capture and stream audio and video.

7.8.2.1 Introduction

Java3D is based on a directed acyclic graph-based scene structure, known as scene graph, that is used for representing and rendering the scene. The scene structure is a tree-like hierarchical diagram that contains nodes with all the necessary information to create and render the scene. In particular, the scene graph contains the nodes that are used to represent and transform all objects in the scene, and all viewing control parameters, i.e. all objects with information related to the viewing of the scene. The scene graph can be manipulated very easily and quickly allowing efficient rendering by following a certain optimal order and bypassing hidden parts of objects in the scene.

Java 3D API has been developed under a joint collaboration between Intel, Silicon Graphics, Apple, and Sun, combining the related knowledge of these companies. It has been designed to be a platform-independent API concerning the host's operating system (PC/Solaris/Irix/HPUX/Linux) and graphics (OpenGL/Direct3D) platform, as well as the

input and output (display) devices. Java3D aims at achieving high performance by utilizing the available graphics libraries (OpenGL/Direct3D), using 3D-graphics acceleration where available, and supporting some rendering optimizations (such as scene reorganization and content culling). The current version of the Java 3D API is the Version 1.2, which works together with the Java 2 Platform. Both APIs can be downloaded for free from the Java products page of Sun.

7.8.2.2 Scene graph: content-view branches

A Java3D scene is created as a tree-like graph structure, which is traversed during rendering. The scene graph structure contains nodes that represent either the actual objects of the scene, or, specifications that describe how to view the objects. Usually, there are two branches in Java 3D, the content branch, which contains the nodes that describe the actual objects in the scene, and the view branch, which contains nodes that specify viewing related conditions. Usually, the content branch contains much larger number of nodes than the view branch.

The image of Figure 7.9 shows a basic Java3D graph scene, where the content branch is located on the left and the view branch on the right side of the graph.

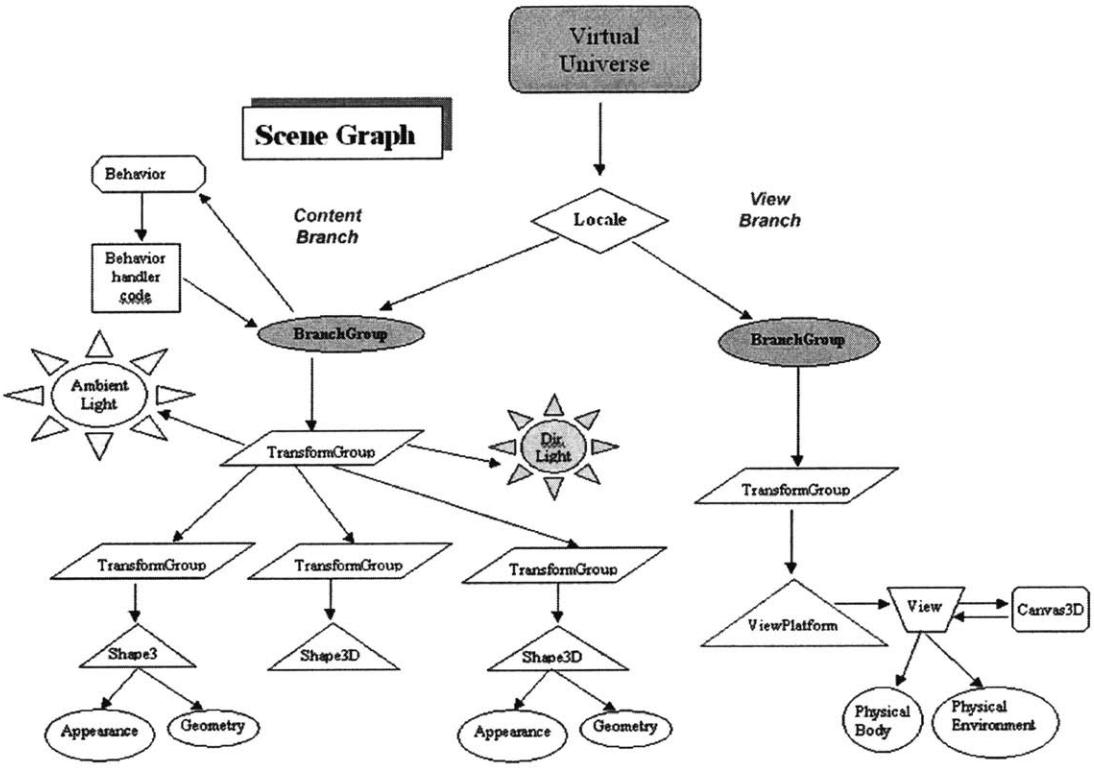


Figure 7.9: Scenegraph example.

Java3D applications construct individual graphic components as separate objects, called nodes, and connects them together into a tree-like scene graph, in which the objects and the viewing of them can easily be manipulated. The scene graph structure contains the description of the virtual universe, which represents the entire scene. All information concerning geometric objects, their attributes, position and orientation, as well as the viewing information are all contained into the scene graph.

The above scene graph consists of superstructure components, in particular a VirtualUniverse and a Locale object, and a two *BranchGroup* objects, which are attached to the superstructure. The one branch graph, rooted at the left *BranchGroup* node, is a content branch, containing all the relevant to the contents of the scene objects. The other branch,

known as view branch, contains all the information related to the viewing and the rendering details of the scene.

The state of a shape node, or any other leaf node, is defined, during rendering, by the nodes that lie in the direct path between that node and the root node, i.e. the VirtualUniverse. For example, a *TransformGroup* node in a path between a leaf node and the scene's root can change the position, orientation, and scale of the object represented by the leaf node.

7.8.2.3 *SceneGraphObject* hierarchy

The Java 3D node objects of a Java 3D scene graph, which are instances of the Node class, may reference node component objects, which are instances of the class *NodeComponent*. The *Node* and *NodeComponent* classes are subclasses of the *SceneGraphObject* abstract class. Almost all objects that may be included in a scene graph are instances of subclasses of the *SceneGraphObject* class. A scene graph object is constructed by instantiating the corresponding class, and then, it can be accessed and manipulated using the provided set and get methods. The following graph shows the class hierarchy of the major subclasses of the *SceneGraphObject* class.

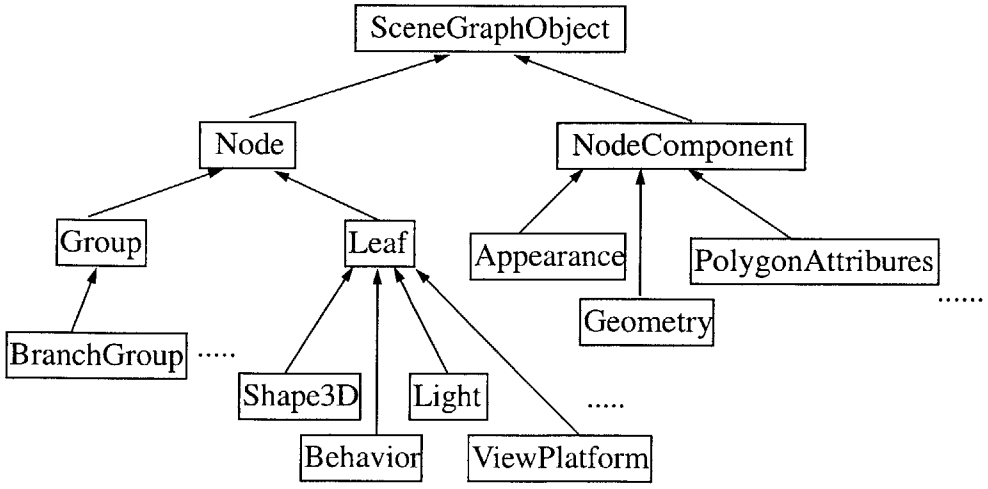


Figure 7.10: Class hierarchy of major subclasses of the *SceneGraphObject* class.

7.8.2.4 Class *Node* and its subclasses

The abstract Class Node is the base class for almost all objects that constitute the scene graph. It has two subclasses the Group, and Leaf classes, which have many useful subclasses. Class Group is a superclass of, among others, the classes *BranchGroup* and TransformGroup. Class Leaf, which is used for nodes with no children, is a superclass of, among others, the classes Behavior, Light, Shape3D, and *ViewPlatform*. The *ViewPlatform* node is used to define from where the scene is viewed. In particular, it can be used to specify the location and the orientation of the point of view.

7.8.2.5 Class *NodeComponent* and its subclasses:

Class *NodeComponent* is the base class for classes that represent attributes associated with the nodes of the scene graph. It is the superclass of all scene graph node component classes, such as the *Appearance*, *Geometry*, *PointAttributes*, and *PolygonAttributes* classes. *NodeComponent* objects are used to specify attributes for a node, such as the color and geometry of a shape node, i.e. a *Shape3D* node. In particular, a *Shape3D* node uses an *Appearance* and a *Geometry* objects, where the Appearance object is used to control how the associated geometry should be rendered by Java 3D.

The geometry component information of a *Shape3D* node, i.e. its geometry and topology, can be specified in an instance of a subclass of the abstract Geometry class. A *Geometry* object is used as a component object of a *Shape3D* leaf node. *Geometry* objects consist of the following four generic geometric types. Each of these geometric types defines a visible object, or a set of objects.

- *CompressedGeometry*
- *GeometryArray*
- *Raster*
- Text3D

For example, the *GeometryArray* is a subclass of the class *Geometry*, which itself extends the *NodeComponent* class, that is extended to create the various primitive types such as lines, triangle strips and quadrilaterals.

The *IndexedGeometryArray* object, shown in Figure 7.11, contains separate integer arrays that index, among others, into arrays of positional coordinates specifying how vertices are connected to form geometry primitives. This class is extended to create the various indexed primitive types, such as *IndexedLineArray*, *IndexedPointArray*, and *IndexedQuadArray*.

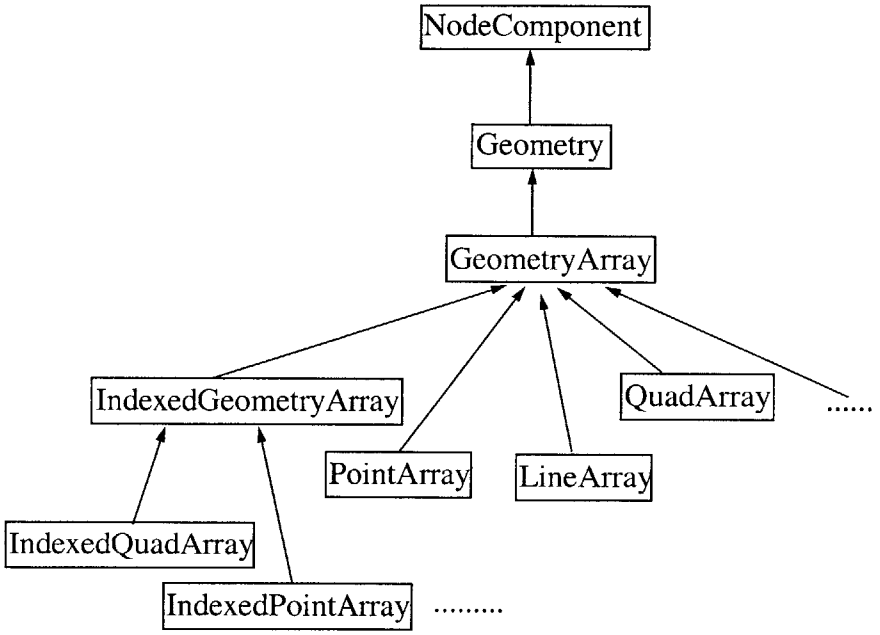


Figure 7.11: Class hierarchy of the *IndexedGeometryArray* class.

Vertex data may be passed to the geometry array either by copying the data into the array using the existing methods, which is the default mode, or by passing a reference to the data.

The methods for setting positional coordinates, colors, normals, and texture coordinates, such as the method *setCoordinates()*, copy the data into the *GeometryArray*, which offers much flexibility in organizing the data.

Another set of methods allows data to be passed and accessed by reference, such as the *setCoordRef3d()* method, set a reference to user-supplied data, e.g. coordinate arrays. In order to enable the passing of data by reference, the *BY_REFERENCE* bit in the *vertexFormat* field of the constructor for the corresponding *GeometryArray* must be set accordingly. Data in any array that is referenced by a live or compiled *GeometryArray* object may only be modified using the *updateData()* method assuming that the *ALLOW_REF_DATA_WRITE* capability bit is set accordingly, which can be done using the *setCapability()* method.

The *Appearance* object defines all rendering state that control the way the associated geometry should be rendered. The rendering state consists of the following:

- Point attributes: a *PointAttributes* object defines attributes used to define points, such as the size to be used
- Line attributes: using a *LineAttributes* object attributes used to define lines, such as the width and pattern, can be defined
- Polygon attributes: using a *PolygonAttributes* object the attributes used to define polygons, such as rasterization mode (i.e. filled, lines, or points) are defined
- Coloring attributes: a *ColoringAttributes* object is used to defines attributes used in color selection and shading.
- Rendering attributes: defines rendering operations, such as whether invisible objects are rendered, using a *RenderingAttributes* object.
- Transparency attributes: a *TransparencyAttributes* defines the attributes that affect transparency of the object
- Material: a Material object defines the appearance of an object under illumination, such as the ambient color, specular color, diffuse color, emissive color, and shininess. It is used to control the color of the shape.
- Texture: the texture image and filtering parameters used, when texture mapping is

enabled, can be defined in a *Texture* object.

- Texture attributes: a *TextureAttributes* object can be used to define the attributes that apply to texture mapping, such as the texture mode, texture transform, blend color, and perspective correction mode.
- Texture coordinate generation: the attributes that apply to texture coordinate generation can be defined in a *TexCoordGeneration* object.
- Texture unit state: array that defines texture state for each of N separate texture units allowing multiple textures to be applied to geometry. Each *TextureUnitState* object contains a *Texture* object, *TextureAttributes*, and *TexCoordGeneration* object for one texture unit.

7.8.2.6 Superstructure components: *VirtualUniverse* and *Locale*

A scenegraph consists of the superstructure components, in particular a *VirtualUniverse* and one or more *Locale* objects, and a set of branch graphs that are attached to the *Locale* object(s). After constructing a subgraph, it can be attached to a *VirtualUniverse* object through a high-resolution *Locale* object, which is itself attached to the virtual universe. The *VirtualUniverse* is the root of all Java 3D scenes, while *Locale* objects are used for basic spatial placement. The attachment to a *Locale* object makes all objects in the attached subgraph live (i.e. drawable), while removing it from the locale reverses the effect. Any node added to a live scene graph becomes live. However, in order to be able to modify a live node the corresponding capability bits should be set accordingly.

Typically, a Java3D program has only one *VirtualUniverse* which consists of one, or more, *Locale* objects that may contain collections of subgraphs of the scene graph that are rooted by *BranchGroup* nodes, i.e. a large number of branch graphs. Although a Locale has no explicit children, it may reference an arbitrary number of *BranchGroup* nodes. The subgraphs contain all the scene graph nodes that exist in the universe. A *Locale* node is used to accurately position a branch graph in a universe specifying a location within the

virtual universe using high-resolution coordinates (*HiResCoord*), which represent 768 bits of floating point values. A *Locale* is positioned in a single *VirtualUniverse* node using one of its constructors.

The *VirtualUniverse* and *Locale* classes, as well as the *View* class, are subclasses of the basic superclass Object, as shown in Figure 7.12.

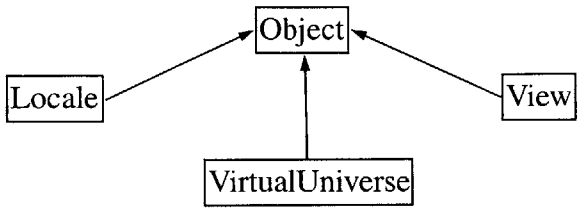


Figure 7.12: VirtualUniverse, Locale and View classes.

7.8.2.7 Branch graphs

A branch graph is a scene graph rooted in a *BranchGroup* node and can be used to point to the root of a scene graph branch. A graph branch can be added to the list of branch graphs of a *Locale* node using its *addBranchGraph(BranchGroup bg)* method. *BranchGroup* objects are the only objects that can be inserted into a Locale's list of objects.

A *BranchGroup* may be compiled by calling its compile method, which causes the entire subgraph to be compiled including any *BranchGroup* nodes that may be contained within the subgraph. A graph branch, rooted by a *BranchGroup* node, becomes live when inserted into a virtual universe by attaching it to a *Locale*. However, if a *BranchGroup* is contained in another subgraph as a child of some other group node, it may not be attached to a *Locale* node.

7.8.2.8 Capability bits, making live and compiling:

Certain optimizations can be done to achieve better performance by compiling a subgraph into an optimized internal format, prior to its attachment to a virtual universe. However, many set and get methods of objects that are part of a live or compiled scene graph cannot

be accessed. In general, the set and get methods can be used only during the creation of a scene graph, except where explicitly allowed, in order to allow certain optimizations during rendering. The set and get methods that can be used when the object is live or compiled should be specified using a set of capability bits, which by default are disabled, prior to compiling or making live the object. The methods *isCompiled()* and *isLive()* can be used to find out whether a scene graph object is compiled or live. The methods *setCapability()* and *getCapability()* can be used to set properly the capability bits to allow access to the object's methods. However, the less the capability bits that are enabled, the more optimizations can be performed during rendering.

7.8.2.9 Viewing branch: *ViewPlatform*, *View*, *Canvas3D*, and *Screen3D*

The view branch has usually the structure shown in Figure 7.9, consisting of nodes that control the viewing of the scene. The view branch contains some scene graph viewing objects that can be used to define the viewing parameters and details, such as the *ViewPlatform*, *View*, *Canvas3D*, *Screen3D*, *PhysicalBody*, and *PhysicalEnvironment* classes.

Java 3D uses a viewing model that can be used to transform the position and direction of the viewing while the content branch remains unmodified. This is achieved with the use of the *ViewPlatform* and the *View* classes, to specify from where and how, respectively, the scene is being viewed.

The *ViewPlatform* node controls the position, orientation and scale of the viewer. A viewer can navigate through the virtual universe by changing the transformation in the scene graph hierarchy above the *ViewPlatform* node. The location of the viewer can be set using a *TransformGroup* node above the *ViewPlatform* node. The *ViewPlatform* node has an activation radius that is used together with the bounding volumes of Behavior, Background and other nodes in order to determine whether the latter nodes should be sched-

uled, or turned on, respectively. The method *setActivationRadius()* can be used to set the activation radius.

A View object connects to the *ViewPlatform* node in the scene graph, and specifies all viewing parameters of the rendering process of a 3D scene. Although it exists outside of the scene graph, it attaches to a *ViewPlatform* leaf node in the scene graph, using the method *attachViewPlatform(ViewPlatform vp)*. A *View* object contains references to a *PhysicalBody* and a *PhysicalEnvironment* object, which can be set using the methods *setPhysicalBody()* and *setPhysicalEnvironment()*, respectively.

A View object contains a list of *Canvas3D* objects where rendering of the view is done. The method *addCanvas3D(Canvas3D c)* of the class *View* can be used to add the provided *Canvas3D* object to the list of canvases of the *View* object. Class *Canvas3D* extends the heavyweight class Canvas in order to achieve hardware acceleration, since a low rendering library, such as *OpenGL*, requires the rendering to be done in a native window to enable hardware acceleration.

Finally, all *Canvas3D* objects on the same physical display device refer to a *Screen3D* object, which contains all information about that particular display device. *Screen3D* can be obtained from the *Canvas3D* using the *getScreen3D()* method.

7.8.2.10 Default coordinate system

The default coordinate system is a right-handed Cartesian coordinate system centered on the screen with the x and y-axes directed towards the right and top of the screen, respectively. The z-axis is, by default directed out of the screen towards the viewer, as shown below. The default distances are in meter and the angles in radians.

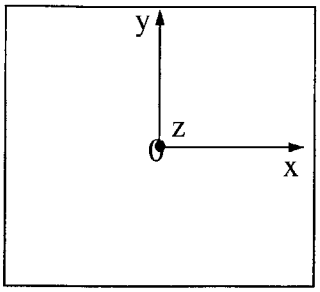


Figure 7.13: Default coordinate system of Java3D

7.8.2.11 Transformations

Class *TransformGroup* which extends the class Group can be used to set a spatial transformation, such as positioning, orientation, and scaling of its children through the use of a *Transform3D* object. A *TransformGroup* node enables the setting and use of a coordinate system relative to its parent coordinate system.

The *Transform3D* object of a *TransformGroup* object can be set using the method set-*Transform(Transform3D t)*, which is used to set the transformation components of the *Transform3D* object to the ones of the passed parameter.

A *Transform3D* object is a 4x4 double-precision matrix that is used to determine the transformations of a *TransformGroup* node, as shown in the following equation. The elements T00, T01, T02, T10, T11, T12,T20, T21, and T22 are used to set the rotation and scaling, and the T03, T13, and T23 are used to set the translation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} T_{00} & T_{01} & T_{02} & T_{00} \\ T_{10} & T_{11} & T_{12} & T_{13} \\ T_{20} & T_{21} & T_{22} & T_{23} \\ T_{30} & T_{31} & T_{32} & T_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{7.1}$$

As the scene graph is traversed by the Java 3D renderer, the transformations specified by any transformation nodes accumulate. The transformations closer to the geometry nodes executed prior to the ones closer to the virtual universe node

7.8.3 Swing

7.8.3.1 Description of Swing

Java Foundation Classes (JFC) provide features to facilitate the development of Graphical User Interfaces (GUIs). Among other, JFC provides Swing, which includes several components that can be used for the development of GUIs, support for a choice on the look and feel that a program uses, and provides Java2D for high quality 2D graphics. Finally, swing allows the specification which look and feel a program should use. This is done with the *setLookAndFeel()* method of the *UIManager* class. Swing is built on top of the AWT (Abstract Window Toolkit) using the AWT infrastructure. However, Swing provides its own graphical user interface (GUI) components, many of which have a close relation or correspondence to the AWT components. Essentially, Swing is an extension and improvement to the AWT.

Swing provides several standard components (e.g. buttons, checkbuttons, radiobuttons, menus, lists, labels, and text areas) to create a program's GUI using one or more containers (e.g. frames, dialogs, windows and tool bars). Most Swing components that begin with *J*, except the top-level containers, are subclasses of the *JComponent* class. The letter *J* is used to differentiate the actual extra user interface classes provided by Swing from the support classes that it provides. Swing components inherit many features from the *JComponent* class, such as a configurable look and feel, borders, and tool tips, as well as many methods. In addition, some Swing components can display images on them. The *JComponent* class extends the *Container* class (provides support for adding and laying out compo-

nents), which itself extends the *Component* class (provides support for painting, events, layout etc.).

JComponent is the base class for almost all lightweight components. Therefore, all Swing lightweight components (derived from the *JComponent* class) are subclasses of the Container class of AWT.

Every program with a Swing GUI contains at least one top-level Swing container, i.e., in general, an instance of a *JApplet*, *JFrame*, or *JDialog*, which enables the painting and event handling of the Swing components. The *JApplet*, *JFrame*, or *JDialog*, are considered top-level containers because they are used to provide an area in which the other containers and components can appear. Other containers, such as the *JPanel*, are used to facilitate the positioning and sizing of other containers and components.

Between a top-level container and an intermediate container, a content pane (*JRootPane*), which is also an intermediate container, is indirectly provided. The content pane contains all of visible components of its GUI, except a menu bar. One of the add() methods of a container can be used to add a component to it. The component to be added is used as argument to the add method. In some cases another argument, which has to do with the layout, is used with the add() method.

Besides the top-level containers (*JApplet, JFrame, JDialog*, and *JWindow*) there are special-purpose containers (such as the *JInternalFrame, JLayeredPane*, and *JRootPane*) that are used for special purposes, and general-purpose containers (such as *JPanel, JScroll Pane, JSplitPane, JTabbedPane*, and *JToolBar*) that are used for other any general purpose.

The *JApplet* and *JFrame* classes should be used to implement Swing applets or applications, respectively. Both *JApplet* and *JFrame* classes are containers that contain an instance of the *JRootPane* class. the latter contains the content pane, which is a container,

that contains all the components contained in the applet or application. Therefore, components should be added and layout managers should be set to the content pane.

Most Swing components that begin with *J*, except the top-level containers, are subclasses of the *JComponent* class. The letter *J* is used to differentiate the actual extra user interface classes provided by Swing from the support classes that it provides. Swing components inherit many features from the *JComponent* class, such as a configurable look and feel, borders, and tool tips, as well as many methods. In addition, some Swing components can display images on them. The *JComponent* class is the base class for almost all lightweight J-components of Swing. The *JComponent* class extends the Container class (provides support for adding and laying out components), which in turn extends the *Component* class (provides support for painting, events, layout etc.). Therefore, all Swing J-components are AWT containers and inherit all methods from the Container and Component classes. Any instance of a *JComponent* subclass can contain both AWT and Swing components, since *JComponent* extends the *java.awt.Container* class.

For the rendering of lightweight Swing components, the paint() method of the *JComponent* class is used. In particular, a *Graphics* object is passed to the paint() method in which it draws the component, the component's border, and the component's children in order. When the component has a UI delegate, the delegate's paint() method is invoked, which clears the background in case of an opaque component, and, then, paints the component. For double buffered components it paints the component into an offscreen buffer and then copies it into the component's onscreen area. Since double buffered is provided by Swing there is no need to override the paint() method, as it is for AWT to achieve avoid flickering. To redefine the way a component is painted the *paintComponent()* method of the *JComponent* class should be overwritten, which usually needs to invoke the *super.paintComponent()* method.

In contrast to AWT components, it is not necessary to override the update() method, which erases the background of a component and then invokes paint(), to avoid flickering. The *JComponent* class overrides the update() method invoking directly the paint() method, while the UI delegate is responsible for erasing the background. The flickering is avoided since double buffering is used by the Swing components, i.e. the components are repainted first in an offscreen buffer and, then, copied to the screen. Only *JRootPane* and *JPanel* of the Swing lightweight components are by default double buffered. Components that reside in a double buffered container are automatically double buffered. The method *setDoubleBuffered()* of the *JComponent* class can be used to set whether the receiving component should use a buffer to paint. The offscreen buffer used for the double buffering can be obtained using the method *RepaintManager.getOffscreenBuffer()*.

7.8.3.2 Mixing Java3D and Swing:

Care should be taken when using Swing together with Java3D as the latter uses the *Canvas3D*, a heavyweight component. In particular, when lightweight components overlap with heavyweight components, the heavyweight components are always painted on top. This is caused because of the way "depth" at which components are displayed in a container is represented by the Z-order. The latter is determined by the order with which each component is added to the container, i.e. the first component to be added to a container has the highest Z-order which means that it is displayed in front of all other components added to that container. When lightweight and heavyweight components are mixed, the lightweight components, which need to reside in a heavyweight container, have the same Z-order of their container, and within it the order of each of the lightweight components is determined by the order in which they are added to the container.

The heavyweight *Canvas3D* of Java 3D can be kept apart from lightweight Swing components using different containers to avoid such problems. Furthermore, to avoid

heavyweight components overlapping Swing popup menus, which are lightweight, the popup menus are forced to be heavyweight using the method *setLightWeightPopupEnabled()* of the *JPopupMenu* class. Similarly, problems with tooltips can be avoided by invoking the following method: *ToolTipManager.sharedInstance().setLightWeightPopupEnabled(false)*.

7.8.3.3 Swing and multithreading

Swing, in general, is not thread safe, because a Swing component can be accessed by only one thread at a time, which, in general, is the event-dispatching thread. Because Swing is not thread safe, Swing components, in general, should be accessed only from the event-dispatching thread. The event-dispatching thread is the thread that invokes callback methods, e.g. the *paint()* and *update()* functions, and event handler methods. Swing has been designed to be not thread safe in order to avoid the overhead of multithreading (e.g. obtaining and releasing locks) and to simplify the subclassing of its component. It is not allowed to access Swing components from any thread other than the event-dispatching thread.

However, some Swing components support multithreaded access. Actually after the realization of a Swing component code that might affect or depend on the state of that component should be executed in the event-dispatching thread. Realization of a component means after the component is available for painting on screen, after it is painted or become ready to be painted using one of the methods: *setVisible(true)*, *show()*, or *pack()*.

The access only through the event-dispatching thread is not required in the following cases:

- when dealing with thread safe methods (as specified in the Swing API documentation) to construct and show a GUI in the main thread of an application, as long as no components have been realized in the current runtime environment

- constructing and manipulating the GUI in an applet's *init()* method, as long as the components have not been made visible, i.e. the method *show()* or *setVisible(true)* has never been called on the actual applet object
- methods *repaint()* and *revalidate()* are safe to call from any thread

In general, Swing it is not safe to access Swing components from any thread other than the event-dispatching thread. However, there are times that it is preferable to update Swing components from another thread, or perform time-consuming operations on a separate thread, and not use the event-dispatching thread. In those cases, Swing provides the methods *invokeLater()* and *invokeAndWait()* in the *SwingUtilitiesclass*, which can be used to queue a runnable object on the event dispatch thread. They essentially allow a block of code from another thread to be invoked and executed by the event-dispatching thread. Both methods can be used to access Swing components from a thread other than the event-dispatching thread. Method *invokeLater()* queues the runnable object and returns immediately, while *invokeAndWait()* waits until the runnable object's *run()* method has started before returning. Only the *invokeLater()* (and not *invokeAndWait()*) method can be called from the event dispatching thread.

7.8.4 Java2D API

The Java2D API provides 2D graphics capabilities and image manipulation. It is part of the Java Foundation Classes (JFC).

It uses an object of the class *Graphics2D*, which is an extension of the class *Graphics*, as its rendering machine. A *Graphics2D* reference can be used by casting a *Graphics* reference to a *Graphics2D* reference

Chapter 8

Applications

8.1 Introduction

A series of simple examples have been performed in order to demonstrate the correctness of the solution, within some acceptable numerical errors. Since theoretical solutions are available, or can be derived, only for simple problems involving a pair of bodies, such simulations are considered. As long as a solution for a pair of bodies is sufficiently accurate, it is evident that the solution of the corresponding problem involving multiple bodies should also be sufficiently accurate.

The usefulness of the simulations is better demonstrated through either an actual execution of the simulation or a movie created during a prior simulation. However, that way of qualitatively testing the correctness of the simulation cannot be presented in the context of a document. However, snapshots of simulations have been taken and presented in the following examples.

8.2 Verification Examples

8.2.1 Exchange of momentum

A body such as the one shown on the left of Figure 8.1.a (infinitely rigid case) and Figure 8.1.b (deformable bodies case) with an initial velocity of 1m/sec collides with a motionless body (on the right side of these figures. Other than the deformability, which only the right pair of bodies (Figure 8.1.b) posses both bodies in both cases have the same geomet-

ric and material properties.

In the case of the infinitely rigid bodies, when the left body collides with the second they two bodies exchange momentum, the right body moves to the right with a velocity equal to that that the left body had prior to the collision, while the latter remains still at the point where the collision took place.

a. b.

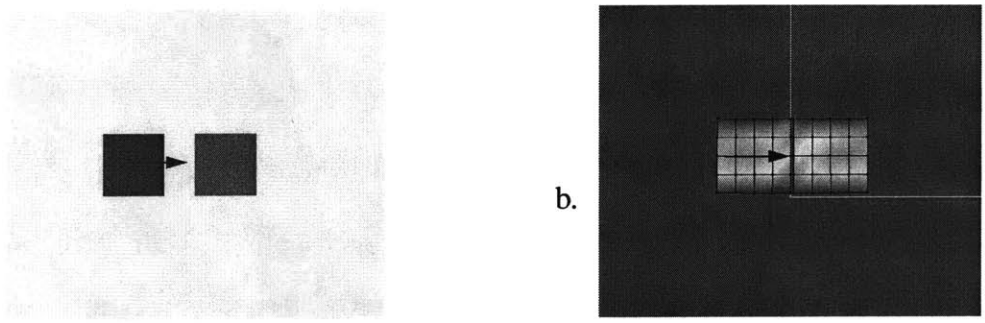


Figure 8.1: A body with an initial velocity colliding centrically with another still body.

In contrast, in the case of the deformable bodies, when the left body collides with the second, the two bodies also begin vibrating. However, the bodies exchange most of the momentum, as the second body moves with a velocity slightly less than the initial velocity of the right body, while the right body does not come to a complete still position. The following figure shows the velocities of all bodies.

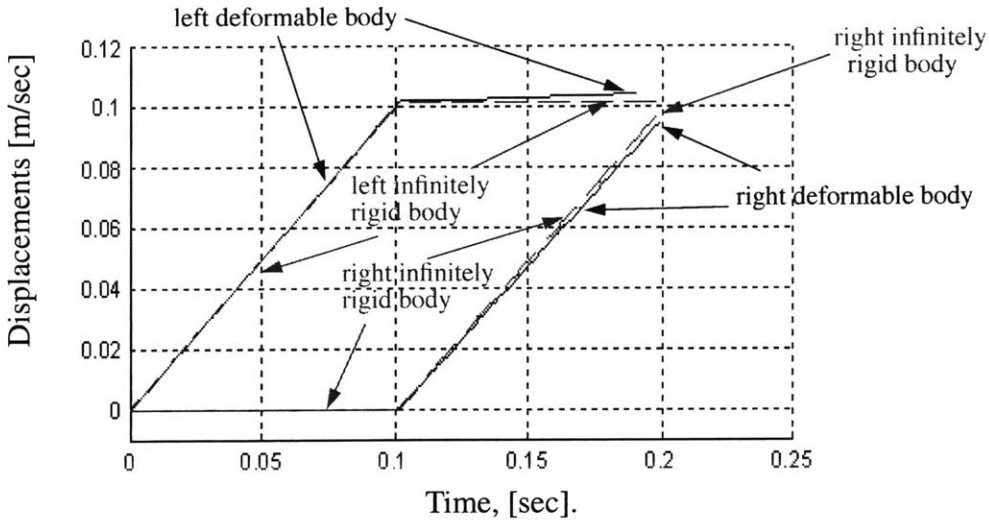


Figure 8.2: Displacements of the centroid of the bodies.

Figure 8.3 presents the kinetic energies of the system of infinitely rigid bodies. It is evident that there is conservation of energy as the kinetic energy prior and after the collision is the same. Note, that the kinetic energy during collision is less than the overall energy due to the action of the contact springs which deform possessing elastic energy, which has not been drawn in this figure.

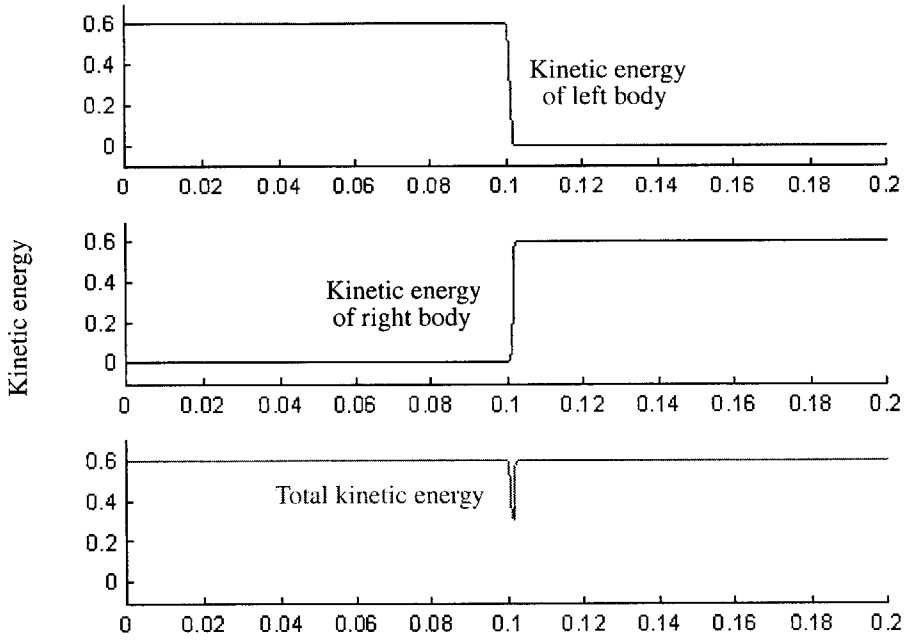


Figure 8.3: Kinetic energy of the two infinitely rigid bodies (without considering the energy of the contact springs) during simulation.

Figure 8.4 illustrates the kinetic and elastic energy of the two deformable bodies. Again the energy of the contact springs is not drawn as it is approximately the difference between the total energy, i.e. $0.5mv_{initial}^{2} = 0.6$, and the sum of the kinetic and the elastic energy of the deformable bodies. The elastic energy is the one that keeps the bodies vibrating, which results in the stress and strain distributions within the bodies, after the collision.

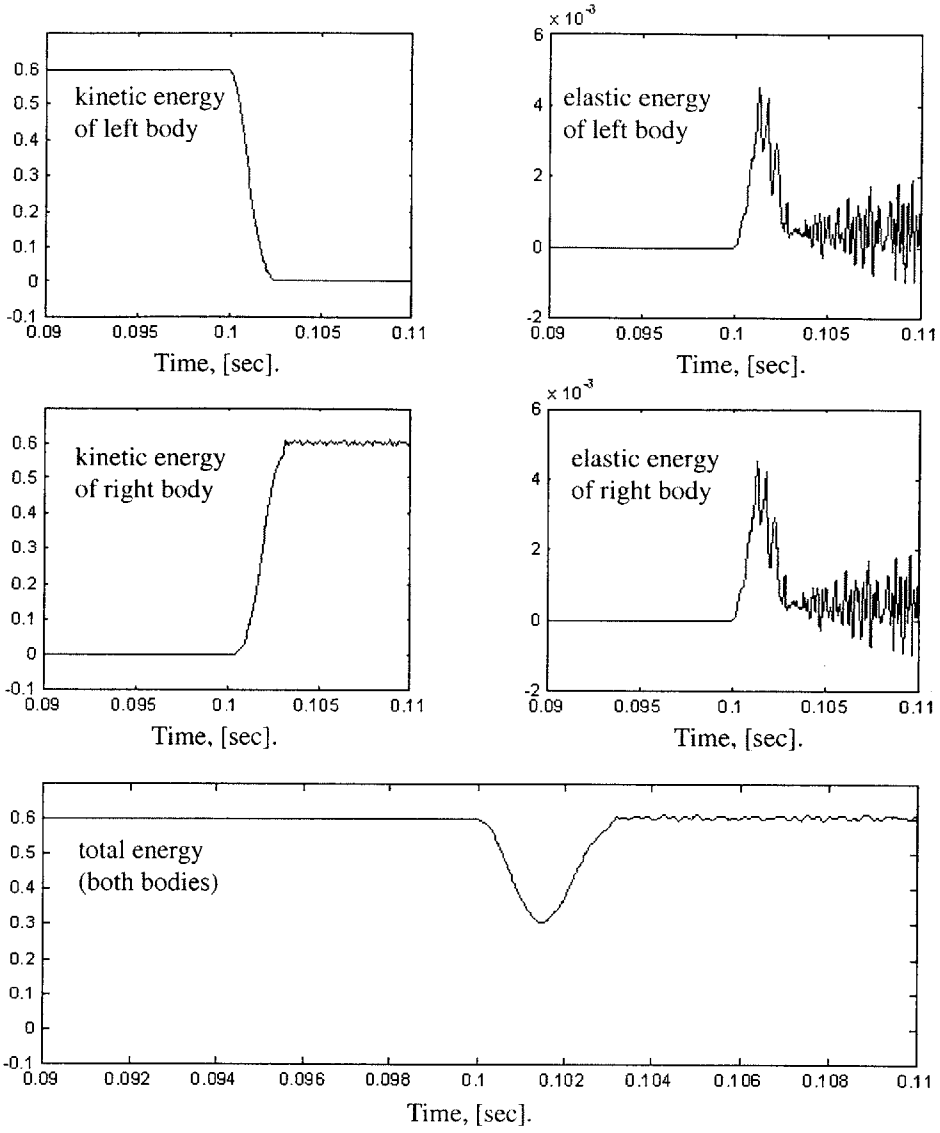


Figure 8.4: Energy of the two deformable bodies (excluding the deformation energy of the contact springs during collision) during simulation.

8.2.2 Wave propagation

A body with a shape of a rod (shown on the left of Figure 8.5) is driven with an initial velocity against an infinitely rigid body (shown on the right of Figure 8.5) that is stationary and fixed.

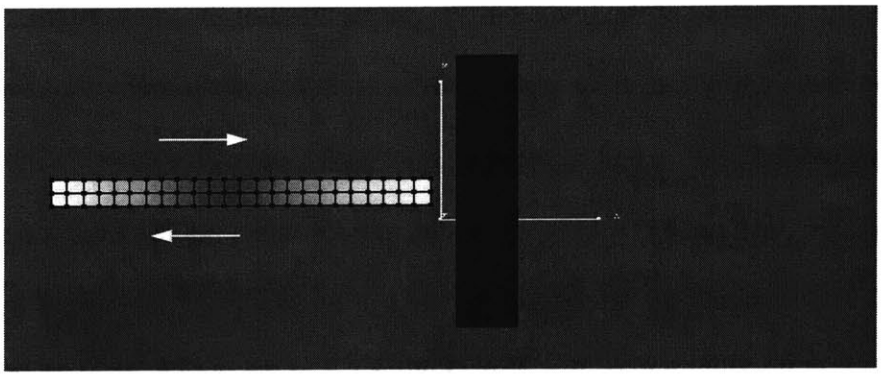


Figure 8.5: Rod-shape deformable body colliding to a fixed infinitely rigid body.

As the deformable body collides with the rigid fixed body it compressively deforms generating compression stresses that propagate from right to left. Then, as the rod bounces off the wall, under the action of the contact forces, the compressive stresses are bounced off the other end and a wave propagates back and forth from the one end of the rod to the other. This results in compression and tension within the body as the stress wave propagate through it and vibration of the body according to its natural frequency.

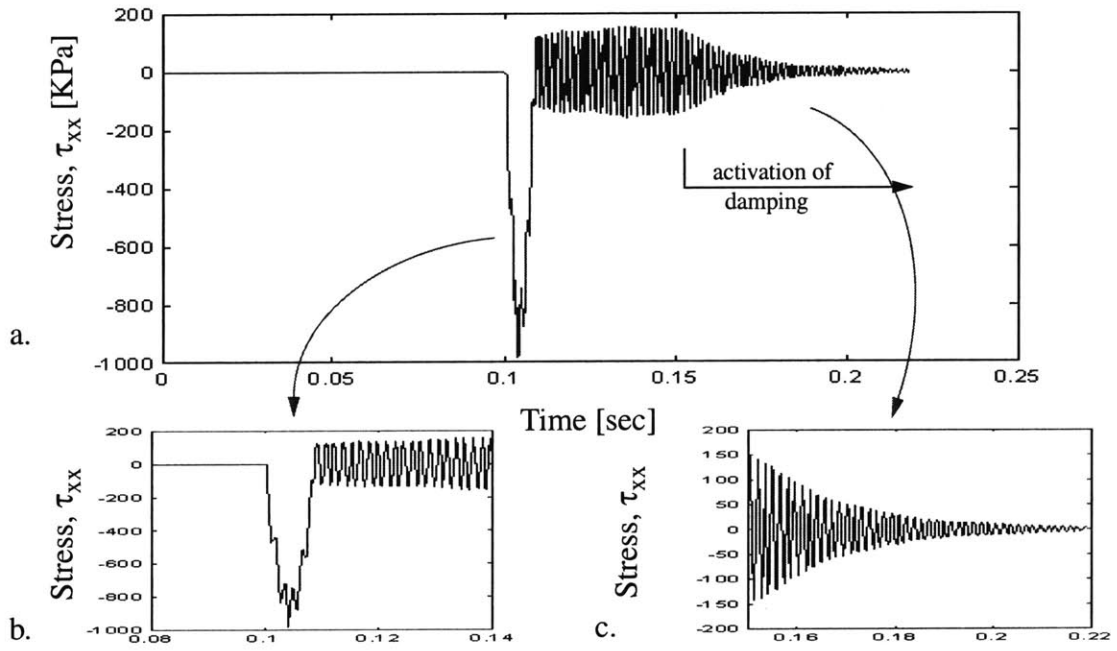


Figure 8.6: Stresses τ_{xx} at a center node of the deformable body.

At time *t=0.15 sec* damping is activated, resulting in dissipation of elastic energy, which causes the vibration of the body and the propagation of the stress wave. Soon after the activation of damping the stress wave and the vibration of the body are dissipated as shown in Figure 8.6.c.

The stresses τ_{xx} at a node at the right end, i.e. at the contact point, of the rod are shown in Figure 8.7.a. Comparing the stresses at the node at the center of the body (Figure 8.6.a) and at the node at the right end of the body (Figure 8.7.a) we notice, as expected, a delay in the initiation of the stress pulse. This delay is presented in the graphs *b* and *c* of Figure 8.7, which are zooms of the stress histories of both nodes during the time interval when the first stress increment is initiated.

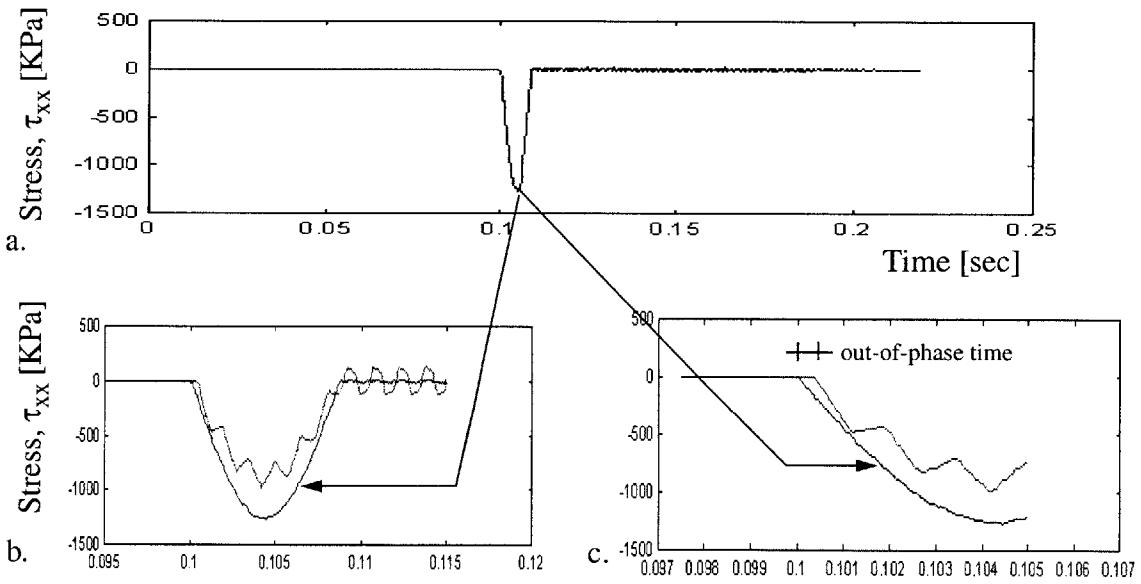


Figure 8.7: Stresses τ_{xx} at an edge node of the deformable body.

The time of delay between the node at the right end of the rod and the node the center, i.e the out-of-phase, time, agrees well with the theoretical time that is required to travel this distance using the wave propagation speed of the particular material that is used. In particular, the modulus of elasticity, Poisson ratio and density of the body have been considered to be equal to:

$E = 20\ GPa$
$v = 0.1$
$\rho = 2000\ Kg/m3$
$length = 2.4m$

Therefore, the stress wave speed, in the longitudinal direction, is equal to C_L

$$C_L = \sqrt{\frac{E \cdot (1-v)}{\rho \cdot (1+v) \cdot (1-2v)}} = 2558\frac{m}{s} \tag{8.1}$$

which means that the stress wave to travel a distance equal to the half length of the rod, which is equal to 2.4 m, requires time equal to:

$$t_{required} = \frac{1.2}{2558} = 0.00047 \quad seconds$$

8.2.3 Rigid body colliding on a constrained deformable body

A square infinitely rigid body is driven with an initial velocity into a constrained deformable body, as shown in Figure 8.9. The body on the left is an infinitely rigid body that moves to the right towards the constrained body. The latter is modeled using FEM as a plane stress body that is supported on its two ends, as shown in the following figure.

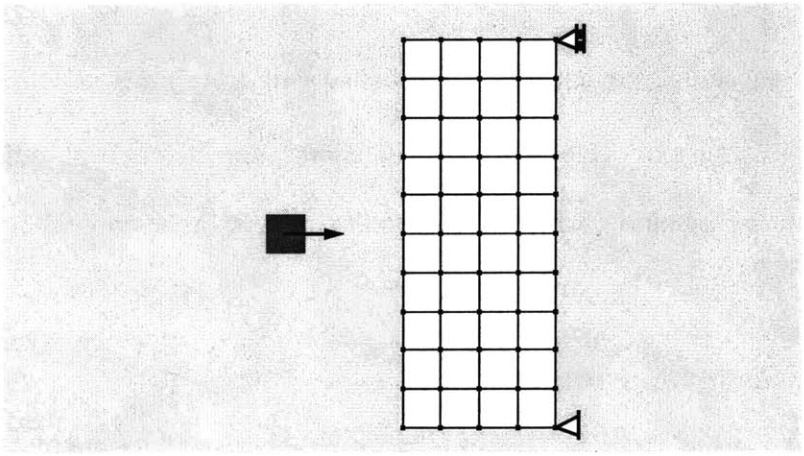


Figure 8.8: Rigid body (left) moving towards a constrained deformable body.

Figure 8.9 shows snapshots taken during the simulation as the rigid body on the left moves, with an initial velocity, towards the restrained deformable body, collides with it and is bounced back with a velocity that has magnitude close to that of its initial velocity but in the opposite direction.

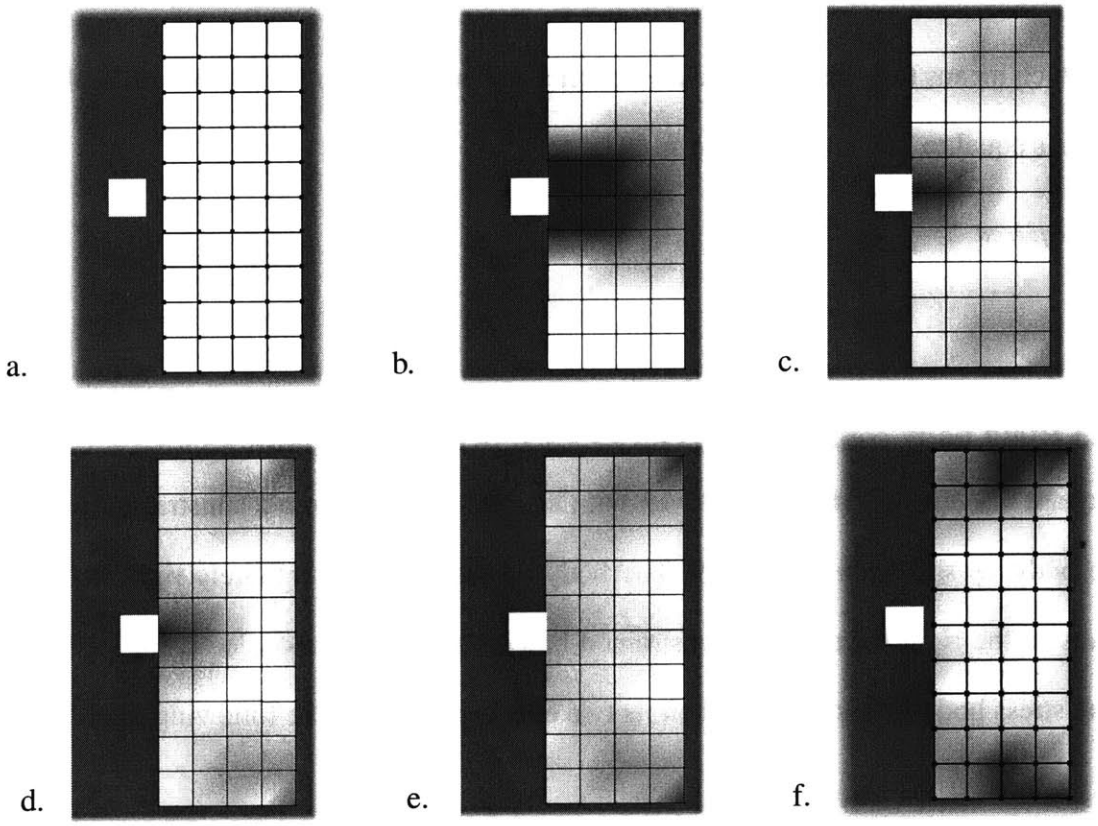


Figure 8.9: Snapshots from a rigid body collision with a constrained deformable body.

Simulations as this one are better presented either interactively or through movies that are created during simulation, which is impossible to show in the context of a thesis document.

8.2.4 Energy dissipation during contact

Two deformable bodies are left to fall under gravity on two other restrained deformable bodies, as shown in Figure 8.10.

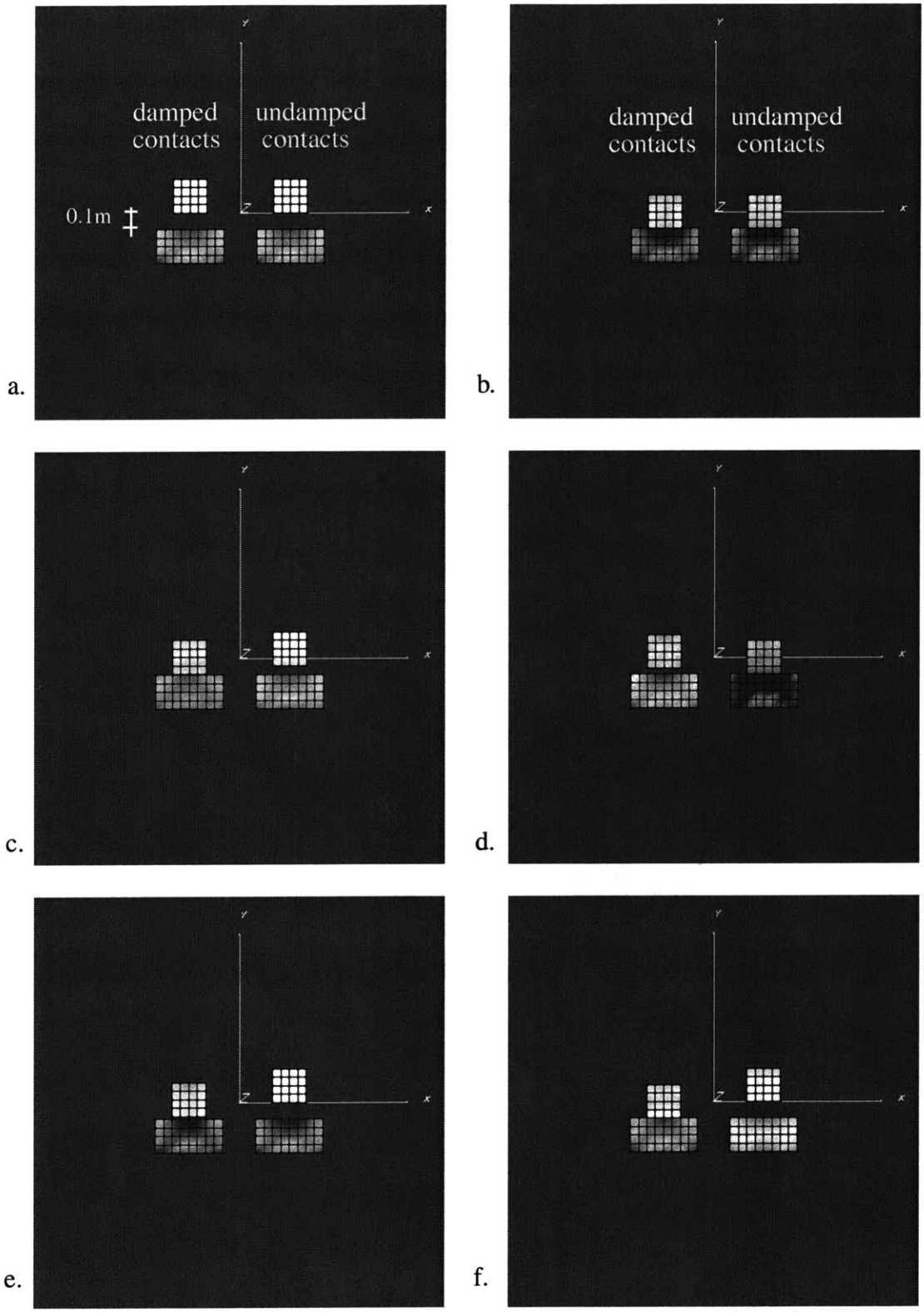


Figure 8.10: Deformable bodies falling under gravity on restrained deformable bodies.

The contacts of the left pair of bodies has damping taken into account, while for the right pair of bodies no dampers are considered during collisions. For both pairs of bodies no damping within the bodies, i.e. damping associated with the dissipation of strain energy inside the bodies was taken into account. The graphs Figure 8.10.a-Figure 8.10.f are subsequent timely snapshots taken during simulation.

As it is shown in the following graph, Figure 8.11, the falling body of the damped system eventually comes to a stationary position over the restrained body, while the falling body of the undamped system continues the bouncing off the restrained body.

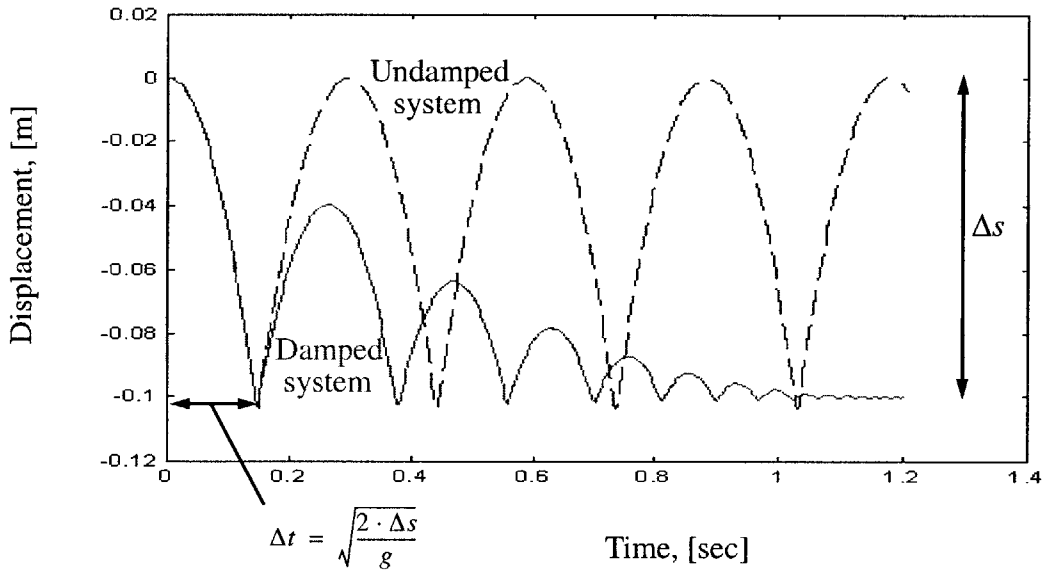


Figure 8.11: Displacements of falling bodies under gravity on restrained bodies.

Chapter 9

Concluding Remarks

9.1 Summary

The major objective of this thesis was to extend the capabilities of DEM to enable the efficient consideration of deformable bodies. This has been achieved by the combination of FEM and DEM as it is described in one of the main chapters of this thesis. However, there are many more enhancements to this work that can be implemented as described in a following paragraph.

The first chapter of this thesis provides a brief introduction to the DEM, the motivation to develop and use these methods, and applications of these methods. The limitations of DEM, regarding the rigidity of the simulated bodies, are presented as a motivation for the research described in this thesis. The chapter concludes with the objectives of this thesis work.

Chapter 2 discusses briefly the various object representation schemes and presents the specific object representation that has been selected to be used in the development of the combined DEM/FEM formulation and the implementation of the corresponding software.

Chapter 3 presents the major categories of contact detections algorithms followed by the selected contact detection scheme. A spatial reasoning algorithm using bounding boxes is used to identify pairs of bodies that may be in contact, followed by pairwise contact detection tests, which first verify the existence of contact and then determine the contact point using a contact resolution phase.

The following chapter, Chapter 4 describes the physics related to the contact effects. A soft contact approach is used, allowing the simulated bodies to overlap assuming local deformations, in order to evaluate the contact forces. The problem is significantly simplified by modeling the contact effects using idealized contact springs to remove the associated with contacts nonlinearities from the problem. These simplified contact springs provide equivalent contact forces to avoid overlapping between the colliding bodies. The contact forces are determined, using the positions and velocities of the previous time step according to the specified spring properties, and applied as external surface tractions.

Chapter 5 presents the major issue that is addressed in this thesis, the incorporation of a simplified FE formulation in the DEM in order to enable the consideration of deformability of certain simulated bodies and the computation of stress and strain distributions. The mesh generation is described in Chapter 6.

Chapter 7 discusses the software design and implementation. In particular, after an overview of the software design, computational, graphical-user interface and computed graphics issues are presented. Then, the utilization of database technology and its advantages are discussed. Finally, the structure of the implemented software and certain issues related to the technologies that have been used are described in detail. Portability and machine architecture independence has been achieved using solely Java technology in the implementation allowing the developed program to run on any platform that supports the Java technology. Visualization is very important for this kind of simulations of multibody systems since it enables the user to observe, investigate and record the interactions between the discrete bodies. Java3D has been used and certain recording modules have been developed allowing the user to create movies as a series of successive snapshots taken at user-selected and controlled frame rates during simulation.

Examples using the developed software are presented in Chapter 8. These applications demonstrate basic physics concepts that can be captured by the software, such as exchange of momentum, wave propagation, collisions of deformable and infinitely rigid bodies with restrained ones, and energy dissipation.

The current chapter provides an outline of this thesis, presents the main contributions of this research effort and discusses its potential future extensions.

The three appendices of the thesis provide a description of the graphical-user interface of the developed program, the formulation that should be used in case of an extension to large strains, and the source code that has been used for the performance comparison of C++ with Java, respectively.

9.2 Contributions and Conclusions

Clearly the main contribution of this thesis is the incorporation of deformability in DEM, using FEM, allowing the computation of stresses and strains distributions. The updated-Lagrangian formulation and an explicit time integration scheme are used together with some simplifying assumptions in order to render the simulation of multibody systems realistic with the current computational resources.

In particular, the equations of motion are considered at time t, taking into account large displacements and using the configuration at time t. The masses are lumped into nodal masses and the damping is considered mass proportional, resulting in diagonal mass and damping matrices. This greatly simplifies the solution of the problem as the equations of motion can be solved uncoupled without the need of any matrices inversion or multiplications.

The unknown displacements at time $t+\Delta t$ are computed using an explicit integration scheme, the central difference method, which can handle unconstrained systems such as

the ones under consideration. Having determined the new displacements, the new positions and orientations of the simulated bodies are determined and used for the contact detection. The latter is performed in the same way as it is applied when infinitely rigid bodies are under consideration. Having computed and applied the contact forces on the bodies in contact, as well as any other kind of external forces, the next step of the integration takes place computing the new displacements.

In addition, database technology has been utilized in order to efficiently store, search and manipulate the input and output data. The latter are typically stored in flat files, which can be used only by the users of the program that are familiar with the application's specific file format.

Finally, Java technology has been used to develop an architecturally neutral, platform independent, robust object-oriented software that can be easily extended. Potential extensions of the currently developed software are discussed in the following paragraph.

9.3 Future Work

There are many potential future extensions of this research. Some of them are discussed in this paragraph. The developed simulation tool has been designed and implemented to be easily extended using a pure object-oriented programming language and an easily extendable class hierarchy.

First, the developed software can be extended to 3D, which is conceptually easy as everything in the current version of DAFES is defined as 3D. Although the current implementation can be used to perform only 2D simulations, all vectors and matrices that are used, the underlying computed graphics and, in general, the overall simulation flow are the same as the ones that need to be used in 3D simulations. Only certain methods, such as the

contact resolution, and certain data, such as the 3D elasticity matrix need to be provided in order to extend the program to 3D.

A natural extension of this software regarding the continuum mechanics part that takes into account the deformability of simulated bodies, is the consideration of large strains. In addition, nonlinear material may be considered by using the proper FE formulation, something that cannot be done with the current formulation, which assumes linear elastic and isotropic material. Both large-strains and material nonlinearity can be taken into account using a total Lagrangian formulation. The latter requires use of new stress and strain measures, in particular the second Piola-Kirchhoff stresses and the Green Lagrange strains, which are briefly described in Appendix B. In addition, there are many potential applications if fracturing and fragmentation capabilities are also incorporated.

It would be also useful to compare DEM simulations with solutions provided by traditional FEM programs that allow simulations of systems with multiple unrestrained bodies, such as DYNA3D, in order to verify the correct behavior. FEM contact simulations can also be used to determine proper values for the stiffness used for the normal and tangential contact springs, which are currently selected arbitrarily.

Each DEM tool available has a single contact detection algorithm which is usually efficient only for the kind of problems that have been addressed with the specific code. It would be useful to have different contact detection algorithms and allow the user to select according to the nature of the problem under consideration the most efficient approach. Both grid subdivision and spatial sorting approaches will be implemented and used according to the input problem characteristics.

Another extension of this work is regarding the utilization of the database technology. A very simple relational database has been used as it was readily available. A better approach is to use an object-oriented database system, which is more powerful and offers

a natural correspondence between the objects of the simulation and the objects of the database.

Finally, the incorporation of simple mechanical constraints to allow simulations that also involve mechanical system is another potential area for future research work. Incorporation of mechanical system capabilities in DEM programs will allow simulations of systems of bodies together with mechanical parts that may move during simulation, such as moving or vibrating hoppers. Vibration of multibody systems can then be studied in a more realistic way with capabilities to observe and record the state of the simulated bodies.

Appendix A

DAFES Graphical User Interface (GUI)

The following figure shows a snapshot of the developed application, named DAFES (discrete and finite element simulator) during a simulation.

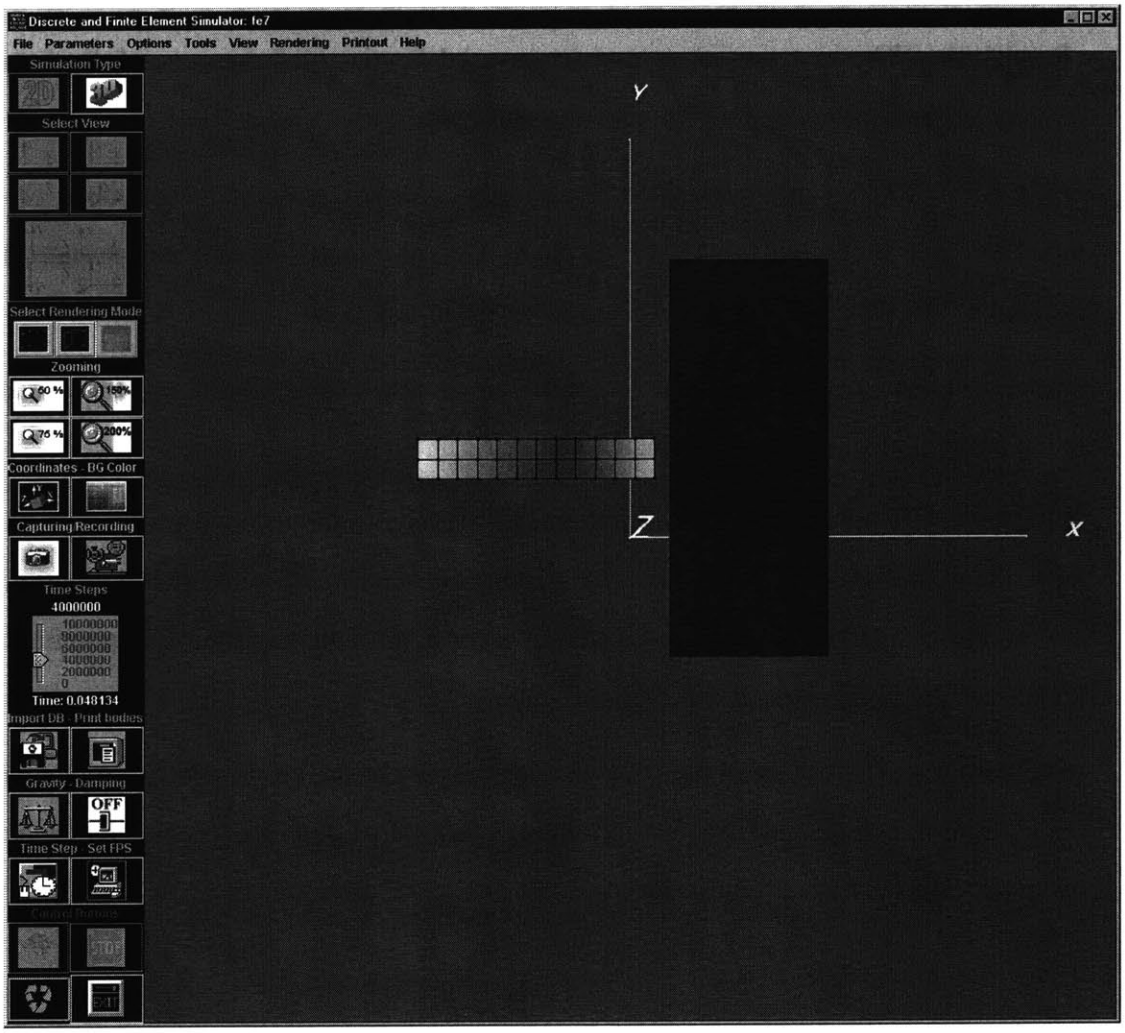


Figure A.1: DAFES.

The following paragraphs describe the main elements of the GUI of the program and their functionality. The user interface consists of two main components the menu bar and the controller, a set of buttons on the left site of the program's main window.First, the menubar and its submenus, *File*, *Parameters*, *Options*, *Tools*, *View*, *Rendering*, *Printout*, and *Help*, are described, followed by a description of the functionality of the controller.

Several factors have been considered in order to design a user-friendly GUI that enables an intuitive use of the software. These factors include:

- Learnability: the GUI has been designed to be intuitive in order to enable easy familiarization. An on-line help system has been provided as well.
- Readability: The text messages and icons used by the GUI have carefully been selected so as to provide more information and control to the user.
- Efficiency: Toolbar and buttons for frequently-used functionalities have been provided to achieve efficient use of the program. Default options have been set and can be used or modified according to the needs of the simulation.
- Autonomy: The user is provided with sufficient status information at any time so as to have full control of the program.
- Consistency: Buttons with related functionalities have been grouped together. Similarly, the menubar has been divided into groups of menu items with related functionalities. The buttons are enabled and disabled according to whether their functionalities are accessible or non-accessible, respectively.
- Explorability: The design of the GUI enables the user to explore it with the provision of warning messages and verification dialogs in case of potentially irrational selections. Tooltips are used to indicate the functionality of each button.

A.1 DAFES Menu

A.1.1 File submenu

The *File* submenu (Figure A.2.a) has options to import the input data from a database (Figure A.2.b), to export user-selected results to a database (Figure A.2.c), or to a file (Figure A.2.d), to print the screen (Figure A.2.e), and exit the program(Figure A.2.f).

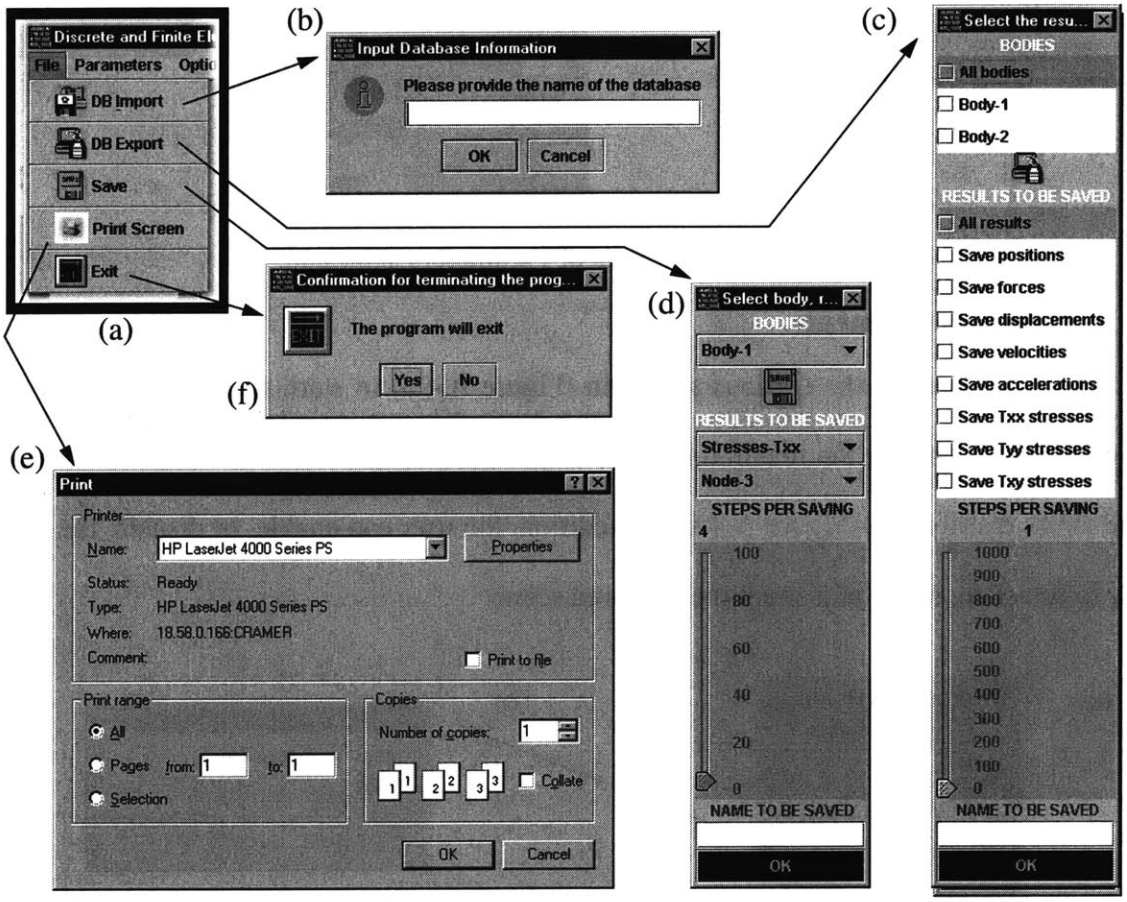


Figure A.2: Options of File submenu.

A.1.2 Parameters submenu

The *Parameters* submenu (Figure A.3.a) enables the setting of the time step (Figure A.3.b), which should be used in the simulation, the selection of whether gravity or/and damping should be taken into account during simulation, the specification of the FE type (Figure A.3.c), when deformable bodies are used, and the imposition of constraints on user selected degrees of freedom.

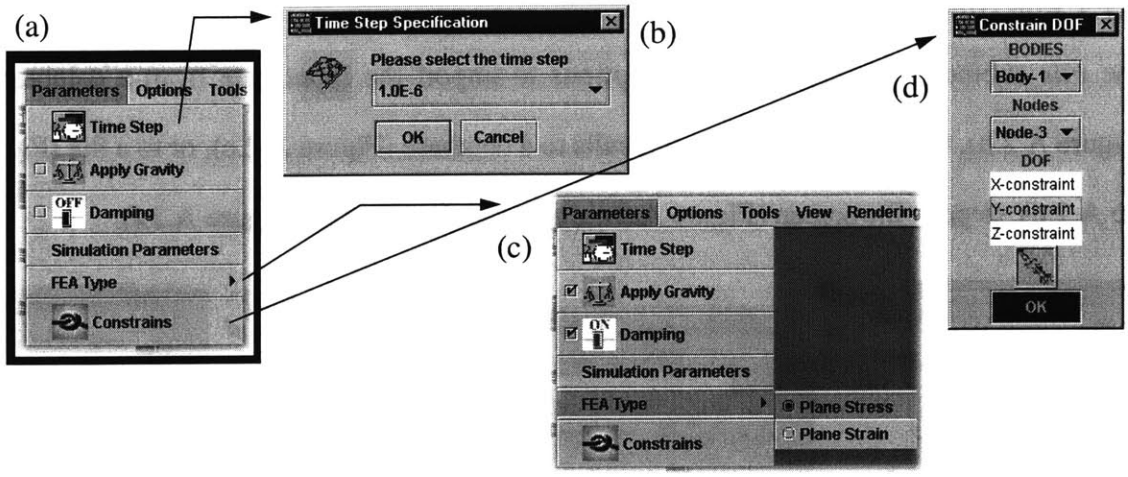


Figure A.3: Options of Parameters submenu.

A.1.3 Options submenu

The user can use the *Options* submenu (Figure A.4.a) to start, stop and continue the simulation, to specify the maximum stress value that should be used to calibrate the plotted stress distribution (Figure A.4.b). In addition, the user can enable, or disable, the ability to translate, rotate and zoom the rendered scene.

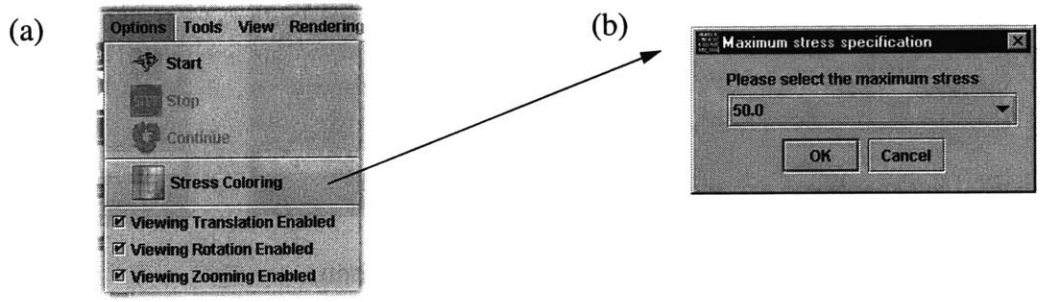


Figure A.4: Options submenu.

A.1.4 Tools submenu

The *Tools* submenu (Figure A.5.a) enables the plotting of a user-selected quantity (Figure A.5.a) during simulation, the grabbing of snapshots during simulations (Figure A.5.b) and the recording (Figure A.5.c) of the simulation in a form of a series of JPEG

files. In addition, the user can explicitly sort the bodies in any direction, and perform a contact detection in order to identify bodies that are in contact.

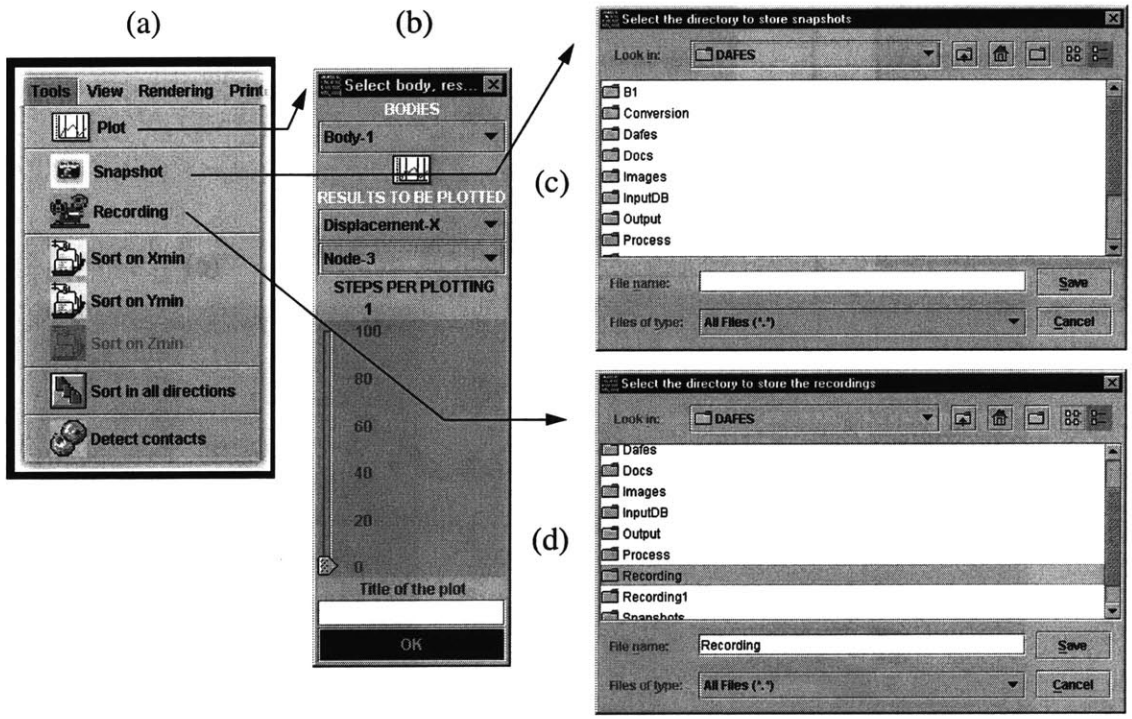


Figure A.5: Tools submenu.

A.1.5 View submenu

The user can use the *View* submenu (Figure A.6.a) to zoom in and out of the scene-graph add or remove the coordinate system from the scene, and defined the background color (Figure A.6.b) of the canvas(es). Furthermore, the user can specify the polygon attributes (Figure A.6.c) of the simulated bodies, in particular whether the bodies should be drawn filling up their faces, or painting only their edges or corners. In addition, there is an option to define the viewing aspect, i.e. the number of canvas to render in and the viewing location and direction, and an option to minimize the program.

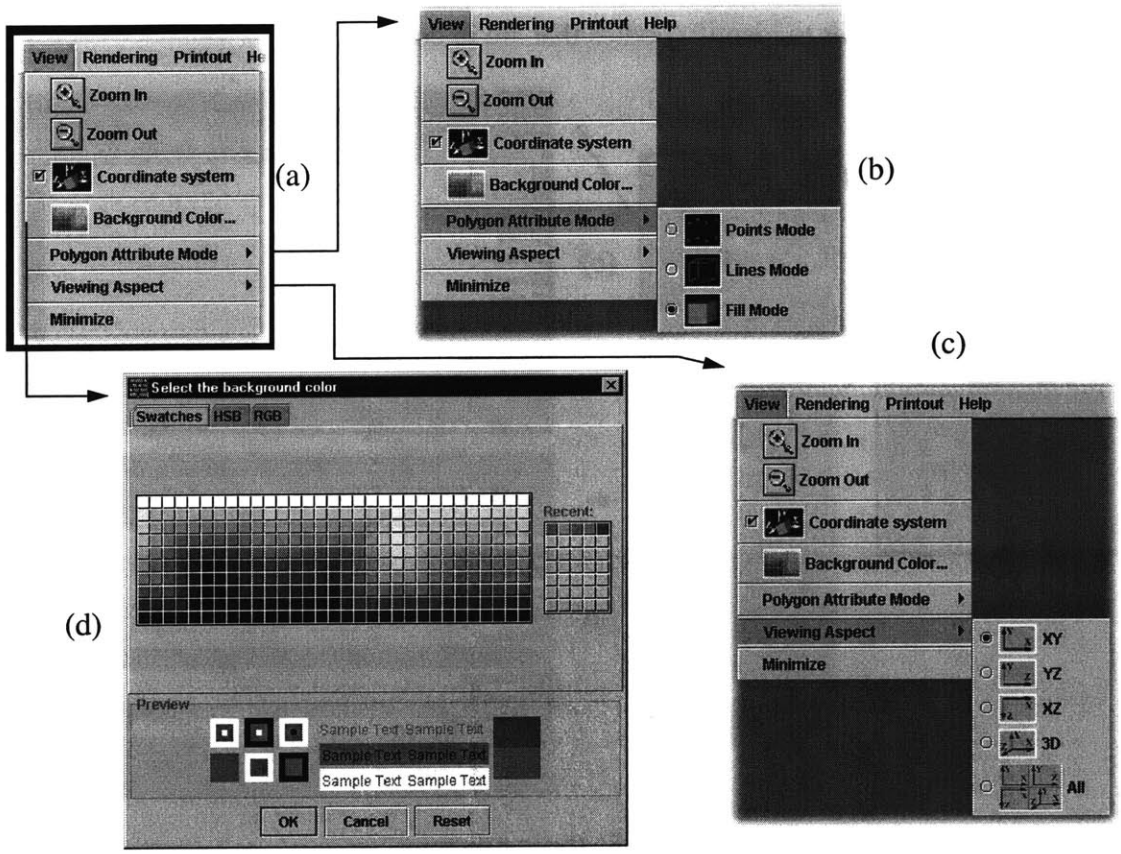


Figure A.6: View submenu.

A.1.6 Rendering submenu

The *Rendering* submenu (Figure A.7.a) allows the selection of the thread priority (Figure A.7.b) for the Java3D renderer during simulation, and the number of time steps per rendering (Figure A.7.c).

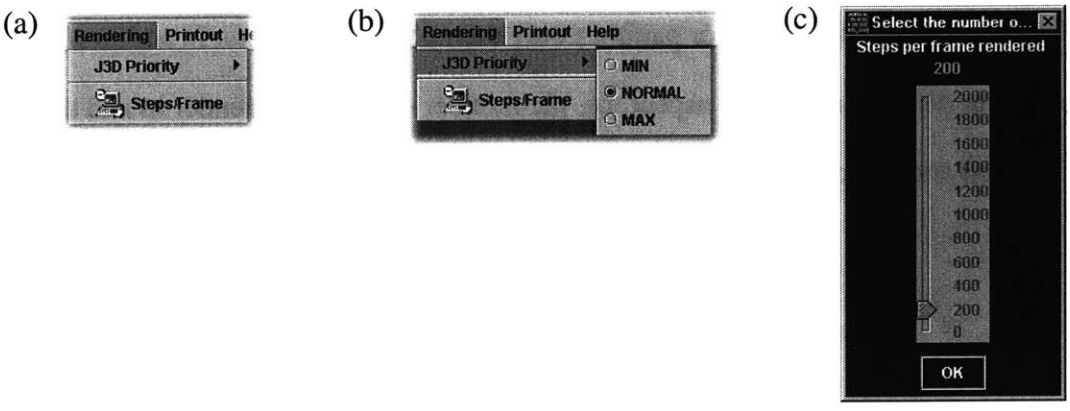


Figure A.7: Rendering submenu.

A.1.7 Printout submenu

The *Printout* submenu (Figure A.8) can be used to printout the simulated bodies, either sorted in a user-selected direction or unsorted, and any contacts between the simulated bodies.

Figure A.8: Printout submenu.

A.2 DAFES controller: control buttons

The DAFES has a set of buttons, named DAFES *controller*, on its left side (Figure A.1) with which the simulation can be controlled. The buttons are presented in more detail in Figure A.9, which makes their purpose self-evident. In particular, the following buttons are available:

- Simulation type buttons (Figure A.9.b), which control the type (2D/3D) of simulation to be performed.
- View selection buttons (Figure A.9.c) to specify the view to be used for rendering.
- Rendering mode buttons (Figure A.9.d) to select the type of polygon attributes.
- Zooming buttons (Figure A.9.e) for zooming in or out.
- A button to indicate the presence of a coordinate system (Figure A.9.f)
- A button to specify the color of the background of the canvas(es) (Figure A.9.f)
- A button to take a single snapshot of the simulation (Figure A.9.g, left)
- A button to specify that the simulation should be recorded (Figure A.9.g, right)
- A slider to select the number of time steps (Figure A.9.h)

- A button to read in the input data from a user-specified database (Figure A.9.i, left)
- A button to print the current simulation data and results (Figure A.9.i, right)
- A button to apply gravity on the simulated bodies (Figure A.9.k, left)
- A button to select whether damping should be taken into account (Figure A.9.k, right)
- A button to specify the simulation time step (Figure A.9.i, left)
- A button to specify the number of steps between successive renderings (Figure A.9.i, right)
- Four buttons, namely the start, stop, continue and exit buttons, that can be used to control the progress of a simulation (Figure A.9.i, left)

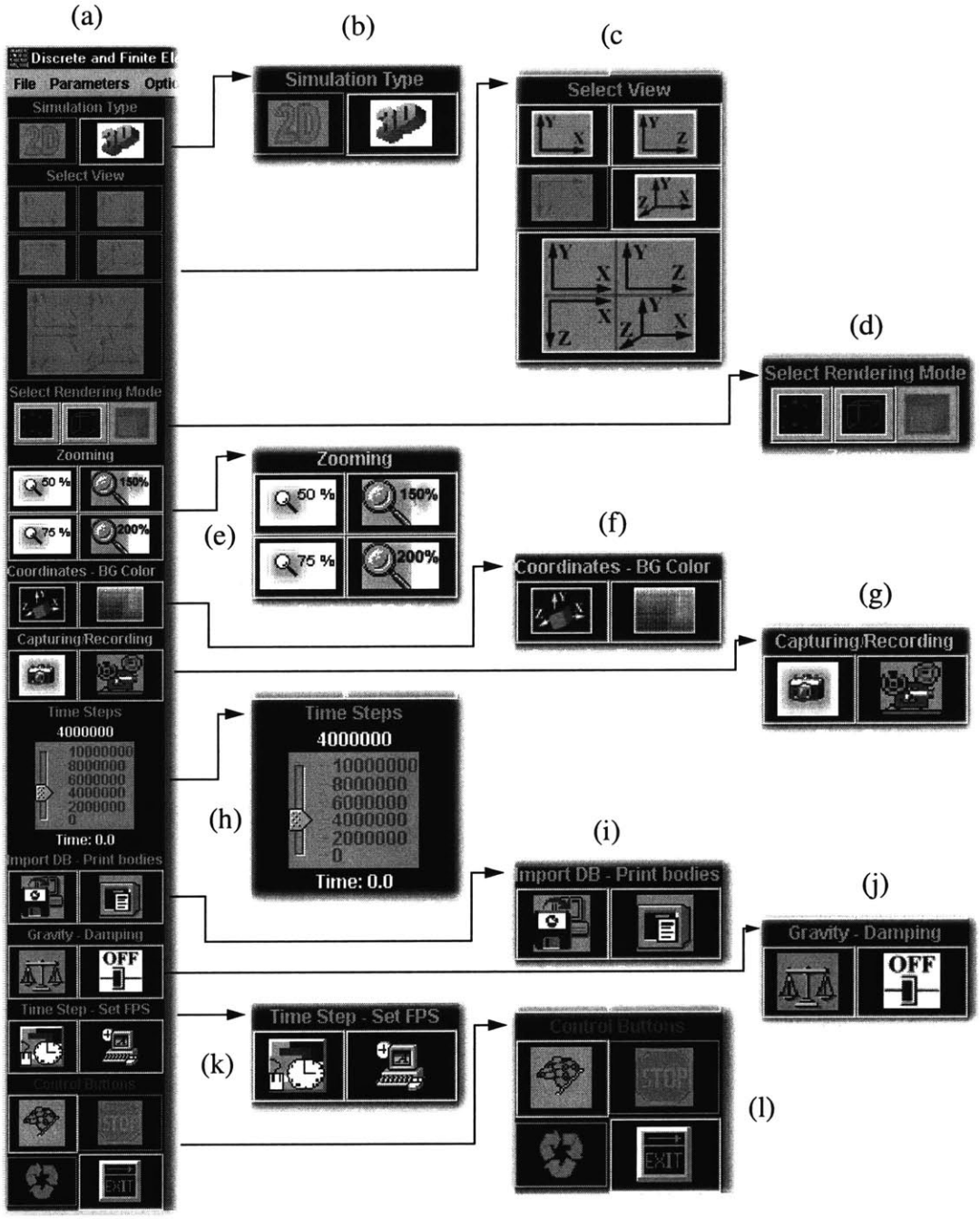


Figure A.9: Controller, i.e. a set of buttons to control the simulation, of DAFES.

Appendix B

Large Strain Considerations

B.1 Introduction

When the strains are large the UL FE formulation cannot be used. A much more computationally intensive nonlinear FEA is required using the Total Lagrangian FE formulation should be employed using different stresses and strains measures. A detail description of this approach is provided in Bathe [1]. Here, a brief description of these stresses and strains measures is presented.

B.2 Total Lagrangian Formulation

Assuming that we know the solution up to time t, we want to determine the solution for time $t+\Delta t$, i.e. the displacements, strains and stresses and, in general, the state of the simulated bodies at the new time instant. The governing equations of dynamic equilibrium at time $t+\Delta t$ are of the form:

$$ {}^{t+\Delta t}\boldsymbol{F}_I + {}^{t+\Delta t}\boldsymbol{F}_D + {}^{t+\Delta t}\boldsymbol{F}_E = {}^{t+\Delta t}\boldsymbol{R} \tag{B.1} $$

where: ${}^{t+\Delta t}\mathbf{F}_I$: inertia forces, due to dynamic effects

${}^{t+\Delta t}\mathbf{F}_D$: damping forces, due to energy dissipation

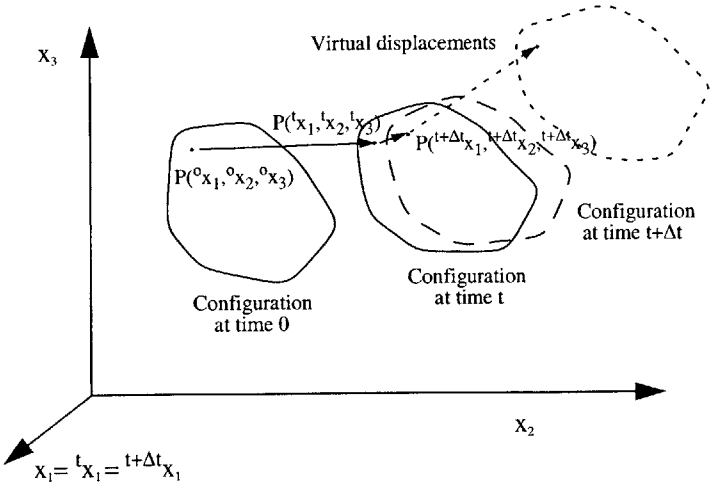
${}^{t+\Delta t}\mathbf{F}_E$: nodal point forces corresponding to the internal element stresses, and,

${}^{t+\Delta t}\boldsymbol{R} = {}^{t+\Delta t}\boldsymbol{R}_B + {}^{t+\Delta t}\boldsymbol{R}_s + {}^{t+\Delta t}\boldsymbol{R}_c$: externally applied loads, other than inertia and damping forces. In particular, ${}^{t+\Delta t}\mathbf{R}_B$ are the body forces, ${}^{t+\Delta t}\boldsymbol{R}_s$ are the surface tractions, and, ${}^{t+\Delta t}\mathbf{R}_c$ are externally applied concentrate forces.

The above governing equations can be derived using the PVW which states that: 'the internal virtual work (LHS of the following equation) is equal to the external virtual work (RHS) for any arbitrary virtual displacements that satisfy the displacement boundary conditions:

$$^{t+\Delta t}f = {}^{t+\Delta t}\Re \tag{B.2}$$

The following figure shows a general body in its configuration at time *0* (original configuration), at time *t*, and, at time $t+\Delta t$. It also shows the virtual displacements that are applied on the latter configuration. Note that the cartesian coordinate axes are stationary, i.e. ${}^{0}X_i = {}^{t}X_i = {}^{t+\Delta t}X_i$.



$$^{t}x_i = {}^{0}x_i + {}^{t}u_i$$

$$^{t+\Delta t}x_i = {}^{0}x_i + {}^{t+\Delta t}u_i$$

$$\Delta^{t}u_i = {}^{t+\Delta t}x_i - {}^{t}x_i$$

$$\Delta^{t}u_i = {}^{t+\Delta t}u_i - {}^{t}u_i$$

$$^{t+\Delta t}x_i = {}^{t}x_i + \Delta^{t}u_i$$

(for i = 1, 2, 3)

Figure B.1: Configurations at time *0*, *t* and $t+\Delta t$.

The internal virtual work, i.e. the work due to the internal stresses, can be computed as:

$$^{t+\Delta t}f = \int_{^{t+\Delta t}V} {}^{t+\Delta t}\tau_{ij} \cdot \delta_{t+\Delta t}e_{ij} \cdot d^{t+\Delta t}V \tag{B.3}$$

where: ${}^{t+\Delta t}\tau_{ij}$: Cauchy stress tensor components, actual stresses which we want to compute

$\delta_{t+\Delta t}\mathbf{e}_{ij} = \frac{1}{2}\cdot\left(\frac{\partial}{\partial^{t+\Delta t}x_j}\delta u_i + \frac{\partial}{\partial^{t+\Delta t}x_i}\delta u_j\right)$: infinitesimal virtual strain tensor components corresponding to the virtual displacements δu_i which refer to configuration at time $t+\Delta t$

δu_i : virtual displacements, i.e. variations of the real displacements ${}^{t+\Delta t}u_i$

${}^{t+\Delta t}x_i$: Cartesian coordinates of a point at time $t+\Delta t$

${}^{t+\Delta t}V$: Volume of the simulated body at time $t+\Delta t$

The external virtual work ${}^{t+\Delta t}\Re$ at time $t+\Delta t$ is equal to:

$$
\begin{aligned}
{}^{\Delta t}\Re = & \int_{t+\Delta t_V} {}^{t+\Delta t}f_i^B \cdot \delta u_i \cdot d^{t+\Delta t}V + \int_{t+\Delta t_{S_f}} {}^{t+\Delta t}f_i^S \cdot \delta^S u_i \cdot d^{t+\Delta t}S + \sum_i R_c^i \cdot \delta u_i \\
& - \int_{t+\Delta t_V} {}^{t+\Delta t}\rho_i \cdot {}^{t+\Delta t}\ddot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V - \int_{t+\Delta t_V} {}^{t+\Delta t}\xi \cdot {}^{t+\Delta t}\dot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V
\end{aligned}
\qquad \text{(B.4)}
$$

where: ${}^{t+\Delta t}f_i^B$: body force components, not including inertia and damping, (force per unit volume)

${}^{t+\Delta t}f_i^S$: surface tractions (force per unit area)

R_c^i : concentrate forces

${}^{t+\Delta t}V$: volume at time $t+\Delta t$

${}^{t+\Delta t}S_f$: surface at time $t+\Delta t$, where surface tractions are applied

${}^{t+\Delta t}\rho$: mass density

${}^{t+\Delta t}\xi$: damping property parameter

$\dot{u}_i$, and $\ddot{u}_i$: (unknown) velocity, and, acceleration components, respectively, at $t+\Delta t$

δu_i, $\delta\dot{u}_i$, and $\delta\ddot{u}_i$: virtual displacement, velocity, and, acceleration components, respectively, imposed on configuration at time $t+\Delta t$

The inertia and damping forces, which are formally given by the following expressions, are essentially body forces. However, in the analysis, instead of using these expressions to derive, according to the FE formulation, the consistent mass and damping matrices, a diagonal mass matrix is selected, and the damping is assumed to be mass proportional which significantly simplifies the solution.

$${}^{t+\Delta t}\boldsymbol{F}_I = \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\rho \cdot {}^{t+\Delta t}\ddot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V$$: inertia forces, resisting any change of momentum

$${}^{t+\Delta t}\boldsymbol{F}_D = \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\xi \cdot {}^{t+\Delta t}\dot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V$$: damping forces, resisting motion by dissipating energy

Substituting in the PVW the expressions for the internal and external virtual work we get the following equation:

$$\begin{aligned} \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\tau_{ij} \cdot \delta_{t+\Delta t}e_{ij} \cdot d^{t+\Delta t}V = & \int_{t+\Delta t V} {}^{t+\Delta t}f_i^B \cdot \delta u_i \cdot d^{t+\Delta t}V + \\ & \int_{t+\Delta t S_f} {}^{t+\Delta t}f_i^S \cdot \delta^S u_i \cdot d^{t+\Delta t}S + \sum_i R_c^i \cdot \delta u_i \\ - \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\rho_i \cdot {}^{t+\Delta t}\ddot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V - & \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\xi \cdot {}^{t+\Delta t}\dot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V \end{aligned} \tag{B.5}$$

Rearranging the terms we get the familiar equations of dynamic equilibrium. These equations express the equilibrium and compatibility requirements of any general body at time $t+\Delta t$. In addition, assuming that the proper constitutive relations are used the stress-strain law is also satisfied.

$$\int_{t+\Delta t V} {}^{t+\Delta t}\rho \cdot {}^{t+\Delta t}\ddot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V + \int_{t+\Delta t V} {}^{t+\Delta t}\xi \cdot {}^{t+\Delta t}\dot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V + \int_{t+\Delta t V} {}^{t+\Delta t}\tau_{ij} \cdot \delta_{t+\Delta t} e_{ij} \cdot d^{t+\Delta t}V = \int_{\Delta t V} {}^{t+\Delta t}f_i^B \cdot \delta u_i \cdot d^{t+\Delta t}V + \int_{t+\Delta t S_f} {}^{t+\Delta t}f_i^S \cdot \delta^S u_i \cdot d^{t+\Delta t}S + \sum_i R_c^i \cdot \delta u_i \tag{B.6}$$

However, is difficult to use the above expression of the PVW, because the integration is over the unknown and deformed volume at time $t+\Delta t$. The PVW can be rewritten the in a more convenient way, by using proper stress and strain measures. In particular, the Second Piola-Kirchhoff stresses, ${}^{t+\Delta t}_{t}\boldsymbol{S}$, and the Green-Lagrange strains, ${}^{t+\Delta t}_{t}\varepsilon$, are employed to express the internal virtual work in terms of an integral over a known volume. In addition, these stress and strain measures can be very effectively decomposed into known and unknown parts leading to an incremental formulation that facilitates an efficient nonlinear analysis. Then, the total Lagrangian formulation, in which all variables are referred to the initial configuration can be used.

The Second Piola-Kirchhoff stresses ${}^{t+\Delta t}_{t}\boldsymbol{S}$ are stresses at time $t+\Delta t$ referring, i.e. measured with respect, to configuration at time t. They are given, in tensor notation[1], and in matrix form, by the following equations:

$$\text{tensor notation: } {}^{t+\Delta t}_{t}S_{i,j} = \frac{{}^{t}\rho}{{}^{t+\Delta t}\rho} \cdot {}^{t}_{t+\Delta t}x_{i,m} \cdot {}^{t+\Delta t}\tau_{mn} \cdot {}^{t}_{t+\Delta t}x_{j,n} \tag{B.7}$$

1. A left superscript indicates at which time the quantity occurs and a left subscript indicates the configuration with respect to which the quantity is measured in case its different from the former.

$$
{}^{t+\Delta t}_{t}S_{ij} = \frac{{}^{t}\rho}{{}^{t+\Delta t}\rho} \cdot \left(\frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_1} \cdot {}^{t+\Delta t}\tau_{11} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_1} + \right. \tag{B.8}
$$

$$
\frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_1} \cdot {}^{t+\Delta t}\tau_{12} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_2} + \frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_1} \cdot {}^{t+\Delta t}\tau_{13} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_3} +
$$

$$
\frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_2} \cdot {}^{t+\Delta t}\tau_{21} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_1} + \frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_2} \cdot {}^{t+\Delta t}\tau_{22} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_2} +
$$

$$
\frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_2} \cdot {}^{t+\Delta t}\tau_{23} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_3} + \frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_3} \cdot {}^{t+\Delta t}\tau_{31} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_1} +
$$

$$
\left. \frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_3} \cdot {}^{t+\Delta t}\tau_{32} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_2} + \frac{\partial {}^{t}x_i}{\partial {}^{t+\Delta t}x_3} \cdot {}^{t+\Delta t}\tau_{33} \cdot \frac{\partial {}^{t}x_j}{\partial {}^{t+\Delta t}x_3} \right)
$$

$$
\text{matrix form: } {}^{t+\Delta t}_{t}\boldsymbol{S} = \frac{{}^{t}\rho}{{}^{t+\Delta t}\rho} \cdot {}^{t}_{t+\Delta t}\boldsymbol{X}^{T} \cdot {}^{t+\Delta t}\tau \cdot {}^{t}_{t+\Delta t}\boldsymbol{X} \tag{B.9}
$$

The Cauchy stress tensor ${}^{t+\Delta t}\tau$ at time $t+\Delta t$, whose components ${}^{t+\Delta t}\tau_{mn}$ are used above, can be expressed, in matrix form, as:

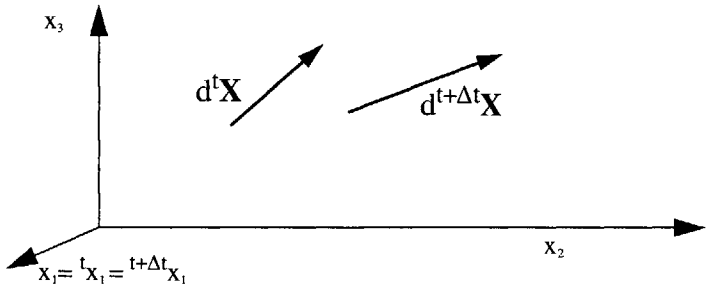
$$
{}^{t+\Delta t}\tau = \frac{{}^{t+\Delta t}\rho}{{}^{t}\rho} \cdot {}^{t+\Delta t}_{t}\mathbf{X}^{T} \cdot {}^{t+\Delta t}_{t}\boldsymbol{S} \cdot {}^{t+\Delta t}_{t}\mathbf{X} \tag{B.10}
$$

${}^{t}_{t+\Delta t}\mathbf{X}$ is the inverse of the deformation gradient, i.e. ${}^{t}_{t+\Delta t}\mathbf{X} = ({}^{t+\Delta t}_{t}\mathbf{X})^{-1}$. The deformation gradient, is given below, is a measure of the deformation of the body. In particular, it measures how much a material fiber has been rotated and stretched.

The deformation gradient tensor: describes the deformations (stretches) and rotations of material fibers from time t to time $t+\Delta t$

$$
{}^{t+\Delta t}_{t}\mathbf{X} = {}_{t}\nabla \cdot {}^{t+\Delta t}X = \begin{bmatrix} \frac{\partial {}^{t+\Delta t}x_1}{\partial {}^{t}x_1} & \frac{\partial {}^{t+\Delta t}x_1}{\partial {}^{t}x_2} & \frac{\partial {}^{t+\Delta t}x_1}{\partial {}^{t}x_3} \\ \frac{\partial {}^{t+\Delta t}x_2}{\partial {}^{t}x_1} & \frac{\partial {}^{t+\Delta t}x_2}{\partial {}^{t}x_2} & \frac{\partial {}^{t+\Delta t}x_2}{\partial {}^{t}x_3} \\ \frac{\partial {}^{t+\Delta t}x_3}{\partial {}^{t}x_1} & \frac{\partial {}^{t+\Delta t}x_3}{\partial {}^{t}x_2} & \frac{\partial {}^{t+\Delta t}x_3}{\partial {}^{t}x_3} \end{bmatrix} \tag{B.11}
$$

As shown in the following figure, the deformation gradient ${}^{t+\Delta t}_{t}\mathbf{X}$ relates the differential fiber vector $d^{t+\Delta t}\boldsymbol{X}$ at time $t+\Delta t$, with the differential fiber at time t, $d^{t}\boldsymbol{X}$, according to the chain rule of differentiation.



$$d^{t+\Delta t}\boldsymbol{X} = {}^{t+\Delta t}_{t}\mathbf{X} \cdot d^{t}\boldsymbol{X}$$

$$d^{t}\boldsymbol{X} = {}^{t}_{t+\Delta t}\mathbf{X} \cdot d^{t+\Delta t}\boldsymbol{X}$$

Figure B.2: Physical interpretation of the deformation gradient.

It can be proved that mass densities at time t and time $t+dt$, ${}^{t}\rho$ and ${}^{t+\Delta t}\rho$, respectively, are related through the deformation gradient ${}^{t+\Delta t}_{t}\mathbf{X}$:

$${}^{t}\rho = {}^{t+\Delta t}\rho \cdot det({}^{t+\Delta t}_{t}\mathbf{X})$$

Another important property of the deformation gradient is that it can always be decomposed into a symmetric stretch matrix ${}^{t+\Delta t}_{t}\mathbf{U}$, and an orthogonal rotation matrix, ${}^{t+\Delta t}_{t}\mathbf{R}$. This is called polar decomposition of the deformation gradient:

$${}^{t+\Delta t}_{t}\mathbf{X} = {}^{t+\Delta t}_{t}\mathbf{R} \cdot {}^{t+\Delta t}_{t}\mathbf{U} \tag{B.12}$$

The Green-Lagrange strains, ${}^{t+\Delta t}_{t}\varepsilon$, are strains at time $t+\Delta t$ referring to configuration at time t. The Green-Lagrange strain tensor ${}^{t+\Delta t}_{t}\varepsilon$ is a symmetric matrix which measures the stretching deformations of material fibers, and is given in terms of the stretch matrix ${}^{t+\Delta t}_{t}\mathbf{U}$ as:

$${}^{t+\Delta t}_{t}\varepsilon = \frac{1}{2} \cdot ({}^{t+\Delta t}_{t}\boldsymbol{U}^{T} \cdot {}^{t+\Delta t}_{t}\boldsymbol{U} - \boldsymbol{I}) \tag{B.13}$$

The product of the transpose of the deformation gradient with itself is equal to the Cauchy-Green deformation tensor, which is invariant under rigid body rotation.

$${}^{t+\Delta t}_{t}\boldsymbol{C} = {}^{t+\Delta t}_{t}\boldsymbol{X}^{T} \cdot {}^{t+\Delta t}_{t}\boldsymbol{X} \tag{B.14}$$

For a rigid body motion, i.e. without any fiber stretching or change of the angle between two fibers starting from the same point, the Cauchy-Green deformation tensor is equal to the identity matrix $\boldsymbol{I}$.

Considering the orthogonality of the rotation matrix, the Green-Lagrange strain tensor can be expressed as:

$$ {}^{t+\Delta t}_{t}\varepsilon = \frac{1}{2}\cdot({}^{t+\Delta t}_{t}\boldsymbol{X}^{T}\cdot{}^{t+\Delta t}_{t}\boldsymbol{R}^{T}\cdot{}^{t+\Delta t}_{t}\boldsymbol{R}\cdot{}^{t+\Delta t}_{t}\boldsymbol{X}-\boldsymbol{I}) = \frac{1}{2}\cdot({}^{t+\Delta t}_{t}\boldsymbol{C}-\boldsymbol{I}) \qquad \text{(B.15)} $$

Both the Cauchy-Green deformation tensor and the Green-Lagrange strain tensor are symmetric. For a rigid body motion between times t_1 and t_2: ${}^{t_1}_{0}\varepsilon = {}^{t_2}_{0}\varepsilon$, while for a general rigid body motion (all the way from the time *0*): ${}^{t}_{0}\varepsilon = \boldsymbol{0}$

The components of the Green-Lagrange strain tensor can also be expressed in terms of displacement derivatives, by substituting the expressions for ${}^{t+\Delta t}x_i = {}^{t}x_i + \Delta^{t}u_i$ as:

$$ {}^{t+\Delta t}_{t}\varepsilon_{ij} = \frac{1}{2}\cdot({}^{t+\Delta t}_{t}u_{i,j} + {}^{t+\Delta t}_{t}u_{j,i} + {}^{t+\Delta t}_{t}u_{k,i}\cdot{}^{t+\Delta t}_{t}u_{k,j}) = \qquad \text{(B.16)} $$

$$ {}^{t}_{t}\varepsilon_{ij} + \Delta_t\varepsilon_{ij} = \Delta_t\varepsilon_{ij} = \Delta_t e_{ij} + \Delta_t\varepsilon^{nl}_{ij} $$

$$ {}^{t+\Delta t}_{t}\varepsilon_{ij} = \Delta_t e_{ij} + \Delta_t\varepsilon^{nl}_{ij} = \frac{1}{2}\cdot\left(\frac{\partial}{\partial^{t}x_j}\Delta^{t}u_i + \frac{\partial}{\partial^{t}x_i}\Delta^{t}u_j + \right. \qquad \text{(B.17)} $$

$$ \left.\frac{\partial}{\partial^{t}x_i}\Delta^{t}u_1\cdot\frac{\partial}{\partial^{t}x_j}\Delta^{t}u_1 + \frac{\partial}{\partial^{t}x_i}\Delta^{t}u_2\cdot\frac{\partial}{\partial^{t}x_j}\Delta^{t}u_2 + \frac{\partial}{\partial^{t}x_i}\Delta^{t}u_3\cdot\frac{\partial}{\partial^{t}x_j}\Delta^{t}u_3\right) $$

where: ${}^{t+\Delta t}_{t}u_{i,j} = \frac{\partial}{\partial^{t}x_j}({}^{t+\Delta t}u_i - {}^{t}u_i) = \frac{\partial}{\partial^{t}x_j}\Delta^{t}u_i$ is a spatial derivative of the incremental displacement,

and, the strain at time t with respect to the configuration at time t is equal to zero,

${}^{t}_{t}\varepsilon_{ij} = 0$, since ${}^{t}_{t}u_{i,j} = \frac{\partial}{\partial^{t}x_j}({}^{t}u_i - {}^{t}u_i) = 0$

Note, that the differentiations are with respect to the known configuration at time t.

The strain increment can be decomposed into a linear and a nonlinear part, which facilitates the linearization that has to be done to solve this nonlinear problem:

Linear strain increment: $\Delta_t e_{ij} = \frac{1}{2} \cdot ({}_t^{t+\Delta t}u_{i,j} + {}_t^{t+\Delta t}u_{j,i})$

Nonlinear strain increment: $\Delta_t \varepsilon_{ij}^{nl} = \frac{1}{2} \cdot ({}_t^{t+\Delta t}u_{k,i} \cdot {}_t^{t+\Delta t}u_{k,j})$

Considering virtual displacements δu_i, applied on the configuration at time t+Δt, there should be corresponding variations of the Green-Lagrange strain tensor ${}_t^{t+\Delta t}\varepsilon$, and the infinitesimally small strain tensor ${}_{t+\Delta t}e$. It can be shown that these variations are related as follows:

$$\delta {}_t^{t+\Delta t}\varepsilon = {}_t^{t+\Delta t}\mathbf{X}^T \cdot \delta_{t+\Delta t}e \cdot {}_t^{t+\Delta t}\mathbf{X}, \tag{B.18}$$

where: $\delta_{t+\Delta t}e_{ij} = \frac{1}{2} \cdot \left(\frac{\partial \delta u_j}{\partial {}^{t+\Delta t}x_i} + \frac{\partial \delta u_i}{\partial {}^{t+\Delta t}x_j} \right)$

Both the Second Piola-Kirchhoff stress tensor and the Green-Lagrange strain tensor are symmetric matrices, and, most importantly, they do not change under rigid body motion. Furthermore, using these stress and strain measures we can rewrite the internal virtual work as follows, where the integration is over a known volume:

$$\int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\tau_{ij} \cdot \delta_{t+\Delta t}e_{ij} \cdot d^{t+\Delta t}V = \int_{{}^{0}V} {}^{t+\Delta t}_{0}S_{ij} \cdot \delta {}^{t+\Delta t}_{0}\varepsilon_{ij} \cdot d^{t}V \tag{B.19}$$

The PVW, which expresses the equilibrium and compatibility requirements, as well as the stress-strain law assuming that proper constitutive relations have been employed, can be written as:

$$\int_{{}^{0}V} {}^{t+\Delta t}_{0}S_{ij} \cdot \delta {}^{t+\Delta t}_{0}\varepsilon_{ij} \cdot d^{t}V = \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}f_i^B \cdot \delta u_i \cdot d^{t+\Delta t}V + \int_{{}^{t+\Delta t}S_f} {}^{t+\Delta t}f_i^S \cdot \delta^S u_i \cdot d^{t+\Delta t}S +$$
$$\sum_i R_c^i \cdot \delta u_i - \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\rho \cdot {}^{t+\Delta t}\ddot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V - \int_{{}^{t+\Delta t}V} {}^{t+\Delta t}\xi \cdot {}^{t+\Delta t}\dot{u}_i \cdot \delta u_i \cdot d^{t+\Delta t}V$$

Since the problem is nonlinear, at least due to the large displacements and rotations, we need to linearize the governing equations and use iterations to determine within sufficient accuracy the solution for each individual time instant.

A linearization of the unknown terms that are nonlinear in displacements is required in order to be able to determine an approximate displacement increment. Then, from the calculated displacements we can determine the corresponding strains, and stresses. The corresponding to the internal stresses nodal point forces can then be computed and used for the new iteration. This procedure continues until convergence within an allowable tolerance is reached. Therefore, we need to linearize the terms which are nonlinear in terms of the unknown displacement increments:

The resulting system of nonlinear equations is solved using a Newton-Raphson procedure where an approximation for the solution at the new time step is obtained by doing some linearizations. Iterations need to be used to improve the approximation and satisfy the equilibrium equations within some allowable tolerance, before proceeding to the next time step. The iterations continue until the out-of-balance load and the associated displacement increments are sufficiently small. Having determine the displacements for each time step, the deformations and stresses of each deformable body can also be computed. The motion of each discrete body during each new time step is determined, and updating the positions of all discrete bodies a new contact detection procedure defines the new contacts and evaluates the corresponding forces to be used in the next time step.

Appendix C

Source Code

C.1 Mergesort algorithm to sort an N-Size array

The same sorting algorithm, mergesort, was implemented both in C++ and Java in order to evaluate the relative performance of the two languages, see Paragraph 8.3.

C.1.1 C++ version

```
void sortArray(double data[], double tmp[], int start, int end)
{
  if(start < end)
    {
       int length = end - start + 1;
       int split   start + length/2;

       sortArray(data, tmp, start, split-1);
       sortArray(data, tmp, split, end);

       merge(data, tmp, start, split, end);
    }
}

void merge(double data[], double tmp[], int start, int split, int end)
{
   int length, i, j, a, b;

   i=j=0;
   length = end - start + 1;
   a = start;
   b = split;

  while(a < split && b <= end)
   {
      if(data[a] > data[b])
       {
          tmp[i] = data[b];
```

```
        b++;
      }
    else
     {
        tmp[i] = data[a];
        a++;
      }
    i++;
  }

 if(a >= split)
    for( ;b <= end;b++,i++)
       tmp[i] = data[b];
 else
     for( ;a <split;a++,i++)
          tmp[i] = data[a];

 for(i=start;i<=end;i++,j++)
    data[i] = tmp[j];
}
```

C.1.2 Java version

```
void sortArray(double data[], int start, int end)
  {
    if(start < end)
      {
        int length = end - start + 1;
        int split = start + length/2;

        sortArray(data, start, split-1);
        sortArray(data, split, end);

        merge(data, start, split, end);
      }
  }

void merge(double data[], int start, int split, int end)
  {
    int length, i, j, a, b;

    i=j=0;
    length = end - start + 1;
    a = start;
    b = split;
```

```
    while(a < split && b <= end)
      {
        if(data[a] > data[b])
          {
            tmp[i] = data[b];
            b++;
          }
        else
          {
            tmp[i] = data[a];
            a++;
          }
        i++;
      }

    if( a >= split)
      for( ;b <= end;b++,i++)
        tmp[i] = data[b];
    else
      for( ;a <split;a++,i++)
          tmp[i] = data[a];

    for(i=start;i<=end;i++,j++)
       data[i] = tmp[j];
}
```

References

[1] Bathe Klaus-Jurgen, *"Finite Element Procedures"*, Prentice-Hall Inc. Englewood Cliffs, New Jersey, 1996.

[2] Brown Kirk and Petersen Daniel, *"Ready to Run Java 3D"*, Wiley & Sons, Inc., 1999.

[3] Chiou Jen-Diann, *"A Distributed Environment for Multibody Physics"*, Ph.D. thesis, Department of Civil and Environmental Engineering, MIT, 1998.

[4] Cook Ben, *"A Numerical Framework for the Direct Simulation of Solid-Fluid Systems"*, Ph.D. thesis, Department of Civil and Environmental Engineering, MIT, 2001.

[5] Cundall Peter, *"A computer model for simulating progressive large scale movements in block rock systems"*. Symp. Intl. Society of Rock Mechanics, Nancy, France, 1971.

[6] Cundall P. and Strack O, *"A distinct element model for granular assemblies"*. Geotechnique, 29:47,65, 1979.

[7] Cundall Peter and Hart Roger, *"Numerical modeling of discontinua"*, in Proceedings of the 1st U.S. Conference on Discrete Element Methods (DEM), 1989.

[8] Fahl, G., Risch, T., Skold, M., *"AMOS - An Architecture for Active Mediators"*, The International Workshop on Next Generation Information Technologies and Systems (NGITS' 93), Haifa, Israel, June 28-30, 1993, pp. 47-53.

[9] Flodin S., Karlsson J., Orsborn, K., Risch T., Skold M., and Werner M., *"AMOS User's Guide"*, http://www.dis.uu.se/~udbl,/amos/doc/amos_users_guide.html, EDSLAB, Linkoping University, Linkoping, 1997.

[10] Flodin S., Karlsson J., Risch T., Skold, M., and Werner M., *"AMOS System Manual"*, EDSLAB, Linkoping University, Linkoping, 1997.

[11] George Paul, *"Automatic Mesh Generation: application to finite element methods"*, J. Wiley, 1991.

[12] Goyal S., Pinson E., and Sinden W., *"Simulation of dynamics of interacting rigid bodies including friction 1: General problem and contact model"*, Engineering with Computers, Vol. 10, pp. 162-174, 1994.

[13] Goyal S., Pinson E., and Sinden W., *"Simulation of dynamics of interacting rigid bodies including friction 2: Software system design and implementation"*, Engineering with Computers, Vol. 10, pp. 175-195, 1994.

[14] Klosek T. Justin, *"The Integration of Fluid Dynamics with a Discrete-Element Modeling System"*, S.M. thesis, Department of Civil and Environmental Engineering, MIT, 1997.

[15] Knupp P. and Steinberg S., *"Fundamentals of Grid Generation"*, CRC Press, 1993.

[16] Leiserson C., Cormen T., and Rivest R., *"Introduction to Algorithms"*, MIT Press, McGraw-Hill, 1990.

[17] Munjiza Ante, *"Discrete elements in transient dynamics of fractured media"*, Ph.D. thesis, Department of Civil Engineering, University College of Swansea, 1992.

[18] Munjiza A., Owen D., and Williams J., *"On a Rational Approach to Rock Blasting"*, Proceedings of the 8th International Conference on Computer Methods and Advances in Geomechanics, Editors H.J. Siriwardane and M.M. Zaman, Vol.1, pp857-862, 1994.

[19] Mustoe G., Henriksen M., and Huttelmaier H., editors. *Proceedings of the 1st U.S. Conference on Discrete Element Methods (DEM)*, Colorado School of Mines, Golden, CO, 1989.

[20] Mustoe G., Hocking G, and Williams J., *"Validation of CICE discrete element code for ride-up and ice ridge/cone interaction"*, in Proceedings ARTIC '85 San Francisco. ASCE, New York, 1985.

[21] O'Connor M. Ruaidhri, *"A Distributed Discrete Element Modeling Environment - Algorithms, Implementation, and Applications"*, Ph.D. thesis, Department of Civil and Environmental Engineering, MIT, 1996.

[22] Orsborn Kjell, *"On Extensible and Object-Relational Database Technology for Finite Element Analysis"*, Ph.D. Thesis, Department of Computer and Information Science, Linkoping University, Sweden, 1996.

[23] Pante G., Beer G, and Williams John, *"Numerical methods in rock mechanics*, Wiley, 1992.

[24] Perkins Eric, *"Discrete element computation: algorithms and architecture"*, Ph.D. Thesis, Department of Civil and Environmental Engineering, MIT, 2001.

[25] Preparata Franco and Shamos Michael, *"Computational Geometry: An introduction"*, Springer-Verlag, 1988.

[26] Rege V. Nabha, *"Computational Modeling of Granular Materials"*, Ph.D. thesis, Department of Civil and Environmental Engineering, MIT, 1996.

[27] Rumbaugh J. et al., *"Object-Oriented Modeling and Design"*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1991.

[28] Sedgewick R., *"Algorithms in C++"*, Addison-Wesley, 1992.

[29] Shi G., *"Discontinuous deformation analysis - a new numerical model of the statics and dynamics of deformable block structures"*, Proceedings of the First US Conference on discrete element methods, 1989.

[30] Shi G., *"Block system modeling by discontinuous deformation analysis"*, Computational Mechanics Publications, 1993.

[31] Taylor L. and Preece D., *"Simulation of blasting induced rock motion using spherical element models"*, Engineering computations, 9(2), 1992.

[32] Thompson J., Soni B., ad Weatherill N., *"Handbook of Grid Generation"*, CRC Press, 1999.

[33] Samet H., *"The quadtree and realted hierarchical data structures"*, CM Computing Survey, Vol. 16, No. 2, 1984.

[34] Tisell, C. and Orsborn K., *"A System for Multibody Analysis Based on Object-Relational Database Technology"*, Proceedings of the First International Conference on Engineering Computational Technology (EST '98), Edinburgh, Scotland, 18-20 August 1998. In Advances in Engineering Computational Technology, Topping, B.H.V. (ed.), Civil-Comp. Press, ISBN 0-948749-55-5, pp. 273-285, 1998.

[35] United States Department of Defense, *"Department Of Defense Requirements for High Order Computer Programming Languages: Steelman"*, June, 1978.

[36] Weiss Mark, *"Algorithms, Data Structures, and Problem Solving with C++"*, Addison-Wesley, 1996.

[37] Wheeler David, *"Ada, C, C++, and Java vs. The Steelman Institute for Defense Analyses"*, Alexandria, Virginia, July 1997.

[38] Williams John, Hocking Grant and Mustoe Graham, *"The theoretical basis of the discrete element methods"*, Proceedings of the 1985 Conference on Numerical Methods in Engineering, Theory and Application, pp. 897-906, 1985.

[39] Williams John *"Contact analysis of large numbers of interacting bodies using discrete modal methods for simulating material failure on the microscopic scale"*, International Journal of Computer Aided Engineering, Engineering computations, 5(3), 1988.

[40] Williams John, Chen Andrie, and Petrie Dennis, *"Ice island interaction with conical production structures"*, Proceedings of the 1st Conference on Discrete Element Methods, 1989.

[41] Williams John and Pentland Alex, *"Superquadric and modal dynamics for discrete elements in interactive design"*, Engineering Computations, 9-2, pp. 115-127, 1992.

[42] Williams John and Mustoe Graham, editors. *Proceedings of the 2nd International Conference on Discrete Element Methods (DEM)*, Dept. of Civil and Environmental Engineering, Massachusetts Institute of Technology, 1993. IESL Publications.

[43] Williams John and Rege Nabha, "*Circulation cells in a deforming granular material*", Powder Technology, 1996.

[44] Williams John and Rege Nabha, "*A simulation system for computational materials*". Technical report, Massachusetts Institute of Technology, Dept. of Civil and Environmental Engineering, IESL report 95-01, 1995.

[45] Williams J. and O'Connor R., *"A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries"*, Engineering Computations, Vol. 12, pp. 185-201, 1995.

Index