Linköping Studies in Science and Technology

Dissertation No. 1009

Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analyses

by

Iakov Nakhimovski

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2006

Abstract

The motivation for this thesis was the need for further development of multibody dynamics simulation packages focused on detailed contact analysis. The three parts of the thesis make contributions in three different directions:

Part I summarizes the equations, algorithms and design decisions necessary for dynamics simulation of flexible bodies with moving contacts. The assumed general shape function approach is presented. It is expected to be computationally less expensive than FEM approaches and easier to use than other reduction techniques. Additionally, the described technique enables studies of the residual stress release during grinding of flexible bodies. The proposed set of mode shapes was also successfully applied for modeling of heat flow.

The overall software system design for a flexible multibody simulation system SKF BEAST (Bearing Simulation Tool) is presented and the specifics of the flexible modeling are specially addressed.

An industrial application example is described. It presents results from a case where the developed system is used for simulation of flexible ring grinding with material removal.

Part II is motivated by the need to reduce the computation time. The availability of the new cost-efficient multiprocessor computers triggered the development of the presented hybrid parallelization framework.

The framework includes a multilevel scheduler implementing work-stealing strategy and two feedback based loop schedulers. The framework is designed to be easily portable and can be implemented without any system level coding or compiler modifications.

Part III is motivated by the need for inter-operation with other simulation tools. A co-simulation framework based on the Transmission Line Modeling (TLM) technology was developed. The main contribution here is the framework design. This includes a communication protocol specially developed to support coupling of variable time step differential equations solvers.

The framework enables integration of several different simulation components into a single time-domain simulation with minimal effort from the simulation components developers. The framework was successfully used for connecting MSC.ADAMS and SKF BEAST simulation models. Some of the test runs are presented in the text.

Throughout the thesis the approach was to present a practitioner's road-map. The detailed description of the theoretical results relevant for a real software implementation is put in focus. The software design decisions are discussed and the results of real industrial simulations are presented.

Acknowledgments

Many people contributed and rendered possible the work presented in this thesis.

First, I would like to thank my supervisor Dag Fritzson at SKF who suggested many of the ideas realized in this work, encouraged me to tackle the problems during the whole period of studies, and gave many important comments on the text of the thesis.

Many thanks to the co-authors of the papers that were presented over the years of studies: Alexander Siemers, Lars-Erik Stacke, Mikael Holgerson, Jonas Ståhl, Stathis Ioannides and Dag Fritzson.

I also want to express my gratitude to all the other members of the Analytics team at SKF and my colleagues at PELAB, Linköping University who have created a great working atmosphere and provided a lot of useful feedback.

Special thanks to Bodil Mattsson-Kihlström for handling all the administrative work caused by my frequent travels to Göteborg. Furthermore, I am grateful to all the administrative staff at IDA for providing the necessary support.

I would like to thank SKF, ECSEL (Excellence Center for Computer Science and Systems Engineering in Linköping), The Knowledge Foundation (KK-stiftelsen), and The Foundation for Strategic Research (SSF/ProViking) for the financial support.

Finally, I would like to thank my wife Olga, my parents in Russia, and my relatives and friends in different countries around the world for constant moral support and belief in my ability to do the work and write this thesis.

Iakov Nakhimovski
Göteborg, January 2006

Contents

Chapter 1

Thesis Overview

1.1 Introduction

Simulation technology is becoming increasingly important to industry. Modeling and simulation environments and tools can be seen as virtual test rigs that facilitate replacement of long and costly experiments on a real test rig with faster, cheaper, and more detailed investigations using a computer.

Mechanical systems are obvious candidates for simulation for many reasons, some of them are listed here:

- Building a prototype mechanical system just to test a new design is very expensive and time consuming. Sometimes the manufacturing of a slightly modified part at a factory or workshop requires several weeks.
- Many new machine designs may prove dangerous to use without simulation in advance. Therefore, simulation nowadays is a must, e.g. in airplane or car design.
- Some measurements in a real dynamic system are very difficult or even impossible to perform. However in a simulation all the variables are accessible.
- Tuning the system parameters is much easier to perform in a simulated system than in a real machine.
- Some particular effects, which in real systems are obscured by some other phenomena, can be isolated and carefully studied in a simulation. On the other hand, in other situations such small effects can be completely neglected to allow more careful study of the main relationships.

The general development of simulation software goes in the direction of simulation of complete systems with more and more detailed and accurate analysis. Multi-domain and multi-physics simulations have become the reality. The models for control systems, mechanical components, hydraulics, etc, are brought into integrated simulation environments. At the same time more advanced and accurate models are becoming available in each particular field.

An industrial simulation system is always the result of interdisciplinary research where contributions from different fields are integrated into a single generally useful system. In the case of flexible multibody simulation systems, contributions from mechanics, numerical analysis, and computer science are necessary. Here, mechanics provides the mathematical models of physical phenomena in the form of equations; numerical analysis comes up with the methods and algorithms for solution of the equations, and computer science gives the design guidelines for efficient and well structured implementation of the model on a computer.

The motivation for this thesis work is the need for multibody dynamics simulation tools supporting detailed contact analysis.

Part I of the thesis is primarily motivated by the need for a structural elasticity model for dynamics simulations with moving contacts on multiple surfaces. Modeling of distortion, due to the initial residual stresses in the surface layer combined with uneven material removal, is an additional requirement.

Part II is motivated by the need to reduce computation time. The availability of new cost-efficient multiprocessor computers triggered the development of the presented hybrid parallelization technique.

The need for inter-operation with other simulation tools has motivated the development of the co-simulation framework presented in Part III. The idea is to enable different tools to model and simulate subsystems (e.g., different parts of a mechanical product) and develop a framework to allow dynamic information exchange between these tools.

Throughout this thesis the approach has been to present a practitioner's roadmap. The detailed description of the theoretical results relevant for a real software implementation is put in focus. The software design decisions are discussed and the results of real industrial simulations are presented.

1.2 Contributions

This thesis contributes to techniques and methods for modeling and simulation of specialized mechanical systems. The techniques have been implemented in the BEAST (BEAring Simulation Tool) system described in the appendix on page 9 and evaluated on industrial applications. The three parts of the thesis contribute in three different directions.

Part I contributes with:

- A new set of deformation mode shapes based on series of mathematical functions. The proposed approach can be efficiently used for flexible dynamics simulation involving detailed contact analysis.

- The same set of mode shapes is then successfully used for heat-flow simulation.
- The developed model is currently actively used for industrial projects.

Part I starts by providing a detailed practical introduction into the modeling of flexible mechanical components using the floating reference frame formulation. Both the mathematical model and the software design approach are presented. The intention is to provide the equations in a form suitable for straight-forward software implementation. A description of the system architecture and software class design is also provided. Issues specific for flexible multibody systems as compared to systems with only rigid bodies are specially addressed. The text, therefore, may serve as a guide for a developer of a flexible multibody simulation system. A reader interested in stricter mathematical development and proofs would, however, need to consult other references.

The thesis subsequently focuses on the development of a new set of deformation mode shapes based on well-known series of mathematical functions. Such mode shapes are particularly suitable for modeling of flexible rings in grinding simulations. More generally, the proposed approach can be efficiently used for flexible dynamics simulation involving detailed contact analysis.

The same set of mode shapes is then successfully used for simulation of heat flow. This again demonstrates the simplicity of the approach with the general shapes providing a way to simulate new physical phenomena in a stand-alone tool.

The developed model has been implemented in the BEAST industrial strength simulation tool at SKF.

Part II presents the work that improves the computational efficiency of the simulation by presenting algorithms that enable the use of shared-memory parallel computers by the simulation code. The contributions are:

- Framework design enabling realization of modern work-stealing scheduling approaches in the context of implementations using any standard *C++* compiler.
- Two feedback based dynamic loop scheduling algorithms.
- General guidelines on the use of hybrid, i.e., mixed distributed and shared memory, parallelization.

The legacy code has previously been parallelized for distributed memory computers. The thesis presents an approach that enables both plain shared-memory parallelization of the serial code, and hybrid parallelization of the distributed memory code.

Two feedback based dynamic loop scheduling algorithms are also presented. Of the two algorithms one algorithm requires timing data for each loop iteration and is suitable for loops with minimal dependencies between the iterations. If the iterations have more significant dependencies, the second algorithm, which is a generalization of the Feedback Guided Dynamic Loop Scheduling (FGDLS) [1] should be used. The generalized algorithm renders FGDLS suitable for a work-stealing framework by tracking the time that different processors spend working on the loop.

The described approaches were implemented within the BEAST toolbox and reduced wall-clock computation time by 1.8 on a cluster of two-processor SMP nodes for some real simulation cases. The theoretically maximum improvement in this case is 2.

Part III describes a coupled simulation framework based on the Transmission Line Modeling (TLM) technology [7]. The work is focused on co-simulation of mechanical systems.

The main contributions here are:

- TLM based co-simulation framework design.
- A new communication protocol specially developed to support coupling of variable time step differential equations solvers.

The framework design includes a TLM plugin interface design enabling easy integration of different simulation tools into the framework.

The framework has been successfully used for connecting MSC.ADAMS [5] and SKF BEAST simulation tools running coupled sub-models. Some of the test runs are presented in the text.

1.3 Papers

The papers below cover part of the material presented in this thesis.

- I. Nakhimovski, D. Fritzson, and M. Holgerson. Dynamic Simulation of Grinding with Flexibility and Material Removal. *Proceedings of Multibody Dynamics in Sweden 2001*, http://www.sm.chalmers.se/MBDSwe_Sem01/Pdfs/IakovNakhimovski.pdf, 2001.
- A. Siemers and I. Nakhimovski. RunBeast- Managing Remote Simulations. *Proceedings of SIMS 2001*, pages 39–46, 2001.
- A. Siemers, I. Nakhimovski, and D. Fritzson. Meta-modelling of mechanical systems with transmission line joints in Modelica. *Proc. of Modelica Conference*, 2005.

- M. Holgerson, L.-E. Stacke, and I. Nakhimovski. Cage Temperature Prediction through Dynamics Simulation. Extended Abstract. *STLE Annual Meeting*, Las Vegas, Nevada, 15-19 May, 2005.
- Iakov Nakhimovski and Dag Fritzson. Modeling of Flexible Rings for Grinding Simulation. *Multibody Dynamics 2005*, Madrid, Spain, 2005. ECCOMAS Thematic Conference.
- S. Ioannides, L.-E. Stacke, D. Fritzson, and I. Nakhimovski. Multibody rolling bearing calculcations. *World Tribology Congress III*, 2005.
- D. Fritzson, J. Ståhl, and I. Nakhimovski. Transmission line co-simulation of rolling bearing applications. Submitted to Journal of Multibody Dynamics, 2006.
- I. Nakhimovski and D. Fritzson. Hybrid Parallelization of Multibody Simulation with Detailed Contacts. Submitted to EuroPar'06 conference, 2006.

1.4 Work not Published in Papers

The work presented in the following parts of this thesis is not presented in the papers mentioned in the previous section:

- Chapters 3-5, 8-9, 11, A Practitioners Road-map to Flexible Multibody Simulations.
- Chapter 6, Flexible Body Exchange Formats for Systems Engineering.
- Section 7.5, Reducing the Number of Flexible Shapes.
- Chapter 13, Thermal Analysis with General Shape Functions.
- Chapter 19, Feedback Based Loop Scheduling Strategies.
- Section 24.5 and Chapter 25, Design of TLM Based Coupled Simulation Framework
- Section 25.1, Supporting Variable Time-step Differential Equations Solvers in a TLM Co-simulation.

1.5 Conclusions

More detailed discussion of the results of each Part can be found in the corresponding chapters. Here, we would like to highlight some of them.

A practical overview of the floating frame of reference formulation constitutes a large fraction of Part I. We believe that such an overview may be useful for a developer working on extending legacy rigid body simulation code with the flexible body model. The discussion of the system design changes is also highly relevant in this case.

A set of deformation mode shapes utilizing well known series of mathematical function are proposed as a basis for a modes set. The advantages of using the well known mathematical functions are calculation efficiency, ease of implementation and use. The proposed mode set is particularly suitable for simulations involving detailed contact analysis such as the grinding simulation presented in the thesis. The same set of mode shapes has been applied for modeling of temperature distribution inside bodies.

It should, however, be noticed that the general mathematical functions may not be optimal when complex geometry is involved or if flexible bodies are connected with ideal joints and not contacts. Therefore, import of reduced standardized finite element models into the presented flexible multibody framework is supported and discussed.

Linux clusters of multiprocessor computers are becoming increasingly popular. Efficient use of such computer architectures is an important problem for many applications. The hybrid parallelization approach proposed in Part II enables higher speed-up for some important application cases.

The presented approach utilizes a modern work-stealing strategy developed for other frameworks [4, 9]. The strategy is modified so that it can be put into a portable *C++* library. This enables incremental parallelization of the legacy code while providing the features found in more advanced packages.

The developed parallelization approach complemented with the presented feedback loop scheduling algorithms may be useful for most time-dependent process simulation codes. The approach is particularly suited for cases where speedup of legacy message-passing code is limited by task granularity.

The approach to a coupled simulation framework described in Part III is an effective solution for the connection of mechanical simulation tools. The discussion of the communication protocol for variable time step differential equation solvers should also be interesting for other types of simulations.

1.6 Future Work

Dynamics simulations with detailed analysis of moving contacts involving flexible bodies with complex geometries and large potential contact surfaces is an open research area. Further research is needed to identify appropriate sets of deformation shapes.

Both heat transfer and structural deformation models are presented in this thesis. Simulation of coupled thermo-mechanical processes is a natural continuation of this work.

A key issue for successful industrial utilization of an approach in the area of hybrid parallelization is its transparency for the application user. Therefore, an algorithm for automatic detection of the best parallelization approach to be used for a particular simulation case and hardware combination seems to be the most important continuation project.

Supporting tools are necessary to simplify meta-modeling when complex submodels are involved in a co-simulation. Such tools should enable faster meta-model creation and minimize the potential for modeling mistakes.

As TLM approach proved to be useful in different physical domains further expansion of the supporting framework is an important development. Such expansion is also necessary to further prove the general applicability of the presented design.

Introduction Bibliography

[1] J. M. Bull. Feedback guided loop scheduling: Algorithms and experiments. In *Proceedings of Euro-Par'98*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[2] Scott D. Cohen and Alan C. Hindmarsh. CVODE User Guide. Netlib, 1994.

[3] D. Fritzson, L.-E. Stacke, and P. Nordling. BEAST - a Rolling Bearing Simulation Tool. *Proc. Instn Mech Engrs, Part K*, 213, 1999.

[4] Yonezawa Laboratory. StackThreads/MP Home Page. *http://www.yl.is.s.u-tokyo.ac.jp/sthreads/*.

[5] MSC Software Corporation homepage. *http://www.mscsoftware.com/*.

[6] Patrik Nordling. The Simulation of Rolling Bearing Dynamics on Parallel Computers. Licentiate thesis, Linkoping University, Sweden, 1996.

[7] Krus P. Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints. *Transactions of ASME, Journal of Dynamic Systems Measurement and Control.*, Dec 1999.

[8] A. Siemers and I. Nakhimovski. RunBeast- Managing Remote Simulations. In *Proceedings of SIMS 2001*, pages 39–46, 2001.

[9] MIT Supercomputing Technologies Group. The Cilk Project. *http://supertech.lcs.mit.edu/cilk/*.

Appendix: The BEAST Toolbox

BEAST (BEAring Simulation Tool) [3] is a modeling and simulation program developed at SKF for the simulation of fully three-dimensional mechanical models. It was originally used to perform simulations of bearing dynamics on any bearing type. The BEAST tool has made possible studies of internal motions and forces in a bearing under any given loading condition. The bearing can be put under load in any way the user required. BEAST can be viewed as a virtual test rig where the user has full insight into the dynamic behavior of the bearing components.

The toolbox development in terms of design generalization has widened the application area of the program. Later versions of BEAST include models of grinding machines, experimental engines, transmissions, etc. BEAST can now be seen as a general multibody simulation tool specialized in detailed contact analysis.

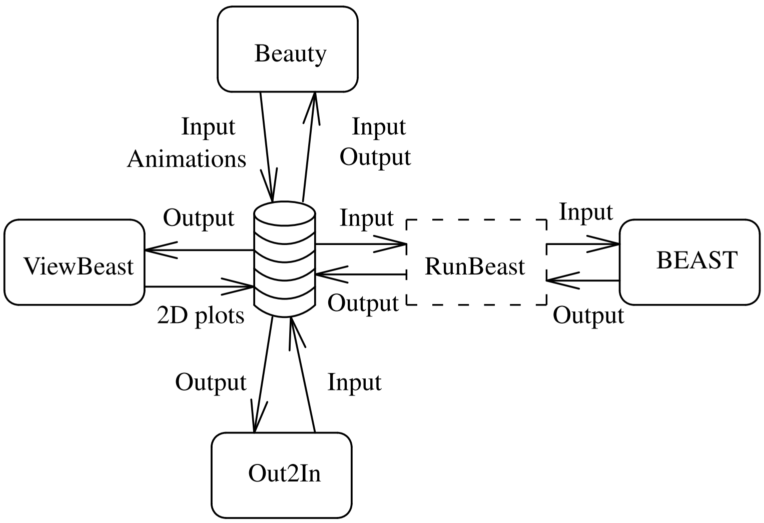


Figure 1-1: *Programs in the BEAST toolbox.*

The BEAST toolbox includes a set of tools that are interacting with each other by means of different kinds of files. Figure 1-1 shows the programs in the toolbox and their interactions with the file storage. The tools are:

Beauty is an advanced 3D graphical tool designed for setting up the model and simulation parameters in the input files and visualizing the output files. Some of the features of the tool include animating the simulated sequence from different viewpoints, visualization of force vectors and surface associated data (such as pressure distribution). The visual representation of the simulation results in Beauty in an easily understandable way contributes to a quicker interpretation of the result and popularity of the complete toolbox among the users. Figure 1-2 shows a snapshot of the Beauty working on an input file for a ball screw model.

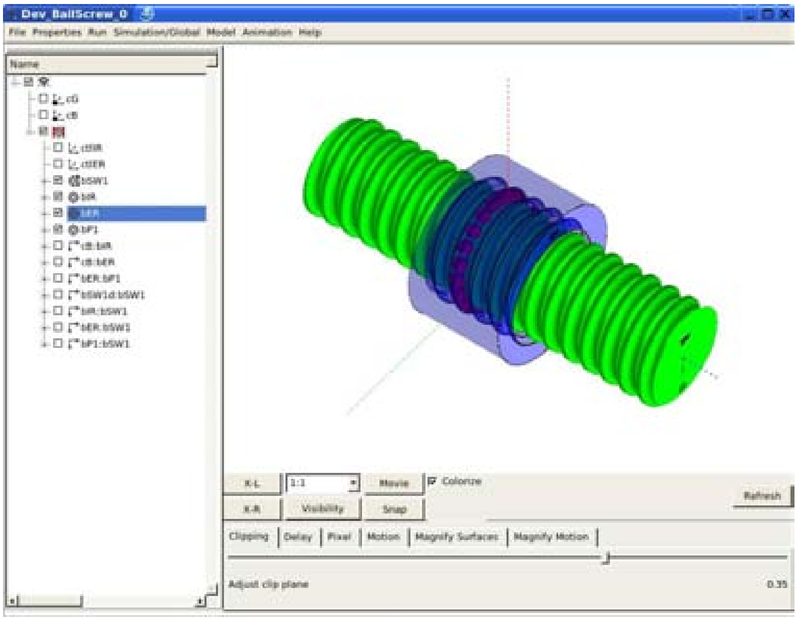


Figure 1-2: *A snapshot of the Beauty displaying a ball screw model.*

ViewBeast is specially designed for 2D plot presentation and analysis. In addition to the basic functionality, i.e., curve plotting, different operations on the simulation results are possible. For example, the user can apply Fourier transforms to a variable or specify his/her own function on several variables.

RunBeast is a remote simulation interface system. Its architecture is described in [8]. The tool provides a user-friendly interface for submitting a simulation on a remote computation server which is often a parallel computer.

BEAST is the main simulation program. It reads in a model specification from input files, performs the simulation and generates a set of output files containing the results. Most of the developments of this thesis were realized within the framework of this tool.

Out2In is a small utility that allows generation of new input files from the simulation output files. In this way the user is given a chance to interrupt the simulation, modify some parameters, and continue the simulation from the moment where the previous run was terminated. This means that is the results of one simulation can be used as the initial conditions for another.

From the mathematical point of view the BEAST system solves the simultaneous system of Newton-Euler equations of motion for every body in the mechanical system. The Newton-Euler equations are formulated as second order ordinary differential equations (ODEs) on explicit form. The second order differential equations system is rewritten as a first order system. Typical characteristics of such ODEs are: mathematical stiffness, very high numerical precision of the solution required by the application, and computationally expensive evaluation of the derivatives. See Part I and [6] for more details. The tool uses a modified version of the CVODE [2] differential equation solver for the numerical solution of the resulting system of equations. This solver is one of the most well known free implementations of the implicit backward differentiation formulas (BDF) integration scheme. The most important features of this solver are:

- A variable order multistep method that requires high continuity of the derivatives.
- Adaptive time step changes based on a local error estimate.
- Efficient use of the solver requires a fast procedure for the system Jacobian calculation (i.e., the matrix of partial derivatives of the state derivatives with respect to the state variables).

The original solver has been extended to handle special kinds of Jacobians (block diagonal with borders and sparse) and to perform a simultaneous RHS and Jacobian calculation [6].

For the work described in this thesis, the BEAST system has been the assumed implementation platform. All the ideas for the modeling, the algorithms, and the design decisions were tried for applicability and usability in this system. Hence we can say that the system has become a test implementation of the presented approaches. Its successful industrial usage validates the correctness of the decisions taken.

Part I

Modeling and Simulation of Flexible Bodies for Detailed Contact Analysis in Multibody Systems

Notation

The notation used throughout the thesis is described below. The section can be used as a reference when reading the equations presented in the thesis.

The notation used here is both the *direct matrix notation* of vectors and tensors, and the *vector-dyadic notation*. Therefore, the basic quantities and some formulas are given in both notations. The orthonormal right-handed rectangular Cartesian coordinate system is used.

For further reading we refer to [15, 13, 18, 14] or other well-known standard publications.

Variables & basic definitions

$\boldsymbol{A}_{2/1}$	transformation matrix from coordinate system 1 to system 2.
a	a scalar.
$\vec{\boldsymbol{a}}$	$a_i\hat{\mathrm{e}}_i$ a vector or first-order tensor on which vector transformation rules can be applied.
$\hat{\mathrm{a}}$	a unit vector.
$\tilde{\vec{\boldsymbol{a}}}$	a 3×3 skew symmetric matrix constructed from the components of 3-element vector $\vec{\boldsymbol{a}}$. That is a matrix: $\tilde{\vec{\boldsymbol{a}}} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$ The important properties of a skew symmetric matrix are: $\tilde{\vec{\boldsymbol{a}}} = -\tilde{\vec{\boldsymbol{a}}}^{\mathrm{T}}$, $\vec{\boldsymbol{a}} \times \vec{\boldsymbol{b}} = \tilde{\vec{\boldsymbol{a}}} \cdot \vec{\boldsymbol{b}}$
$\boldsymbol{a}$	normally a second-order tensor or matrix, in some cases an array.
$[a]$	$a_{ij}\hat{\mathrm{e}}_i\hat{\mathrm{e}}_j$ a second-order tensor or matrix.
$\vec{\boldsymbol{a}}^{\mathrm{T}}, \boldsymbol{a}^{\mathrm{T}}$	transpose of $\vec{\boldsymbol{a}}$ and $\boldsymbol{a}$.

$\vec{\boldsymbol{a}}^{(\mathrm{c})}$	a 3-element vector expressed in component form in coordinate system 'c'.
$\dot{\vec{\boldsymbol{a}}}\Big\vert_{\mathrm{c}}, \dfrac{\partial\vec{\boldsymbol{a}}}{\partial t}\Big\vert_{\mathrm{c}}$	time derivative of a vector with respect to coordinate system c. If no coordinate system is specified then the global inertial coordinate system is assumed.
$a = \vec{\boldsymbol{b}} \cdot \vec{\boldsymbol{c}}$	$b_j c_j$ (or $\vec{\boldsymbol{b}}^T \cdot \vec{\boldsymbol{c}}$) dot product of two vectors
$\vec{\boldsymbol{a}} = \vec{\boldsymbol{b}} \times \vec{\boldsymbol{c}}$	cross product of two vectors
$\vec{\boldsymbol{a}} = \boldsymbol{b} \cdot \vec{\boldsymbol{c}}$	$b_{ij} c_j \hat{\mathrm{e}}_i$
$\boldsymbol{a} = \boldsymbol{b} \cdot \boldsymbol{C}$	$b_{ij} c_{jk} \hat{\mathrm{e}}_i \hat{\mathrm{e}}_k$ dot product of two matrices
$C_{1,2}$	a constant that gives the relative stiffness between object 1 and object 2.
$\boldsymbol{C}_{\mathrm{ff}}$	damping matrix.
$\boldsymbol{C}_{\theta\theta}$	heat capacity matrix.
$c1.ctl_1$	a control point ctl_1 defined in the coordinate system $c1$.
$\hat{\mathrm{e}}_i$	unit base vector in a rectangular Cartesian axis system
e_1, e_2, e_3, e_4	Euler parameters
E	elastic (Young's) modulus.
$\boldsymbol{E}$, $\boldsymbol{E_v}$	matrices of elastic and viscous material constants.
$\boldsymbol{H}_{\theta\theta}, \vec{\boldsymbol{H}}_{\theta}, \vec{\boldsymbol{H}}_{\mathrm{ext}}$	generalized heat flow contributions.
h_c	heat transfer coefficient.
$\vec{\boldsymbol{F}}_{1,\mathrm{ext}}$	a external force refering to object 1
$\boldsymbol{I}$	an identity matrix
$\boldsymbol{J}_{2/1}$	moment of inertia for object 2 relative to coordinate system 1
$\vec{\boldsymbol{J}}_1$, $J_{kl}, \bar{\boldsymbol{J}}_{kl}$	some of the inertia shape integrals defined by the equations 3-19, 3-20 and 3-23 respectively.
$\bar{\boldsymbol{J}}_{\mathrm{ee}}, \bar{\boldsymbol{J}}_{\mathrm{ef}}$	sub-blocks of complete mass matrix $\boldsymbol{M}$ associated with angular velocities. The subscipts specify the degrees of freedom (DOF) corresponding to the particular sub-block. 'e' the three rotational DOFs and 'f' - n_f elastic DOFs.
$\boldsymbol{K}_{\mathrm{ff}}$	stiffness matrix.
$\boldsymbol{K}_{\theta\theta}$	thermal conductivity matrix.
L	length or width
$\vec{\boldsymbol{M}}_{1/\mathrm{c}}$	a moment relative to the origin of system c refering to object 1
$\vec{\boldsymbol{M}}_{1,\mathrm{ext}}$	an external force couple, i.e. moment, refering to object 1

$\vec{\boldsymbol{M}}_{1,2}$	a force couple, i.e. moment, refering to object 1 and caused by interaction with object 2
$\boldsymbol{M}$	complete symmetric mass matrix of a body as used in Newton-Euler equations.
$\boldsymbol{M}_{\text{RR}}, \boldsymbol{M}_{\text{ee}}, \boldsymbol{M}_{\text{ff}}$	sub-blocks of complete mass matrix $\boldsymbol{M}$. The subscipts specify the degrees of freedom (DOF) corresponding to the particular sub-block. 'R' specify the three translational DOFs, 'e' the generalized rotational DOFs and 'f' - n_f elastic DOFs. The non-diagonal blocks of the matrix use mixed subscripts, e.g., $\boldsymbol{M}_{\text{Re}}$ stands for the sub-block defining inertia coupling between the translational and rotational DOFs.
m_i, ρ_i	mass and mass density of object i. The subscript i is omitted if the discussion clearly indicates a single object.
$\vec{\boldsymbol{p}}_{\text{a/b}}$	a "point", i.e. the vector between a and b. If b is a coordinate system, then it refers to the origin of system b.
$\vec{\boldsymbol{p}}_{\text{f}}^{(\text{b})}$	a "point" vector expressed in the coordinate system b.
$(\vec{\boldsymbol{p}}_{\text{f}}^{(\text{b})})_k$	an element of the point vector associated with the axis k. Axes are named 'x', 'y', 'z' or '1', '2', '3' in different equations as appropriate in the particular context.
$\vec{\boldsymbol{Q}}$	total generalized force tensor as appears on the right hand side of the Newton-Euler equation defined in 3.4.
$(\vec{\boldsymbol{Q}})_R, (\vec{\boldsymbol{Q}})_\alpha, (\vec{\boldsymbol{Q}})_{\text{f}}$	parts of a generalized force vector corresponding to translational, rotational and elastic DOFs respectively.
$\vec{\boldsymbol{Q}}_e$	external generalized force tensor.
$\vec{\boldsymbol{Q}}_v$	quadratic velocity tensor.
$\vec{\boldsymbol{Q}}_{i_f}$	generalized force due to internal residual stress release.
$\vec{\boldsymbol{R}}_{2/1}$	the location of the origin of coordinate system 2 relative to the origin of coordinate system 1
$\boldsymbol{S}$	a shape matrix, that is a 3 $\times n_f$ matrix representing the deformation field of a flexible body. $\boldsymbol{S}_k$ is used to denote the k-th row of the matrix, that is the row associated with the k-th Cartesian coordinate axis.
$\vec{\boldsymbol{S}}_t$	a dynamic inertia shape vector defined by the Equation 3-25.
$\bar{\boldsymbol{S}}$	one of the inertia shape integrals defined by Equation 3-21.
$\bar{\boldsymbol{S}}_{kl}$	nine inertia shape integrals defined by Equation 3-22. The subscripts k and l specify corresponding coordinate axes.
$\boldsymbol{S}_\theta$	the space dependent shape functions in thermal equation.

T($\vec{\boldsymbol{p}}$,t)	temperature distribution as a space and time dependent function.
u_k	components of the deflection vector $\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})}$ in the direction of coordinate axis k. Where k is one of $(1, 2, 3)$ or (x, y, z) for a Cartesian coordinate system and one of (r, ϕ, z) for a cylindrical coordinate system.
$\vec{\boldsymbol{x}}_{\mathrm{f}}$	the tensor of the generalized elastic coordinates of a flexible body.
V	volume.
$\vec{\boldsymbol{v}}$	velocity.
W, W_s, W_e	potential energy, strain energy and work of external forces.
$\vec{\boldsymbol{\alpha}}_{\mathrm{a}/1}\Big\vert_{2}$	angular acceleration of 'a' relative system 1 differenciated in system 2. If no system 2 is specified then system 1 is assumed.
δ_{ij}	Kronecker delta:δ_{ij} equals one iff $i = j$ and is zero otherwise.
φ	rotation angle.
$\boldsymbol{\varphi}_{2/1}$	a first order tensor containing three angles representing the rotation of coordinate system 2 relative to system 1. This tensor cannot be expressed in a coordinate system.
$\vec{\boldsymbol{\Omega}}_{2/1}$	the angular speed of coordinate system 2 relative to coordinate system 1 .
$\vec{\boldsymbol{\omega}}_{\mathrm{a}/1}$	the angular speed of “a” relative to coordinate system 1.
ω_i	i-th vibration mode frequency [radians/second].
$\vec{\boldsymbol{\epsilon}}$	elastic strain tensor.
$\vec{\boldsymbol{\tau}}$	viscose stress tensor.
$\vec{\boldsymbol{\sigma}}$	elastic stress tensor.
λ, μ	Lame's constants.
γ	Poisson's ratio.
$\vec{\boldsymbol{\theta}}$	infinitesimal rotation angles in Chapter 3.
$\vec{\boldsymbol{\theta}}$	state variables for the thermal equation in Chapter 13.
$\boldsymbol{\Lambda}$	diagonal matrix of eigenvalues.

Chapter 2

Introduction

2.1 Simulation of Flexible Mechanical Components

Historically, two main types of simulation of mechanical systems have dominated:

- Multibody dynamics simulations typically deal with systems of interconnected rigid bodies. In many cases the connections are limited to idealized joints (e.g., revolute, prismatic, etc) and traditional force elements (e.g., spring, damper, external load). Multiple dynamic contacts are often important parts of the model and a relatively simple contact model is most often employed for performance reasons. Multibody systems are mostly used for time domain dynamics simulations.

- Finite element tools are mostly used for static and quasi-static (with constant angular velocities) analysis of systems and components as well as for modal analysis of structures. They are also used for detailed simulations of very small and specific parts of a system, e.g., a single contact. Finite element analysis generally requires more computational effort than multibody analysis for a single time instance.

Following the general trend of making more complete simulations, the finite-element analysis and multibody simulation systems are now exploring each others domain. Multibody systems now include deformable bodies and more detailed analysis in the models. At the same time finite element tools are beginning to be used for more dynamics simulations.

This thesis contributes to this process on the multibody systems side. It discusses the floating-frame of reference formulation, which is gradually becoming the standard way to simulate flexible bodies in the context of a multibody simulation. To further narrow the field of the research we limit our development to multibody systems specializing in detailed contact analysis, such as the BEAST (BEAring Simulation Tool) system [12].

The aim of Part I is to describe an approach for building a flexible multibody simulation system as an extension to the existing rigid body system. Therefore, all aspects of the system development have to be considered. We discuss the necessary mathematical model for deformable bodies simulation, provide some important numerical algorithms, and suggest an object-oriented design of the computer implementation.

2.2 Model Requirements

The main objective of Part I is to provide the elastic body model suitable for dynamic simulations involving multiple moving contacts. The main issue is an efficient model of the overall structural deformation. Local deformation in a contact is covered by the contact model discussed in other reports on the BEAST (*BEAring Simulation Tool*) and GRIT(*GRInding simulation Tool*) systems [12, 34, 21].

The model requirements can be subdivided into three equally important groups. They are: computational performance or efficiency, accuracy, and potential to incorporate the analysis requisite for grinding.

- The computational complexity of a flexible system model depends mostly on the number of state variables used for the flexible body. Hence, an efficient model should use as few states to describe flexibility as possible. The system of ODEs (ordinary differential equations) generated in dynamic contact analysis problems is in most cases mathematically stiff. Commonly used numerical integrators for such systems are multistep variable time step algorithms. Important consideration for such solvers is the length of the time step in the integration. Longer time steps mean fewer computations and shorter simulation times. Therefore an efficient flexible body model for the case of moving contact should not force the numerical solver to take significantly smaller time steps compared to the same model with rigid components. That is, higher frequencies in the system are expected to be associated with the contact forces calculations and not the flexible body eigenfrequencies.

- Let us provide a very simple example to illustrate the need for accuracy. In Figure 2-1 a rotating elastic ring pressed between two plates is shown. From the physical point of view the radial stiffness of the ring is independent of its orientation. Hence, assuming the constant force acting on the ring, the elastic deflection - measured as the distance between the plates - should become constant after some transient process. When simulating this in the time domain, it is important that we do not introduce vibrations due to modeling or numerical errors. The situation can get worse for a more complex model where small, induced vibrations can occasionally resonate with some resonance frequency of the system leading to completely unrealistic results.

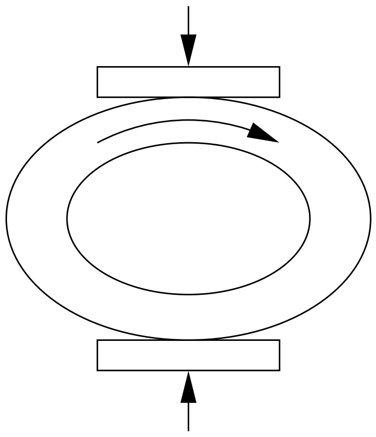


Figure 2-1: *A rotating flexible ring pressed between two plates.*

- In the third group of requirements one can mention distortion, due to the initial residual stresses in the surface layer combined with uneven material removal.

2.3 Related Work

A 3-D FEM model can be used to model flexible bodies. The required procedures are described in a wide variety of articles and books. References [3, 9] provide extensive discussions of the methods. However, finite element methods for transient analysis often suffer from long (days, weeks) computation times, and the radial stiffness computed at the element nodes will slightly differ from the stiffness between the nodes. This is due to the fact that the deflection between finite element nodes is normally computed using a low order interpolation scheme. In our simple example described above, variation in radial stiffness leads to induced non-physical vibration with the frequency dependent on the number of elements on the circumference of the ring and rotational speed. Without doubt the effect can be minimized and rendered negligible by using a large number of finite elements but that would lead to an even longer computational time. The effects of the discretization errors in dynamic simulations are discussed in [26].

A commonly used approach to efficiently simulate structural elasticity is to employ some kind of reduction technique [42, 9, 41, 43] from a detailed finite element model. Reduction schemes often dramatically decrease the number of state variables used in dynamic simulation. The reduction is done prior to the simulation using a FEM-tool. The result of a reduction procedure is a set of assumed displacement shape functions that is used in the dynamics simulation.

The main challenge of a reduction with mode synthesis is to select a small set of superimposed mode shapes that realize acceptable calculation precision for the simulation. It is one of the most complicated and discussed problems in flexible multibody simulation (see [11, 30]). The mode shapes are calculated using data from static load cases and/or cases where load varies with certain frequency superimposed with eigenshapes. These resulting mode shapes can only be calculated for some specific boundary conditions, i.e., given attachment or interface nodes. See Section 5 for more discussions.

However, in the case of a moving contact the boundary conditions change dynamically. Interaction between any two points of the contacting surfaces is possible. Hence many nodes on the surfaces have to be marked as interface nodes. A large number of interface nodes leads to a large number of state variables and makes dynamics simulation slower.

The reduced model still has the same vibration problem as the FEM model, due to the discrete interface nodes on the contact surfaces. The solution is to employ global shape functions for the interface motion, i.e., control the surface interface nodes by the shape functions. This can be done directly in the reduction scheme [43], or as a separate transformation step after a standard reduction with all interface nodes.

These reduced models cannot handle the requirements from the third group mentioned above in Section 2.2 well, i.e., non-linear effects due to initial stresses and material removal.

2.4 Part I Overview

The dynamics equations for the rigid bodies in a multibody system can be defined in terms of body mass, the inertia tensor, and the forces and torques vectors acting on the body. Dynamics equations for linear structural systems require definition of the system mass and stiffness matrices as well as the vector of generalized forces. Here we summarize the equation of motion for deformable bodies that undergo large translational and rotational displacements using the floating frame of reference formulation. The set of the inertia shape integrals required for the equations is defined. These inertia shape integrals, that depend on the assumed displacement field, appear in the non-linear terms that represent the inertia coupling between the reference motion and the elastic deformation of the body. Some numerical procedures required to perform a dynamics simulation involving flexible bodies, including numerical volume integration and Jacobian matrix evaluation, are also described.

There are several kinds of analyses that are possible for flexible body models but not rigid body models. We describe three general types of such analysis: free-body eigenmodes analysis, body deformation under static load and quasi-static conditions. Results of these kinds of analysis can be used to quickly assert the basic properties of the flexible body model before running a computationally heavy dynamics simulation. There is also a discussion on a more specialized kind of analysis: ring distortion due to the residual stress release during grinding.

We provide some examples that were used to verify the correctness of the mathematical model presented.

The main intention of Part I is to provide a complete and possibly compact description of the equation and procedures necessary to perform a dynamics simulation of a flexible multibody system and some simpler kinds of single body analysis. The test is therefore system implementation oriented and does not generally cover the mathematical derivation of the presented equations. The complete derivation of these equations can be found in most books on flexible dynamics (see [31]).

Part I is structured in the following way. Section Notation provided in the beginning can be used as a reference when reading the equations presented in the rest of the text. Chapter Introduction then provides an overview of Part I along with motivations for the mathematical model development.

The mathematical model of an elastic body using the floating frame of reference formulation and assumed shape function is presented in Chapter 3 where all the important equations that define the model are listed and explained. Appendix B shows how the presented equations can be used for modeling a flexible ring.

Using the presented general equations some basic single body analysis types can be formulated. Three types of such analysis that are important in our target application are presented in Chapter 4.

Interfacing Finite Element Analysis (FEA) tools to generate mode shapes for an elastic body model is an important aspect for flexible multibody codes. Therefore we continue with two chapters on this subject. Different FEA methods, typical for the generation of mode shapes, are presented, two common interface formats for importing FEA data into multibody-dynamics frameworks are considered, and an example of how such a format can be used.

As we have already mentioned in Section 2.2, detailed contact analysis imposes some special requirements on the mode shapes in elastic body models. To fulfill these requirements we propose the use of series of global shape functions, e.g, Fourier series and Chebyshev polynomials. Chapter 7 describes the motivations and principles of the approach.

The thesis continues with a presentation of the mean-axis condition for the most advantageous choice of the floating reference frame for our application.

In the next chapter we get closer to the implementation of the approach in a real simulation framework. Some design decisions that were made to provide an efficient user interface for specifying boundary and external loading conditions in simulations involving flexible bodies are introduced.

Chapter Verification demonstrates the correctness of the selected model for some theoretical cases.

The system design discussion continues in the next chapter. Here we focus on system architecture and design for a multibody simulation system that includes flexible components. We specially address the problem of transition from a multibody system that allows only rigid bodies into the system with both kinds of components.

The chapter on simulation of grinding presents an application example for the developed system. The importance of the structural flexibility and its influence on the simulation results for the particular application are discussed.

We go on by presenting application of the general shape functions approach for the modeling of dynamic thermal processes. The provided application examples strengthen the claim of practical importance of the approach. Appendix C demonstrates the use of the approach for a simple heat transfer problem.

The last two chapters summarize the results of the performed work and discuss possible directions for the continuation of research and development in this area.

Chapter 3

Key Equations in Flexible Body Dynamics

This section introduces the mathematical model of an elastic body using the floating frame of reference formulation and assumed shape function. The text is strongly based on the results of [31] and therefore no derivation for the most of the presented equations is given. The relations that are developed in this chapter and cannot be found in the reference above are:

- Representation in the different coordinate systems.
- Generalized viscosity forces and damping modeling.
- Generalized external moment.
- Generalized external body load.
- Generalized force from the residual stress release.
- An interpolation method for forces with discontinuities.
- Jacobian calculations.

The intention of this text is to provide the equations necessary for an implementation of a multibody dynamic simulation package with flexible components. The derivation of the most common equations is intentionally skipped. A reader interested in a more complete mathematical development should consult the referenced book [31] or one of the other papers developing the complete model, e.g., [37, 23].

In the following sections the subscript that normally indicates the body number will be omitted with the understanding that all vectors and matrices are associated with some particular single body. Since the equations involve only two coordinate systems - the global and body ones - one corresponding letter ('g' or 'b') will be used to specify the coordinate system where the vector is expressed or differentiated.

3.1 Flexible Body Motion and Shape Functions

A rigid body in space has six degrees of freedom that describe the location and orientation of the body with respect to the inertial reference frame. Since the relative position of two particles within a rigid body is constant the displacement of each of the particles in space can be uniquely identified from the body position. On the other hand, modeling of structural deformation requires evaluation of the displacement for every particle of the body independently. To deal with such problems classical approximation methods can be employed. The displacement of each point due to elastic deformation is expressed in terms of a finite number of coordinates:

$$\left.\begin{array}{l} (\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})})_x \approx \sum_{k=1}^{l} a_k f_k, \text{ where } f_k = f_k(\vec{\boldsymbol{p}}_0^{(\mathrm{b})}) \\ (\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})})_y \approx \sum_{k=1}^{m} b_k g_k, \text{ where } g_k = g_k(\vec{\boldsymbol{p}}_0^{(\mathrm{b})}) \\ (\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})})_z \approx \sum_{k=1}^{n} c_k h_k, \text{ where } h_k = h_k(\vec{\boldsymbol{p}}_0^{(\mathrm{b})}) \end{array}\right\} \tag{3-1}$$

where $\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})}$ gives the displacement of an arbitrary point that has coordinates $\vec{\boldsymbol{p}}_0^{(\mathrm{b})}$ in the undeformed state. The vector of displacement (or deformation vector) $\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})}$ is space- and time- dependent. The coefficients a_k, b_k, and c_k are assumed to depend only on time. The above equations can be written in the following matrix form:

$$\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})} = \boldsymbol{S} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \tag{3-2}$$

where $\boldsymbol{S}$ is the three-rows n_f-columns ($n_f = l + m + n$) space coordinate dependent *shape matrix* whose elements are the functions f_k, g_k, and h_k; and $\vec{\boldsymbol{x}}_{\mathrm{f}}$ is the vector of *elastic coordinates* that contains the time dependent coefficients a_k, b_k, and c_k. The total number of elastic coordinates (which is of course equal to the number of columns of the shape matrix n_f) should be determined from experience depending on the simulation accuracy required and the importance of the elasticity effects for the complete model. The approaches to the selection of the appropriate shapes is further discussed in Sections 5 and 7.1.

By using the outlined approximations, the global position of an arbitrary point P of the body can be defined as

$$\vec{\boldsymbol{p}}_{\mathrm{P/g}}^{(\mathrm{g})} = \vec{\boldsymbol{R}}_{\mathrm{b/g}}^{(\mathrm{g})} + \boldsymbol{A}_{g/b} \cdot \vec{\boldsymbol{p}}_{\mathrm{P/b}}^{(\mathrm{b})} \tag{3-3}$$

where

$\vec{\boldsymbol{R}}_{\mathrm{b/g}}^{(\mathrm{g})}$ defines the origin of the body reference;

$\boldsymbol{A}_{g/b}$ orthogonal rotation matrix;

$\vec{\boldsymbol{p}}_{\mathrm{P/b}}^{(\mathrm{b})}$ is equal to $\vec{\boldsymbol{p}}_0^{(\mathrm{b})} + \vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})} = \vec{\boldsymbol{p}}_0^{(\mathrm{b})} + \boldsymbol{S} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}}$ and gives the displacement of the point P in the body coordinate system

Note that it is generally possible to specify the shape matrix in different coordinate systems and transform it using rotation matrix and normal rules for coordinate transformation. However, use of the body reference frame enables the most natural and efficient choice of the shape functions. For this reason the shape matrix is always assumed to be calculated in the body coordinate system. Hence the coordinate system needs to be specified for all the tensors in the following sections.

3.2 Rotation of a Material Particle due to Deformation

In general during the deformation of a body, any element is changed in shape, translated, and rotated [35]. In order to analyze systems with rotational springs and dampers it is necessary to be able to calculate the orientation of a material point in the deformed body. In order to calculate the orientation of a material particle for the case of large deformations one has to use the properties of the deformation gradient [3]. However for the case of small deformations, where only linear terms are taken into account, the rotation can be analyzed as infinitesimal. The rotation matrix that performs the rotation of a particle from the undeformed state to the infinitesimally deformed orientation can be calculated as (see [31]):

$$\boldsymbol{A}_{f/0} = \boldsymbol{I} + \begin{bmatrix} 0 & \theta_z & -\theta_y \\ -\theta_z & 0 & \theta_x \\ \theta_y & -\theta_x & 0 \end{bmatrix} \tag{3-4}$$

where $\boldsymbol{I}$ is an identity matrix and $(\theta_x, \theta_y, \theta_z)$ are three small angles corresponding to the rotations around the three respective coordinate axes. The angles can be calculated as:

$$\begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \dfrac{\partial \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{f,y}}}{\partial z} - \dfrac{\partial \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{f,z}}}{\partial y} \\ \dfrac{\partial \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{f,z}}}{\partial x} - \dfrac{\partial \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{f,x}}}{\partial z} \\ \dfrac{\partial \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{f,x}}}{\partial y} - \dfrac{\partial \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{f,y}}}{\partial x} \end{bmatrix} \tag{3-5}$$

3.3 Using Intermediate Coordinate System

It is sometimes convenient, and/or more efficient, for the numerical solver to observe the motion of a body in a coordinate system other than the inertial one. For example, if the simulated body is a part of a complex system and it is desirable to analyze the body movements from a certain known position in the system. From the mathematical point of view, this known position is a coordinate system with known motion. In such a case a special care must be taken when working with time derivatives of position and orientation of a body. Below, the transformation rules for taking time derivatives with respect to different coordinate systems, are given.

To be more specific let us assume that two coordinate systems are involved in the analysis.

- System $'G'$ is the inertial coordinate system where the Newton-Euler equations are valid.
- System $'B'$ is a coordinate system with know motion in $'G'$. That is, the position vector $\vec{\boldsymbol{R}}_{B/G}$, linear velocity $\dot{\vec{\boldsymbol{R}}}_{B/G}\big|_G$, and linear acceleration $\ddot{\vec{\boldsymbol{R}}}_{B/G}\big|_G$ as well as rotation matrix $\boldsymbol{A}_{B/G}$, relative angular velocity $\vec{\boldsymbol{\Omega}}_{B/G}$ and relative angular acceleration $\dot{\vec{\boldsymbol{\Omega}}}_{B/G}\big|_G$ are known.

Then for a physical vector $\vec{\boldsymbol{R}}$ the following transformation is valid:

$$\dot{\vec{\boldsymbol{R}}}\big|_G = \dot{\vec{\boldsymbol{R}}}\big|_B + \vec{\boldsymbol{\Omega}}_{B/G} \times \vec{\boldsymbol{R}} \tag{3-6}$$

Applying Equation 3-6 on the relative angular velocity vector shows that the time derivative is invariant with respect to these two systems:

$$\dot{\vec{\boldsymbol{\Omega}}}_{B/G}\big|_G = \dot{\vec{\boldsymbol{\Omega}}}_{B/G}\big|_B + \vec{\boldsymbol{\Omega}}_{B/G} \times \vec{\boldsymbol{\Omega}}_{B/G} = \dot{\vec{\boldsymbol{\Omega}}}_{B/G}\big|_B \tag{3-7}$$

Therefore the coordinate system where differentiation is done can be omitted from the specification the angular acceleration vector $\dot{\vec{\boldsymbol{\Omega}}}_{B/G}$.

Applying Equation 3-6 twice we arrive to the relation for the second time derivative:

$$\ddot{\vec{\boldsymbol{R}}}\big|_G = \ddot{\vec{\boldsymbol{R}}}\big|_B + \dot{\vec{\boldsymbol{\Omega}}}_{B/G} \times \vec{\boldsymbol{R}} + 2\vec{\boldsymbol{\Omega}}_{B/G} \times \dot{\vec{\boldsymbol{R}}}\big|_B + \vec{\boldsymbol{\Omega}}_{B/G} \times (\vec{\boldsymbol{\Omega}}_{B/G} \times \vec{\boldsymbol{R}}) \tag{3-8}$$

Now we will consider a body coordinate system $'c'$ defined relative to the intermediate coordinate system $'B'$. That is the position of the body is defined with the vector $\vec{\boldsymbol{R}}_{c/B}$ and its orientation with a rotation matrix $\boldsymbol{A}_{c/B}$. Hence the position of the body in the coordinate system $'G'$ is defined as

$$\vec{\boldsymbol{R}}_{c/G} = \vec{\boldsymbol{R}}_{B/G} + \vec{\boldsymbol{R}}_{c/B} \tag{3-9}$$

Then Equation 3-6 leads to the following expression for the linear velocity of the body:

$$\dot{\vec{\boldsymbol{R}}}_{c/G}\big|_G = \dot{\vec{\boldsymbol{R}}}_{B/G}\big|_G + \dot{\vec{\boldsymbol{R}}}_{c/B}\big|_B + \vec{\boldsymbol{\Omega}}_{B/G} \times \vec{\boldsymbol{R}}_{c/B} \tag{3-10}$$

Equation 3-8 gives the transformation for the linear acceleration:

$$\begin{aligned} \ddot{\vec{\boldsymbol{R}}}_{c/G}\Big|_G = & \ \ddot{\vec{\boldsymbol{R}}}_{B/G}\Big|_G + \ddot{\vec{\boldsymbol{R}}}_{c/B}\Big|_B + \dot{\vec{\boldsymbol{\Omega}}}_{B/G} \times \vec{\boldsymbol{R}}_{c/B} \\ + & \ 2\vec{\boldsymbol{\Omega}}_{B/G} \times \dot{\vec{\boldsymbol{R}}}_{c/B}\Big|_B + \vec{\boldsymbol{\Omega}}_{B/G} \times (\vec{\boldsymbol{\Omega}}_{B/G} \times \vec{\boldsymbol{R}}_{c/B}) \end{aligned} \tag{3-11}$$

Similarly, for the angular velocity of the body $\vec{\boldsymbol{\omega}}_{c/G}$ and angular acceleration $\vec{\boldsymbol{\alpha}}_{c/G}\Big|_G$:

$$\vec{\boldsymbol{\omega}}_{c/G} = \vec{\boldsymbol{\Omega}}_{B/G} + \vec{\boldsymbol{\omega}}_{c/B} \tag{3-12}$$

$$\vec{\boldsymbol{\alpha}}_{c/G}\Big|_G = \dot{\vec{\boldsymbol{\omega}}}_{c/G}\Big|_G = \dot{\vec{\boldsymbol{\Omega}}}_{B/G} + \vec{\boldsymbol{\alpha}}_{c/B}\Big|_B + \vec{\boldsymbol{\Omega}}_{B/G} \times \vec{\boldsymbol{\omega}}_{c/B} \tag{3-13}$$

Note that the equations above do not specify the coordinate system where the components of the vectors are expressed. Therefore appropriate transformation matrices (e.g., $A_{B/G}, A_{c/B}$) may be necessary to apply to bring all the vectors in the same coordinate system (e.g., 'B').

3.4 Generalized Newton-Euler Equation

The generalized Newton-Euler equation for the unconstrained motion of the deformable body that undergoes large reference displacement is given by:

$$\begin{bmatrix} \boldsymbol{M}_{\mathrm{RR}} & \boldsymbol{A}_{g/b}\tilde{\bar{\boldsymbol{S}}}_t^T & \boldsymbol{A}_{g/b}\bar{\boldsymbol{S}} \\ & \bar{\boldsymbol{J}}_{\mathrm{ee}} & \bar{\boldsymbol{J}}_{\mathrm{ef}} \\ \multicolumn{2}{c}{symmetric} & \boldsymbol{M}_{\mathrm{ff}} \end{bmatrix} \begin{bmatrix} \ddot{\vec{\boldsymbol{R}}}_{\mathrm{b/g}}^{(\mathrm{g})} \\ \vec{\boldsymbol{\alpha}}^{(\mathrm{b})} \\ \ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} \vec{\boldsymbol{Q}}_{\mathrm{R}}^{(g)} \\ \vec{\boldsymbol{Q}}_{\alpha}^{(b)} \\ \vec{\boldsymbol{Q}}_{\mathrm{f}} \end{bmatrix} \tag{3-14}$$

Where

$$\begin{bmatrix} \vec{\boldsymbol{Q}}_{\mathrm{R}}^{(g)} \\ \vec{\boldsymbol{Q}}_{\alpha}^{(b)} \\ \vec{\boldsymbol{Q}}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} (\vec{\boldsymbol{Q}}_e)_{\mathrm{R}}^{(g)} \\ (\vec{\boldsymbol{Q}}_e)_{\alpha}^{(b)} \\ (\vec{\boldsymbol{Q}}_e)_{\mathrm{f}} - \boldsymbol{K}_{\mathrm{ff}} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} - \boldsymbol{C}_{\mathrm{ff}} \cdot \dot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \end{bmatrix} + \begin{bmatrix} (\vec{\boldsymbol{Q}}_v)_{\mathrm{R}}^{(g)} \\ (\vec{\boldsymbol{Q}}_v)_{\alpha}^{(b)} \\ (\vec{\boldsymbol{Q}}_v)_{\mathrm{f}} \end{bmatrix} \tag{3-15}$$

Alternatively using body coordinate system for all vectors:

$$\begin{bmatrix} \boldsymbol{M}_{\mathrm{RR}} & \tilde{\bar{\boldsymbol{S}}}_t^T & \bar{\boldsymbol{S}} \\ & \bar{\boldsymbol{J}}_{\mathrm{ee}} & \bar{\boldsymbol{J}}_{\mathrm{ef}} \\ \multicolumn{2}{c}{symmetric} & \boldsymbol{M}_{\mathrm{ff}} \end{bmatrix} \begin{bmatrix} \ddot{\vec{\boldsymbol{R}}}_{\mathrm{b/g}}^{(\mathrm{b})} \\ \vec{\boldsymbol{\alpha}}^{(\mathrm{b})} \\ \ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} \vec{\boldsymbol{Q}}_{\mathrm{R}}^{(b)} \\ \vec{\boldsymbol{Q}}_{\alpha}^{(b)} \\ \vec{\boldsymbol{Q}}_{\mathrm{f}} \end{bmatrix} \tag{3-16}$$

where

$\ddot{\vec{\boldsymbol{R}}}_{\mathrm{b/g}}\Big|_{\mathrm{g}}$ is the acceleration of the offset of the body reference coordinates relative to the global coordinate system.

$\vec{\boldsymbol{\alpha}}^{(\mathrm{b})}$ is the angular acceleration vector of the reference of the deformable body defined in the body coordinate system.

$\ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}}$ second derivative of the vector of time-dependent *elastic generalized coordinates* of the body $\vec{\boldsymbol{x}}_{\mathrm{f}}$.

$\boldsymbol{A}_{g/b}$ - orthogonal rotation matrix, which for the case of Euler parameters is defined as:

$$\boldsymbol{A}_{g/b} = \begin{bmatrix} 1-2(e_2)^2-2(e_3)^2 & 2(e_1e_2-e_3e_4) & 2(e_1e_3+e_2e_4) \\ 2(e_1e_2+e_3e_4) & 1-2(e_1)^2-2(e_3)^2 & 2(e_2e_3-e_1e_4) \\ 2(e_1e_3-e_2e_4) & 2(e_2e_3+e_1e_4) & 1-2(e_1)^2-2(e_2)^2 \end{bmatrix}, \tag{3-17}$$

where $e_{1..4}$ are the four Euler rotation parameters, defined for the rotation around vector $\hat{v} = [\hat{v}_x, \hat{v}_y, \hat{v}_z]$ by angle ϕ as

$$\left.\begin{array}{l} e_1 = \hat{v}_x \sin\frac{\phi}{2},\;\; e_2 = \hat{v}_y \sin\frac{\phi}{2}, \\ e_3 = \hat{v}_z \sin\frac{\phi}{2},\;\; e_4 = \cos\frac{\phi}{2} \end{array}\right\} \tag{3-18}$$

The other parameters of the equation are the mass matrix, the generalized forces and quadratic velocity vector. The components of the mass matrix are defined in the next section, generalized forces are discussed in Section 3.7, and the quadratic velocity vector that includes the effects of Coriolis and centrifugal forces is defined in Section 3.6.

The Newton-Euler equation given in this section can be further simplified by using the shape functions that satisfy mean-axis condition. See Chapter 8 for further development and an efficient solution approach.

3.5 Mass Matrix

Even though the mass matrix depends on the elastic states, the deformable body inertia can be defined in terms of a set of constant inertia integrals that depend on the assumed displacement field.

$$\vec{\boldsymbol{J}}_1 = \int_V \rho\, \vec{\boldsymbol{p}}_0^{(\mathrm{b})} dV \tag{3-19}$$

$$J_{kl} = \int_V \rho\, p_{0,k}^{(b)}\, p_{0,l}^{(b)} dV,\;\; k,l = 1,2,3 \tag{3-20}$$

$$\bar{\boldsymbol{S}} = \int_V \rho\, \boldsymbol{S} dV \tag{3-21}$$

$$\bar{\boldsymbol{S}}_{kl} = \int_V \rho\, \boldsymbol{S}_k^T \boldsymbol{S}_l dV,\;\; k,l = 1,2,3 \tag{3-22}$$

$$\bar{\boldsymbol{J}}_{kl} = \int_V \rho\, p_{0,k}^{(b)} \boldsymbol{S}_l dV,\ k,l = 1,2,3 \tag{3-23}$$

where

$\vec{\boldsymbol{p}}_0^{(\mathrm{b})}$ is the undeformed position of an arbitrary point on the deformable body;

ρ is the mass density of the body;

V is the volume of the body;

$\boldsymbol{S}_k$ is the k-th row in the body shape matrix $\boldsymbol{S}$, defined by Equation 3-2.

Note that in the special case of rigid body analysis the shape integrals are given by Equations 3-19 and 3-20 only.

The mass matrix components now can be defined:

- $\boldsymbol{M}_{\mathrm{RR}} = \int_V \rho \boldsymbol{I} dV = m\boldsymbol{I}$, where $\boldsymbol{I}$ - is the 3×3 identity matrix for the spatial case, m - mass of the body. For the case of conservation of mass this matrix is the same for both cases of rigid and deformable bodies.

- $\tilde{\vec{\boldsymbol{S}}}_t$ is given by

$$\begin{bmatrix} 0 & -S_{t,3} & S_{t,2} \\ S_{t,3} & 0 & -S_{t,1} \\ -S_{t,2} & S_{t,1} & 0 \end{bmatrix} \tag{3-24}$$

where

$$\vec{\boldsymbol{S}}_t = [S_{t,1}, S_{t,2}, S_{t,3}]^T = \vec{\boldsymbol{J}}_1 + \bar{\boldsymbol{S}} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \tag{3-25}$$

- $\bar{\boldsymbol{J}}_{\mathrm{ee}}$ is a 3×3 symmetric matrix with the elements given by

$$\int_V \rho \begin{bmatrix} (p_{P/b,2}^{(b)})^2 + (p_{P/b,3}^{(b)})^2 & -p_{P/b,2}^{(b)} p_{P/b,1}^{(b)} & -p_{P/b,3}^{(b)} p_{P/b,1}^{(b)} \\ & (p_{P/b,1}^{(b)})^2 + (p_{P/b,3}^{(b)})^2 & -p_{P/b,3}^{(b)} p_{P/b,2}^{(b)} \\ symmetric & & (p_{P/b,1}^{(b)})^2 + (p_{P/b,2}^{(b)})^2 \end{bmatrix} dV \tag{3-26}$$

Let us now show how this matrix can be calculated efficiently by using pre-calculated inertia shape integrals. First we will analyze the components of the diagonal elements:

$$\begin{aligned} \int_V \rho[(p_{P/b,k}^{(b)})^2]dV &= \int_V \rho[(p_{0,k}^{(b)} + \boldsymbol{S}_k \cdot \vec{\boldsymbol{x}}_{\mathrm{f}})^2]dV \\ &= \int_V \rho[(p_{0,k}^{(b)})^2 + \vec{\boldsymbol{x}}_{\mathrm{f}}^T \cdot \boldsymbol{S}_k^T \cdot \boldsymbol{S}_k \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} + 2 \cdot p_{0,k}^{(b)} \cdot \boldsymbol{S}_k \cdot \vec{\boldsymbol{x}}_{\mathrm{f}}]dV \\ &= \int_V \rho(p_{0,k}^{(b)})^2 dV + \vec{\boldsymbol{x}}_{\mathrm{f}}^T \cdot \int_V \rho[\boldsymbol{S}_k^T \cdot \boldsymbol{S}_k]dV \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \\ &\qquad + \ 2\int_V \rho[p_{0,k}^{(b)} \cdot \boldsymbol{S}_k]dV \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \\ &= J_{kk} + 2\bar{\boldsymbol{J}}_{kk} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} + \vec{\boldsymbol{x}}_{\mathrm{f}}^T \cdot \bar{\boldsymbol{S}}_{kk} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \end{aligned} \tag{3-27}$$

Using the procedure similar to the one shown above we can come up with the following compact expressions for the diagonal and non-diagonal members of $\bar{\boldsymbol{J}}_{\text{ee}}$:

$$\begin{aligned}
&\bar{J}_{\text{ee},kk} = \sum_l J_{ll} + 2(\sum_l \bar{\boldsymbol{J}}_{ll}) \cdot \vec{\boldsymbol{x}}_{\text{f}} + \vec{\boldsymbol{x}}_{\text{f}}^T \cdot (\sum_l \bar{\boldsymbol{S}}_{ll}) \cdot \vec{\boldsymbol{x}}_{\text{f}} \\
&\bar{J}_{\text{ee},kl} = -(J_{lk} + (\bar{\boldsymbol{J}}_{lk} + \bar{\boldsymbol{J}}_{kl}) \cdot \vec{\boldsymbol{x}}_{\text{f}} + \vec{\boldsymbol{x}}_{\text{f}}^T \cdot \bar{\boldsymbol{S}}_{lk} \cdot \vec{\boldsymbol{x}}_{\text{f}}) \\
&\text{where } \; k, l = 1, 2, 3; \; l \neq k
\end{aligned} \tag{3-28}$$

Note that for the case of rigid body analysis the elastic coordinates are zero and so the matrix $\bar{\boldsymbol{J}}_{ee}$ is constant.

- $\bar{\boldsymbol{J}}_{\text{ef}}$ has three rows defined as follows

$$\bar{\boldsymbol{J}}_{\text{ef}} = \begin{bmatrix} \vec{\boldsymbol{x}}_{\text{f}}^T \cdot (\bar{\boldsymbol{S}}_{23} - \bar{\boldsymbol{S}}_{32}) \\ \vec{\boldsymbol{x}}_{\text{f}}^T \cdot (\bar{\boldsymbol{S}}_{31} - \bar{\boldsymbol{S}}_{13}) \\ \vec{\boldsymbol{x}}_{\text{f}}^T \cdot (\bar{\boldsymbol{S}}_{12} - \bar{\boldsymbol{S}}_{21}) \end{bmatrix} + \begin{bmatrix} (\bar{\boldsymbol{J}}_{23} - \bar{\boldsymbol{J}}_{32}) \\ (\bar{\boldsymbol{J}}_{31} - \bar{\boldsymbol{J}}_{13}) \\ (\bar{\boldsymbol{J}}_{12} - \bar{\boldsymbol{J}}_{21}) \end{bmatrix} \tag{3-29}$$

- $\boldsymbol{M}_{\text{ff}}$ is independent on the generalized coordinates of the body and, therefore, constant:

$$\boldsymbol{M}_{\text{ff}} = \bar{\boldsymbol{S}}_{11} + \bar{\boldsymbol{S}}_{22} + \bar{\boldsymbol{S}}_{33} \tag{3-30}$$

3.6 Quadratic Velocity Vector

The quadratic velocity vector can be defined as

$$\vec{\boldsymbol{Q}}_v = [(\vec{\boldsymbol{Q}}_v)_{\text{R}}^T \; (\vec{\boldsymbol{Q}}_v)_{\alpha}^T \; (\vec{\boldsymbol{Q}}_v)_{\text{f}}^T]^T \tag{3-31}$$

In the three-dimensional analysis the components of the vector $\vec{\boldsymbol{Q}}_v$ are defined as

$$\left.\begin{aligned}
&(\vec{\boldsymbol{Q}}_v)_{\text{R}}^{(g)} = -\boldsymbol{A}_{g/b} \cdot [(\tilde{\boldsymbol{\omega}})^2 \bar{\boldsymbol{S}}_t + 2\tilde{\boldsymbol{\omega}} \bar{\boldsymbol{S}} \dot{\vec{\boldsymbol{x}}}_{\text{f}}] \\
&(\vec{\boldsymbol{Q}}_v)_{\alpha} = -\vec{\boldsymbol{\omega}}^{\,(\text{b})} \times (\bar{\boldsymbol{J}}_{\text{ee}} \cdot \vec{\boldsymbol{\omega}}^{\,(\text{b})}) - \dot{\bar{\boldsymbol{J}}}_{\text{ee}} \cdot \vec{\boldsymbol{\omega}}^{\,(\text{b})} - \vec{\boldsymbol{\omega}}^{\,(\text{b})} \times (\bar{\boldsymbol{J}}_{\text{ef}} \cdot \dot{\vec{\boldsymbol{x}}}_{\text{f}}) \\
&(\vec{\boldsymbol{Q}}_v)_f = -\int_V \rho \{ \boldsymbol{S}^T [(\tilde{\boldsymbol{\omega}})^2 \cdot \vec{\boldsymbol{p}}^{(\text{b})} + 2 \cdot \tilde{\boldsymbol{\omega}} \cdot \dot{\vec{\boldsymbol{p}}}_{\text{f}}^{(\text{b})}] \} dV
\end{aligned}\right\} \tag{3-32}$$

where $\vec{\boldsymbol{\omega}}^{\,(\text{b})}$ is the angular velocity vector defined in the body coordinate system. and $\tilde{\boldsymbol{\omega}}$ is a skew symmetric matrix given by

$$\tilde{\boldsymbol{\omega}} = \tilde{\vec{\boldsymbol{\omega}}}^{\,(\text{b})} = \begin{bmatrix} 0 & -\omega_3^{(\text{b})} & \omega_2^{(\text{b})} \\ \omega_3^{(\text{b})} & 0 & -\omega_1^{(\text{b})} \\ -\omega_2^{(\text{b})} & \omega_1^{(\text{b})} & 0 \end{bmatrix} \tag{3-33}$$

The quadratic velocity vector includes the effect of Coriolis and centrifugal forces as nonlinear functions of the system generalized coordinates and velocities.

In order to express $(\vec{\boldsymbol{Q}}_v)_{\mathrm{f}}$ in terms of the shape integrals we need to evaluate the symmetric $(\tilde{\boldsymbol{\omega}})^2$ matrix:

$$(\tilde{\boldsymbol{\omega}})^2 = \begin{bmatrix} -((\omega_3^{(\mathrm{b})})^2 + (\omega_2^{(\mathrm{b})})^2) & \omega_1^{(\mathrm{b})}\omega_2^{(\mathrm{b})} & \omega_1^{(\mathrm{b})}\omega_3^{(\mathrm{b})} \\ \omega_1^{(\mathrm{b})}\omega_2^{(\mathrm{b})} & -((\omega_1^{(\mathrm{b})})^2 + (\omega_3^{(\mathrm{b})})^2) & \omega_3^{(\mathrm{b})}\omega_2^{(\mathrm{b})} \\ \omega_1^{(\mathrm{b})}\omega_3^{(\mathrm{b})} & \omega_3^{(\mathrm{b})}\omega_2^{(\mathrm{b})} & -((\omega_1^{(\mathrm{b})})^2 + (\omega_2^{(\mathrm{b})})^2) \end{bmatrix} \tag{3-34}$$

Hence, $(\vec{\boldsymbol{Q}}_v)_{\mathrm{f}}$ can be given explicitly in terms of the flexible states $\vec{\boldsymbol{x}}_{\mathrm{f}}$ and $\dot{\vec{\boldsymbol{x}}}_{\mathrm{f}}$:

$$(\vec{\boldsymbol{Q}}_v)_f = -\sum_{i=1}^{3}\sum_{j=1}^{3}(\tilde{\boldsymbol{\omega}})^2_{ij}(\bar{\boldsymbol{J}}_{ij}^T + \bar{\boldsymbol{S}}_{ij} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}}) - 2 \cdot \sum_{i=1}^{3}\sum_{j=1}^{3}\tilde{\boldsymbol{\omega}}_{ij}\bar{S}_{ij}\dot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \tag{3-35}$$

The last term $\sum_{i=1}^{3}\sum_{j=1}^{3}\tilde{\boldsymbol{\omega}}_{ij}\bar{S}_{ij}\dot{\vec{\boldsymbol{x}}}_{\mathrm{f}}$ can be further simplified to

$$\omega_3^{(\mathrm{b})}(\bar{\boldsymbol{S}}_{12} - \bar{\boldsymbol{S}}_{21}) \cdot \dot{\vec{\boldsymbol{x}}}_{\mathrm{f}} + \omega_1^{(\mathrm{b})}(\bar{\boldsymbol{S}}_{23} - \bar{\boldsymbol{S}}_{32}) \cdot \dot{\vec{\boldsymbol{x}}}_{\mathrm{f}} + \omega_2^{(\mathrm{b})}(\bar{\boldsymbol{S}}_{31} - \bar{\boldsymbol{S}}_{13}) \cdot \dot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \tag{3-36}$$

3.7 Generalized Forces

3.7.1 Generalized Elastic Forces

The virtual work due to elastic forces can be written as

$$\delta W_s = -\int_V \vec{\boldsymbol{\sigma}}^T \delta\vec{\boldsymbol{\epsilon}}\, dV \tag{3-37}$$

where $\vec{\boldsymbol{\sigma}}$ and $\vec{\boldsymbol{\epsilon}}$ are the stress and strain tensors, and δW_s is the virtual work of the elastic forces.

Strain is a dimensionless parameter describing deformation. Strain displacement relations can be formulated as:

$$\vec{\boldsymbol{\epsilon}} = \boldsymbol{D}\vec{\boldsymbol{p}}_{\mathrm{f}}^{(\mathrm{b})} \tag{3-38}$$

where $\boldsymbol{D}$ is a differential operator defined according to:

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i} + \sum_{k=1}^{3} u_{k,i}u_{k,j}),\ u_{i,j} = \frac{\partial u_i}{\partial x_j},\ i,j = 1,2,3 \tag{3-39}$$

or, in case of cylindrical coordinate system with coordinates (r, ϕ, z) (see [35]):

$$\begin{aligned}
\epsilon_{11} &= \frac{\partial u_r}{\partial r} \\
\epsilon_{22} &= \frac{1}{r}(\frac{\partial u_\phi}{\partial \phi} + u_r) \\
\epsilon_{33} &= \frac{\partial u_z}{\partial z} \\
\epsilon_{12} &= \frac{1}{r}(\frac{\partial u_r}{\partial \phi} - u_\phi) + \frac{\partial u_\phi}{\partial r} \\
\epsilon_{13} &= \frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r} \\
\epsilon_{23} &= \frac{1}{r}\frac{\partial u_z}{\partial \phi} + \frac{\partial u_\phi}{\partial z}
\end{aligned} \tag{3-40}$$

For a linear homogeneous isotropic material, the constitutive equations relating the stress and strains can be written as

$$\vec{\boldsymbol{\sigma}} = \boldsymbol{E}\vec{\boldsymbol{\epsilon}} \tag{3-41}$$

where $\vec{\boldsymbol{\epsilon}} = [\epsilon_{11}, \epsilon_{22}, \epsilon_{33}, \epsilon_{12}, \epsilon_{13}, \epsilon_{23}]^T$, $\vec{\boldsymbol{\sigma}} = [\sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}, \sigma_{13}, \sigma_{23}]^T$, and $\boldsymbol{E}$ is the symmetric matrix of elastic coefficients, defining the material properties.

Substituting Equation 3-41 into Equation 3-37 we can get:

$$\delta W_s = -\vec{\boldsymbol{x}}_{\mathrm{f}}^{\mathrm{T}} \cdot \boldsymbol{K}_{\mathrm{ff}} \cdot \delta\vec{\boldsymbol{x}}_{\mathrm{f}} \tag{3-42}$$

where

$$\boldsymbol{K}_{\mathrm{ff}} = \int_V (\boldsymbol{DS})^T \boldsymbol{EDS} dV \tag{3-43}$$

$\boldsymbol{K}_{\mathrm{ff}}$ is a symmetric positive semidefinite stiffness matrix associated with the elastic states of the body. Boundary conditions, discussed in Chapter 8, ensure that the stiffness matrix becomes positive definite.

Since the virtual work of the elastic forces does not depend on the rigid body states the corresponding components of the generalized force vector are zero. The resulting vector is therefore given by:

$$\begin{bmatrix} \vec{\boldsymbol{Q}}_{\mathrm{R}}^{(g)} \\ \vec{\boldsymbol{Q}}_{\alpha}^{(b)} \\ \vec{\boldsymbol{Q}}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\boldsymbol{K}_{\mathrm{ff}} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \end{bmatrix} \tag{3-44}$$

To complete the description we show how the matrix $\boldsymbol{E}$ can be expressed in terms of Lame's constants:

$$\boldsymbol{E} = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu \end{bmatrix} \tag{3-45}$$

The Lame's constants are related to the engineering constants E - Young's modulus and γ - Poisson's ration with the following equations:

$$\begin{cases} E = \dfrac{\mu(3\lambda + 2\mu)}{\lambda + \mu} \\ \gamma = \dfrac{\lambda}{2(\lambda + \mu)} \end{cases} \qquad \begin{cases} \mu = \dfrac{E}{2(\gamma + 1)} \\ \lambda = \dfrac{E\gamma}{(1 - 2\gamma)(1 + \gamma)} \end{cases} \tag{3-46}$$

It is important to mention that the stresses and strains we were describing above are the second Piola-Kirchhoff stress and Green-Lagrange strain tensors. The most important observation about these two measures is that the components of the tensors do not change under rigid body translation and rotation and so they provide a natural material description for the case of large rigid body motion but small elastic deflection and strain analysis. The observation is of practical importance since Hook's law is applicable only to small strains and because in the particular problems we are interested in the condition of large rigid body motion, accompanied by small strains, holds.

3.7.2 Generalized Viscosity Forces

We start the description by mentioning a big similarities between the elastic and viscosity forces and the related laws. For instance, for a linear homogeneous isotropic material, the constitutive equations relating the rate of strain change to the viscosity stress can be written as

$$\vec{\boldsymbol{\tau}} = \boldsymbol{E_v}\dot{\vec{\boldsymbol{\epsilon}}} \tag{3-47}$$

And so following the procedure outlined in the previous subsection we can compute the damping matrix $\boldsymbol{C}_{\text{ff}}$ similarly to the stiffness matrix $\boldsymbol{K}_{\text{ff}}$. However, the material constants' matrix $\boldsymbol{E_v}$ is often not so well known for damping and so we will describe one of the approximation techniques - Rayleigh damping. This damping model is discussed in more details for the case of finite-element analysis in [3]. The damping matrix in this case is calculated as:

$$\boldsymbol{C}_{\text{ff}} = \alpha \cdot \boldsymbol{M}_{\text{ff}} + \beta \cdot \boldsymbol{K}_{\text{ff}} \tag{3-48}$$

where α (1/sec) and β (sec) are constants to be determined from two given damping ratios that correspond to two unequal frequencies of vibration. Assuming that for two vibration frequencies ω_1 and ω_2 we know the damping ratios ξ_1 and ξ_2 respectively (in fractions of critical damping), the Rayleigh coefficients can be found from the equations:

$$\begin{cases} \alpha + \omega_1^2 \cdot \beta = 2\,\omega_1 \cdot \xi_1 \\ \alpha + \omega_2^2 \cdot \beta = 2\,\omega_2 \cdot \xi_2 \end{cases} \tag{3-49}$$

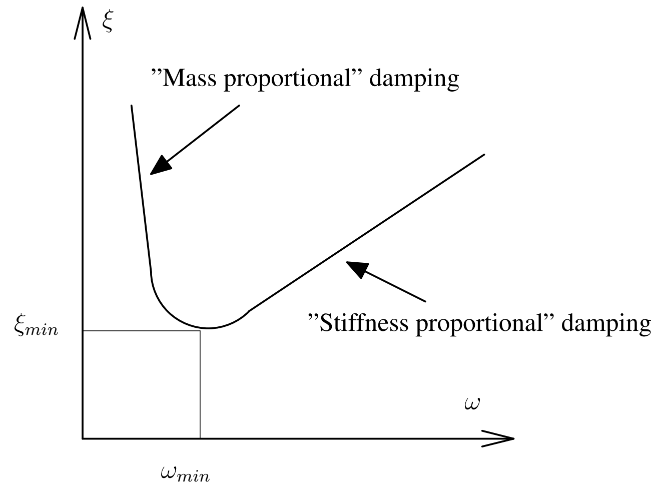


Figure 3-1: *Damping coefficient as a function of frequency.*

Similar to the previous section the resulting generalized force vector can be formulated as:

$$\begin{bmatrix} \vec{\boldsymbol{Q}}_{\mathrm{R}}^{(g)} \\ \vec{\boldsymbol{Q}}_{\alpha}^{(b)} \\ \vec{\boldsymbol{Q}}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\boldsymbol{C}_{\mathrm{ff}} \cdot \dot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \end{bmatrix} \tag{3-50}$$

Note that given the α and β coefficients one can establish the damping ratio that is specified at a value of ω_i:

$$\xi_i = \frac{\alpha + \beta \cdot \omega_i^2}{2\,\omega_i} \tag{3-51}$$

The general view of this curve is shown in Figure 3-1.

Simple analysis of this curve tells us that damping coefficient has its minimum value $\xi_{min} = \sqrt{\alpha \cdot \beta}$ at the frequency $\omega_{min} = \sqrt{\alpha/\beta}$ and that the coefficient grows linearly for higher and hyperbolically for lower frequencies. In fact, one of the features of Rayleigh damping is that the higher modes are considerably more damped than the lower modes, for which Rayleigh constants have been selected.

The solution of the linear system Equation 3-49 gives the following relations for the α and β coefficients:

$$\begin{cases} \alpha = \dfrac{2}{\omega_2^2 - \omega_1^2}\,\omega_1 \cdot \omega_2\,(\xi_1 \cdot \omega_2 - \xi_2 \cdot \omega_1) \\ \beta = \dfrac{2}{\omega_2^2 - \omega_1^2}\,(\xi_2 \cdot \omega_2 - \xi_1 \cdot \omega_1) \end{cases} \tag{3-52}$$

When choosing the damping ratios and frequencies it is important to make sure that the resulting α and β coefficients are positive. The negative value of one of the coefficients leads to the negative damping ratio for a range of flexible body frequencies. Thus the system will become non-dissipative and generate vibrations during the simulation.

Since all the parameters used (ξ_1, ω_1, ξ_2 and ω_2) are positive the following conditions follow from Equation 3-52 if we assume $\omega_2 > \omega_1$:

$$\begin{cases} \xi_1 \cdot \omega_2 - \xi_2 \cdot \omega_1 > 0 \\ \xi_2 \cdot \omega_2 - \xi_1 \cdot \omega_1 > 0 \end{cases} \tag{3-53}$$

or, alternatively:

$$\frac{\omega_1}{\omega_2} < \frac{\xi_1}{\xi_2} < \frac{\omega_2}{\omega_1} \tag{3-54}$$

In practice, the values of the coefficients are often selected based on the experience with simulation of similar structures when the results of a simulation were fitted to the same experimental results. That is the same α and β are used in the analysis of similar structures. The magnitude of the Rayleigh coefficients is to large extent determined by the energy dissipation characteristics of the construction, including the material.

Another use of Rayleigh damping might be to improve the numerical performance of the simulation by damping high frequency vibrations outside the frequency band that the user is interested in. For example, suppose that only one damping ratio ξ_e for the first body eigen-frequency ω_e is known. Then, it might be reasonable to select some high frequency ω_h that limits the frequency band and set the corresponding damping ration ξ_h to one thus damping out all the high frequencies. Having the two damping ratios ξ_e and ξ_h for two distinct frequencies the Equation 3-52 can be used to obtain the α and β coefficients.

3.7.3 Generalized External Forces

The virtual work of all external forces acting on a body can be written in compact form as

$$\delta W_e = \vec{\boldsymbol{Q}}_e^T \delta\vec{\boldsymbol{x}} \tag{3-55}$$

where $\vec{\boldsymbol{Q}}_e$ is the vector of generalized external forces associated with the body generalized coordinates. In partitioned form, the virtual work can be written as

$$\delta W_e = [(\vec{\boldsymbol{Q}}_{\mathrm{e}}^{(\mathrm{g})})_{\mathrm{R}}^T \; (\vec{\boldsymbol{Q}}_e)_{\mathrm{e}}^T \; (\vec{\boldsymbol{Q}}_e)_{\mathrm{f}}^T] \begin{bmatrix} \delta\vec{\boldsymbol{R}}^{(\mathrm{g})} \\ \delta\vec{\boldsymbol{e}} \\ \delta\vec{\boldsymbol{x}}_{\mathrm{f}} \end{bmatrix} \tag{3-56}$$

where $(\vec{\boldsymbol{Q}}_{\mathrm{e}}^{(\mathrm{g})})_{\mathrm{R}}$ and $(\vec{\boldsymbol{Q}}_e)_{\mathrm{e}}$ are the generalized forces associated, respectively, with the translational and rotational coordinates of the selected body reference, and $(\vec{\boldsymbol{Q}}_e)_{\mathrm{f}}$ is the vector of generalized forces associated with the elastic generalized coordinates of the body. Generalized forces may depend on the system's generalized coordinates $\vec{\boldsymbol{x}}$, velocities, and on time.

Later in this section in order to provide simpler and more compact equations the small angles $\vec{\boldsymbol{\theta}}$ and actual moments vector $(\vec{\boldsymbol{Q}}_e)_\alpha$ are used instead of generalized rotational coordinates (Euler parameters) $\vec{\boldsymbol{e}}$ and associated part of the generalized force $(\vec{\boldsymbol{Q}}_e)^T_{\mathrm{e}}$. That is we will use:

$$\delta W_e = [(\vec{\boldsymbol{Q}}_{\mathrm{e}}^{(\mathrm{g})})^T_{\mathrm{R}} \; (\vec{\boldsymbol{Q}}_e)^T_\alpha \; (\vec{\boldsymbol{Q}}_e)^T_{\mathrm{f}}] \begin{bmatrix} \delta \vec{\boldsymbol{R}}^{(\mathrm{g})} \\ \delta \vec{\boldsymbol{\theta}} \\ \delta \vec{\boldsymbol{x}}_{\mathrm{f}} \end{bmatrix} \tag{3-57}$$

3.7.4 External Point Force

Let us assume that a force $\vec{\boldsymbol{F}}^{(\mathrm{g})} = \vec{\boldsymbol{F}}^{(\mathrm{g})}(\vec{\boldsymbol{x}}, t)$ acts at point $\vec{\boldsymbol{p}}^{(\mathrm{g})}$ of the deformable body. The virtual work of this force is defined as

$$\delta W_e = (\vec{\boldsymbol{F}}^{(\mathrm{g})})^T \delta \vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}} \tag{3-58}$$

where $\vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}}$ is the global position of point P of the deformable body. Let the position of the point be defined by the vector $\vec{\boldsymbol{p}}^{(\mathrm{g})}$.

$$\vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}} = \vec{\boldsymbol{R}}^{(\mathrm{g})}_{\mathrm{b/g}} + \boldsymbol{A}_{g/b} \cdot \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{P/b}} \tag{3-59}$$

where $\vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{P/b}}$ is the local position of point P with respect to the body coordinate system. The virtual change $\delta \vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}}$ is then defined as

$$\begin{aligned} \delta \vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}} = \delta \vec{\boldsymbol{R}}^{(\mathrm{g})}_{\mathrm{b/g}} + \frac{\partial}{\partial \vec{\boldsymbol{\theta}}}[\boldsymbol{A}_{g/b} \cdot \vec{\boldsymbol{p}}^{(\mathrm{b})}_{\mathrm{P/b}}] \cdot \delta \vec{\boldsymbol{\theta}} + \boldsymbol{A}_{g/b} \cdot \boldsymbol{S} \cdot \delta \vec{\boldsymbol{x}}_{\mathrm{f}} = \\ \delta \vec{\boldsymbol{R}}^{(\mathrm{g})} - \boldsymbol{A}_{g/b} \cdot \tilde{\vec{\boldsymbol{p}}}^{(\mathrm{b})}_{\mathrm{P/b}} \cdot \delta \vec{\boldsymbol{\theta}} + \boldsymbol{A}_{g/b} \cdot \boldsymbol{S} \cdot \delta \vec{\boldsymbol{x}}_{\mathrm{f}} \end{aligned} \tag{3-60}$$

The vector $\delta \vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}}$ can be written in a partitioned form as

$$\delta \vec{\boldsymbol{p}}^{(\mathrm{g})}_{\mathrm{P/g}} = [\boldsymbol{I}, \; -\boldsymbol{A}_{g/b} \cdot \tilde{\vec{\boldsymbol{p}}}^{(\mathrm{b})}_{\mathrm{P/b}}, \; \boldsymbol{A}_{g/b} \cdot \boldsymbol{S}] \begin{bmatrix} \delta \vec{\boldsymbol{R}}^{(\mathrm{g})} \\ \delta \vec{\boldsymbol{\theta}} \\ \delta \vec{\boldsymbol{x}}_{\mathrm{f}} \end{bmatrix} \tag{3-61}$$

where the shape matrix $\boldsymbol{S}$ is defined at the point P. Thus, the virtual work δW_e is defined as

$$\delta W_e = (\vec{\boldsymbol{F}}^{(\mathrm{g})})^T [\boldsymbol{I}, \; -\boldsymbol{A}_{g/b} \cdot \tilde{\vec{\boldsymbol{p}}}^{(\mathrm{b})}_{\mathrm{P/b}}, \; (\boldsymbol{A}_{g/b} \cdot \boldsymbol{S})] \begin{bmatrix} \delta \vec{\boldsymbol{R}}^{(\mathrm{g})} \\ \delta \vec{\boldsymbol{e}} \\ \delta \vec{\boldsymbol{x}}_{\mathrm{f}} \end{bmatrix} \tag{3-62}$$

and so the generalized forces in Equation 3-57 can be recognized as

$$(\vec{\boldsymbol{Q}}_{\mathrm{e}}^{(\mathrm{g})})_R = \vec{\boldsymbol{F}}^{(\mathrm{g})}, \;\; (\vec{\boldsymbol{Q}}_e)_\alpha = \tilde{\vec{\boldsymbol{p}}}^{(\mathrm{b})}_{\mathrm{P/b}} \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{F}}^{(\mathrm{g})}, \;\; (\vec{\boldsymbol{Q}}_e)_f = \boldsymbol{S}^T \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{F}}^{(\mathrm{g})} \tag{3-63}$$

where the facts that $\tilde{\vec{\boldsymbol{p}}} = -\tilde{\vec{\boldsymbol{p}}}^T$ and $\boldsymbol{A}^T_{g/b} = \boldsymbol{A}_{b/g}$ were used to simplify the equations.

3.7.5 External Point Moment

Let us assume that an external moment $\vec{\boldsymbol{M}} = \vec{\boldsymbol{M}}^{(\mathrm{g})}(\vec{\boldsymbol{x}}, t)$ is applied at point P of the deformable body. Let the position of the point be defined by the vector $\vec{\boldsymbol{p}}^{(\mathrm{g})}$. The virtual work of this moment is defined as

$$\delta W_e = (\vec{\boldsymbol{M}}^{(\mathrm{g})})^T \delta\vec{\boldsymbol{\theta}}^{(\mathrm{g})} \tag{3-64}$$

where $\delta\vec{\boldsymbol{\theta}}^{(\mathrm{g})}$ is a virtual change of the the global orientation of the point P. Since vector rules apply to the infinitesimal rotations one can write:

$$\delta\vec{\boldsymbol{\theta}}^{(\mathrm{g})} = \delta\vec{\boldsymbol{\theta}}_{\mathrm{r}}^{(\mathrm{g})} + \boldsymbol{A}_{g/b} \cdot \delta\vec{\boldsymbol{\theta}}_{\mathrm{f}}^{(\mathrm{b})} \tag{3-65}$$

where $\delta\vec{\boldsymbol{\theta}}_{\mathrm{r}}^{(\mathrm{g})}$ is the virtual change of the reference frame orientation and $\delta\vec{\boldsymbol{\theta}}_{\mathrm{f}}^{(\mathrm{b})}$ is the virtual change of the material particle orientation inside the flexible body as defined in Section 3.2.

In terms of the generalized coordinates the virtual orientation change is then defined as:

$$\delta\vec{\boldsymbol{\theta}}^{(\mathrm{g})} = [0, \boldsymbol{A}_{g/b}, \boldsymbol{A}_{g/b}\frac{\partial\vec{\boldsymbol{\theta}}_{\mathrm{f}}^{(\mathrm{b})}}{\partial\vec{\boldsymbol{x}}_{\mathrm{f}}}] \begin{bmatrix} \delta\vec{\boldsymbol{R}}^{(\mathrm{g})} \\ \delta\vec{\boldsymbol{\theta}} \\ \delta\vec{\boldsymbol{x}}_{\mathrm{f}} \end{bmatrix} \tag{3-66}$$

where $\frac{\partial\vec{\boldsymbol{\theta}}_{\mathrm{f}}^{(\mathrm{b})}}{\partial\vec{\boldsymbol{x}}_{\mathrm{f}}}$ is a $3 \times n_f$ matrix calculated by differentiation of Equation 3-5 with respect to the elastic states:

$$\frac{\partial\vec{\boldsymbol{\theta}}_{\mathrm{f}}^{(\mathrm{b})}}{\partial\vec{\boldsymbol{x}}_{\mathrm{f}}} = \frac{1}{2}\begin{bmatrix} \frac{\partial\boldsymbol{S}_y}{\partial z} - \frac{\partial\boldsymbol{S}_z}{\partial y} \\ \frac{\partial\boldsymbol{S}_z}{\partial x} - \frac{\partial\boldsymbol{S}_x}{\partial z} \\ \frac{\partial\boldsymbol{S}_x}{\partial y} - \frac{\partial\boldsymbol{S}_y}{\partial x} \end{bmatrix} \tag{3-67}$$

$\boldsymbol{S} = [\boldsymbol{S}_x^T, \boldsymbol{S}_y^T, \boldsymbol{S}_z^T]^T$ is the shape matrix defined at point P.

Finally the resulting generalized force can be recognized as:

$$(\vec{\boldsymbol{Q}}_{\mathrm{e}}^{(\mathrm{g})})_{\mathrm{R}} = 0, \;\; (\vec{\boldsymbol{Q}}_e)_\alpha = \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{M}}^{(\mathrm{g})}, \;\; (\vec{\boldsymbol{Q}}_e)_{\mathrm{f}} = (\frac{\partial\vec{\boldsymbol{\theta}}_{\mathrm{f}}^{(\mathrm{b})}}{\partial\vec{\boldsymbol{x}}_{\mathrm{f}}})^T \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{M}}^{(\mathrm{g})} \tag{3-68}$$

where the fact that $\boldsymbol{A}_{g/b}^T = \boldsymbol{A}_{b/g}$ was used to simplify the equations.

3.7.6 External Volume Load

In this subsection we will analyze the external loads acting on all of the particles of the body. The most common cases of such loads are gravity, magnetic and electrostatic forces. For generality we will assume that some load vector $\vec{\boldsymbol{L}}$ with the dimension N/m^3 (Newton per cubic meter) is acting on each volume particle dV. For the case of gravity this load is given as $\rho \cdot \vec{\boldsymbol{g}}$, where $\vec{\boldsymbol{g}}$ is the gravity acceleration constant vector and ρ is the mass density of the body. Using the equations developed in Section 3.7.4 and integrating over the volume we can get the components of the generalized body load $\vec{\boldsymbol{Q}}_{\boldsymbol{L}}$:

$$(\vec{\boldsymbol{Q}}_{\boldsymbol{L}})_{\mathrm{R}} = \int_V \vec{\boldsymbol{L}}^{(g)} dV = \vec{\boldsymbol{L}}^{(g)} \int_V dV = \vec{\boldsymbol{L}}^{(g)} \cdot V \tag{3-69}$$

$$(\vec{\boldsymbol{Q}}_{\boldsymbol{L}})_{\alpha} = \int_V \left[\tilde{\vec{\boldsymbol{p}}}^{(b)} \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{L}}^{(g)}\right] dV = \int_V \left[\tilde{\vec{\boldsymbol{p}}}^{(b)}\right] dV \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{L}}^{(g)} = \frac{1}{\rho}\, \tilde{\vec{\boldsymbol{S}}}_t \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{L}}^{(g)} \tag{3-70}$$

$$(\vec{\boldsymbol{Q}}_{\boldsymbol{L}})_{\mathrm{f}} = \int_V \left[\boldsymbol{S}^T \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{L}}^{(g)}\right] dV = \int_V \left[\boldsymbol{S}^T\right] dV \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{L}}^{(g)} = \frac{1}{\rho}\, \bar{\boldsymbol{S}} \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{L}}^{(g)} \tag{3-71}$$

where $\tilde{\vec{\boldsymbol{S}}}_t$ and $\bar{\boldsymbol{S}}$ were defined in Section 3.5. For the case of the gravity force the equations can be further simplified considering that $m = \rho \cdot V$:

$$\begin{aligned} (\vec{\boldsymbol{Q}}_{\boldsymbol{g}})_{\mathrm{R}} &= m \cdot \vec{\boldsymbol{g}}^{(g)} \\ (\vec{\boldsymbol{Q}}_{\boldsymbol{g}})_{\alpha} &= \tilde{\vec{\boldsymbol{S}}}_t \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{g}}^{(g)} \\ (\vec{\boldsymbol{Q}}_{\boldsymbol{g}})_{\mathrm{f}} &= \bar{\boldsymbol{S}} \cdot \boldsymbol{A}_{b/g} \cdot \vec{\boldsymbol{g}}^{(g)} \end{aligned} \tag{3-72}$$

Note that the derivation as presented in this section easily can be generalized for other kinds of loads that do not depend on the elastic states. That is as long as the load (force) vector does not depend on the point position within the body one can sum (or integrate) over the points once at the start up and use the precalculated shape integrals during the simulation. One useful application of this approach is area load, where a constant load acts over a surface area.

From the user interface design point of view, the body and area loads rise an issue of sub-volume and sub-area specification in the input. That is, a simulation tool needs to provide some means to define the geometry to be integrated. Both for the area and volume specification two approaches for the user interface design are possible.

Closed boundary A sub-volume can be defined by a set of surfaces and a sub-area - by a set of curves on the surfaces. User must explicitly define all the boundary surfaces and curves. The boundary must be closed to make integration possible.

Parametric specification The sub-volume and sub-area can be specified by coordinate or parametric intervals. This approach is essentially a special case of the closed boundary specification where the boundaries have simple configuration (planes and strait lines). For some cases, e.g., when the specified sub-volume is completely inside the body, the parametric specification simplifies the integration procedures since the integration ranges become constant.

3.7.7 Generalized Forces from the Internal Stress Release during Grinding

The structural elasticity model presented in this thesis was incorporated into a grinding simulation tool [21]. One of the special effects that is of interest for this simulation is the internal stress release during grinding of rings. The internal residual stresses in the material can be caused by material phase changes in the hardening process. When stressed material is removed the released stresses result in elastic distortion of the body. These distortions are especially significant for large rings where they often cause manufacturing problems and therefore should be incorporated into the flexible model [6].

The internal stresses do not affect the elastic behavior of the body as long as the material is intact. When the material is removed they result in additional generalized force that affects the generalized elastic coordinates but does not influence the rigid body part of the system. Following the procedure described in Section 3.7.1 we can get:

$$\vec{\boldsymbol{Q}}_{i_f} = \int_{V_g} \vec{\boldsymbol{\sigma}}^T \boldsymbol{D}\boldsymbol{S} dV_g, \tag{3-73}$$

where the volume integration is done over the material volume V_g that is ground away during the simulation and $\vec{\boldsymbol{\sigma}}$ stands for the residual stress tensor.

For simulation purposes it is often possible to assume that all the surface particles are subjected to the same treatment and only a thin layer of material is removed during grinding. That is why it is realistic to model the internal stress tensor $\vec{\boldsymbol{\sigma}}$ as a constant tensor in the surface layer when the local coordinate system (s) normal to the surface is used to calculate the strains and stresses. Assuming that a rotation matrix $\boldsymbol{A}_{s/b}$ transforms the body coordinates into the local surface coordinates one can perform integration of the shape matrix independently:

$$\vec{\boldsymbol{Q}}_{i_f}^{(c)} = (\vec{\boldsymbol{\sigma}}^{(s)})^T \int_{V_g} \boldsymbol{D}^{(s)}(\boldsymbol{A}_{s/b}\boldsymbol{S}) dV_g \tag{3-74}$$

Note that for the case of ring grinding where a cylindrical coordinate system (r, ϕ, z) is used as the body coordinate system the matrix $\boldsymbol{A}_{s/b}$ does not depend on the angle ϕ.

The problem with the evaluation of the generalized force $\vec{\boldsymbol{Q}}_{i_f}$ comes from the fact that the volume V_g representing the ground material changes with time. In a simulation the ground surface is described by a mesh. Every node of this mesh keeps track of the value of the height $h_g(u, v)$ ground at the point (u, v) on the surface. The integration in Equation 3-74 should obviously be performed over this surface mesh. During a grinding simulation the values $h_g(u, v)$ are updated only once for every ODE solver time step and are kept constant when the solver does trial time steps during internal solver iterations. It means that the generalized force $\vec{\boldsymbol{Q}}_{i_f}$ can also be evaluated only once per time step and can not be accurately evaluated between the steps. The values $h_g(u, v)$ are accumulated during the simulation run and can not be directly computed from the state variables. Therefore the extrapolation procedures inside the solver can not predict the change of the resulting force. Hence, from the ODE solver point of view the force changes discontinuously. For high-order solvers with adaptive time step size, like CVODE [8], force discontinuity results in convergence problems and shorter time steps, leading to much longer simulation times.

In order to avoid this discontinuity problem an interpolation procedure was developed. The procedure is actually general and can be used for smoothing of any kind of force input that causes numerical problems. The procedure is discussed in Section 3.7.8

3.7.8 An Interpolation Method for Forces with Discontinuities

Some non-linear effects modeled in dynamic simulations make it impossible, or very hard, to evaluate the force function at every time instant. See Section 3.7.7 for an example of such a force. In such cases some extrapolation or interpolation procedure must be used to provide a continuous force function in time domain. High order ODE solvers used for dynamic simulations assume high continuity of all the functions involved in the problem. Some experiments done in the BEAST tool show that at least continuity of the second order time derivatives is needed in order to get reasonably large time steps in the solution.

The interpolation procedure described here can be used for the forces which are changing with lower frequencies than some other fast processes in the simulated system. The main idea of this procedure is to delay the real force value for a small time interval and to use cubic interpolation to get a smooth change of the force function between the current and the delayed values.

Before using the procedure the interpolation time delay parameter δ_t must be selected. The only requirement on this parameter is to be larger than the maximum possible time step of the ODE solver. Initial value of the force variable to be interpolated must be given. The initial first and second time derivatives of the variable are normally assumed to be zero if no other information is available.

The interpolation procedure can be seen as a filter block shown in Figure 3-2. The only input for the block is the calculated force $F^c(t_i)$. This value is delayed and assigned to the variable F_i^f which is seen as the expected force value for time $t + \delta_t$. So, at every time t, such that $t_i \leq t < (t_i + \delta_t)$ the interpolation block is using the following four variables:

- Force variable current approximating value $F_i = F(t_i)$;
- First and second time derivatives $\dot{F}_i = \dot{F}(t_i)$ and $\ddot{F}_i = \ddot{F}(t_i)$;
- Delayed value of force F_i^f.

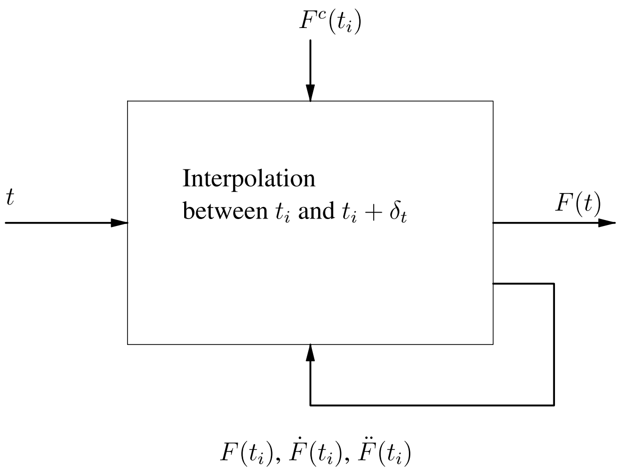

$F^c(t_i)$
t
Interpolation between t_i and $t_i + \delta_t$
$F(t)$
$F(t_i)$, $\dot{F}(t_i)$, $\ddot{F}(t_i)$

Figure 3-2: *Force interpolation as a filter block. A continuous force function $F(t)$ is generated for the discreet input $F^c(t_i)$.*

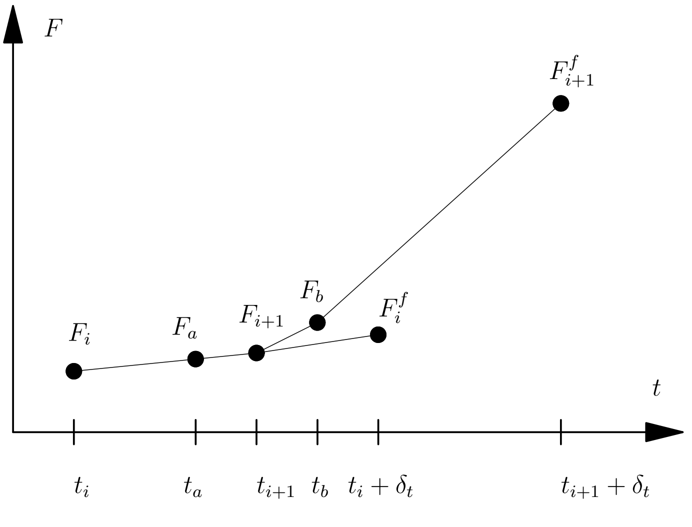


Figure 3-3: *One dimensional force interpolation. The values of the force variable F for the time instances t_a and t_b are calculated using third order interpolation between F_i and F_i^f for F_a and between F_{i+1} and F_{i+1}^f for F_a.*

Having the four parameters above (two values and two derivatives) it is possible to calculate the approximate value $F(t)$ by using cubic interpolation:

$$\begin{aligned} F(t) = \ & F_i(1 - \frac{1}{\delta_t^3}(t-t_i)^3) + F_i^f \frac{1}{\delta_t^3}(t-t_i)^3 \\ & + \dot{F}_i(t-t_i)(1 - \frac{1}{\delta_t^2}(t-t_i)^2) - \ddot{F}_i \frac{1}{2\delta_t}(t-t_i)^2(t-t_i-\delta_t) \end{aligned} \tag{3-75}$$

Note that $F(t)$ is a continuous function which is the essential output of the interpolation block and should be used instead of the discreet values $F^c(t_i)$.

When the numeric ODE solver has done some iteration and decided the step size and state variables values for the time t_{i+1} the force variable value F_{i+1}^f can be calculated. In order to move to the next time step and to interpolate the values for times between t_{i+1} and t_{i+2} we need to evaluate $F_{i+1} = F_x(t_{i+1})$ and the derivatives $\dot{F}_{i+1}$ and $\ddot{F}_{i+1}$ according to the equations:

$$\begin{aligned} \dot{F}_{i+1} = \ & -3F_i \frac{(t_{i+1}-t_i)^2}{\delta_t^3} + 3\ F_i^f \frac{(t_{i+1}-t_i)^2}{\delta_t^3} \\ & + \dot{F}_i(1 - 3\frac{(t_{i+1}-t_i)^2}{\delta_t^2} + \ddot{F}_i(t_{i+1}-t_i)(1 - \frac{3\ (t_{i+1}-t_i)}{2\delta_t} \\ \ddot{F}_{i+1} = \ & -6\ F_i \frac{(t_{i+1}-t_i)}{\delta_t^3} + 6F_i^f \frac{(t_{i+1}-t_i)}{\delta_t^3} \\ & - 6\ \dot{F}_i \frac{(t_{i+1}-t_i)}{\delta_t^2} + \ddot{F}_i(1 - \frac{3(t_{i+1}-t_i)}{\delta_t}) \end{aligned} \tag{3-76}$$

Figure 3-3 shows an interpolation scenario where the ODE solver evaluates one extra time instance t_a between two time step t_i and t_{i+1}. The corresponding value of the force variable F_a is interpolated from F_i, F_i^f, $\dot{F}_i$ and $\ddot{F}_i$ using the procedure above.

The interpolation procedure described in this section provides continuity of the second time derivative of the force vector which is sufficient for most cases. It is certainly possible to achieve higher order continuity by using higher order interpolation methods.

The interpolation procedure can be applied element-wise to vector variables and hence can be used for generalized forces' interpolation.

3.8 Calculating Jacobian

Jacobian computation is an important procedure that influences the efficiency of an implicit solver. The ODE solver receives the differential equation $\dot{\vec{z}} = f(\vec{z}, t)$ that represents the mechanical system behavior and it requires the partial derivatives $\frac{\partial \dot{\vec{z}}}{\partial \vec{z}}$ in order to construct the Jacobian. From the application point of view, knowing that

$\vec{z} \equiv \{\vec{x}, \vec{v}\}$ it is convenient to view the partial derivatives as a set of four matrices:

$$\begin{bmatrix} \frac{\partial \dot{\vec{x}}}{\partial \vec{x}} & \frac{\partial \dot{\vec{x}}}{\partial \vec{v}} \\ \frac{\partial \dot{\vec{v}}}{\partial \vec{x}} & \frac{\partial \dot{\vec{v}}}{\partial \vec{v}} \end{bmatrix}$$

where $\vec{x}$ includes all the translational, rotational and elastic coordinates, and $\vec{v}$ contains all the velocity terms. The blocks in the first row of this matrix ($\frac{\partial \dot{\vec{x}}}{\partial \vec{x}}$ and $\frac{\partial \dot{\vec{x}}}{\partial \vec{v}}$) can be computed analytically. Since for the elastic part $\dot{\vec{x}}_{\mathrm{f}} \equiv \vec{v}_{\mathrm{f}}$, the "elastic" parts of $\frac{\partial \dot{\vec{x}}}{\partial \vec{x}}$ are zero and corresponding parts of $\frac{\partial \dot{\vec{x}}}{\partial \vec{v}}$ are an identity matrix:

$$\frac{\partial \dot{\vec{x}}}{\partial \vec{x}_{\mathrm{f}}} = 0; \; \frac{\partial \dot{\vec{x}}}{\partial \vec{v}_{\mathrm{f}}} = \boldsymbol{I}$$

Calculation of the second row blocks ($\frac{\partial \dot{\vec{v}}}{\partial \vec{x}}$ and $\frac{\partial \dot{\vec{v}}}{\partial \vec{v}}$) is more complicated and involves differentiation of the acceleration terms with respect to the state variables. Let us rewrite the Newton-Euler equations presented in Section 3.4 for some body i:

$$\boldsymbol{M}_i \cdot \dot{\vec{v}}_i^{(b)} = \vec{\boldsymbol{Q}}_i^{(b)} \qquad (3\text{-}77)$$

Differentiating both parts of the equation by the complete state vector $\vec{z}$ we get:

$$\frac{\partial \dot{\vec{v}}_i^{(b)}}{\partial \vec{z}} = \boldsymbol{M}_i^{-1} \cdot \left(\frac{\partial \vec{\boldsymbol{Q}}_i^{(b)}}{\partial \vec{z}} - \frac{\partial \boldsymbol{M}_i}{\partial \vec{z}} \cdot \dot{\vec{v}}_i^{(b)} \right) \qquad (3\text{-}78)$$

The term involving the mass matrix derivatives introduces the specifics of flexible body analysis where the mass matrix is not constant. Another property of the elastic case is the dependency of the generalized force vector $\vec{\boldsymbol{Q}}_i^{(b)}$ on the elastic states.

Obviously the derivatives of the velocity states are nonzero only when differentiated with respect to the same body states or with respect to the states of a directly interacting body. In this context interaction means existence of a force acting between the bodies. Figure 3-4 illustrates the case of a force acting between points fixed in the material of the elastic bodies i and j. Let us describe the procedure to calculate partial derivatives of the $\dot{\vec{v}}_i$ vector with respect to the states of the both bodies.

For the application efficiency the most important aspect is to minimize the number of the force function calculation. This is done by using chain differentiation rules and using force function only to compute a low-dimensional matrix $\frac{\partial \vec{\boldsymbol{F}}}{\partial \vec{\boldsymbol{\Delta}}_{j/i}^{(i)}}$, where $\vec{\boldsymbol{\Delta}}_{j/i}^{(i)}$ is the relative motion tensor, that includes the distance between the points, relative orientation and relative velocities of the points including the effects of structural deformation.

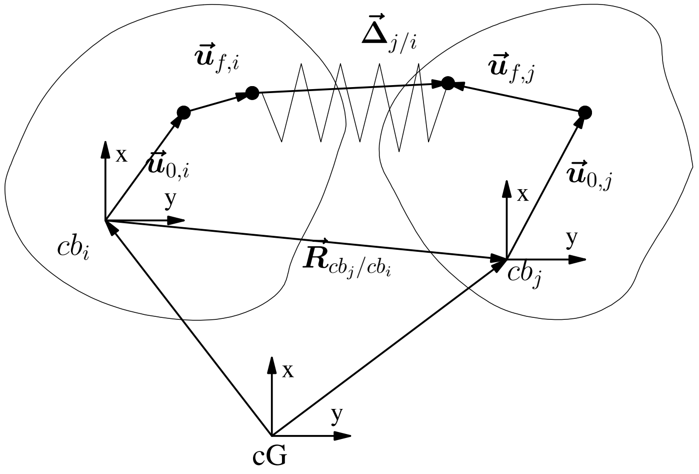


Figure 3-4: *A simple force acting between two flexible bodies.*

The procedure for calculating $\frac{\partial \dot{\vec{v}}_i}{\partial \vec{z}_j}$, when $i \neq j$ (mixed sub-Jacobian) differs from the case $i = j$.

We will first describe the simpler algorithm that is used for mixed sub-Jacobians. For this case in Equation 3-78 only the generalized force vector is dependent on z_j and we can immediately write the differentiation chain:

$$\frac{\partial \dot{\vec{v}}_i}{\partial \vec{z}_j} = \frac{\partial \dot{\vec{v}}_i}{\partial \vec{Q}_{i/j}} \cdot \frac{\partial \vec{Q}_{i/j}}{\partial \vec{z}_j} \tag{3-79}$$

where $\vec{Q}_{i/j}$ is the contribution to the generalized force $\vec{Q}_i$ from the force acting on body i from body j. Note that from the differentiation of Equation 3-77 with respect to $\vec{Q}_{i/j}$ it follows that the matrix $\frac{\partial \dot{\vec{v}}_i}{\partial \vec{Q}_{i/j}} = \boldsymbol{M}_i^{-1}$ is a constant matrix that needs to be computed only once.

Continuing the expansion we get:

$$\frac{\partial \vec{Q}^{(i)}_{i/j}}{\partial \vec{z}_j} = \frac{\partial \vec{Q}^{(i)}_{i/j}}{\partial \vec{F}^{(i)}_{i/j}} \cdot \frac{\partial \vec{F}^{(i)}_{i/j}}{\partial \vec{\Delta}^{(i)}_{j/i}} \cdot \frac{\partial \vec{\Delta}_{j/i}}{\partial \vec{z}_j} \tag{3-80}$$

The partial derivatives $\frac{\partial \vec{\Delta}^{(i)}_{j/i}}{\partial \vec{z}_j}$ reflect both the motion caused by rigid body motion of the geometry center of body j and the elastic deflection motion of the material point due to the changes of the elastic states.

Current implementation uses forward differences to calculate the partial derivatives on the right hand side of Equation 3-80.

Differentiation with respect to the same body state, i.e., the calculation of the derivatives $\frac{\partial \dot{\vec{\boldsymbol{v}}}_i}{\partial \vec{\boldsymbol{z}}_i}$, is more complicated for the following two reasons:

- Mass matrix $\boldsymbol{M}_i$is a function of the state tensor $\vec{\boldsymbol{z}}_i$

- Generalized force is a function of both applied force and the state tensor $\vec{\boldsymbol{z}}_i$.

Recall, for instance, from Section 3.7.4 that the generalized external point force can be written as:

$$\vec{\boldsymbol{Q}}_{i/j} = \begin{bmatrix} \vec{\boldsymbol{F}}_{i/j} \\ (\vec{\boldsymbol{u}}_{0,i} + \vec{\boldsymbol{u}}_{f,i}) \times \vec{\boldsymbol{F}}_{i/j} \\ \boldsymbol{S}^T(\vec{\boldsymbol{u}}_{0,i}) \cdot \vec{\boldsymbol{F}}_{i/j} \end{bmatrix} \tag{3-81}$$

and so the partial derivatives for the elastic states become (all vectors and matrices are in the body i geometric coordinate system):

$$\frac{\partial \vec{\boldsymbol{Q}}_{i/j}}{\partial \vec{\boldsymbol{z}}_{i,f}} = \begin{bmatrix} \frac{\vec{\boldsymbol{F}}_{i/j}}{\partial \vec{\boldsymbol{z}}_{i,r}} \\ \boldsymbol{S}(\vec{\boldsymbol{u}}_{0,i}) \times \vec{\boldsymbol{F}}_{i/j} + (\vec{\boldsymbol{u}}_{0,i} + \vec{\boldsymbol{u}}_{f,i}) \times \frac{\partial \vec{\boldsymbol{F}}_{i/j}}{\partial \vec{\boldsymbol{z}}_{i,f}} \\ \boldsymbol{S}^T(\vec{\boldsymbol{u}}_{0,i}) \cdot \frac{\partial \vec{\boldsymbol{F}}_{i/j}}{\partial \vec{\boldsymbol{z}}_{i,f}} \end{bmatrix} \tag{3-82}$$

Currently, in order to get a possibly simple implementation we use the following general rule. Given some general function $F(g(z), z)$ the partial derivatives with respect to z can be calculated as:

$$\frac{\partial F(g(z_0), z_0)}{\partial z} = \frac{F(g(z_0) + \frac{\partial g(z_0)}{\partial z} \cdot \delta z, z_0 + \delta z) - F(g(z_0), z_0)}{\delta z} \tag{3-83}$$

So, having computed $\frac{\partial \vec{\boldsymbol{F}}_{i/j}}{\partial \vec{\boldsymbol{\Delta}}_{j/i}}$ and having functions $\vec{\boldsymbol{Q}}_{i/j}(\vec{\boldsymbol{F}}_{i/j}, \vec{\boldsymbol{z}}_{f,i})$ and $\vec{\boldsymbol{\Delta}}_{j/i}(\vec{\boldsymbol{z}}_i, \vec{\boldsymbol{z}}_j)$ we can construct a function for numerical differentiation that computes $\frac{\partial \vec{\boldsymbol{Q}}_{i/j}}{\vec{\boldsymbol{z}}_{i,f}}$. The pseudo code for the function follows:

```
for each component of $\vec{\boldsymbol{z}}_{i,f}$ $z_k$ do:
    Choose an increment delta for forward differences $\delta z$
    Calculate:
        $z_k^F = z_k + \delta z$, compose $\vec{\boldsymbol{z}}_i^F$
        $\vec{\boldsymbol{\Delta}}_{j/i}^F = \Delta_{j/i}(\vec{\boldsymbol{z}}_i^F, \vec{\boldsymbol{z}}_j)$
        $\vec{\boldsymbol{F}}_{i/j}^F = \frac{\partial \vec{\boldsymbol{F}}_{i/j}}{\partial \vec{\boldsymbol{\Delta}}_{j/i}} \cdot (\vec{\boldsymbol{\Delta}}_{j/i}^F - \vec{\boldsymbol{\Delta}}_{j/i}^0) + \vec{\boldsymbol{F}}_{i/j}^0$
        $\vec{\boldsymbol{Q}}_{i/j}^F = \vec{\boldsymbol{Q}}_{i/j}(\vec{\boldsymbol{F}}_{i/j}^F, \vec{\boldsymbol{z}}_i^F)$
        $\frac{\partial \vec{\boldsymbol{Q}}_{i/j}}{\partial \vec{\boldsymbol{z}}_{i,f}} = (\vec{\boldsymbol{Q}}_{i/j}^F - \vec{\boldsymbol{Q}}_{i/j}^0)/\delta z$
end for
```

Chapter 4

Static, Eigenmode and Quasi-static Single Body Analysis

Before running a complete multibody dynamic simulation with flexible bodies it is often useful to perform some simple analysis on a single body model. This analysis can help answering the question of the required number of deformation shapes for the particular big simulation case. Note that the presented analysis types are only meaningful for flexible body models. None of them deals with the rigid body motion and mass.

Three kinds of single body analysis were implemented in the BEAST system. They are:

- *Static loading.* In this analysis some constant (time independent) loads are applied on the body and body's deformation in the equilibrium is studied. Since the loads are static no inertia forces are active in the equilibrium state. The discussion of this case later in this chapter shows that only the properties of the stiffness matrix and deformation field can be studied with static loading. Note that in order to perform static analysis the applied loads must be balanced so that no rigid body acceleration is necessary to resolve them.

- *Eigenmode analysis.* Free body eigenfrequencies and eigenmodes are computed in this analysis mode. In case of general shape functions some additional boundary conditions can be implicitly introduced by the selection of the shapes. For instance, specifying no (or too few) shape functions in a certain coordinate direction results in implicit 'fixed-ends' boundary condition. This happens since lack of shapes reduces the number of degrees of freedom for the material particles of the body, often resulting in eigenfrequencies much higher than anticipated. Eigenmode analysis helps in studying the properties of both mass and stiffness matrices of the body as well as the properties of the selected deformation field but does not give any information about the centrifugal forces acting on the body.

- *Quasi-static analysis.* Generally this kind of analysis assumes constant velocity analysis. The most interesting cases are constant angular velocity revolutions around different rotations axes. This analysis helps the study of the centrifugal forces as well as mass matrix, stiffness matrix and deformation field of a body.

It must be clearly understood that no time domain integration is required to perform single body analysis. That makes the analysis much less computationally demanding than complete simulations. Hence, users are strongly recommended to carefully study the results of single body analysis before performing the computationally heavy time domain runs.

The following sections discuss the equations necessary to perform all three kind of analysis described above. They are all based on the generalized Newton-Euler equation that was discussed in Section 3.4. The equation with all the vectors and matrices computed in body's local coordinate system is repeated below for convenience:

$$\begin{bmatrix} \boldsymbol{M}_{RR} & \tilde{\vec{\boldsymbol{S}}}_t^T & \bar{\boldsymbol{S}} \\ & \bar{\boldsymbol{J}}_{ee} & \bar{\boldsymbol{J}}_{ef} \\ symmetric & & \boldsymbol{M}_{\mathrm{ff}} \end{bmatrix} \begin{bmatrix} \ddot{\vec{\boldsymbol{R}}}_{\mathrm{b/g}}^{(\mathrm{b})} \\ \vec{\boldsymbol{\alpha}}^{(\mathrm{b})} \\ \ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}} \end{bmatrix} = \begin{bmatrix} (\vec{\boldsymbol{Q}}_{\mathrm{e}}^{(\mathrm{b})})_{\mathrm{R}} \\ (\vec{\boldsymbol{Q}}_e)_\alpha \\ (\vec{\boldsymbol{Q}}_e)_{\mathrm{f}} - \boldsymbol{K}_{\mathrm{ff}} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} - \boldsymbol{C}_{\mathrm{ff}} \cdot \dot{\vec{\boldsymbol{x}}}_{\mathrm{ff}} \end{bmatrix} + \begin{bmatrix} (\vec{\boldsymbol{Q}}_v)_{\mathrm{R}} \\ (\vec{\boldsymbol{Q}}_v)_\alpha \\ (\vec{\boldsymbol{Q}}_v)_{\mathrm{f}} \end{bmatrix} \tag{4-1}$$

4.1 Static Loading Cases

The equilibrium equation for the static analysis can be derived from Equation 4-1 by setting all the accelerations and velocities to zero, and further assuming that no rigid-body motion is required to resolve the loading case. The resulting equation follows:

$$0 = (\vec{\boldsymbol{Q}}_e)_{\mathrm{f}} - \boldsymbol{K}_{\mathrm{ff}} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} \tag{4-2}$$

This is a linear equation for the elastic state vector that has a unique solution:

$$\vec{\boldsymbol{x}}_{\mathrm{f}} = \boldsymbol{K}_{\mathrm{ff}}^{-1} \cdot (\vec{\boldsymbol{Q}}_e)_{\mathrm{f}} \tag{4-3}$$

The generalized external force vector $(\vec{\boldsymbol{Q}}_e)_{\mathrm{f}}$ gives the specific of the loading case and normally stands for superposition of several point forces and body loads. See Sections 3.7.4 and 3.7.6 for the calculation procedure for these cases.

Typical example static load cases for rings include symmetrical n-point loads and for beams they are point or distributed loads acting on a simply supported beam.

4.2 Eigenmode Analysis

The eigenmode analysis discussed in this section is limited to the discussion of the free-vibration equilibrium analysis with damping neglected. More discussions on eigenmode analysis can be found in [3, 9].

Since rigid body frequencies (which are always zero for a free body) are not interesting to analyze, Equation 4-1 reduces to:

$$\boldsymbol{M}_{\text{ff}} \cdot \ddot{\vec{\boldsymbol{x}}}_{\text{f}} = -\boldsymbol{K}_{\text{ff}} \cdot \vec{\boldsymbol{x}}_{\text{f}} \tag{4-4}$$

In vibration analysis it is assumed that the solution of this equation is represented in the form:

$$\vec{\boldsymbol{x}}_{\text{f}} = \vec{\boldsymbol{a}} \sin \omega t \tag{4-5}$$

where $\vec{\boldsymbol{a}}$ is a vector with the number of elements equal to the number of degrees of freedom (or number of elastic coordinates) n_{f} used in the system, t is the time variable and ω is a constant identified to represent the frequency of vibration (radians/second) of the vector $\vec{\boldsymbol{a}}$. In most cases the $\vec{\boldsymbol{a}}$ vector is called *eigenmode* and the associated ω constant is called *eigenfrequency*. Substituting Equation 4-5 into Equation 4-4 a generalized eigenproblem is obtained, from which $\vec{\boldsymbol{a}}$ and ω must be determined:

$$\boldsymbol{K}_{\text{ff}} \cdot \vec{\boldsymbol{a}} = \omega^2 \boldsymbol{M}_{\text{ff}} \cdot \vec{\boldsymbol{a}} \tag{4-6}$$

The eigenproblem yields the n_{f} eigensolutions $(\omega_i^2, \vec{\boldsymbol{a}}_i)$, where eigenvectors are M-orthonormalized, i.e.,:

$$\vec{\boldsymbol{a}}_i^T \cdot \boldsymbol{M}_{\text{ff}} \cdot \vec{\boldsymbol{a}}_j = \begin{cases} 1, \ i = j \\ 0, \ i \neq j \end{cases} \tag{4-7}$$

and

$$0 \leq \omega_1 \leq \omega_2 \leq ... \leq \omega_{n_{\text{f}}} \tag{4-8}$$

The vector $\vec{\boldsymbol{a}}_i$ is called the i-th mode shape vector, and ω_i is the corresponding vibration frequency. In order to calculate the positions of material particles for a certain frequency one just need to substitute the vector $A \cdot \vec{\boldsymbol{a}}_i$ (A is an arbitrary real constant) for $\vec{\boldsymbol{x}}_{\text{f}}$ in deflection calculations (Equations 3-2). A typical eigenmode vibration visualisation procedure should repeat this calculations for different values of A obtaining an eigenvibration animation sequence.

The numerical algorithms for the solution of eigenproblems have attracted large attention in numerical analysis. Most numerical linear algebra packages include these algorithms. In case only algorithms for the solution of standard eigenvalue problem are included in the package it is possible to transform the generalized eigenproblem into the standard form (see [3] for the discussion).

4.3 Quasi-static Analysis

The quasi-static equilibrium equation is derived from Equation 4-1 by setting the acceleration vector, the velocity vectors $\dot{\vec{\boldsymbol{R}}}_{\text{b/g}}^{(\text{b})}$ and $\dot{\vec{\boldsymbol{x}}}_{\text{f}}$, and the external force vectors to zero. Then for a given, constant angular velocity ω:

$$0 = -\boldsymbol{K}_{\text{ff}} \cdot \vec{\boldsymbol{x}}_{\text{f}} + (\vec{\boldsymbol{Q}}_v)_{\text{f}} \tag{4-9}$$

This linear equation can be used to get approximate solution for the static case. Such solution is quite accurate for the cases with weak coupling between the elastic deformation and quadratic velocity vector, and can be considered satisfactory for most cases.

To obtain the exact solution it is necessary to recall Equation 3-35:

$$(\vec{\boldsymbol{Q}}_v)_\mathrm{f} = -\sum_{i=1}^{3}\sum_{j=1}^{3}(\tilde{\boldsymbol{\omega}})^2_{ij}(\bar{\boldsymbol{J}}^T_{ij} + \bar{\boldsymbol{S}}_{ij}\cdot\vec{\boldsymbol{x}}_\mathrm{f}) - 2\cdot\sum_{i=1}^{3}\sum_{j=1}^{3}\tilde{\boldsymbol{\omega}}_{ij}\bar{S}_{ij}\dot{\vec{\boldsymbol{x}}}_\mathrm{f} \tag{4-10}$$

Substituting Equation 4-10 into Equation 4-9 and simplifying, we come to a linear equation:

$$(\boldsymbol{K}_\mathrm{ff} + \sum_{i=1}^{3}\sum_{j=1}^{3}(\tilde{\boldsymbol{\omega}})^2_{ij}\bar{\boldsymbol{S}}_{ij})\cdot\vec{\boldsymbol{x}}_\mathrm{f} = -\sum_{i=1}^{3}\sum_{j=1}^{3}(\tilde{\boldsymbol{\omega}})^2_{ij}\bar{\boldsymbol{J}}^T_{ij} \tag{4-11}$$

If the equation above is solvable then the solution vector gives the deformation shape for the analyzed quasi-static situation. Absence of the solution indicates coupling between the specified rotation and some of the elastic states' time derivatives $\dot{\vec{\boldsymbol{x}}}_\mathrm{f}$. In such a case no unique general solution exists to the quasi-static problem.

Chapter 5

Generation of Mode Shapes from Finite Element Analysis

The majority of the modern simulation tools use a combination of finite element analyses to generate the required shape functions $\boldsymbol{S}(\vec{\boldsymbol{p}}_0)$. Note that from a general point of view both linear finite element methods and floating frame of reference formulation use the same variable separation approach. The space dependent shape functions for the whole body correspond to the combination of all the local element shape functions in a finite element model. The list of all the node positions and orientations corresponds to the complete vector of elastic coordinates.

The problem with the finite element representation is the large number of degrees of freedom in a typical finite element model. The goal of the procedures described in this section is to *reduce* the number of DOFs to be used in a multibody simulation by constructing a transformation matrix $\boldsymbol{V}$. The process is also called *substructuring*. Every column in the $\boldsymbol{V}$ matrix gives a combination of motions for all the nodes in the FE model and in such a way constructs a new deformation shape as a linear combination of the original FE shapes. Normally the new set of deformation shapes is the result of FE analysis for some sets of loading conditions. The choice of $\boldsymbol{V}$ matrix corresponds therefore to the choice of loading conditions that are expected to be typical for the particular multibody simulation. Such loading conditions are relatively easy to identify if the flexible component has some fixed *attachment points* or *interface nodes* where the external load is acting. The number of different loading conditions (hence, number of columns in $\boldsymbol{V}$) is always chosen to be as small as possible to represent the interesting complete deformation shape with desired accuracy.

Most FE calculations start with the assembly of body mass and stiffness matrices $\boldsymbol{M}_{\text{FEM}}$ and $\boldsymbol{K}_{\text{FEM}}$. The corresponding reduced mass and stiffness matrices for a linear model can then be calculated according to:

$$\begin{aligned} \boldsymbol{M}_{\text{ff}} &= \boldsymbol{V}^T \boldsymbol{M}_{\text{FEM}} \boldsymbol{V} \\ \boldsymbol{K}_{\text{ff}} &= \boldsymbol{V}^T \boldsymbol{K}_{\text{FEM}} \boldsymbol{V} \end{aligned} \tag{5-1}$$

In order to recover the motion of all the nodes in the FE model from the reduced set of coordinates the following relation can be used:

$$\vec{\boldsymbol{x}}_{\text{FEM}} = \boldsymbol{V}\vec{\boldsymbol{x}}_f \tag{5-2}$$

A classification of the different approaches to the reduction of FE models can be found in [11]. The author identifies four different categories of the reduction methods depending on the used combinations of loading conditions. This section, instead, presents the FE analysis types used to generate the shape modes. The presented two analysis types are the most commonly used.

Eigen-mode Shapes Many approaches suggest using either the eigen-modes of the free floating struture while others prefer the eigen-analysis with the attachment points fixed. The problem to be solved can be formulated as a generalized eigen-problem:

$$\boldsymbol{M}_{\text{FEM}}\boldsymbol{V}\boldsymbol{\Lambda} = \boldsymbol{K}_{\text{FEM}}\boldsymbol{V} \tag{5-3}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix containing eigenvalues. In case of fixed attachment points the rows and columns of the mass and stiffness matrices corresponding to the interface nodes DOFs are removed from the mass and stiffness matrices.

Particular Modes Particular modes can be generated by specifying specific boundary conditions. To achieve that the complete set of DOFs associated with the FE model is subdivided into two parts:

- $\vec{\boldsymbol{x}}^{\text{ifc}}_{\text{FEM}}$ is associated with the interface nodes;
- $\vec{\boldsymbol{x}}^{\text{int}}_{\text{FEM}}$ is associated with the internal nodes.

The most general equation used to specify the load cases for the particular modes can then be formulated as the following linear equation:

$$\omega_0^2\boldsymbol{M}_{\text{FEM}}\begin{bmatrix}\vec{\boldsymbol{x}}^{\text{ifc}}_{\text{FEM}}\\ \vec{\boldsymbol{x}}^{\text{int}}_{\text{FEM}}\end{bmatrix} + \boldsymbol{K}_{\text{FEM}}\begin{bmatrix}\vec{\boldsymbol{x}}^{\text{ifc}}_{\text{FEM}}\\ \vec{\boldsymbol{x}}^{\text{int}}_{\text{FEM}}\end{bmatrix} + \vec{\boldsymbol{Q}}_{\text{ext}}(\omega_0) = 0 \tag{5-4}$$

where the unknowns are $\vec{\boldsymbol{x}}^{\text{int}}_{\text{FEM}}$ while the values of the interface DOFs $\vec{\boldsymbol{x}}^{\text{ifc}}_{\text{FEM}}$ and some additional external load $\vec{\boldsymbol{Q}}_{\text{ext}}(\omega_0)$ are assumed. In most cases one load case is generated per each interface DOF.

An an example, of choice of the vectors $\vec{\boldsymbol{x}}^{\text{ifc}}_{\text{FEM}}$, consider the Craig-Bampton method [10]. The method is implemented, e.g., in ADAMS/Flex. Here in each set $\vec{\boldsymbol{x}}^{\text{ifc}}_{\text{FEM}}$ only one variable has a unit value while others are set to zero. The method does not include dynamic excitation analysis, that is the ω_0 is assumed to be zero.

A more advanced method, called optimized-substructuring, tries to improve the quality of the solution by using the responses around the main excitation frequencies in the multibody simulation. The thesis [11] presents the approach in details.

Chapter 6

Flexible Body Exchange Formats for Systems Engineering

6.1 Different Formats for Mode Data

Unfortunately, there is no common standard for the storage of mode data produced as a result of FE analyses. The two most known industrial formats are *MNF* (Modal Neutral File) used by MSC.ADAMS [19] and *SID* (Standard Input Data) used by SimPack [32].

MSC.ADAMS MNF The format is proprietary for MSC Software and is used exclusively by ADAMS related packages (ADAMS/View, ADAMS/Flex, etc) [20, 19]. The file can be generated in all the popular FE packages (Nastran, Ansys, ABAQUS, Marc, I-DEAS). The format is originally designed to keep information about Craig-Bampton modes, but other kinds of modes can be potentially stored in the same format as well. Some advanced features of the format are available when it is exported from Nastran. This is due to the fact that MSC Software owns both Nastran and ADAMS and active development of an integrated simulation environment is going on. The toolkit is available for free from the MSC.Software for all the FE tool developers.

The MNF format was chosen as the most well documented and well supported for further studies and test implementation in BEAST. Even though the format is proprietary no big changes from MSC are expected thanks to the support in other FEA tools from independent developers.

SIMPACK SID files The SID format (Standard Input Data) [39] was proposed as a standard for data exchange between FEA tools and MBS software packages. It has advanced optional features for storing information necessary for modeling of second order effects (geometric stiffening). The format uses frequency response modes for generating deformation shapes at interfaces. This is a more general loading condition when compared to the basic Craig-Bampton method that uses only static deformation modes. The frequency dependent mode shapes give higher precision in certain dynamic models.

Unfortunately, only SimPack [32] simulation package from Intec Gmbh utilizes the format. The FemMBS program is distributed as a part of Simpack toolbox. The program enables the data exchange with most FE tools (Ansys, Nastran, ABAQUS, Marc, I-DEEAS, PERMAS). The FemMBS program can be purchased separately from SimPack. There is a substantial annual license fee for the code.

6.2 Generating MNF

The MSC.ADAMS uses MNF (Modal Neutral File) exchange format to define the interface with FE software. The MNF file format is not published. Instead, it is defined by the C and Fortran programming interfaces in the MNF Toolkit. The toolkit is available for free from the MSC.Software for all the FE tool developers and is supported by most industrial FEA packages. BEAST is using the toolkit to read the file and not to modify it.

Within the scope of this project investigation were done to check the possibilities of MNF files generation in FEA software packages ANSYS and ABAQUS. Those are the two FEA packages used by SKF engineers.

External FEM consultants were asked to generate MNF files using the MNF toolkit from ANSYS [2] and ABAQUS [1]. This was done to get an insight into the user scenario for an expert FE analyst when generating MNF files. Detailed reports were ordered and received on the generation of MNF files from ANSYS. A more concise report was ordered and received on ABAQUS.

Our experience from those results follows:

1. The interfaces are implemented and easy to use for the classical application of the methodology: complex geometry with small number of interfaces.

2. Some modifications to the basic FE model are often necessary before a good component model is created. The modifications concern the creating of auxiliary interface nodes used to distribute the load to a group of nodes in the original model. There is a good manual describing the methods.

3. Most FE analysts lack the experience in the area of creating components for the multibody simulations. It might take some time for them to become familiar with the concepts of interface nodes in general and user interfaces to the MNF toolkit within a particular tool. When this basic difficulty is passed the generation of MNF files becomes strait forward and does not require much time.

4. FE-packages lack the ability to retrieve information from an MNF-file. Hence,

a trivial mistake introduced by an FE tool user is only discovered when a component is imported into an MBS framework.

5. The MNF toolkit experience problems when a large number (hundreds) of interface nodes are defined. The difficulties are caused by the necessary large data sets.

6.3 BEAST-MNF Interface

The MNF toolkit is quite versatile most likely for historical reasons. The BEAST use of it is limited to the following:

1. All the node coordinates are read. Note that the information might be excessive since only surface nodes are used for visualization and external loading of the component. However, availability of all the nodes enables modeling of some additional effects, e.g., distributed volume loading.

2. Nodal masses are read and used for calculation of necessary integrals over the component volume. Note that the inertia integrals that may be stored in the MNF are ignored. This was decided after some experiments demonstrating that the lumped formulation (nodal masses) is used to generate the integrals. In such a case using the integrals stored in MNF does not increase accuracy. An alternative approach to set those integrals by doing a specialized assembly and reduction over the FE model is not implemented.

3. Information about surface nodes is read and used for visualization. The information about finite element types is not available in MNF. The surfaces are represented with facets and no information about surface curvature is present. This leads to some visual artifacts when the component is shown in Beast.

4. Interface nodes are identified and marked in the BEAST model. The BEAST control points are used to represent the interfaces making the model structure familiar to BEAST users.

6.4 Model Scaling

Additional feature implemented in BEAST is model scaling. The FE models are often done in non-SI units, which make it necessary to scale the data from the MNF file with some factor. This is also convenient when the same FE model is used for the components of identical geometries but different sizes.

Suppose that the model dimensions need to be scaled with a scalar coefficient a. Let us examine how such a scaling affect different modal data. Note that the $\boldsymbol{M}$-orthonormality of the mode shapes needs to be maintained. This is important for the reasons discussed in Section 8.3.

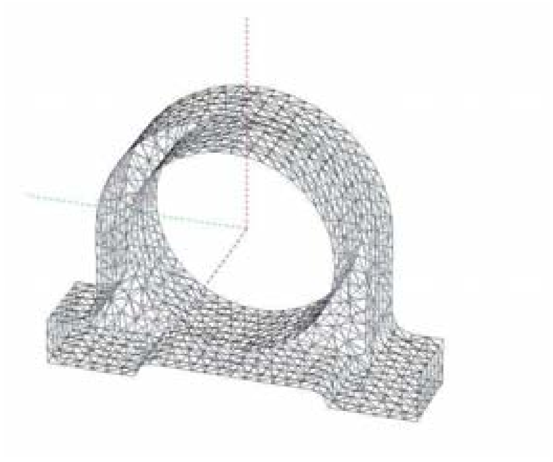


Figure 6-1: *Housing surface mesh imported into MBS tool.*

A naive scaling would just update the mass and stiffness matrices assuming that mass grows cubically with the scale and stiffness grows linearly:

$$\begin{aligned} \boldsymbol{M}^{a}_{\text{ff}} &= a^3 \boldsymbol{M}_{\text{ff}}, \quad \boldsymbol{M}_{\text{ff}} = \boldsymbol{I} \\ \boldsymbol{K}^{a}_{\text{ff}} &= a \boldsymbol{K}_{\text{ff}}, \quad K_{\text{ff}}(i,j) = \delta_{i,j}\ \omega_i^2 \end{aligned} \tag{6-1}$$

where $\boldsymbol{M}_{\text{ff}}$ and $\boldsymbol{K}_{\text{ff}}$ are the original matrices; $\delta_{i,j}$ is Kronecker delta, ω_i is the frequency of the i-th mode.

The mode shapes are non-dimensional and, therefore, do not require scaling. However scaling has changed the mass matrix and therefore has broken the orthonormality condition. In order to restore it a normalization coefficient $\hat{a}$ needs to be introduced:

$$\hat{a} = a^{-1.5} \tag{6-2}$$

Scaling the original mode shapes with the normalization coefficient we get a set of orthonormal modes and corresponding matrices for the scaled geometry:

$$\begin{aligned} \boldsymbol{S}^{\hat{a}} &= \hat{a}\boldsymbol{S} \\ \boldsymbol{M}^{\hat{a}}_{\text{ff}} &= \frac{1}{a^3}\boldsymbol{M}^{a}_{\text{ff}} = \boldsymbol{I} \\ \boldsymbol{K}^{\hat{a}}_{\text{ff}} &= \frac{1}{a^3}\boldsymbol{K}^{a}_{\text{ff}} = \frac{1}{a^2}\boldsymbol{K}_{\text{ff}} \end{aligned} \tag{6-3}$$

6.5 Simulation example

A FE model of bearing housing was chosen to run a demonstrative example. The model was processed in ABAQUS and an MNF file was generated. The file was then imported in a BEAST model and a dynamics simulation was run.

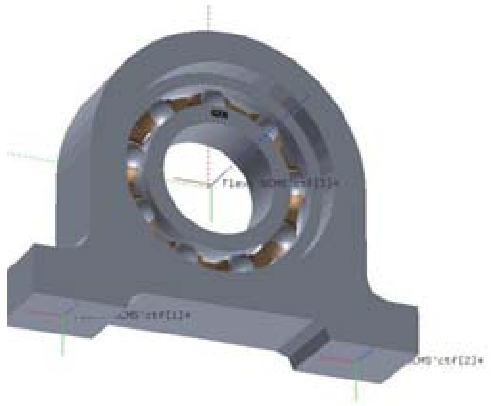


Figure 6-2: *A MBS model with the reduced body. Interface nodes are marked with coordinate systems at the foundation and in the center.*

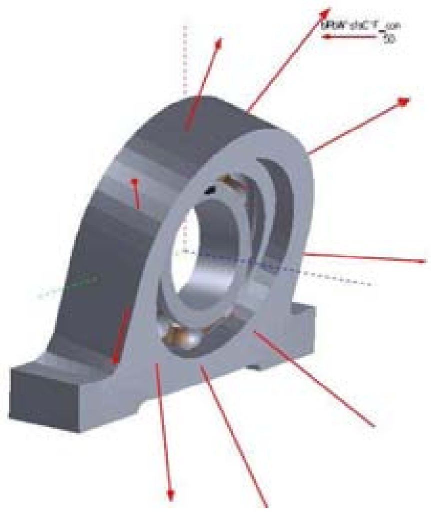


Figure 6-3: *Housing deformation (magnification 1000). Results shown after one revolution with axial load on the inner ring of DGBB.*

The Figures 6-1, 6-2 and 6-3 present the results of a simulation where an axial load was applied to the inner ring of a DGBB bearing. The outer ring of the bearing was attached to the housing. Foundation of the housing was fixed to the ground.

6.6 Summary

There is no common standard for the storage of mode data produced as a result of FE analyses. However, an implementation in a MBS framework based on the formats utilized in other tools, e.g, `MNF` (Modal Neutral File) or SID (Standard Input Data), is strait forward.

Export of flexible models of components with point interfaces is well established in simulation industry. The export functionality is implemented in all the popular FEA tools. The export procedure is automatic, but some modifications to the original FE model are often necessary (e.g., definition of the interface nodes). Familiarity with the export functionality among FEA tools users is low. Some specific training is often necessary.

The exchange file formats are mostly suitable for the components with a few interface nodes, since large amount of data is required otherwise. The most common approach is to utilize the Craig-Bampton method [10]. The method is known to work well for applications with low velocities [11].

Chapter 7

General Shape Functions

7.1 Motivation

We observed that there is a large class of applications with bodies that are constrained by surfaces that can have a contact. The essence of these constraints is the need to have a continuous shape function description for deflection of each of such potential contact surface.

Having considered the limitations with the reduction techniques we decided to use the classic approximation method and sets of general functions to approximate the deflection. Particularly we used Chebyshev polynomials to model deformation in radial and longitudinal directions and Fourier series for the circumference direction of rotating bodies. We use the proposed set of functions to describe the deformation through the material. The use of the well-known analytical functions enables efficient calculation of deformation shapes. The series are known to be efficient in approximating smooth functions and we can expect the real deformation shapes to be smooth and so accurately represented even with low order polynomials and small number of waves. In [29] it was pointed out that the same deflection solution can be obtained using different sets of eigenmodes as mode shapes with high accuracy. Since general functions are able to accurately approximate any set of eigenmodes this assertion also supports the idea of using general shapes.

The flexible model that employs general shape functions is easy to use since no interaction with other tools are needed, and the flexibility is controlled by a few input parameters. An additional attractive feature of the approach is the ability to select specific elastic degrees of freedom in an easy way and disable the deformations that are known to be insignificant.

The shape function integrals are automatically calculated at start up since we have a full geometric description of the body. The integrals may be modified during the simulation. Thus, it makes it possible to include effects of initial stresses and material removal.

Since we have a continuous description of the deformation field, we can have contact or connections anywhere on the body, and there will be no induced vibrations due to modeling errors.

Of course, there is a practical limit when this technique is not suitable, i.e., for components with very little constrained, complex geometry, but those limits are yet to be found. In such cases standard reduction techniques will be more suitable. We believe that different complementing methods need to be available within a large general purpose MBS system.

- For bodies with well defined interfaces the mode shapes should be selected using eigenmode, static and frequency response analysis as described in [28, 41, 40].
- In the case of small area of potential contact for a body with complicated geometry one should complement the reduction method with the continuous shape function description for the contacting surface [43].
- For the case of multiple contacting surfaces and relatively simple geometry the general shape functions approach presented in this thesis should be used. This is also the only approach known to work in simulations of prestressed structures during material removal.

7.2 Choice of Shape Functions

For the majority of real-life problems the exact deflection shape functions are not known and different methods to select shapes that provide the best approximation of the exact solution are often discussed in the literature. It is also known that different sets of functions give approximately the same results when sufficient number of degrees of freedom is included in the system.

The approach taken in this work was to use series of general functions that are known to approximate any continuous function. The approach is based on the idea first suggested in [43] for the representation of interface surface deformation shape only. In this work the same set of functions was used to represent deformation field through the entire volume of elastic body.

Specifically, when working in cylindrical coordinate system series of Chebyshev polynomials were used for the radial and axial directions and Fourier series for the circumferential direction. That is the deflection shape along certain coordinate direction k (radial, tangential or axial) is approximated as:

$$\vec{p}^{(b)}_{f,k} = u_k = \sum_{i_r=0}^{n_{r,k}} \sum_{i_\phi=0}^{n_{\phi,k}} \sum_{i_z=0}^{n_{z,k}} x_{f,k}^{i_r,2i_\phi,i_z} \cdot g_{i_r}(p_{0,r}) \cdot \sin(i_\phi\, p_{0,\phi}) \cdot g_{i_z}(p_{0,z})$$

$$+ \sum_{i_r=0}^{n_{r,k}} \sum_{i_\phi=0}^{n_{\phi,k}} \sum_{i_z=0}^{n_{z,k}} x_{f,k}^{i_r,2i_\phi+1,i_z} \cdot g_{i_r}(p_{0,r}) \cdot \cos(i_\phi\, p_{0,\phi}) \cdot g_{i_z}(p_{0,z}) \quad (7\text{-}1)$$

Where $n_{r,k}, n_{\phi,k}, n_{z,k}$ give the number of functions used in each direction, g_j is the Chebyshev polynomial of order j and $\vec{\boldsymbol{x}}_f$ is the vector of elastic coordinates.

For the case of Cartesian coordinates Chebyshev polynomials were used for all the three coordinate directions. Hence the real deflection shape along certain coordinate direction k (x, y or z) is approximated as:

$$\vec{\boldsymbol{p}}^{(b)}_{f,k} = u_k = \sum_{i_x=0}^{n_{x,k}} \sum_{i_y=0}^{n_{y,k}} \sum_{i_z=0}^{n_{z,k}} x_{f,k}^{i_x,i_y,i_z} \cdot g_{i_x}(p_{0,x}) \cdot g_{i_y}(p_{0,y}) \cdot g_{i_z}(p_{0,z}) \tag{7-2}$$

where $n_{x,k}, n_{y,k}, n_{z,k}$ give the number of polynomials used in each direction.

An important property of the both orthogonal polynomials and Fourier series is the availability of the recurrent relations providing a way for fast series calculation. For instance, for the Chebyshev polynomials the relation is given below:

$$\begin{cases} g_0(x) = 1 \\ g_1(x) = x \\ g_m(x) = 2x g_{m-1}(x) - g_{m-2}(x) \end{cases} \tag{7-3}$$

Note that having the recurrent relation for the polynomials we can easily get recurrent relations for the derivatives of the polynomials as well. These derivatives are required for the strain and stress calculation. Again using the Chebyshev polynomials as an example:

$$\begin{cases} \dfrac{dg_0(x)}{dx} = 0 \\ \dfrac{dg_1(x)}{dx} = 1 \\ \dfrac{dg_m(x)}{dx} = 2(g_{m-1}(x) + x\dfrac{dg_{m-1}(x)}{dx}) - \dfrac{dg_{m-2}(x)}{dx} \end{cases} \tag{7-4}$$

7.3 Special Shape Functions for Solid Bodies in Cylindrical Coordinates

One problem with the general shape functions concerns the use of cylindrical coordinate system as a body coordinate system, which is quite natural for the bodies with rotation symmetry. The problem here is the singular point in the centre of the coordinate system. This point is not important for the bodies with no material at the centre (rings and hollow shafts), but it is unavoidable for the solid bodies (solid shafts).

Let us recall the deflection to strain equations for the cylindrical coordinates in

Equation 3-40:

$$\begin{aligned}
\epsilon_{11} &= \frac{\partial u_r}{\partial r} \\
\epsilon_{22} &= \frac{1}{r}(\frac{\partial u_\phi}{\partial \phi} + u_r) \\
\epsilon_{33} &= \frac{\partial u_z}{\partial z} \\
\epsilon_{12} &= \frac{1}{r}(\frac{\partial u_r}{\partial \phi} - u_\phi) + \frac{\partial u_\phi}{\partial r} \\
\epsilon_{13} &= \frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r} \\
\epsilon_{23} &= \frac{1}{r}\frac{\partial u_z}{\partial \phi} + \frac{\partial u_\phi}{\partial z}
\end{aligned} \tag{7-5}$$

In order for the strain components ϵ_{22}, ϵ_{12} and ϵ_{23} to be finite in the centre of the coordinate system, that is for $r = 0$, the following equation must be satisfied for any angle $\phi \subset [0, 2\pi]$ and for any cross-section cut z:

$$\begin{aligned}
&\frac{\partial u_\phi}{\partial \phi} + u_r = 0 \\
&\frac{\partial u_r}{\partial \phi} - u_\phi = 0 \\
&\frac{\partial u_z}{\partial \phi} = 0
\end{aligned} \tag{7-6}$$

From the physical point of view the Equation 7-6 require the continuity of the displacement functions for every point on the centre line of the body.

In order to realize this constrain both sets of shape functions defined in Equations 7-1 and 7-2 were used. The cylindrical shape functions were used to describe the deflection shape of the cross-section and the Cartesian shapes were dedicated to the central line deflection. That is, the cylindrical shape functions were modified so that they never result in a deflection of the centre line of the body. This is done by removing the constant term from the Chebyshev polynomials that have $p_{0,r}$ as an argument. At the same time the Cartesian shape functions were restricted to the centre line deflections by removing the dependency of the x and y coordinates thus leaving only a single polynomial $g(p_{0,z})$ in the series. The Cartesian shapes were then transformed into the cylindrical coordinates by using ordinal coordinate transformation:

$$\begin{bmatrix} u_r \\ u_\phi \\ u_z \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{7-7}$$

7.4 Volume Integration

Volume integrals representing the stiffness, damping and inertia properties of elastic objects are needed for simulation.

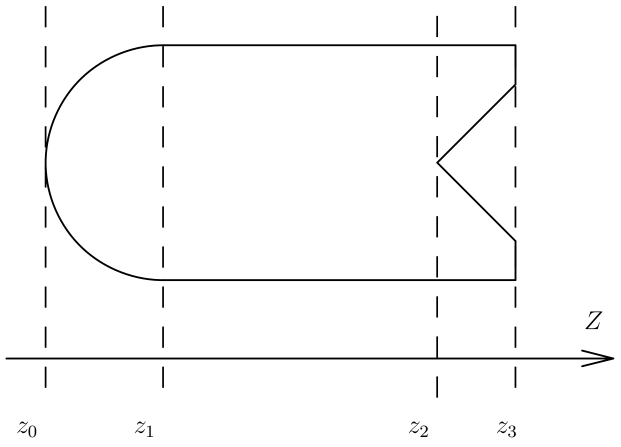


Figure 7-1: *A profile curve used to define a rotation symmetry body.*

Here, most work has been done on integration of bodies defined with a rotating profile curve. Cylindrical system (r, ϕ, z) is said to be the 'natural' coordinate system for such bodies. To define rotating profile body a profile curve $(r(u, \phi), z(u, \phi))$ must be given for any angle ϕ. The profile curve is defined piecewise and each segment has a separate parameter u going along the curve. We assume that the curve is continuous both in value and derivatives within each segment.

The integrals are calculated in the cylindrical space and not in the profile curve parameters' space, since the deformation shape functions that we integrate are defined in the physical space and are not related to the boundary parameters' spaces.

The integration of rotating profile bodies has to be performed in cylindrical coordinates for accuracy reasons. Integration of a rotating profile body, that often has some kind of rotation symmetry, in Cartesian coordinates gives low accuracy and asymmetric stiffness matrix which results in non-physical vibrations in simulation output.

Integration is performed first in the radial direction, then in longitudinal direction and finally in circumferential direction. There are two basic ideas behind this ordering. First, we want to put the most varying boundary (radial) into the innermost integral to simplify the implementation and be able to use high precision integration scheme for it. Secondly, having integral in circumferential direction as outermost ensures the rotational symmetry properties of the resulting matrices. We use extended Simpson's (trapezoidal) integration scheme in circumferential direction since it is known to give good results for periodic functions and we have Fourier series as a basic set of functions in this direction. Both in radial and longitudinal direction we use modified Clenshaw-Curtis quadratures [25].

The complete algorithm follows. We will refer to Figure 7.4 to illustrate the algorithm. The figure shows a six segments profile curve with four straight and two curved segments.

1. Select the number of integration points in circumferential direction. The number of points must be more than the double of the number of waves used in the shape functions or in geometry deviation in circumferential direction. Integration bounds are $[-\pi, \pi]$.

2. For each angle ϕ we have a cylindrical sector to integrate. We start by going through all the segments of the profile curve and evaluating the boundary for the specified ϕ. Our purpose is to find minimum and maximum z coordinate for each segment as well as the edge values between the segments. These values are used to define integration intervals in z direction. In Figure 7.4 we would find 3 integration intervals with bounds z_0, z_1, z_2 and z_3. In the general case we might get several non-adjacent integration intervals.

3. For each integration interval in z direction, calculate abscissas and weights according to the quadrature formula. The order of integration (number of points) is double the highest order of polynomial used as a deformation shape in longitudinal direction. Note that each integration interval is treated separately.

4. We need to go through all the segments once again to find the radii for each z abscissa detected in the previous step. An ordered list of radii is build for each abscissa. If the closed volume condition is satisfied then the number of elements in every radii list will be even for hollow bodies and odd for bodies with material in the centre of a cross-section. Every pair of radii in the lists define an integration interval in radial direction and quadrature formula can be applied again to get the radial abscissas and weights. Note that since we have split the integration along z axis in intervals the number of integration intervals in radial direction will be constant within each z interval. In the example, Figure 7.4, we have just one radial integration interval between z_0 and z_2 and two intervals between z_2 and z_3

Having computed the set of integration points and weights as described above we can approximate all the required integrals with the weighted sums of function values computed at the specified points multiplied with the corresponding weights.

There are still some uncertainties regarding the integration process. How does the complexity of the boundary geometry affects the accuracy of the integrals? How do we select the optimal number of integration points? Is adaptive strategy necessary?

7.5 Reducing the Number of Flexible States

The series defined by the Equations 7-1 and 7-2 provide a complete mathematical set of functions suitable for approximation of any continuous deformation shape. However in a real simulation only some specific deformation shapes determined by the flexible body geometry and boundary conditions are present. That is why contributions of some of the general shape functions to the particular solution are small. By detecting such shape functions and eliminating the state variables associated with them we can reduce the number of flexible states for a body and speed up the simulation. Note that this approach is fundamentally different from the traditional reduction techniques. We do not try to find the most suitable shapes and change the functional basis, instead we just eliminate some of the shape functions in the complete series.

Let us define a *characteristic set* of shape vectors to be the set of elastic state vectors corresponding to the deformation shapes typical for the specific simulation. The characteristic set can be constructed using static, eigenmode, or quasi-static analysis (see Chapter 4), or using a relatively short trial simulation in the following ways:

Static shapes The state vectors corresponding to the static solutions of a typical boundary conditions problem should be a part of the characteristic set.

Eigenvectors If the user specifies a frequency band then the eigenvectors corresponding to the free body eigenfrequencies in this band should be included into the characteristic set. Alternatively a user might want to specify the number of eigenmodes (e.g., specify that only first three eigenmodes are of interest). In such a case just the specified number of eigenvectors should be included into the characteristic set.

Quasi-static shapes In rolling-bearing applications simulation for a certain specified angular velocity is common task. The solution vectors for the corresponding quasi-static case should be included into the characteristic set.

Trial simulation In a rotating machinery simulations there are often some periodic processes where the difference between the two periods is relatively small. For instance, two complete revolutions of the workpiece in a grinding simulation is an example of such a repetition. In such cases a trial simulation of exactly one revolution can be run and all the state vectors during this trial simulation can be recorded into the characteristic set. Note that the vectors generated during the simulation often correspond to the same shape with different amplitude. Then, after the reduction, the following revolutions of the machine can be simulated faster with smaller number of elastic states.

Having selected the state vectors into the characteristic set and for a specified relative threshold ϵ (which is an input parameter) the following filtering algorithm should be applied.

```
Mark all the elastic states as inactive

for each state vector in the characteristic set
```

```
        Calculate the threshold:
            $x_\epsilon = \epsilon \cdot \left( \frac{1}{n_f} \sum_{i=1}^{n_f} |x_{f,i}| \right)$    (7-8)
        Remove the inactive mark for the elastic states that have the absolute
        value larger than $x_\epsilon$.
    end for
```

Note that it is assumed that all the shape functions are scaled with the real geometrical dimensions of the body and therefore all the state variables has the same dimension (meters) and can be directly compared.

As the result of the algorithm only the states that have very small contributions to the vector in the characteristic set retain the inactive mark. Those states have therefore become the candidates for the reduction. The choice of the threshold ϵ is normally determined by the anticipated modeling error. That is, if one percent error is acceptable, then ϵ can be set to $1e - 2/n_f$.

Assuming that the contributions of the states with inactive mark are negligible they can be safely eliminated from the problem. Elimination of an elastic state i means removal of the i-th element from all the vectors that have a part associated with the elastic states. Hence the i-th element should be removed from the elastic state vector $\vec{\boldsymbol{x}}_f$, velocity vector $\dot{\vec{\boldsymbol{x}}}_{\mathrm{f}}$, acceleration vector $\ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}}$ as well as from the generalized forces vector $\vec{\boldsymbol{Q}}_{\mathrm{f}}$ and precomputed integrals $\bar{\boldsymbol{S}}$ and $\bar{\boldsymbol{J}}_{kl}$. Besides the i-th row and i-th column should be removed from all the matrices associated with the elastic states: $\boldsymbol{M}_{\mathrm{ff}}$, $\boldsymbol{K}_{\mathrm{ff}}$, $\boldsymbol{C}_{\mathrm{ff}}$ and $\bar{\boldsymbol{S}}_{kl}$.

The technique has not been used in the production runs yet. However, some of our test runs showed that up to the half of the elastic states generated by the general functions from Equations 7-1 and 7-2 for the user-specified number of degrees of freedom can be eliminated later on by using the presented reduction approach. This is due to the fact that the general interpolation series introduce many shapes that correspond to very high frequencies for the particular body.

Chapter 8

Mean-Axis Conditions on the Reference Frame

8.1 Separation of Elastic and Rigid Body Motion Modes

Typically in an assumed mode shape method the selected mode shapes always satisfy some specific boundary conditions. This is not true for the general mathematical functions used in this work. Thus we have to impose some extra conditions in order to make the selected set of mode shapes admissible for the problem.

In this thesis the case with no fixed external boundary conditions is considered - all the bodies have all the degrees of freedom. That means the only condition imposed on the shape field is the absence of rigid body motion degrees of freedom in the elastic deformation [28].

In order to understand the importance of this condition consider a small example in 2D. Suppose that only two deflection functions are selected: $u_x(x,y) = x_{\mathrm{f}}^{(1)} p_y$ and $u_y(x,y) = x_{\mathrm{f}}^{(2)} p_x$. If we follow the standard procedure for calculating the strain energy of an object we will get: $W(\vec{\boldsymbol{x}}_{\mathrm{f}}) = \mu A (x_{\mathrm{f}}^1 + x_{\mathrm{f}}^2)^2$ where μ is the Lame material constant and A - area of the body. The relation means that when $x_{\mathrm{f}}^{(1)} = -x_{\mathrm{f}}^{(2)}$ no strain energy is created. The immediate conclusion that this shape combination corresponds to the rigid body motion is only partially correct. In Figure 8-1 we show the changes to a simple square when the shape coefficients are $\vec{\boldsymbol{x}}_{\mathrm{f}} = [1, -1]$. It is obvious that the area of the square grows, for our example it is doubled. The example shows that the use of general functions without proper condition will not just slow down the simulation, but can result in some non-physical effects. The discussed example, for instance, would lead to the 'explosion' of a body, since the change of the area doesn't create any energy and consequently doesn't provide any stiffness.

The conditions that were chosen to remove the rigid body motion terms from the deflection functions correspond to elastic body mean-axis condition for the reference frame. The mean-axis conditions are obtained by minimizing the kinetic energy with

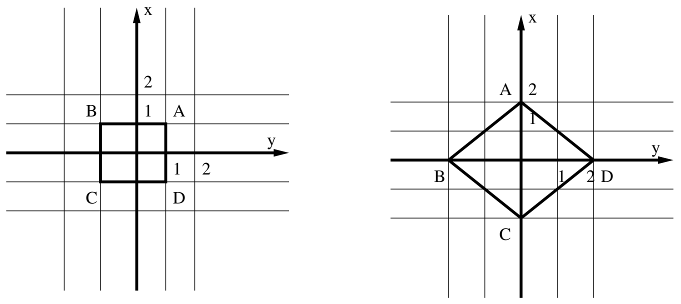


Figure 8-1: *Use of shape functions without boundary conditions.*

respect to an observer stationed on the flexible body [28, 29, 37, 23]. The stronger version of the mean-axis conditions can be formulated in term of the precomputed integrals (see Section 3.5). The total of six conditions can be partitioned into the three conditions for the translational degrees of freedom:

$$\vec{\boldsymbol{J}}_1 + \bar{\boldsymbol{S}} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} = 0 \tag{8-1}$$

And three conditions for the rotational degrees of freedom:

$$\begin{bmatrix} (\bar{\boldsymbol{J}}_{23} - \bar{\boldsymbol{J}}_{32}) \\ (\bar{\boldsymbol{J}}_{31} - \bar{\boldsymbol{J}}_{13}) \\ (\bar{\boldsymbol{J}}_{12} - \bar{\boldsymbol{J}}_{21}) \end{bmatrix} \cdot \vec{\boldsymbol{x}}_{\mathrm{f}} = 0 \tag{8-2}$$

Note that the conditions effectively attach the reference frame to the mass center of the body allowing simplifications to the dynamic equations described in Chapter 3. The mass matrix simplifies to (using body coordinate system for all vectors):

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_{\mathrm{RR}} & 0 & \bar{\boldsymbol{S}} \\ & \bar{\boldsymbol{J}}_{\mathrm{ee}} & \bar{\boldsymbol{J}}_{\mathrm{ef}} \\ \multicolumn{2}{c}{\textit{symmetric}} & \boldsymbol{M}_{\mathrm{ff}} \end{bmatrix} \tag{8-3}$$

And the quadratic velocity term $(\vec{\boldsymbol{Q}}_v)_{\mathrm{R}}$ also becomes zero.

Having the mass matrix simplified and noting that from the no mass center motion condition $\bar{\boldsymbol{S}} \cdot \ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}} = 0$ one can also avoid the inverse of the complete mass matrix when finding the acceleration tensor from the Newton-Euler equation and get:

$$\begin{aligned} &\ddot{\vec{\boldsymbol{R}}}^{(\mathrm{b})}_{\mathrm{m/g}} = \frac{1}{m}\vec{\boldsymbol{Q}}_{\mathrm{r}} \\ &\vec{\boldsymbol{\alpha}}^{(\mathrm{b})} = (\bar{\boldsymbol{J}}_{\mathrm{ee}} - \bar{\boldsymbol{J}}_{\mathrm{ef}} \cdot \boldsymbol{M}_{\mathrm{ff}}^{-1} \cdot \bar{\boldsymbol{J}}_{\mathrm{ef}}^{T})^{-1}(\vec{\boldsymbol{Q}}_{\alpha} - \bar{\boldsymbol{J}}_{\mathrm{ef}} \cdot \boldsymbol{M}_{\mathrm{ff}}^{-1}(\cdot\vec{\boldsymbol{Q}}_{\mathrm{f}} - \bar{\boldsymbol{S}}^{T} \cdot \ddot{\vec{\boldsymbol{R}}}^{(\mathrm{b})}_{\mathrm{m/g}})) \\ &\ddot{\vec{\boldsymbol{x}}}_{\mathrm{f}} = \boldsymbol{M}_{\mathrm{ff}}^{-1} \cdot (\vec{\boldsymbol{Q}}_{\mathrm{f}} - \bar{\boldsymbol{J}}_{\mathrm{ef}}^{T} \cdot \alpha - \bar{\boldsymbol{S}}^{T} \cdot \ddot{\vec{\boldsymbol{R}}}^{(\mathrm{b})}_{\mathrm{m/g}}) \end{aligned} \tag{8-4}$$

Even though the expressions in Equation 8-4 might look more complicated than the original equation in Section 3.4 one should note that the simplified expression require inverse only of two matrices: a constant matrix $\boldsymbol{m}_{\text{ff}}$ and a small 3×3 matrix to find the angular acceleration. Hence the time for the solution of this linear system grows as $O(n_f^2)$ instead of $O(n_f^3)$ in the original case with full mass matrix.

8.2 Imposing Constraints for Rigid Body Motion

There are several possible approaches to impose boundary conditions discussed earlier in this chapter on the shape functions. The three most popular methods are mentioned in this section. They are: Lagrange multipliers method, penalty method and elimination method.

Lagrange multipliers method is very popular in multibody simulation tools [3, 31]. It provides a general way to deal with boundary conditions. In this method one extra state variable is associated with every boundary condition. These extra state variables are called Lagrange multipliers. A Lagrange multiplier becomes nonzero if the boundary condition is active and results in a reaction force that forces the condition to be satisfied. The drawback of this method is the necessity to have extra state variables and keep larger mass matrix and force vector. If the number of constraints is large (comparable to the number of the elastic shape functions) the number of extra state variables becomes comparable to the total number of elastic degrees of freedom. That is the size of the system of ODE becomes significantly larger and makes the simulation slower.

The idea of a penalty approach is to represent each boundary condition as a spring with very high stiffness K_b [3, 7]. The stiffness matrix of the elastic body and the force vector are modified to incorporate the effect of this spring. By choosing the K_b parameter large enough it is possible to get sufficiently accurate solution, i.e., a solution where the error in boundary condition lies within the numerical tolerance. The penalty approach is general and simple to implement.

A general linearized boundary condition can be written as:

$$\vec{\boldsymbol{B}}^T \cdot \vec{\boldsymbol{x}}_{\text{f}} = b_0 \tag{8-5}$$

where elements of $\vec{\boldsymbol{B}}$ are known constants. In reality, constraints often restrict only a few variables and so the $\vec{\boldsymbol{B}}$ vector is sparse.

The modified stiffness matrix is given by:

$$\bar{\boldsymbol{K}}_{\text{ff}} = \boldsymbol{K}_{\text{ff}} + K_b \cdot \vec{\boldsymbol{B}}^T \cdot \vec{\boldsymbol{B}} \tag{8-6}$$

and an extra reaction force vector can be computed as:

$$\vec{\boldsymbol{Q}}_b = K_b \cdot b_0 \cdot \vec{\boldsymbol{B}} \tag{8-7}$$

The large boundary condition spring constant is typically chosen to be several orders of magnitude higher than the maximum diagonal element of the original stiffness matrix $\boldsymbol{K}_{\text{ff}}$. In this case boundary constraints introduce frequencies that are much higher than the body's eigenfrequencies and are in practice completely damped out when even a small stiffness proportional Rayleigh damping is used.

It must be noted that the introduction of multiple constraints with the penalty approach results in large diagonal elements of the stiffness matrix. The drawback of this, according to [7] is the sensitivity of the results to the computational errors. To avoid this problem the K_b constant should not be set to extremely high values and double precision arithmetics must be used. In out test runs it was noted that the numerical error introduced by the penalty constants is several order of magnitude lower than the general modeling error and other kinds of computational errors.

The elimination method is based on the idea that the set of m boundary conditions can be used as a set of equations to express the values of some m elastic state variables as functions of other states of the body. In this case corresponding m degrees of freedom can be considered as redundant and may be safely removed from the system. In general, it is very hard or impossible to determine which degrees of freedom can be eliminated in a large simulation model. However, use of orthonormalization procedure described in the next section makes elimination strait-forward for the case of removing rigid body motion. Since orthonormalization brings some additional benefits this method was chosen the implementation in the BEAST system.

8.3 Orthonormalization of Deformation Shapes

There are generally two options on how to realize the mean axis conditions. One approach is to follow the procedures defined in the previous section and enforce the condition on the pre-selected shape functions (see also [23]). Another, more efficient approach, is to utilizes the orthonormalization procedure to change the set of shape functions so, that the mean-axis conditions is always fulfilled.

Generalized eigen-problem for the mass and stiffness matrices can be formulated as following:

$$\boldsymbol{M}_{\text{ff}}\boldsymbol{V}\boldsymbol{\Lambda} = \boldsymbol{K}_{\text{ff}}\boldsymbol{V} \tag{8-8}$$

Solving it gives a set of eigen-vectors stored in the columns of matrix $\boldsymbol{V}$ and a set of eigen-values stored in the diagonal matrix $\boldsymbol{\Lambda}$. Recall from the Section 4.2 that each eigen-value corresponds to an eigen-frequency of the body:

$$\boldsymbol{\Lambda} = \{\delta_{ij}\omega_i^2\} \tag{8-9}$$

where δ_{ij} is Kronecker delta and ω_i is the i-th frequency.

The important properties of the solution of a generalized eigen-problem are $\boldsymbol{M}$-orthonormality and $\boldsymbol{K}$-orthogonality of eigenvectors:

$$\begin{aligned} \boldsymbol{V}^T\boldsymbol{M}_{\text{ff}}\boldsymbol{V} &= \boldsymbol{I} \\ \boldsymbol{V}^T\boldsymbol{K}_{\text{ff}}\boldsymbol{V} &= \boldsymbol{\Lambda} \end{aligned} \tag{8-10}$$

If we assume that the original set of shape functions $\boldsymbol{S}$ included the rigid body motion shapes, then six of the eigen-frequencies ω_i in the resulting solution will be zero. It is then possible to identify the set of eigen-vectors corresponding to the rigid body motion and remove them from matrix $\boldsymbol{V}$. Let us call the resulted reduced matrix $\hat{\boldsymbol{V}}$. Note that reduced matrix still satisfy $\boldsymbol{M}$-orthonormality and $\boldsymbol{K}$-orthogonality conditions.

It is now possible to construct a new set of shape functions $\hat{\boldsymbol{S}}$ such as:

$$\hat{\boldsymbol{S}} = \boldsymbol{S}\hat{\boldsymbol{V}} \tag{8-11}$$

From the Eq 8-10 it follows that the mass matrix corresponding to the new set of shapes is identity and the stiffness matrix is diagonal:

$$\begin{aligned} \hat{\boldsymbol{M}}_{\text{ff}} &= \boldsymbol{I} \\ \hat{\boldsymbol{K}}_{\text{ff}} &= \{\delta_{ij}\omega_i^2\}, \end{aligned} \tag{8-12}$$

.

Furthermore, if we associate the reference frame of the body with its mass center then the shape functions $\hat{\boldsymbol{S}}$ will automatically satisfy the principles of the mean axis conditions (see [37]).

As the mean axis conditions are satisfied most of the inertia coupling terms between the deformation and rigid body translation and rotation vanish:

$$\begin{aligned} \bar{\boldsymbol{S}} &= 0 \\ \bar{\boldsymbol{J}}_{\text{ef}} &= 0 \end{aligned} \tag{8-13}$$

The only coupling is caused by the dependency of the rotation inertia tensor $\bar{\boldsymbol{J}}_{\text{ee}}$ from the body deformation.

The equations presented in Sections 3.4-3.6 can then be simplified.

The inertia integrals associate with the mass-center reference frame and the new set of shape functions are:

$$\begin{aligned} \vec{\boldsymbol{J}}_1 &= 0 \\ \hat{\bar{\boldsymbol{S}}} &= 0 \\ \hat{\bar{\boldsymbol{S}}}_{kl} &= \boldsymbol{V}^T \bar{\boldsymbol{S}}_{kl} \boldsymbol{V} \\ \hat{\bar{\boldsymbol{J}}}_{kl} &= \bar{\boldsymbol{J}}_{kl} \boldsymbol{V} \end{aligned} \tag{8-14}$$

The Newton-Euler equation reduces to:

$$\begin{bmatrix} \boldsymbol{M}_{\text{RR}} & 0 & 0 \\ & \bar{\boldsymbol{J}}_{\text{ee}} & 0 \\ \multicolumn{2}{c}{symmetric} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \ddot{\vec{\boldsymbol{R}}}^{(\text{b})}_{\text{cm/g}} \\ \vec{\boldsymbol{\alpha}}^{(\text{b})} \\ \ddot{\vec{\boldsymbol{x}}}_{\text{f}} \end{bmatrix} = \begin{bmatrix} \vec{\boldsymbol{Q}}^{(b)}_{\text{R}} \\ \vec{\boldsymbol{Q}}^{(b)}_{\alpha} \\ \vec{\boldsymbol{Q}}_{\text{f}} \end{bmatrix} \tag{8-15}$$

Additionally the expressions for the quadratic-velocity vector may be simplified taking into account the $\bar{\boldsymbol{J}}_{\text{ef}}$ block of the new mass matrix is zero.

It is important to understand that the orthonormalization does not change the properties of the original discretization, which is done when the set of shape functions is chosen. Any deformation shape that can be represented with the original shape function $\boldsymbol{S}$ can be represented with the orthonormalized set $\hat{\boldsymbol{S}}$ as well. The result of orthonormalization followed by removal of rigid body related eigen-vectors is just enforcement of the mean axis condition. One can say that orthonormalization is used to exclude the part of $\boldsymbol{S}$ that can be descibed with rigid body translation and rotation.

Chapter 9

Modeling Ideal Connections with Control Points

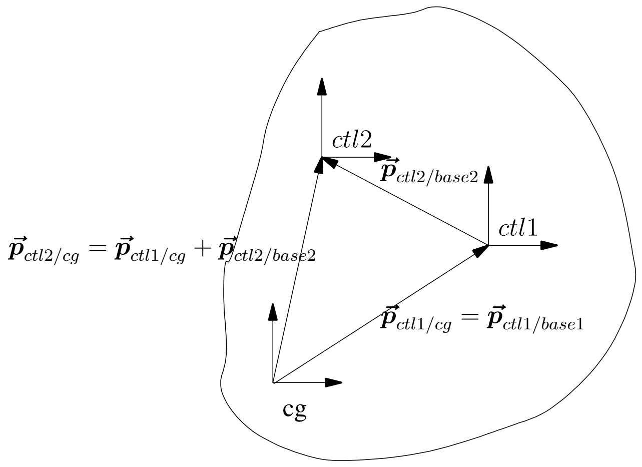
9.1 Control Points and Flexible Bodies

The BEAST system design uses continuous definitions of the complete surface geometry. No explicit interface nodes are predefined. This is in contrast with most other tools where some joint attachment points are included in the basic model design or FEM-based tools where joints and force elements can be connected only to the finite element mesh nodes.

However ideal joints and force elements are often used in multibody modeling. The older version the rigid-body subsystem of BEAST allowed such forces to be introduced in the geometrical centres of the bodies or in a single control point between the bodies. The user of the system had to translate the combination of forces and moments acting between two bodies to the bodies geometry centres or the single control point. Since in the rigid body dynamics the force and moment couple applied at one point of a body can easily be translated to any other point (including the geometry centre of the body), this feature provided enough modeling freedom until the introduction of flexible components.

For a flexible body the relation between the generalized force tensor and applied force is more complicated (see Section 3.7). That is why we cannot require the user to translate the force to the body's geometry centre or any specific coordinate system by hand. Instead, there must be a method to specify any material particle inside the body volume as the point where the force vector is applied. The control point design discussed in this section resolves this problem.

Control points provide a way to specify a particle's position, orientation and speed inside a body (relative body geometrical centre), or in the model reference frame (called cB coordinate system in the BEAST framework). Thus a control point can be seen as an information data structure consisting of a position vector, orthogonal rotation matrix as well as linear and angular velocity vectors of the particle. These points can be used to study the elastic deformation of the specified material particle,



$ctl2$
$\vec{p}_{ctl2/base2}$
$ctl1$
$\vec{p}_{ctl2/cg} = \vec{p}_{ctl1/cg} + \vec{p}_{ctl2/base2}$
$\vec{p}_{ctl1/cg} = \vec{p}_{ctl1/base1}$
cg

Figure 9-1: *Control points.*

and to apply forces at the specified locations (not only at the geometry centre).

First, the *parent system* of a control point must be selected. This is the coordinate system where all the output components of the control point are calculated. Generally, it can be any abstract coordinate frame. The two most common cases are floating model reference frame and a body geometry centre.

For each control point the following relations are defined:

- $\vec{p}_{ctl/parent} = \vec{p}_{base/parent} + \vec{p}_{ctl/base}$ The position of the control point which is normally written in the output file. Every component of this relation should be understood as a combinations of position vector, rotation matrix and speed vectors. The plus operator represents a complete coordinate transformation (translation, rotation and velocity). The $\vec{p}_{ctl/parent}$ tensor provides a common interface to all the different types of control points described below. Parent coordinate system (model reference frame or geometry centre of a body) gives the coordinate system where the output $\vec{p}^{(parent)}_{ctl/parent}$ is calculated.

- $\vec{p}_{base/parent}$ defines the beginning (base coordinate system) for the $\vec{p}_{ctl/base}$ tensor introduced by this control point. $\vec{p}_{base/parent}$ tensor itself can be a control point making it possible to build 'chains' of control points. Such chains can be efficiently used to specify complex motion as shown in the example section of this chapter.

- $\vec{p}_{ctl/base}$ is the relative motion defined by the control point. The specific of this vector calculation determines the type of the control point (see below).

The example Figure 9-1 shows two control points defined in a body. Both control points have body geometry centre as the parent coordinate system. The first control point $ctl1$ is defined relative to the body coordinate centre and so the output for it is equivalent to the $\vec{\boldsymbol{p}}_{ctl1/base1}$. The second control point $ctl2$ is defined relative to the first control point thus forming a 'chains' of two points.

Currently three types of control points are available in BEAST. They are:

Fixed control point. The $\vec{\boldsymbol{p}}_{ctl/base}$ tensor for this kind of control point is constant and so the velocity components are zero. It is semantically identical to a time dependent control point (see below) with all the time dependent parts set to zero. Compared to the time dependent control point the fixed control point is faster to process, requires less input and memory.

Time dependent control point. The components of the $\vec{\boldsymbol{p}}_{ctl/base}$ tensor are functions of time. Note that it is impossible to specify both velocity and position as independent time functions and so user must either prescribe the position - letting the program to calculate velocity or provide the velocity and let the program compute the trajectory.

Flexible control point. This type of control points can be used only within flexible bodies. They provide a mean to examine the deflection of a specific material particle. Unlike the two other types of control points the flexible control point does not have any input and it must be the last control point in a chain. That is it is not allowed to define other control points having the flexible control point as a base. The $\vec{\boldsymbol{p}}_{ctl/base}$ vector is computed by BEAST and gives the deflection vector and deflecting speed and the output $\vec{\boldsymbol{p}}_{ctl/parent}$ tensor specifies the deflected position of the material point that had $\vec{\boldsymbol{p}}_{base/parent}$ coordinates in the undeformed body. In the current implementation we do not calculate the orientation of the deflected point. Flexible control point define extra output tensors: $strain$ and $stress$. They are the second Piola-Kirchhoff stress and Green-Lagrange strain tensors discussed earlier in Section 3.7.1.

9.2 Boundary and Loading Conditions

Stiffness-dampers, force-moment, and simple parametric bearings can be used to set the boundary and loading conditions for flexible bodies. The required input is identical to the one needed to set the conditions for rigid bodies. Special attention however must be paid to the type of control point where the force is applied. The reason for that is the different deflections caused by the same force applied to different parts of a flexible body. If some loading or boundary condition should cause deflections then it needs to be applied at a flexible control point in the body. If some specific loading should only influence rigid body motion then it should be applied at a time dependent or fixed control point.

9.3 An Example of Control Points Usage

Figure 9-2 shows an example model that uses control points to apply complicated loading conditions on a flexible ring. The purpose of this model is to force satellite-like motion of the ring and deform it elliptically in the same time.

Eight control points are used in the model. Four of the points have the model (reference frame 'cB') as parent and four others are defined in the ring. In both cases chains of control points are used. Parent is indicated by the prefix in a control point name.

The $cB.ctl1$ point is a time dependent control point that defines a reference frame with constant translation relative to the cB system and rotation with constant angular velocity $\vec{\omega}_1$. Thus all the control points that follow $cB.ctl1$ in the chain will rotate around this control point position. The next control point, $cB.ctl2$, has $cB.ctl1$ as the base system. It is also a time dependent control point with constant translation and constant velocity rotation given by $\vec{\omega}_2$. Both $cB.ctl3$ and $cB.ctl4$ are fixed control points having base system in $cB.ctl2$. Since they have the same base system an imaginary line segment between the two points will not change the length but will follow the satellite-like motion as specified by the velocities $\vec{\omega}_1$ and $\vec{\omega}_2$. Having arranged this motion it is now necessary to relate the control points to the material particles of the ring.

The structure of the control points' chains used inside the ring is simple. Just two fixed control points ($bR.ctl1$ and $bR.ctl2$) and two flexible control points ($bR.ctlf1$ and $bR.ctlf2$) are used. The fixed control points define the positions of the material particles inside the ring in the undeformed state. The flexible control points that are based on the respective fixed control points follow the deformation of the ring.

It is shown in Figure 9-2 that flexible control points are connected to the control points in cB system. This connection can be of different nature. Most often a combination of springs and dampers is used to prescribe a body motion with certain stiffness and damping coefficients.

The discussed example shows how a complicated loading condition can be described with the control points. It must be noted, however, that some experience is required from the user in order to understand all the features of the control points and apply them correctly and efficiently.

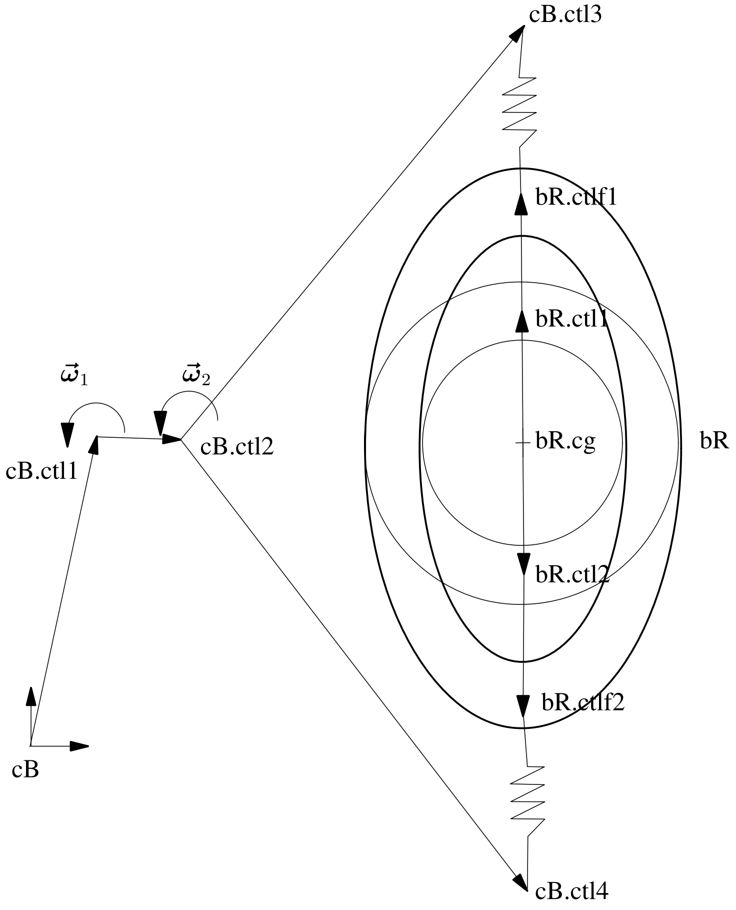


Figure 9-2: *Specifying loading conditions with control points. Object-oriented notation is used. The model (reference frame cB) and body bR are the parent objects and the control points and geometry centre 'cg' are the child objects.*

Chapter 10

Verification

In order to verify the theory described in this thesis some simple test cases with known analytical solutions had to be selected. We have chosen to verify the solutions for the following cases:

1. Static deflection of a simply supported beam.
2. First eigenfrequency of a free beam.
3. Thin ring under symmetric point force load.
4. Eigenfrequencies of a free ring.

10.1 Long Beams

Distributed gravity load mg acts on a simply supported beam (see Figure 10-1). The value of the maximum deflection of the beam centre in equilibrium can be computed analytically [45]. For our case (both ends simply supported, distributed load) the maximum deflection is given:

$$y_{max} = \frac{-m\ g\ L^3}{48E\ I}, \tag{10-1}$$

where m is the mass of the beam, g - gravity constant, L - length of the beam, E - Youngs' modulus, and I - moment of area, which for a round cross-section with

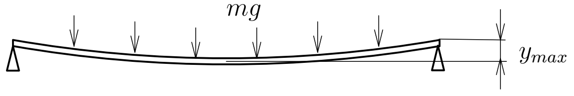


Figure 10-1: *Distributed gravity load acting on a simply supported beam.*

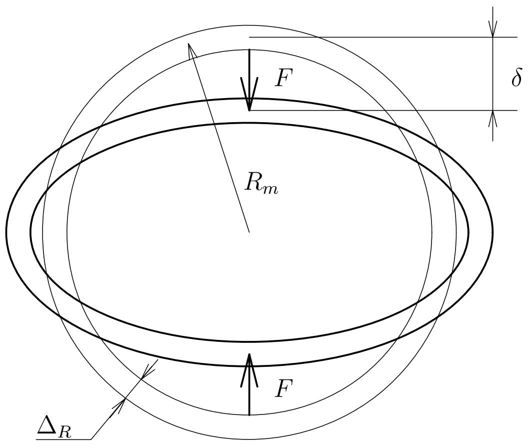


Figure 10-2: *Symmetric constant forced acting on a thin ring.*

radius r is given by

$$I = \frac{\pi r^4}{4} \tag{10-2}$$

The error in the model can be measured in percents as

$$\epsilon = \frac{y_{model} - y_{theory}}{y_{theory}} * 100\% \tag{10-3}$$

We were interested to get error below one percent. To get the correct results for the complete 3D model with volume integration we had to use second order polynomial in radial direction, two-waves in circumferential direction and the 4-th order polynomial for the axial direction.

Note that the same test cases were verified for the beams with the rectangular cross-section where Chebyshev polynomials were used all three coordinate directions.

10.2 A Thin Ring

Two symmetric constant forces act opposite to each other in the plane of the ring. The forces are applied at the middle of the cross-section one from the top of the ring downward and other from the bottom of the ring upward (see Figure 10-2). Theoretical solution for the maximum radial deflection value exists for this load case and is given by:

$$\delta = 0.0744 \frac{F R_m^3}{E \cdot w \cdot \Delta_R^3/12} \tag{10-4}$$

where F - stands for applied force in radial direction, R_m - medium radius, E - Young modulus, w - width of the ring cross-section in axial direction, Δ_R - thickness of the cross-section.

Free body eigenfrequencies in the plane of the ring are given by:

$$f = \sqrt{\frac{E \cdot (1 + j^2)}{\rho \cdot R_m^2}} \tag{10-5}$$

where j is used to compute j-th eigenfrequency.

For the verification we used model with second order shapes in radial and axial directions and Fourier series up to 8-waves on circumference. The error in static solution became 0.3%, errors in eigenfrequencies up to the 8th in the plane of the ring were also below 1%. The total number of flexible states used in the model was 460.

Due to the large number of states required for verification at higher eigenfrequencies we had to reduce the order of polynomials used in our model. We used only linear term in radial direction keeping second order in axial direction and extended Fourier series up to the 30 waves. The total number of elastic coordinates in the model become 1189. The model resulted in a higher error in static solution (8.6%) but we were able to calculate up to the 30th eigenfrequecy. The relative error in the highest eigenfrequecy was 2.5%.

We have to mention that the solution close to the theoretical one for the higher eigenfrequencies could be achieved only for very thin rings (cross section width of about 1% of the ring diameter). We believe that the reason for that is the simplification used in theoretical approach for thin rings that only allows cross-section rotation. Such simplifications becomes inaccurate for the higher eigenshapes in thicker rings and consequently leads to deviations in eigenfrequecies solution.

Chapter 11

Simulation System Design

In this chapter the software design of the simulation system is discussed. Note that the class and object diagrams presented in this chapter follow the standard UML (Unified Modeling Language) notations [24].

11.1 System Overview

In this section a general overview of the system is provided. This overview is quite general and is applicable to most multibody systems, not only to the BEAST system. The concepts described in this section do not change when transferring from rigid to flexible modeling.

The BEAST system uses object-oriented design and programming concepts. The ideas behind the object-oriented implementation of a multibody simulation system were developed in a close collaboration with the ObjectMath project [38].

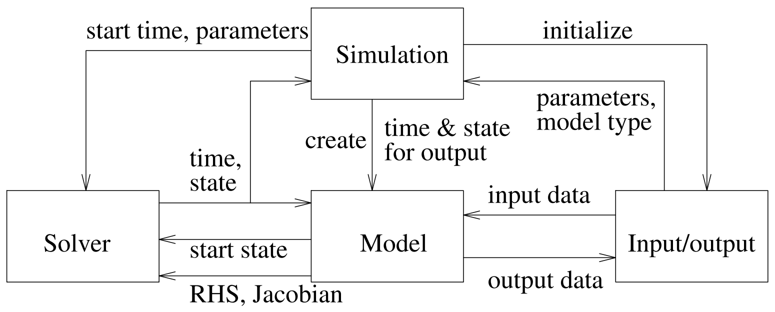


Figure 11-1: *Collaboration of components in the simulation system executable. Arrows denote information flow.*

The main components of the complete simulation system are shown in Figure 11-1 and described below:

Simulation This component manages the dynamic simulation process. During the setup phase the class initializes the input/output classes, reads in the model type and general simulation parameters (such as tolerance, start and end time, etc). Then the numerical solver is setup and the specific model object is created and initialized. As the simulation runs the simulation class monitors the progress, issuing the commands to calculate the output for some time steps and to terminate the simulation when the user specified end time is reached or an error occurs.

Solver This is the numerical ODE solver component of the system which is accessed through a set of wrapper routines providing interface between the standard CVODE numerical solver and the system.

Model This is an object created by the Simulation object. This object essentially defines the multibody system to be simulated. From the mathematical point of view, its responsibility is just to evaluate the time derivatives of the state variables during ODE's RHS (right-hand side) evaluation and Jacobian matrix for the Newton iterations inside the solver. Note that there is a special mode of RHS calculation when the data is not returned to the solver but written on the disk via the input/output classes. The internal structure of a model object will be discussed later in more detail.

Input/output These objects are responsible for the file I/O interface. They provide the functionality for reading the simulation parameters and model input data and for writing the simulation output data.

The differences in the rigid and flexible modeling conceptually affect the Model object only. However, in a real implementation some details must be checked when going into flexible modeling. It must be ensured that none of the objects rely on the constant number of the state variables inside every body in the model. Such an assumption could be used in a multibody simulation tool with rigid bodies only for the optimal implementation of different service routines transferring data between the Model object and other components of the system. Flexible bodies have a variable number of state variables and therefore can not be handled by such service methods.

11.2 Rigid Body Model Classes

The class diagram of the conceptually most important classes in a multibody model is shown in Figure 11-2. The explanation of the responsibilities and purpose of each class follows.

The Model class. The general `Model` class is inherited by all other model classes. It is an abstract class providing the interface to any model from other parts of the system. The class implements necessary standard operations for data exchange with the

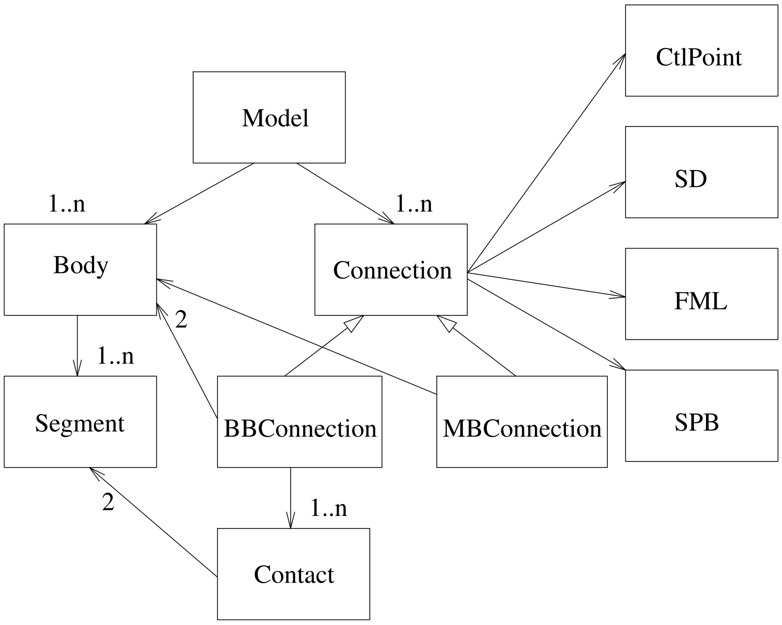


Figure 11-2: *Rigid Body Model Classes.*

numerical solver and with the input/output classes. The child classes of this abstract class are specific models containing specific sets of bodies and specific connections between the bodies.

The Body class. This class has attributes, such as inertia characteristics and material properties, describing those properties of a physical body that are not dependent on the interactions with other bodies. The methods of this class implement the Newton-Euler equations.

The `Body` class is also an abstract class. Specific body classes contain particular kinds of segments. For instance, a Ring class contains several rotational segments.

The Segment class. Segment is another important abstraction in BEAST. A segment represent a continuous surface that can get in contact with other surfaces. The surfaces are normally defined in parametric space (u, v). There are functions evaluating the geometric point coordinates in the body coordinate space and functions that for any point in space can find the (u, v) parameters of the nearest point on the surface which is an important feature for contact forces calculations. The geometry of every body in BEAST is defined by a set of segments.

The Connection class This class is a container class that encapsulates all the objects acting between two bodies or between a body and an abstract coordinate system defined in the model space. The purpose of such a collection is to accumulate all the forces acting between the two bodies or between the body and the external coordinate system. The two child classes represent the two cases that we just defined. The `BBConnection` class represents a connection between two bodies and `MBConnection` class represents a connection from a coordinate system in model. Only `BBConnection` class will be discussed later in the text, since `MBConnection` can be seen as a simplification of the `BBConnection` class (where there are no contacts and motion of one of the bodies is known) .

The CtlPoint class. This class represent a single control point or coordinate system as discussed in Chapter 9. The position, orientation and speed properties of the point are provided by the user. The single control point notion provided enough modeling power for the case of rigid body modeling.

The SD, FML and SPB classes. These classes represent simple force element: stiffness and damping, force and moment loading, as well as simple-parametric bearing respectively. The SPB is essentially a non-linear kind of spring that is parameterized in the typical bearing terms, such as clearance and radial stiffness.

11.3 Flexible System Design

11.3.1 Limitations of the Rigid Body Model Design

The transition to the flexible body model revealed some limitations of the original system design. First, the single control point in a connection between two bodies is not sufficient as we discussed earlier in Chapter 9. Other changes concern the `Body` class design that should now implement the equations for the flexible model case and work with the shape functions. Our design solutions to these problems are presented in the rest of this section.

11.3.2 Implementation of the Control Point Architecture

Figure 11-3 shows the class diagram that illustrates the more general design suitable for the flexible case. Comparing to the original design presented in Section 11.2 the most important change is the transfer of the `ControlPoint` class from the `Connection` class to the `Body` class and generalization of the simple forces into the `PointTie` class.

The `PointTie` class generalizes the concept of a force element acting between two points. The general characteristic of such a force element is its locality. That

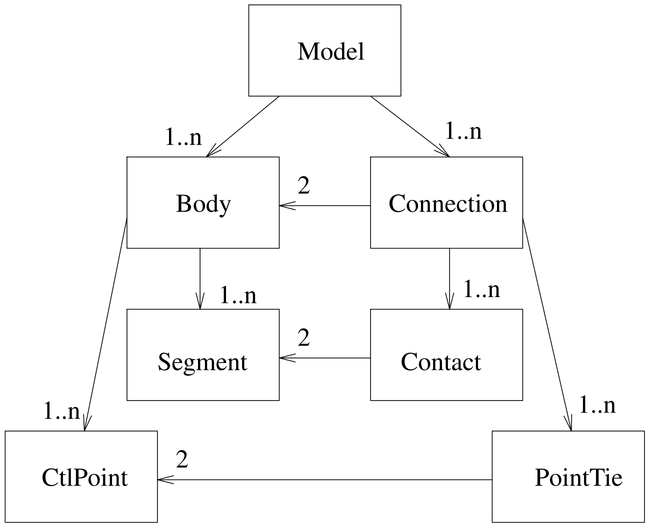


Figure 11-3: *Flexible Body Model Classes.*

is, the resulting force and moment at a given time instant depend only on the relative position, orientation and speeds of the two points that the `PointTie` connects. The nature of the points and the state of the bodies are not relevant for such a force element. The typical example of such force elements are springs and dampers (both translational and rotational). The `PointTie` class is therefore an abstract class with a single abstract method that evaluates the force and moment acting at the connected points for the specified relative motion of the points and time. The generic class is responsible for the necessary coordinate transformations and numerical differentiation of the forces for the Jacobian calculations.

In the context of the BEAST tool, the PointTie abstraction was realized in the three child classes that correspond to the SD, FML and SPB classes discussed earlier for the rigid body system in Section 11.2:

- `SDPointTie` encapsulate the force and moment calculation for general springs and dampers connecting the specified points.

- `FMPointTie` generates forces and moments that are pure functions of time and do not depend on the relative motion of the points.

- `SPBPointTie` (Simple Parametric Bearing tie) normally used to connect centres of two bodies that are joined by a bearing. It is essentially a non-linear kind of spring that is parameterized in the typical bearing terms, such as clearance and radial stiffness.

The obvious benefit of the generalization of the simple forces is the ability to add new kind of force elements in an easier way without concern for the rest of the system. For instance, a new `JournalStaticBRGTie` was recently introduced into the system. The new tie is similar to `SPBPointTie`, but has a different set of parameters.

11.3.3 Object-Oriented View of Shape Functions

During a simulation the shape function conceptually only need to provide the following output:

- The shape matrix $\boldsymbol{S}$ for any given point in the body coordinate system according to Equation 3-2.
- The strain operator $\boldsymbol{DS}$ defined in Section 3.7.1.

Therefore the shape function can be designed as an abstract interface class with only two methods as described above.

Specific shape functions can actually work in different coordinate systems (i.e., cylindrical or Cartesian) and use different kind of evaluations. For the purposes of this work, two specific classes of the shape functions were defined:

CylShpFunc is the cylindrical shape function implementing the series defined in Equation 7-1 and the strain operator calculations defined by Equation 3-40.

CartShpFunc is the Cartesian shape function implementing the series in Equation 7-2 and the differentiation operator according to Equation 3-39.

11.3.4 Flexible Body Class Design

Figure 11-4 shows the class diagram illustrating the design of the body class in the flexible modeling part of the system.

The most general FlexBody class implements all the time dependent equations described in Chapter 3 and the analysis on the single flexible body as discussed in Chapter 4. It cannot, however, be used directly since it relies on the child classes for the initial calculation of the inertia integrals as well as stiffness and damping matrices. This class actually can be used in the future to interface the shapes and integrals generated by other systems, e.g., via the data structures defined in the FEMMBS tool [40].

Note that once the integrals are available all the analysis can be done on the abstract FlexBody class without accessing the child objects.

The IntFlexBody class defines the initialization step for the FlexBody class with the volume integration procedures based on weighed integration points. Many numerical integration schemes, including the scheme described in Section 7.4, generate

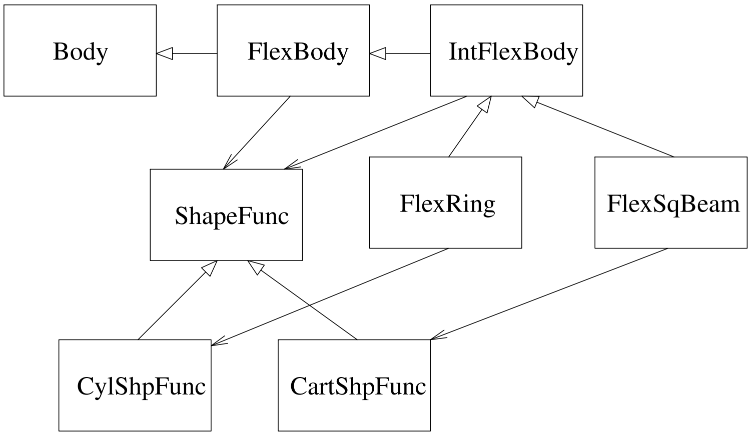


Figure 11-4: *Flexible Body Classes.*

a set of integration points with weights. The initialization procedure of the IntFlex-Body calculates the integrals defined by Equations 3-19 to 3-23 using such sets of integration points.

The descendants of the IntFlexBody class are specific body classes where geometry is defined by a set of segments. The responsibility of these classes is to produce the set of integration points required by the IntFlexBody class. Note that only two child classes are shown in the class diagram in Figure 11-4 just to provide an example of the most simple kinds of bodies. More classes are needed in a real system to accommodate for the variety of different geometries.

11.4 A Simple Model Example

This section gives a simple example of a specific model consisting of two rings as depicted in Figure 11-5. Even though the model is very small all the important elements of larger, more complicated models are present.

The model is called TwoRing. Figure 11-6 presents an object diagram for this model.

The TwoRing model consists of two bodies: the outer ring bER and the inner ring bIR. The outer ring is guided relative to the cB system by use of the stiffness and damping forces cBbER‘SD between the control points cB‘ctl and bER‘ctl.

The rings have a potential contact between the outer segment of the inner ring bIR‘s1 and the inner segment of the outer ring bER‘s3.

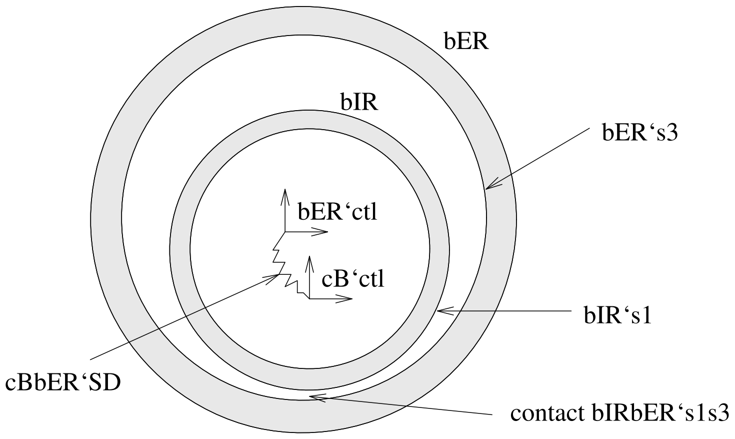


Figure 11-5: *A Simple Multibody Model.*

11.5 Solution Procedure

As was mentioned in the beginning of this chapter the `Model` object is responsible for the evaluation of the RHS vector and Jacobian matrix during the solution of the system of ODE describing the motion in the multibody system. This section discusses the mapping of the dynamic equations of motion onto the software classes and the computations flow between the classes. The differences between the models with rigid bodies only, and the newer models with flexible bodies, are specially addressed.

11.5.1 RHS Evaluation

The evaluation starts by transferring the state variable values from the solver to the objects realizing the `Body` class. At this point the complete state vector as seen by the CVODE is decomposed into variables representing the motion: position, orientation and velocity.

Having the motion state data the `Body` objects perform the calculations independent of the external loading. That includes the terms caused by centrifugal forces and coordinate system transformations. In the newer system design the motion of all the ControlPoint objects can also be calculated at this stage.

Th next step is done by the objects of the `Connection` class. Each `Connection` object calculates the relative motion between the two bodies the object connects. Note that in case of rigid bodies all the forces from the interaction between the two bodies only depend on the calculated relative motion. Absolute position and velocity is not important for the interaction force calculations. However,

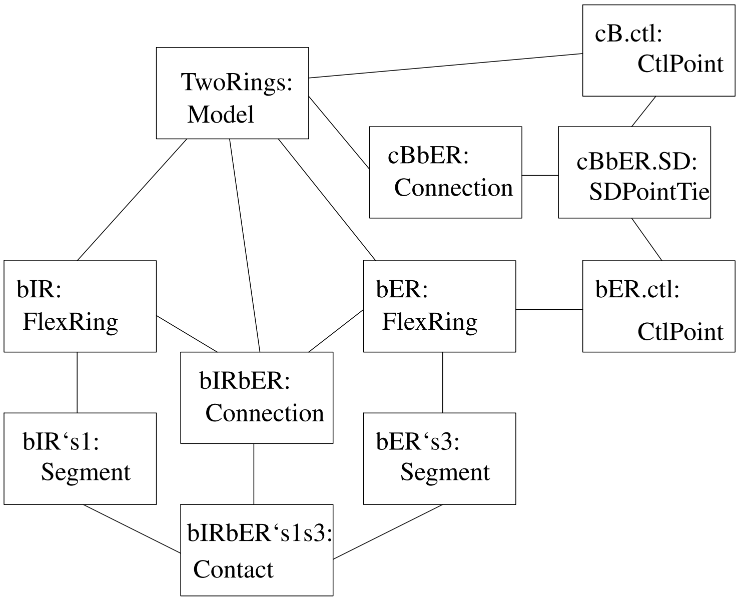


Figure 11-6: *Object Diagram for a Simple Specific Model.*

this is not the case for the flexible body interaction. The resulting force for a flexible body always depends on the elastic state variables. Therefore it can be said that the relative motion between to flexible bodies can be described by the relative motion of the floating reference frames and the elastic state variables of the both bodies.

For the calculated relative motion variables the force elements in the Connection object and corresponding Contact objects evaluate the forces and moments acting between the bodies. These forces and moments are transformed to the local coordinate system of every body. Note that transformation to the local coordinate system for a flexible body includes the evaluation of the elastic part of the generalized force vector as discussed in Section 3.7. All the (generalized) forces acting between two bodies are accumulated by the Connection object before they are passed to the Body objects.

Body objects receive the generalized forces from the Connection object, accumulate them, add the contribution from the centrifugal forces calculated earlier, and use the Newton-Euler equation to calculate the acceleration tensor.

Finally the vector of derivatives is assembled by the Body objects and send to the numerical solver. The RHS evaluation is completed.

In the case of output RHS evaluation some extra output variables (such as power dissipation in a contact) may be computed at different stages of the RHS evaluation algorithm. These variables are then written to the simulation output file.

11.5.2 Jacobian Evaluation

The Jacobian calculation follows the procedures described in Section 3.8. The use of chain differentiation rules leads to the clear separation of several stages in the evaluation where at each stage a different partial derivative used in Equations 3-79 and 3-80 are calculated.

The instances of the `Body` class are responsible for evaluation of all the partial derivatives which involve the particular body states. The analytical part of the Jacobian is also evaluated by the objects of Body class. The following matrices are calculated in a body i: $\frac{\partial \dot{\vec{\boldsymbol{v}}}_i}{\partial \vec{\boldsymbol{Q}}^{(i)}_{i/j}}$ and $\frac{\partial \vec{\boldsymbol{Q}}^{(i)}_{i/j}}{\partial \vec{\boldsymbol{F}}^{(i)}_{i/j}}$.

The instances of the `Connection` class are responsible for the partial differentiation of the forces acting between the bodies with respect to each of the two involved bodies state variables. These objects are also calculating the mixed sub-Jacobians. That is, if an object is a `Connection` between bodies i and j then it is responsible for the evaluation of the following matrices: $\frac{\partial \dot{\vec{\boldsymbol{v}}}_i}{\partial \vec{\boldsymbol{z}}_j}$, $\frac{\partial \dot{\vec{\boldsymbol{v}}}_j}{\partial \vec{\boldsymbol{z}}_i}$, $\frac{\partial \vec{\boldsymbol{Q}}^{(i)}_{i/j}}{\partial \vec{\boldsymbol{z}}_j}$ and $\frac{\partial \vec{\boldsymbol{Q}}^{(j)}_{j/i}}{\partial \vec{\boldsymbol{z}}_i}$.

The numerical differentiation algorithm with respect to the relative motion variables is implemented in the `Connection` class and calculates the most computationally expensive derivatives $\frac{\partial \vec{\boldsymbol{F}}}{\partial \vec{\boldsymbol{\Delta}}^{(i)}_{j/i}}$.

Recall that one of the main differences of the flexible body model compared to the rigid body model is the dependency of the applied force or moment to generalized force conversion from the flexible body states leading to a more complicated partial derivatives evaluation. To resolve this problem the the `Body` class in the flexible case collaborates with the Connection class to implement the algorithm for the evaluation of $\frac{\partial \vec{\boldsymbol{Q}}^{(i)}_{i/j}}{\vec{\boldsymbol{z}}_{i,f}}$ described at the end of Section 3.8.

There are also some technical implementation details that need to be mentioned. The rigid body model uses the same number of states for every body in the model whereas the number of flexible states varies for different bodies. For the partial derivatives matrices it leads to the necessity to deal with arbitrary size matrices. The software developer should therefore carefully consider memory allocation and data transfer issues.

Chapter 12

Dynamic Simulation of Grinding

12.1 Overview

The results presented in this chapter were earlier presented in the articles 'Dynamic Simulation of Grinding with Flexibility and Material Removal' [21] and 'Modeling of Flexible Rings for Grinding Simulation' [22]. The approach presented in this thesis was used to model the workpiece flexibility. In fact, this was the first industrial application of the developed technique.

12.2 Need for Grinding Simulation

The grinding process is important in many high precision manufacture industries. For SKF as a manufacturer of high precision rolling bearings it is about a third of the total manufacturing cost. Thus, continuous development and research in the grinding process is needed. The purpose is quality improvement and cost reduction.

For several reasons it is suitable to study the grinding process with simulations. First, it is very hard or even impossible to measure some characteristics of such a dynamic process as grinding. Second, most machines cannot be dedicated to research activities for sufficiently long periods of time.

Simulation of the grinding process requires simultaneous modeling of several phenomena including:

- Multibody system dynamics. Since the grinding machine is a complex dynamic system, a general MBS modeling framework is necessary, in order to perform the simulation. Detailed and accurate geometric description is needed to model all the local variation in geometry of un-ground rings, as well as the geometry changes during grinding.

- Tribological contacts with material removal. The dynamic behaviour of the grinding machine is primarily determined by the contacts between the workpiece and parts of the machine (various supports and the grinding wheel).

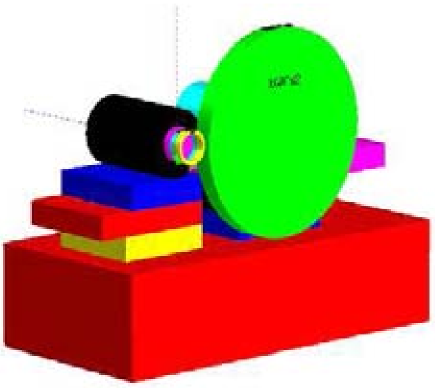


Figure 12-1: *A model of the grinding machine including most of the parts of the machine.*

Hence detailed contact modeling was necessary. The grinding contact needs special treatment since it results in material removal from the workpiece, thus changing the original geometry of the body.

- Elastic deformation. Large forces are applied to the workpiece during grinding and the structural elastic deformation of thin rings is in the same order of magnitude as instantaneous material removal. Hence in such cases the structural deformation must be included.
- Distortion. Initial stress distribution in the surface layers in combination with uneven material removal will cause distortion. The stress distribution is created by material phase changes in the hardening process.

The examples presented in this chapter are concerned with microcentric grinding. Microcentric ring grinding is one of the most common grinding processes in SKF and so the investigations were started with this process. Microcentric arrangement means that the workpiece (bearing ring) is clamped to the drive plane, e.g., with a magnetic force, and driven by friction forces. The drive plane centre has a small offset distance (microcentric) relative to the ring. There are at least two shoes that support the ring and the microcentric forces are holding the ring in position.

12.3 Grinding Machine Model

A grinding machine is a complex and large system consisting of many components. One example of a complete model of a grinding machine is shown in Figure 12-1. However, some parts of the machine can be reduced to a fever number of important bodies for which the movements also can be measured and analysed. The models are used in complete grinding cycle simulations, so efficient computation is essential.

In our example an external ring grinding machine has been modeled. This machine has few major eigenfrequencies and so a limited number of bodies can accurately describe the dynamic behaviour. A measurement of the dynamic response of the grinding machine can identify these dominating eigenfrequencies.

The shoe support arrangement is an essential part of the machine. Detailed definition of the geometry of this arrangement is crucial for the correct modeling of machine dynamics. Even a small deviation can result in malfunction of the arrangement with microcentric forces pushing the simulated ring out of the machine. Shoe supports are also responsible for some filtering effects that minimise the effect of certain imperfections of a ring.

It is important to mention that different complexity levels of the model should be used depending on the purpose of a simulation case. If only the effect of geometrical filtering is of interest it is enough to use pure kinematics. If instead the machine stability and chatter are the only interesting phenomena, a pure dynamic analysis is satisfactory. But if the purpose is to simulate the total grinding cycle and the effects of machine dynamics on the final output shape, a complete model such as the one suggested here is necessary.

Further, it is of importance to include the flexibility of the bodies in order to get accurate results. Flexibility and stress analysis are also very important if effects such as stress release of internal stresses during grinding are of interest.

One significant difference of our models, compared to what is common in other grinding simulations [4], is the use of fully three-dimensional structures.

12.4 Modeling Tribological Contacts with Material Removal

In this section we will outline the most important phenomena that can be observed in a grinding contact. Consider a rotating solid grinding wheel and a rotating ring. Both of them have constant but different rotational speeds. There is a rich supply of cutting fluid (mixture of oil and water) at the contact inlet. Slowly (i.e., without impact) we move the two rings toward each other. The contact is in a different regime depending on the gap between the rings:

- The fluid fills the gap. It is the hydrodynamic regime and a pressure is built up which gives a rolling resistance moment and normal force. However, they are small due to the low viscosity and large gap.

- With smaller gap the fluid pressure causes some elastic deformation and the contact is going into the beginning of the elastohydrodynamic (EHL) regime. Small material damping due to rolling resistance also shows up. However, in the case with a grinding wheel that have quite a rough surface we will quite early get asperity contacts. The load will be shared between the asperities and the fluid film. The asperity contacts also result in a fast growth of friction forces.

- The ”gap” becomes negative, i.e., there is an intersection. The elastic deformation is significant and we are well into the EHL regime. However, due to the low viscosity the film is very thin and nearly all load is carried by the asperities. We have large friction forces.

- At a certain load the grains of the grinding wheel have sufficient pressure to start to cut. The cutting will result in a certain amount of the material removed and, consequently, surface geometry changes. Three different mechanisms are active: some viscous rolling resistance with very thin film building up, elastic deformation, and material removal due to cutting. Each of these phenomena is a non-linear function of load. Besides, the film build up and material removal are also non-linear functions of speed.

- With even more negative gap (larger load), the material removal rate is normally increasing faster than the other parameters, e.g., local deformation and film thickness.

In a more general dynamic situation the velocities normal to the contact surface must also be considered. They will result in additional pressure/forces, e.g., viscous damping, material damping, squeeze film build up, cavitation, etc.

All the physical phenomena mentioned above are modeled and included in the tribological used contact model.

In the literature one can find two directions regarding grinding contact modeling. One direction takes a very detailed look at the cutting of individual grains [44]. However, this is not suitable for dynamic simulations. Another trend is to have quite simple macroscopic models, in many cases linear, for the material removal [27, 36]. Our approach is on the macroscopic level. However, we include some important non-linear behaviour in the model to have good correlation to our experimental data. We also include the EHL part.

12.5 Grinding simulation results

The output from a grinding simulation can be any type of forces and movements. However, it is the shape of the ground piece that is one of the most important outputs. The most typical problem encountered in the final product is waviness of the ground surface. With our simulation tool the shape of the workpiece can be animated for the entire grinding cycle, and it is possible to see how the shape develops during the process. Grinding simulation has now been used for a wide range of investigations and some example studies are presented below.

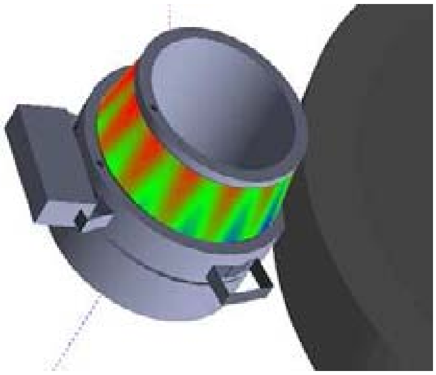


Figure 12-2: *Microcentric shoe support system. The material removal height is shown by color.*

Chatter

Chatter is a regenerative wave creation on the rings. It interacts with the dynamics and eigenfrequencies of the machine. A grinding machine that is not stable enough will produce non-circular, wavy, rings. In Figure 12-2 a snap-shot of a ground ring is shown. The ring and part of the grinding wheel, as well as the shoe support system, can be seen. The colours on the ring surface show the waviness of the ring at this time instance. Another example is shown in Figure 12-3, were the form of the ring and a FFT analysis of the waviness of the ring is shown. In the beginning of the grinding cycle small waves starts to grow on the ring. As the grinding continues these waves becomes bigger and bigger in a regenerative way. It is noticed that the characteristic wave number is 23. This wave number could be correlated to the first (lowest) eigenfrequency of the machine.

The simulations also showed that just an increase in the stiffness or the damping of the machine to a certain level, would make the machine stable. Such a machine will produce much better, rounder rings. This is an example of how this simulation tool can be used to investigate and improve grinding machines with instability problems.

Unbalance

It is very hard to get perfect balance of the grinding wheel, even if the grinding machines have automatic balancing units. If there is a small unbalance this will immediately have an effect on the ring. If the unbalance is too large the waves will become big and the ring will not be approved. These effects have been investigated with this simulation tool. It is not only the unbalance itself that is dangerous to the grinding. The unbalance will create an excitation frequency into the system depend-

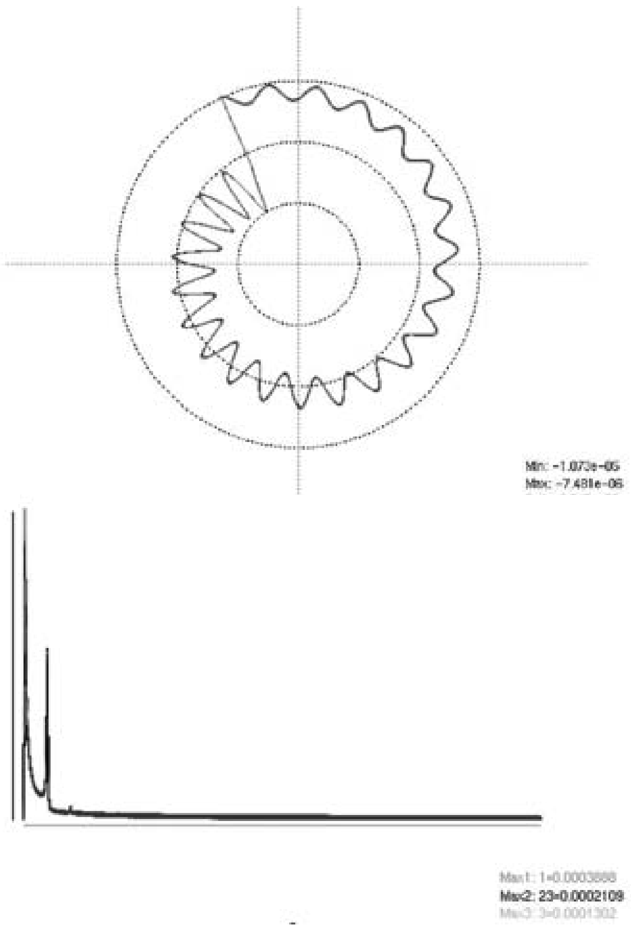


Figure 12-3: *The shape of the ring and FFT analysis of ring waviness.*

ing on the speed of the grinding wheel, and if this frequency lies near some of the eigenfrequencies of the machine, large vibration problems will arise. Thus, it is very important to avoid an unbalance that is interacting with other wave creating effects.

Geometrical Filtering

The grinding machine that is exemplified here uses a microcentric support system. For such a system there will be a filtering effect, which means that some waviness on the ground piece is suppressed, but it also means that another waviness may even be magnified. These wave numbers are depending on the position of the shoes in the shoe support system and of the contacting angles of these shoes.

Wave creation on the ring

From these investigations it was seen that the particular machine had three major effects that affected the waviness on the ground rings. Those were unbalance, eigenfrequencies and geometrical filtering. It was also seen that it is very critical if any of these effects are allowed to interact with each other so that they together create the same wave numbers. Thus it is very important to be able to predict and eliminate such interaction when the machine and its operating conditions are designed.

Feed rates

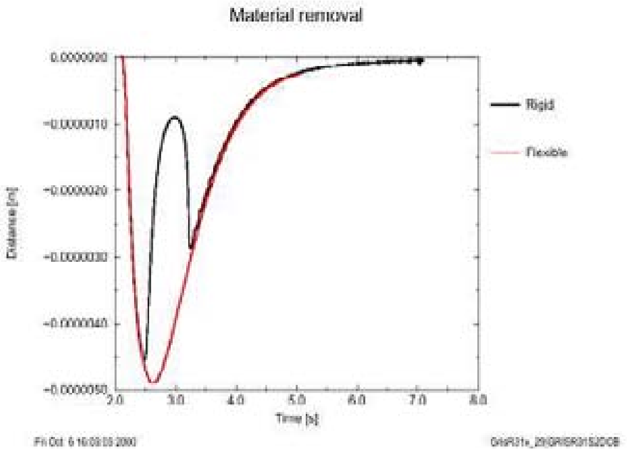


Figure 12-4: *Material removal height for the models with rigid ring in one case and flexible ring in the another one.*

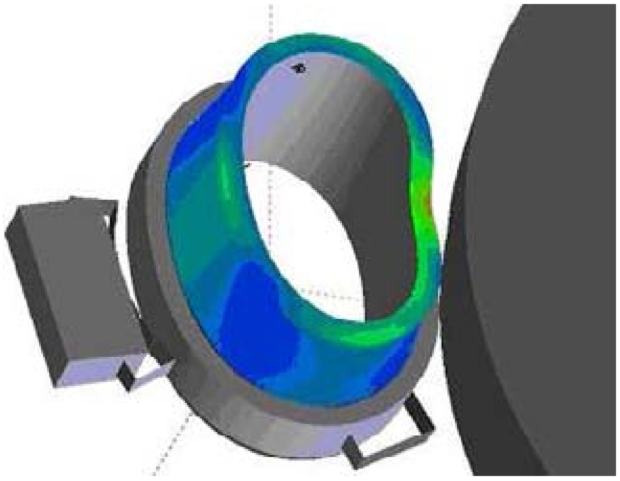


Figure 12-5: *Deformation and stresses of a ring that is being ground. The deformation is magnified.*

For rings that are not very stiff it is important that the ring flexibility can be included in the simulations. An example is a set of simulations to find optimum feed rates for the grinding cycles. In Figure 12-4 the material removal is shown for a case with drastic difference between a rigid and a flexible ring. When the grinding wheel is fed toward the ring during grinding a normal force arise between the workpiece and the grinding wheel. At a certain force level the friction in the grinding contact overcomes the friction on the drive plane for the rigid ring. The ring rotation speed starts to grow when the drive plane loses its grip due to the driving force from the grinding contact. As the ring rotates faster the relative speed in the grinding contact becomes smaller and the material removal rate goes down. However when using flexible ring model an extra effect comes into play: the workpiece gets deformed which means the same feed rate of the grinding wheel results in lower forces in the grinding contact and it will not accelerate. This example clearly shows the importance of flexibility in grinding modeling in some cases. In Figure 12-5 the deformation of a ground ring can be seen (magnified) as well as the effective stresses that are shown as a colour diagram on the ring surface.

Flexibility is also of great importance when the effects of stress release during grinding are investigated.

12.6 Summary

A simulation tool capable of dynamic modeling of the grinding process has been developed and the author contributed to the development with the implementation of the structural elasticity model. The process model used in the tool covers most of the important factors of the grinding process. Completely three-dimensional modeling is used. All the essential components of a grinding machine are modeled as a multibody system. Detailed models for the contacts between the workpiece and other parts of the machine, as well as models for material removal have also been implemented.

The tool enables detailed studies of a complete grinding cycle. It provides an efficient way to investigate possible optimisation of both the machine design and grinding process parameters. The tool is now actively used at SKF for parametric studies of different grinding schemes.

Chapter 13

Thermal Analysis with General Shape Functions

The general shape functions approach presented in Chapter 7 can be applied for the modeling and simulation of other physical processes. In this chapter the approach is applied to the solution of heat transfer equations. Discussions on the similarities between the elasticity and heat transfer modeling can be found in several references, e.g., [3, 16].

13.1 Numerical Solution of Heat Equation

Separation of variables can be used to derive an ODE suitable for transient analysis of heat flow. That is the temperature distribution over a body can be represented as:

$$T(\vec{\boldsymbol{p}}, t) = \boldsymbol{S}_\theta(\vec{\boldsymbol{p}}) \cdot \vec{\boldsymbol{\theta}}(t) \tag{13-1}$$

where $\boldsymbol{S}_\theta$ are the space dependent shape functions and $\vec{\boldsymbol{\theta}}$ are the time dependent state variables. Following the virtual temperatures approach (see, e.g., [3]) the heat equation can be written as:

$$-\boldsymbol{K}_{\theta\theta} \cdot \vec{\boldsymbol{\theta}} - \boldsymbol{H}_{\theta\theta} \cdot \vec{\boldsymbol{\theta}} + \vec{\boldsymbol{H}}_\theta + \vec{\boldsymbol{H}}_{\text{ext}} = \boldsymbol{C}_{\theta\theta} \cdot \dot{\vec{\boldsymbol{\theta}}} \tag{13-2}$$

where

- Thermal conductivity matrix $\boldsymbol{K}_{\theta\theta}$ is computed as:

  $$\boldsymbol{K}_{\theta\theta} = k \int_V (\boldsymbol{D}_t \boldsymbol{S}_\theta)^T \, \boldsymbol{D}_t \boldsymbol{S}_\theta \, dV \tag{13-3}$$

  The differential operator $\boldsymbol{D}_t$ represents the temperature gradient evaluation, which is in 3D Cartesian space is given by space derivatives along each axis: $[\frac{\delta}{\delta x}, \frac{\delta}{\delta y}, \frac{\delta}{\delta z}]$.

- The terms $-\boldsymbol{H}_{\theta\theta} \cdot \vec{\boldsymbol{\theta}} + \vec{\boldsymbol{H}}_\theta$ describes convection from the body surfaces into the surrounding environment, e.g., air:

$$\begin{aligned} \boldsymbol{H}_{\theta\theta} &= h_c \cdot \int_A \boldsymbol{S}_\theta^T \cdot \boldsymbol{S}_\theta \, dA \\ \vec{\boldsymbol{H}}_\theta &= h_c T_{sur} \int_A \boldsymbol{S}_\theta^T \, dA \end{aligned} \tag{13-4}$$

  where the integration is performed over all the surfaces of the body that are open to the environment with temperature T_{sur} and h_c is the heat transfer coefficient.

- The time dependent heat flow due to various effects in contact is given by generalized heat vector $\vec{\boldsymbol{H}}_{\text{ext}}$:

$$\vec{\boldsymbol{H}}_{\text{ext}} = \sum_j \left(\int_{A_j} \boldsymbol{S}_\theta^T h_{cj} \Delta T_j \, dA + \int_{A_j} h_{gen} \boldsymbol{S}_\theta^T \, dA \right) \tag{13-5}$$

  The effects should obviously be computed individually for each contacting body j. The corresponding contact area is designated A_j. The effects that are included in the equation above are:

  - Heat exchange with the heat transfer coefficient h_{cj};
  - Heat generation h_{gen} (e.g., frictional) at every particular contact point.

- The heat capacity matrix $\boldsymbol{C}_{\theta\theta}$ is computed as:

$$\boldsymbol{C}_{\theta\theta} = \rho \cdot c \cdot \int_V \boldsymbol{S}_\theta^T \cdot \boldsymbol{S}_\theta \, dV \tag{13-6}$$

The Equation 13-2 needs to be solved simultaneously with the Newton-Euler equations describing the dynamics of a mechanical system (see Section 3.4). The mechanical processes are normally much faster than the thermal ones. That is the eigenfrequencies associated with the multibody dynamics are normally several orders of magnitude higher than the eigenfrequencies associated with the heat flow. Therefore, for the two time steps t_i and t_{i+1} used by the ODE solver to resolve the equations of multibody dynamics one can assume:

$$\vec{\boldsymbol{H}}(t_i) \approx \vec{\boldsymbol{H}}(t_{i+1}) \tag{13-7}$$

That is the external heat flow change between two time steps is negligibly small. With such an assumption Equation 13-2 can be solved independently for each body. If the implicit Euler method is used, then:

$$\vec{\boldsymbol{\theta}}(t_{i+1}) = (\boldsymbol{C}_{\theta\theta} + \Delta t(\boldsymbol{K}_{\theta\theta} + \boldsymbol{H}_r))^{-1}(\boldsymbol{C}_{\theta\theta} \cdot \vec{\boldsymbol{\theta}}(t_i) + \Delta t \vec{\boldsymbol{H}}(t_i)) \tag{13-8}$$

where $\Delta t = t_{i+1} - t_i$.

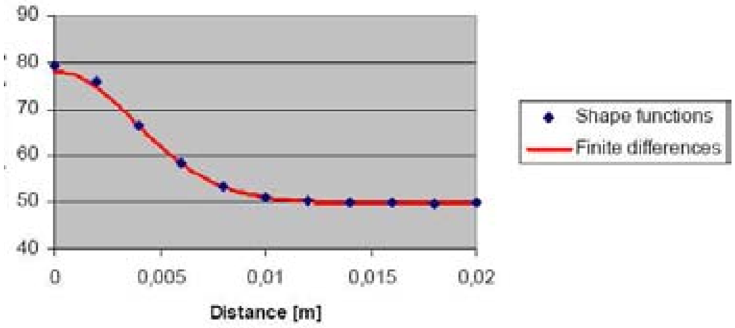


Figure 13-1: *Temperature distribution along a beam after 1 s. Comparison of the solution with final differences v.s. general shape functions.*

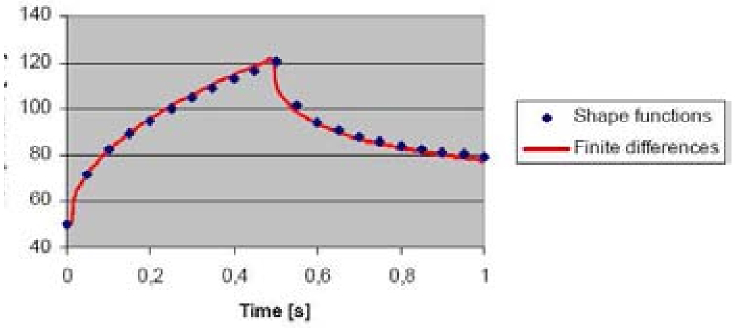


Figure 13-2: *Temperature at the heated end of a beam. Comparison of the solution with final differences v.s. general shape functions.*

Alternatively in some implementations it might be more convenient to use some constant time delay δT for the evaluation of the heat vector $\vec{\boldsymbol{H}}$:

$$\vec{\boldsymbol{H}} = \vec{\boldsymbol{H}}(t - \delta T) \tag{13-9}$$

Since a separate solver is used for the solution of thermal equations and heat transfer is much slower than the mechanical interactions different time scales can be used for the two processes. To implement this a *thermal acceleration factor* was introduced. The time step Δt in Equation 13-8 was magnified with this factor.

13.2 An Example of Transient Heat Conduction

As an example the one-dimensional heat conduction in a beam is solved. The beam is 20 mm long and is heated with a constant heat input at one end for 0.5 s and the other

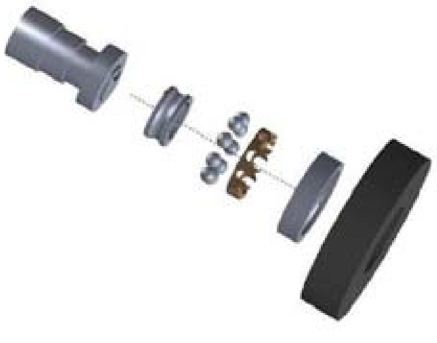


Figure 13-3: *BEAST model of the thermal test rig. Exploded view.*

end is held at a constant temperature of 50°C. A low order (4-th order) shape function is used. The result is compared to a solution with the finite difference scheme. Even if the shape functions are simple in this example the results are very similar in the two cases, see Figures 13-1 and 13-2.

13.3 DGBB Test Rig

Detailed temperature measurements has been performed on a DGBB (deep groove ball bearing) in a test rig at SKF ERC (Engineering Research Center). The test rig consists of a test bearing on a shaft supported by two spherical roller bearings (SRB). The test bearing was a DGBB 6309. The experiments have been used here to verify the BEAST simulations. The Beast model of the test rig consist of the bearing plus a shaft and a housing, see Figure 13-3.

Experiments were made under a number of different operating conditions. In order to verify BEAST, some of these cases were simulated. The BEAST model was kept on a rather simple level with a low level of calibration. The speed was constant at 3000 rpm and the load was increased in four steps, when the load increases also the power from the supporting SRB and the temperature of the air inside the bearing increase.

The power from the SRB and the two seals was estimated with the SKF General Catalogue [33]. Half the SRB power loss is assumed to go into the shaft. The losses from the seals are estimated to totally $60W$. The convection factor of the exposed surfaces of the shaft and the housing is taken from [5] ($50W/m^2K$). The convection factor of the oil and air mixture inside the bearing is put to $1000W/m^2K$. In Figure 13-4 a sketch of the applied boundary conditions can be found. The temperature of the mixture is increasing continuously until it reaches a steady state value for the actual load case. In BEAST this temperature is set to the steady state temperature. The simulations where run with a thermal acceleration factor of 10000.

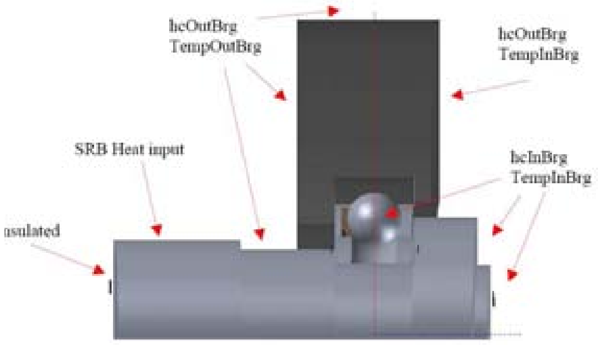


Figure 13-4: *BEAST model of the thermal test rig. Boundary conditions are indicated.*

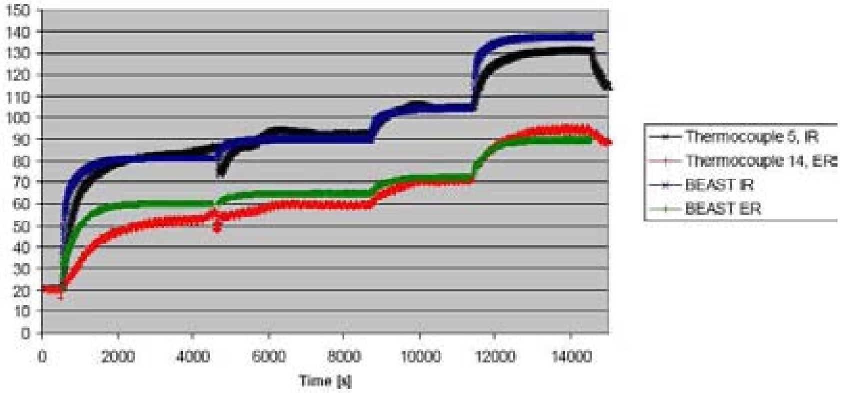


Figure 13-5: *Temperature on inner and outer rings from simulation and experiments.*

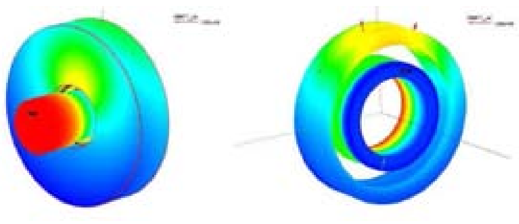


Figure 13-6: *Temperature distribution in the housing and shaft to the left and in the rings - to the right. Color scales are independent for each body.*

The results are presented in Figure 13-5. It can be seen that the major trends are well described in the simulations but that there are some differences in the actual temperatures. Note that the input to the simulations were simplified both with respect to geometry and boundary conditions. For instance, the same thermal boundary conditions are used for many different surfaces. It can clearly be seen form these simulations that it is possible to get reasonable results with rather simple models and estimations of the thermal boundary conditions. This is very important because it means that thermal analysis can be made quite easy within most MBS frameworks.

In Figure 13-6 an example of temperature distribution in the components in the test rig can be seen. Observe that the color scales are separate for each body.

Chapter 14

Conclusions

Dynamic equations of motion of flexible bodies when using the floating frame of reference formulation were presented in this thesis. Some approaches to simplify these equations by choosing the coordinate system for the solution of the equations were discussed. The generalized force vector was defined and procedures to calculate the generalized forces for different kinds of loading were presented. Generalized elastic forces, generalized viscosity forces (with special focus on Rayleigh damping), and generalized external force and body load were presented in particular. Special attention was given to the analysis of the generalized force resulting from the residual stress release during grinding.

We discussed current industry standard technologies for dynamics simulation with flexible bodies. A specific example based on the MNF format import was presented.

A set of assumed mode shapes for modeling deflection of bodies with relatively simple geometrical shapes in dynamics simulation with contacts was introduced. The proposed mode shapes use well known series of mathematical function as the basis for the modes set. The advantage of using the latter functions in terms of calculation efficiency and ease of use were demonstrated. The necessary boundary conditions for the proposed mode shapes were discussed and the methods to enforce such were compared. The ortho-normalization of mode shapes was chosen as the most suitable approach for the discussed case.

The system design that provides the necessary features to specify the external boundary and loading conditions was discussed. Some examples were used to demonstrate how complicated loading conditions can be easily specified using the proposed design.

Additional analysis types (eigenmode and quasistatic) for flexible bodies that are normally found only in finite-element packages were discussed. The procedures necessary to perform these types of analysis inside a dynamics simulation tool were presented.

Finally, some of the single body analysis procedures were used to verify the accuracy of simulations when the proposed set of deformation modes was used. The verification also provided an insight on how to select the number of functions in the function series that are used as the assumed mode shapes.

The approach with general shape functions was applied to the solution of the heat equation. The provided application examples strengthen the claim of practical importance of the approach.

The developed model was implemented in a real simulation system. The thesis describes the system architecture and class design. The issues that are specific for the flexible multibody systems as compared to the system with only rigid bodies are specially addressed.

The system has been successfully used in industrial simulation of the grinding process [21], spindles, geaboxes, and transient heat conduction in bearing applications [17].

Chapter 15

Future Work

The operations on a single flexible body with a large number of states requires manipulation of large matrices and vectors. That means the amount of time spent in linear algebra computations increases quickly. More investigations are needed to improve the performance of these calculations.

The approach with the general shape functions that include Fourier series and Chebyshev polynomials proved to be efficient for the grinding simulations. The question that remains open is the search for other kinds of general mathematical functions that might be more suitable for different geometries. The possible candidates include spline functions and wavelets.

The application of general functions to the modeling of both thermal processes and structural deformation opens up the possibility for coupled thermo-mechanical simulations. Further inverstigations are necessary on possible couplings between the two processes. The fact that thermal processes are significatly slower than mechanical ones should be utilized to simplify the coupling equations.

Bibliography

[1] ABAQUS homepage. *http://www.abaqus.com/*.

[2] ANSYS solutions homepage. *http://www.ansys.com/*.

[3] K.J. Bathe. *Finite Element Procedures*. Prentice-Hall Inc., 1996.

[4] J. Biera, J. Vinolas, and F.J. Nieto. Time-Domain Dynamic Modelling of the External Plunge Grinding Process. *Int. J. Mach. Tools Manufact.*, 37(11):1555–1572, 1997.

[5] J.R. Brown and N. H. Forster. Numerical investigation of the effect of carbon-carbon composite cages on high speed beariong operating temperatures. *Tribology Transactions*, 45, 2002.

[6] Carpinteiro,J. and Cocaño, P. Distortion of Bearing Rings due to Residual Stresses. Master's thesis, Chalmers Univ of Tech, Sweden, 2001.

[7] T.R. Chandrupatla and A.D. Belegundu. *Introduction to Finite Elements in Engineering 2nd ed.* Prentice Hall, 1997.

[8] Scott D. Cohen and Alan C. Hindmarsh. CVODE User Guide. www.netlib.org/ode/cvode.tar.gz, October 1994.

[9] R.D. Cook, D.S. Malkus, and M.E. Plesha. *Concepts and Applications of Finite Element Analysis 3rd ed.* John Wiley & Sons, 1989.

[10] R.R. Craig and Bampton M.C.C. Coupling of Substructures for Dynamic Analysis. *AIAA Journal*, 6(7), 1968.

[11] Stefan Dietz. *Vibration and Fatigue Analysis of Vehicle Systems Using Component Modes*. PhD thesis, Technical Univesity of Berlin, 1999.

[12] D. Fritzson, L.-E. Stacke, and P. Nordling. BEAST - a Rolling Bearing Simulation Tool. *Proc. Instn Mech Engrs, Part K*, 213, 1999.

[13] Y.C. Fung. *Foundations of solid mechanics*. Int. series in dynamics. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1965.

[14] H. Goldstein. *Classical Mechanics*. Addision-Wesley series in physics. Addision-Wesley Publishing Company, Inc., 1988. Second edition.

[15] A.E. Green and W. Zerna. *Theoretical elasticity*. Oxford University Press, Amen House, London, 1954.

[16] Andreas Heckmann and Martin Arnold. Flexible Bodies with Thermoelastic Properties. In *Multibody Dynamics 2005*, Madrid, Spain, 2005. ECCOMAS Thematic Conference.

[17] M. Holgerson, L.-E. Stacke, and I. Nakhimovski. Cage temperature prediction through dynamics simulation. In *STLE Annual Meeting, Las Vegas, Nevada*, 15-19 May 2005.

[18] G. Mase. *Continuum mechanics*. Schaum's outline series. McGraw-Hill Inc, New York, 1970.

[19] MSC Software Corporation homepage. *http://www.mscsoftware.com/*.

[20] Theoretical Backgorund. In *ADAMS/Flex Help*. MSC Software, 2003.

[21] I. Nakhimovski, D. Fritzson, and M. Holgerson. Dynamic Simulation of Grinding with Flexibility and Material Removal. In *Proceedings of Multibody Dynamics in Sweden 2001*, page http://www.sm.chalmers.se/MBDSwe_Sem01/Pdfs/IakovNakhimovski.pdf, 2001.

[22] Iakov Nakhimovski and Dag Fritzson. Modeling of Flexible Rings for Grinding Simulation. In *Multibody Dynamics 2005*, Madrid, Spain, 2005. ECCOMAS Thematic Conference.

[23] Parviz E. Nikravesh. Understanding Mean-Axis Conditions as Floating Reference Frames. In Jorge A.C. Ambrosio, editor, *Advances in Computational Multibody Systems*, pages 185–203. Springer, 2005.

[24] OMG. UML Unified modeling language specification. *http://www.omg.org/uml*, 2001.

[25] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1993.

[26] J.R. Rieker, Y.-H. Lin, and M.W. Trethewey. Discretization consideration in moving load finite element beam models. *Finite Elements in Analysis and Design*, 21:129–144, 1996.

[27] W.B. Rowe, M.N. Morgan, H.S. Qi, and H.W. Zheng. The Effect of Deformation on the Contact Area in Grinding. *Annals of the CIRP*, 43(1):410–412, 1993.

[28] R. Schwertassek, O. Wallrapp, and A.A. Shabana. Flexible Multibody Simulation and Choice of Shape Functions. *Nonlinear Dynamics*, 20:361–380, 1999.

[29] A.A. Shabana. Resonance Conditions and Deformable Body Coordinate Systems. *Journal of Sound and Vibration*, 192(1):389–398, 1996.

[30] A.A. Shabana. Flexible Multibody Dynamics: Review of Past and Recent Developments. *Multibody Systems Dynamics*, 1:189–222, 1997.

[31] A.A. Shabana. *Dynamics of multibody systems 2nd ed.* Cambridge University Press, 1998.

[32] SIMPACK homepage. *http://www.simpack.de*.

[33] 4000 E. SKF, 1989.

[34] L-E. Stacke and D. Fritzson. Dynamic behaviour of rolling bearings: simulations and experiments. *Proc. Instn Mech. Engrs, part J, Journal of Engineering Tribology*, 215:499–508, 2001.

[35] S.P. Timoshenko and J.N. Goodier. *Theory of Elasticity 3rd ed.* McGraw Hill Inc., 1987.

[36] H.K. Tonshoff, J. Peters, I. Inasaki, and T. Paul. Modelling and Simulation of Grinding Process. *Annals of the CIRP*, 41(2), 1992.

[37] B.F. Veubeke. The Dynamics of Flexible Bodies. *International Journal of Engineering Sciences*, 14:895–913, 1976.

[38] Lars Viklund. Contributions to a High-level Programming Environment for Scientific Computing. Licentiate thesis, Linkoping University, Sweden, 1994.

[39] O. Wallrapp. Standardization of Flexible Body Modeling in Multibody System Codes, Part I: Definition of Standard Input Data. *Mech. Struct. & Mach.*, 22(3):283–304, 1994.

[40] O. Wallrapp. Standardization of Flexible Body Modeling in Multibody System Codes, Part I: Definition of Standard Input Data. *Mech. Struct. and Mach.*, 22(3):283–304, 1994.

[41] O. Wallrapp. Flexible Bodies in Multibody System Codes. *Vehicle System Dynamics*, 30:237–256, 1998.

[42] Tamer M. Wasfy and Ahmed K. Noor. Computational strategies for flexible multibody systems. *ASME: Applied Mechanical Reviews*, 56(6):553–613, November 2003.

[43] J.A. Wensing. *On The Dynamics Of Ball Bearing*. PhD thesis, Univ. of Twente, Enschede, The Netherlands, 1998.

[44] Xun Chen , Rowe, W.B., Mills, B., Allanson, D.R. Analysis and Simulation of the Grinding Process Part IV: Effects of Wheel Wear. *Int. J. Mach. Tools Manufact.*, 38(1-2):41–49, 1998.

[45] W.C. Young. *Roark's Formulas For Stress & Strain 6th ed.* McGraw Hill International Edition, 1989.

Appendix

Shapes Generated by General Functions

This appendix is a supplement to Chapter 7. It provides some basic examples of the general shape functions deformation modes for the case of a ring structure.

The most general representation of the deformation shape functions used in BEAST for rotation symmetry bodies can be written as:

$$\begin{cases} u_{f,r}(\vec{\boldsymbol{u}}_0) = \sum_i^{n_{r,r}} \sum_j^{n_{r,\phi}} \sum_k^{n_{r,z}} \mathrm{Polynomial}(\mathrm{r}_0, \mathrm{i}) \cdot \mathrm{Fourier}(\phi_0, \mathrm{j}) \cdot \mathrm{Polynomial}(\mathrm{z}_0, \mathrm{k}) \\ u_{f,\phi}(\vec{\boldsymbol{u}}_0) = \sum_i^{n_{\phi,r}} \sum_j^{n_{\phi,\phi}} \sum_k^{n_{\phi,z}} \mathrm{Polynomial}(\mathrm{r}_0, \mathrm{i}) \cdot \mathrm{Fourier}(\phi_0, \mathrm{j}) \cdot \mathrm{Polynomial}(\mathrm{z}_0, \mathrm{k}) \\ u_{f,z}(\vec{\boldsymbol{u}}_0) = \sum_i^{n_{z,r}} \sum_j^{n_{z,\phi}} \sum_k^{n_{z,z}} \mathrm{Polynomial}(\mathrm{r}_0, \mathrm{i}) \cdot \mathrm{Fourier}(\phi_0, \mathrm{j}) \cdot \mathrm{Polynomial}(\mathrm{z}_0, \mathrm{k}) \end{cases} \tag{15-1}$$

where

- $\vec{\boldsymbol{u}}_0$ vector represents undeformed position of a material point. That is in cylindrical coordinates it has components: (r_0, ϕ_0, z_0).
- $\vec{\boldsymbol{u}}_f$ the deflection vector of the material point. The components of this vector, i.e. $u_{f,r}, u_{f,\phi}$ and $u_{f,z}$, represent deflections in radial, tangential and axial directions. The resulting position of the material point is obviously: $\vec{\boldsymbol{u}}_0 + \vec{\boldsymbol{u}}_f$.
- Polynomial(x, i) generates a Chebyshev polynomial of order i and $\mathrm{Fourier}(\phi, \mathrm{j})$ generates sine and cosine pair for the j-th wave.
- $n_{r,r}$, $n_{r,\phi}$, $n_{r,z}$, $n_{\phi,r}$, $n_{\phi,\phi}$, $n_{\phi,z}$, $n_{z,r}$, $n_{z,\phi}$ and $n_{z,z}$ are the nine parameters of the shape functions that specify the length of each series.

Examples of pipe deformation shapes that can be represented by the modes defined in Equation 15-1 are shown in Figure 15-1. The following parameters should be specified to the series in order to enable these shapes:

- (a) Ring expansion/compression is a deflection in radial direction as a function of radius. Setting $n_{r,r} > 0$ can give such shapes. Note that for a realistic free ring radial expansion mode is normally coupled with changes in axial direction so that ring volume does not change under the deformation. That is why $n_{z,z}$ parameter should also be set greater then zero for this case.

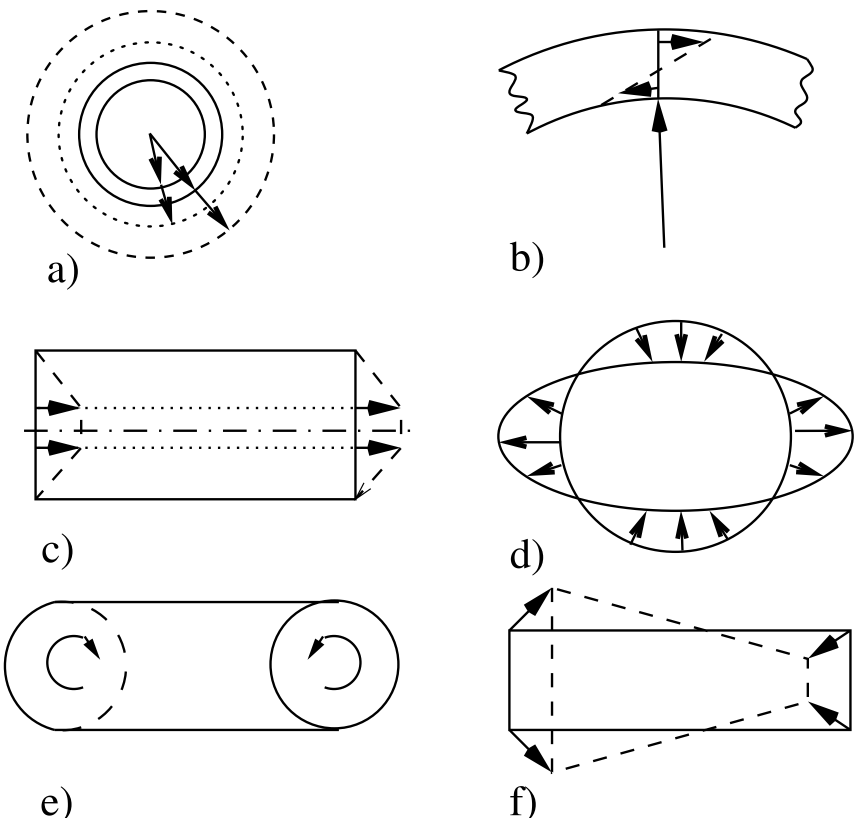


Figure 15-1: *Examples of deformation shapes. (a) Ring expansion/compression. (b) Rotation of ring cross-section. (c) Taper like deformation. (d) Ovality mode of a ring. (e) Pipe torsion. (f) Conical deformation.*

- (b) Rotation of ring cross-section is a result of tangential displacement with respect to the radius and angle. Setting $n_{\phi,r} > 0$ and $n_{\phi,\phi} > 0$ enable such deformation.
- (c) Taper like deformation can occur when axial deflection is a function of radius. Set $n_{z,r} > 0$ to enable it.
- (d) Ovality mode of a ring. To get this mode one should set $n_{r,r} > 0$, $n_{r,\phi} >= 2$, $n_{\phi,r} > 0$, $n_{\phi,\phi} \geq 2$. In general to simulate m-waves deflection $n_{r,\phi}$ and $n_{\phi,\phi}$ must be at least equal to m.
- (e) Torsion of a pipe is enabled when both $n_{\phi,r}$ and $n_{\phi,z}$ are greater then zero.
- (f) Conical deformation shown in the figure is a combination of deformations enabled by non-zero parameters $n_{r,z}$ and $n_{z,z}$

To give a real usage example let us assume that we need to simulate a relatively stiff ring. We only expect ovality mode from it and we don't expect much deformations through the cross-section. The minimal reasonable combination of the input values will be

- $n_{r,r} = 1$ allow radial expansion/compression
- $n_{r,\phi} = 2$ two waves correspond to the ovality mode
- $n_{r,z} = 0$ for a narrow ring taper shape is not important
- $n_{\phi,r} = 1, n_{\phi,\phi} = 2, n_{\phi,z} = 0$ - symmetry with radial parameters to get stress-free central line
- $n_{z,r} = 0, n_{z,\phi} = 0$ - narrow ring has practically no deformation of this kind
- $n_{z,z} = 1$ - allow axial expansion

The parameters specify a model which is stiffer than the real body but can simulate the ovality mode and will run fast due to the low number of shapes used.

In practice it is often worthwhile to try different combinations of shape functions parameters and check the results of single body analysis described in Chapter 4 to select the minimal number of modes satisfying the particular simulation requirements.

Appendix B

This appendix containes a complete implementation of the algorithm for the computation of mass matrix, quadratic velocity vector and generalized forces vector for a flexible ring or shaft in Matematica. The document itself is developed as a Matematica 4.0 notebook.

Setup code

We start by loading some necessary packages and defining a debug output function.

```
DeclarePackage["LinearAlgebra`MatrixManipulation`",
  {"AppendRows", "AppendColumns"}];
DEBUG = True;
Clear[DBGPRINT];
DBGPRINT[m__] := If[DEBUG, Print[m]];
```

Lame constants conversion function

The Lame function computes Lame's constants $\{\lambda, \mu\}$ for the given Young's modulus & Poison's ratio

```
Clear[Lame];
Lame[Emod_, γ_] :=
  {Emod * γ / ((1 - 2 * γ) * (1 + γ)),
   Emod / (2 * (1 + γ))};
```

Volume integration

The IntVol function computes all the necessary volume integrals as required for DynFreeBody: {mRR, mFF, J, Jkl, Sbar, Sbarkl, Jbarkl, Kp}. The naming conversions follow the notation used in the report.

The input parameters are:

Sp - function of (r,ϕ,z) returning the shape matrix 3xN that containes shape tensors for cyllindrical coordinates (ur,uϕ,uz);

ρ - mass density of the body;

L - length of the body;

Rm - medium radius of the cross section of the ring or shaft;

dR - wall thickness of the ring/pipe upto double Rm for a solid body;

λ, μ - Lame's material constants

```
Clear[IntVol];
IntVol[Sp_, ρ_, L_, Rm_, dR_, λ_, μ_] := Module[
  {mRR, mFF, J, Jkl, Sbar, Sbarkl, Jbarkl, Kp, V, m, Sc},
  (* Corresponding shapes in Cartesian system
     assuming x = Sin[ϕ], y = Cos[ϕ] *)
```
$$\texttt{Sc[r\_, ϕ\_, z\_] :=} \begin{pmatrix} \mathrm{Sin}[\phi] & \mathrm{Cos}[\phi] & 0 \\ \mathrm{Cos}[\phi] & -\mathrm{Sin}[\phi] & 0 \\ 0 & 0 & 1 \end{pmatrix} \texttt{.Sp[r, ϕ, z];}$$
```
  DBGPRINT[Sc[r, ϕ, z]];
```
$$V = \int_{-\frac{L}{2}}^{\frac{L}{2}} \int_{0}^{2\pi} \int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} r \, dr \, d\phi \, dz;$$
```
  DBGPRINT["Volume: ", V];
  m = ρ V;
  DBGPRINT["Mass: ", m];
```
$$J1 = \int_{-\frac{L}{2}}^{\frac{L}{2}} \int_{0}^{2\pi} \int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} \rho \{r\,\mathrm{Sin}[\phi],\ r\,\mathrm{Cos}[\phi],\ z\}\ r \, dr \, d\phi \, dz;$$
```
  DBGPRINT["J1: ", J1];
  Jkl =
```
$$\int_{-\frac{L}{2}}^{\frac{L}{2}} \int_{0}^{2\pi} \int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} \rho\,\mathrm{Outer}[\mathrm{Times},\ \{r\,\mathrm{Sin}[\phi],\ r\,\mathrm{Cos}[\phi],\ z\},\ \{r\,\mathrm{Sin}[\phi],\ r\,\mathrm{Cos}[\phi],\ z\}]\ r \, dr \, d\phi \, dz;$$
```
  DBGPRINT["Jkl: ", Jkl];
```

```
Sbar = $\int_{-\frac{L}{2}}^{\frac{L}{2}}\int_0^{2\pi}\int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} \rho\, Sc[r, \phi, z]\, r\, dr\, d\phi\, dz$;
DBGPRINT["Sbar: ", Sbar];
Sbarkl = Table[$\int_{-\frac{L}{2}}^{\frac{L}{2}}\int_0^{2\pi}\int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} \rho$ Outer[Times, Sc[r, ϕ, z]⟦k⟧,
        Sc[r, ϕ, z]⟦l⟧] r ⅆr ⅆϕ ⅆz, {k, 1, 3}, {l, 1, 3}];
DBGPRINT["Sbarkl: ", ScientificForm[Sbarkl, 2]];
DBGPRINT["Symmetry check: Sbarkl[k][
   l]==Transpose[Sbarkl[l][k]] -> ", Table[
  Simplify[Sbarkl⟦k⟧⟦l⟧ == Transpose[Sbarkl⟦k⟧⟦l⟧]],
  {k, 1, 3}, {l, 1, 3}]];
Jbarkl = Table[$\int_{-\frac{L}{2}}^{\frac{L}{2}}\int_0^{2\pi}\int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} \rho$ {r Sin[ϕ], r Cos[ϕ], z}⟦k⟧
       Sc[r, ϕ, z]⟦l⟧ r ⅆr ⅆϕ ⅆz, {k, 1, 3}, {l, 1, 3}];
DBGPRINT["Jbarkl: ", ScientificForm[Jbarkl, 2]];
mRR = m IdentityMatrix[3];
DBGPRINT["mRR: ", mRR];
mFF = $\sum_{k=1}^{3}$ Sbarkl⟦k⟧⟦k⟧; DBGPRINT["mFF: ", mFF];
DSp[r_, ϕ_, z_] :=
 {$\partial_r$ Sp[r, ϕ, z]⟦1⟧,
  $\frac{\partial_\phi Sp[r, \phi, z][[2]]}{r} + \frac{Sp[r, \phi, z][[1]]}{r}$,
  $\partial_z$ Sp[r, ϕ, z]⟦3⟧,
  $\partial_r$ Sp[r, ϕ, z]⟦2⟧ -
   $\frac{Sp[r, \phi, z][[2]]}{r} + \frac{\partial_\phi Sp[r, \phi, z][[1]]}{r}$,
  $\partial_z$ Sp[r, ϕ, z]⟦2⟧ + $\frac{\partial_\phi Sp[r, \phi, z][[3]]}{r}$,
  $\partial_z$ Sp[r, ϕ, z]⟦1⟧ + $\partial_r$ Sp[r, ϕ, z]⟦3⟧};
DBGPRINT["Polar DS operator: ", DSp[r, ϕ, z]];

(* Material constant matrix *)
Ec = {
  {λ + 2 μ, λ, λ, 0, 0, 0},
  {λ, λ + 2 μ, λ, 0, 0, 0},
  {λ, λ, λ + 2 μ, 0, 0, 0},
  {0, 0, 0, μ, 0, 0},
    {0, 0, 0, 0, μ, 0},
    {0, 0, 0, 0, 0, μ}
   };
  Kelp[r_, ϕ_, z_] :=
   Transpose[DSp[r, ϕ, z]].Ec.DSp[r, ϕ, z];
```

$$Kp = \int_{-\frac{L}{2}}^{\frac{L}{2}} \int_{0}^{2\pi} \int_{Rm-\frac{dR}{2}}^{Rm+\frac{dR}{2}} Kelp[r, \phi, z]\, r \, dr\, d\phi\, dz;$$

```
  DBGPRINT["Stiffness matrix: ", Kp];
  Return[{mRR, mFF, J, Jkl, Sbar, Sbarkl, Jbarkl, Kp}];
 ];
```

Skew matrix funtion

The utility function SkewMatrix returns the skew matrix for the given vector.
Important properties of the skew matrix are:
a × b = SkewMatrix[a].b; SkewMatrix[a]= -Transpose[SkewMatrix[a]]

```
Clear[SkewMatrix];
```

$$SkewMatrix[v\_] := \begin{pmatrix} 0 & -v[[3]] & v[[2]] \\ v[[3]] & 0 & -v[[1]] \\ -v[[2]] & v[[1]] & 0 \end{pmatrix};$$

Dynamic inertia calculations

The DynFreeBody function computes the mass matrix and RHS which includes the generalized forces and quadratic velocity tensors for the given state and precomputed integrals. The result is returned as {MM, RHS}. Note that the results are in the body coordinate system.
The required input is:

ω(3) - angular speed in the body coordinate system;
xf(n), vxf(n) - flexible states,
list of integrals as returned from IntVol function

```
Clear[DynFreeBody];
DynFreeBody[ω_, xf_, vxf_, {mRR_, mFF_,
    J_, Jkl_, Sbar_, Sbarkl_, Jbarkl_, Kp_}] :=
  Module[{MM, RHS, St, Stildet, Jbaree, Jbareedot,
    Jbaref, ωtilde, ωt2, Qvr, Qvα, Qvf},
   St = J1 + Sbar.xf;
```

```
DBGPRINT["St: ", St];
Stildet = SkewMatrix[St];
JbareeEq[k_, l_] := Module[{},
  If[k == l,
     3
   ( ∑ (Jkl⟦i⟧⟦i⟧) - Jkl⟦k⟧⟦k⟧ )
    i=1
             3
     + 2 ( ∑ (Jbarkl⟦i⟧⟦i⟧) - Jbarkl⟦k⟧⟦k⟧ ).xf
            i=1
               3
     + xf. ( ∑ (Sbarkl⟦i⟧⟦i⟧) - Sbarkl⟦k⟧⟦k⟧ ).xf,
              i=1
    - (Jkl⟦l⟧⟦k⟧ + (Jbarkl⟦l⟧⟦k⟧ + Jbarkl⟦k⟧⟦l⟧).xf +
       xf.Sbarkl⟦l⟧⟦k⟧.xf)
  ]
 ];
Jbaree = Table[JbareeEq[k, l], {k, 1, 3}, {l, 1, 3}];
DBGPRINT["Jbaree: ", Jbaree];
Jbareedot = Table[
  If[k == l,
          3
    2 ( ∑ (Jbarkl⟦i⟧⟦i⟧) - Jbarkl⟦k⟧⟦k⟧ ).vxf +
         i=1
            3
     vxf. ( ∑ (Sbarkl⟦i⟧⟦i⟧) - Sbarkl⟦k⟧⟦k⟧ ).xf +
           i=1
           3
     xf. ( ∑ (Sbarkl⟦i⟧⟦i⟧) - Sbarkl⟦k⟧⟦k⟧ ).vxf,
          i=1
    - ((Jbarkl⟦l⟧⟦k⟧ + Jbarkl⟦k⟧⟦l⟧).vxf +
       vxf.Sbarkl⟦l⟧⟦k⟧.xf +
       xf.Sbarkl⟦l⟧⟦k⟧.vxf)
  ], {k, 1, 3}, {l, 1, 3}];
DBGPRINT["Jbareedot: ", Jbareedot];
Jbaref =
 {
   xf.(Sbarkl⟦2⟧⟦3⟧ - Sbarkl⟦3⟧⟦2⟧),
   xf.(Sbarkl⟦3⟧⟦1⟧ - Sbarkl⟦1⟧⟦3⟧),
   xf.(Sbarkl⟦1⟧⟦2⟧ - Sbarkl⟦2⟧⟦1⟧)
  }
  + {
    Jbarkl⟦2⟧⟦3⟧ - Jbarkl⟦3⟧⟦2⟧,
    Jbarkl⟦3⟧⟦1⟧ - Jbarkl⟦1⟧⟦3⟧,
    Jbarkl⟦1⟧⟦2⟧ - Jbarkl⟦2⟧⟦1⟧
   };
 DBGPRINT["Jbaref: ", Jbaref];
 (* The components of the
   mass matrix have been computed at this
   point. Calculating quadratic velocity. *)
 ωtilde = SkewMatrix[ω];
 ωt2 = ωtilde.ωtilde;
 Qvr = ωt2.St + 2 ωtilde.Sbar.vxf;
 DBGPRINT["Qvr: ", Qvr];
 Qvα = -ω × (Jbaree.ω) - Jbareedot.ω - ω × (Jbaref.vxf);
 DBGPRINT["Qvα: ", Qvα];
 Qvf =
```

$$-\sum_{i=1}^{3}\sum_{j=1}^{3}\text{ωt2⟦i⟧⟦j⟧ (Jbarkl⟦i⟧⟦j⟧ + Sbarkl⟦i⟧⟦j⟧.xf)} - 2\sum_{i=1}^{3}\sum_{j=1}^{3}\text{ωtilde⟦i⟧⟦j⟧ Sbarkl⟦i⟧⟦j⟧.vxf;}$$

```
 DBGPRINT["Qvf: ", Qvf];
 Return[{AppendColumns[
    AppendRows[mRR, Transpose[Stildet], Sbar],
    AppendRows[Stildet, Jbaree, Jbaref], AppendRows[
     Transpose[Sbar], Transpose[Jbaref], mFF]],
   Flatten[{Qvr, Qvα, Qvf - Kp.xf}]}];
];
```

Generalized external force

GenForce function returns the generalized force tensor for the given cartesian force vector in the body coordinate system (Fx,Fy,Fz), the undeformed point vector in the body coordinate system (r,ϕ,z), shape field (Sp(r,ϕ,z) and elastic state vector xf.

```
Clear[GenForce];
GenForce[{Fx_, Fy_, Fz_}, {r_, ϕ_, z_}, Sp_, xf_] :=
 Module[{F, p, M, p2c, Qf},
               ⎛Sin[ang]  Cos[ang]   0⎞
  p2c[ang_] =  ⎜Cos[ang]  -Sin[ang]  0⎟;
               ⎝   0          0      1⎠
  Sc[rr_, ang_, zz_] := p2c[ang].Sp[rr, ang, zz];
  DBGPRINT["Cartesian shape: ", Sc[r, ϕ, z]];
  p = {r Sin[ϕ], r Cos[ϕ], z} + Sc[r, ϕ, z].xf;
  DBGPRINT["Deformed position of p: ", p];
  F = {Fx, Fy, Fz};
  M = p×F; (* moment *)
  Qf = Transpose[Sc[r, ϕ, z]].F; (* elastic part *)
  Return[Flatten[{{Fx, Fy, Fz}, M, Qf}]];
 ];
```

An example use

```
Clear[Demo];
Demo[] :=
 Module[
  {VolIntegrals, Sp, ρ, L, Router, Rinner, Rm, dR,
   Rs, λ, μ, Emod, γ, ω, xf, vxf, MM, RHS, GF, acc},
  (* Input parameters *)
  Emod = 2.03 10^11; (* Young' s modulus *)
  γ = 0.29;          (* Poison' s ratio *)
  ρ = 7800;          (* mass density *)
  Router = 0.1;      (* outer ring radius *)
  Rinner = 0.9;      (* inner ring radius *)
        (Router + Rinner)
  Rm = ─────────────────; (* median radius *)
               2
  dR = Router - Rinner;    (* wall thickness *)
  (* shape function scaling parameter *)
  Rs = Router;
  L = 0.02; (* width of the ring *)
  (* convert to Lame constants *)
  {λ, μ} = Lame[Emod, γ];
  Print["λ = ", λ, ", μ = ", μ];
  (* Assumed shape field *)
                       ⎛ r/Rs  1     0          0       ⎞
  Sp[r_, ϕ_, z_] :=    ⎜ 0     0  Cos[2 ϕ]  Sin[2 ϕ]   ⎟;
                       ⎝ 0     0     0          0       ⎠
  (* Perform volume integration *)
  VolIntegrals = IntVol[Sp, ρ, L, Rm, dR, λ, μ];
  (* Select angular velocity around
    the ring' s axis *)
  ω = {0, 0, 1000};
  Print["Angular velosity: ", ω];
  (* assume some initial deflection state *)
  xf = {-1. * 10^-6, 1. * 10^-6, -2. * 10^-6, 0};
  Print["Flex state: ", xf];
  vxf = {-1. * 10^-3, 1. * 10^-3, -2. * 10^-3, 0};
  Print["Flex velosity state: ", vxf];
  (* caclulate the mass
    matrix and external load free RHS *)
  {MM, RHS} = DynFreeBody[ω, xf, vxf, VolIntegrals];
  Print["Mass matrix: ",
   ScientificForm[MM, 2] // MatrixForm];
  Print["Free body RHS: ", RHS // MatrixForm];
  (* Symmetric load of 100 Newtons from
    top and bottom at the median radius *)
  GF = GenForce[{-100, 0, 0}, {Rm, π / 2, 0}, Sp, xf] +
    GenForce[{100, 0, 0}, {Rm, -π / 2, 0}, Sp, xf];
  Print["Generalized force: ", GF // MatrixForm];
  RHS += GF;
  (* Calculate acceleration tensor *)
  acc = LinearSolve[MM, RHS];
  Print["Acceleration tensor: ", acc // MatrixForm];
 ];
```

```
Demo[]
```

$\lambda = 1.08656\times10^{11}$, $\mu = 7.86822\times10^{10}$

$$\begin{pmatrix} 10.\,r\sin(\phi) & \sin(\phi) & \cos(\phi)\cos(2\phi) & \cos(\phi)\sin(2\phi) \\ 10.\,r\cos(\phi) & \cos(\phi) & -\cos(2\phi)\sin(\phi) & -\sin(\phi)\sin(2\phi) \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Volume: −0.0502655

Mass: −392.071

J1: {0, 0, 0.}

Jkl: $\begin{pmatrix} -80.3745 & 0 & 0 \\ 0 & -80.3745 & 0 \\ 0 & 0 & -0.013069 \end{pmatrix}$

Sbar: $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Sbarkl:

$$\begin{pmatrix} \begin{pmatrix} -8.\times 10^3 & -1.2\times 10^3 & 0 & -5.9\times 10^2 \\ -1.2\times 10^3 & -2.\times 10^2 & 0 & -9.8\times 10^1 \\ 0 & 0 & -9.8\times 10^1 & 0 \\ -5.9\times 10^2 & -9.8\times 10^1 & 0 & -9.8\times 10^1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & -5.9\times 10^2 & 0 \\ 0 & 0 & -9.8\times 10^1 & 0 \\ -5.9\times 10^2 & -9.8\times 10^1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & -5.9\times 10^2 & 0 \\ 0 & 0 & -9.8\times 10^1 & 0 \\ -5.9\times 10^2 & -9.8\times 10^1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} -8.\times 10^3 & -1.2\times 10^3 & 0 & 5.9\times 10^2 \\ -1.2\times 10^3 & -2.\times 10^2 & 0 & 9.8\times 10^1 \\ 0 & 0 & -9.8\times 10^1 & 0 \\ 5.9\times 10^2 & 9.8\times 10^1 & 0 & -9.8\times 10^1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{pmatrix}$$

Symmetry check: Sbarkl[k][l]==Transpose[Sbarkl[l][k]] -> $\begin{pmatrix} \text{True} & \text{True} & \text{True} \\ \text{True} & \text{True} & \text{True} \\ \text{True} & \text{True} & \text{True} \end{pmatrix}$

Jbarkl: $\begin{pmatrix} \{-8.\times 10^2, -1.2\times 10^2, 0, -5.9\times 10^1\} & \{0, 0, -5.9\times 10^1, 0\} & \{0, 0, 0, 0\} \\ \{0, 0, -5.9\times 10^1, 0\} & \{-8.\times 10^2, -1.2\times 10^2, 0, 5.9\times 10^1\} & \{0, 0, 0, 0\} \\ \{0, 0, 0, 0\} & \{0, 0, 0, 0\} & \{0, 0, 0, 0\} \end{pmatrix}$

mRR: $\begin{pmatrix} -392.071 & 0 & 0 \\ 0 & -392.071 & 0 \\ 0 & 0 & -392.071 \end{pmatrix}$

mFF: $\begin{pmatrix} -16074.9 & -2378.56 & 0 & 0. \\ -2378.56 & -392.071 & 0 & 0. \\ 0 & 0 & -196.035 & 0 \\ 0. & 0. & 0 & -196.035 \end{pmatrix}$

Polar DS operator: $\begin{pmatrix} 10. & 0 & 0 & 0 \\ 10. & \frac{1}{r} & -\frac{2\sin(2\phi)}{r} & \frac{2\cos(2\phi)}{r} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{\cos(2\phi)}{r} & -\frac{\sin(2\phi)}{r} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Stiffness matrix:

$$\begin{pmatrix} -3.76666\times10^{12} & -3.76666\times10^{11} & 0 & 0 \\ -3.76666\times10^{11} & -7.34513\times10^{10} & 0 & 0 \\ 0 & 0 & -1.57765\times10^{11} & 0 \\ 0 & 0 & 0 & -1.57765\times10^{11} \end{pmatrix}$$

Angular velosity: {0, 0, 1000}

Flex state: $\{-1.\times10^{-6}, 1.\times10^{-6}, -2.\times10^{-6}, 0\}$

Flex velosity state: $\{-0.001, 0.001, -0.002, 0\}$

St: {0., 0., 0.}

Jbaree: $\begin{pmatrix} -80.3862 & -0.000237854 & 0. \\ -0.000237854 & -80.3862 & 0. \\ 0. & 0. & -160.746 \end{pmatrix}$

Jbareedot: $\begin{pmatrix} 1.36962 & -0.237852 & 0. \\ -0.237852 & 1.36962 & 0. \\ 0. & 0. & 2.73924 \end{pmatrix}$

Jbaref: $\begin{pmatrix} 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \end{pmatrix}$

Qvr: {0., 0., 0.}

Qvα: $\{0., 0., -2739.24\}$

Qvf: $\{-1.60748\times10^{9}, -2.37854\times10^{8}, 392.071, 0.\}$

Mass matrix: $\begin{pmatrix} -3.9\times10^{2} & 0 & 0 & 0 & 0. & 0. & 0 & 0 & 0 & 0 \\ 0 & -3.9\times10^{2} & 0 & 0. & 0 & 0. & 0 & 0 & 0 & 0 \\ 0 & 0 & -3.9\times10^{2} & 0. & 0. & 0 & 0 & 0 & 0 & 0 \\ 0 & 0. & 0. & -8.\times10^{1} & -2.4\times10^{-4} & 0. & 0. & 0. & 0. & 0. \\ 0. & 0 & 0. & -2.4\times10^{-4} & -8.\times10^{1} & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0 & 0. & 0. & -1.6\times10^{2} & 0. & 0. & 0. & 0. \\ 0 & 0 & 0 & 0. & 0. & 0. & -1.6\times10^{4} & -2.4\times10^{3} & 0 & 0. \\ 0 & 0 & 0 & 0. & 0. & 0. & -2.4\times10^{3} & -3.9\times10^{2} & 0 & 0. \\ 0 & 0 & 0 & 0. & 0. & 0. & 0 & 0 & -2.\times10^{2} & 0 \\ 0 & 0 & 0 & 0. & 0. & 0. & 0. & 0. & 0 & -2.\times10^{2} \end{pmatrix}$

Free body RHS: $\begin{pmatrix} 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ -2739.24 \\ -1.61087\times10^{9} \\ -2.38157\times10^{8} \\ -315138. \\ 0. \end{pmatrix}$

Cartesian shape: $\begin{pmatrix} 5. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. \end{pmatrix}$

Deformed position of p: $\{0.499996, -2.\times10^{-6}, 0.\}$

Cartesian shape: $\begin{pmatrix} -5. & -1. & 0. & 0. \\ 0. & 0. & -1. & 0. \\ 0. & 0. & 0. & 0. \end{pmatrix}$

Deformed position of p: $\{-0.499996, 2.\times10^{-6}, 0.\}$

$$\text{Generalized force:}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0. \\ 0. \\ -0.0004 \\ -1000. \\ -200. \\ 0. \\ 0. \end{pmatrix}$$

$$\text{Acceleration tensor:}\begin{pmatrix} 0. \\ 0. \\ 0. \\ 0. \\ 0. \\ 17.0408 \\ 100941. \\ -4942.62 \\ 1607.56 \\ 0. \end{pmatrix}$$

Conclusions

Analysing the resulting acceleration vector and the specified shape field and displacements it is possible to see two main tendencies:

1. Under the specified condition the ring starts to expand due to centrifugal forces. We can see that from the large positive acceleration for the shapes acting uniformaly in radial direction.
2. The elastic force pushes the disturbed ring back to be round. Note the initial values and accelerations for the shapes with angular dependencies.

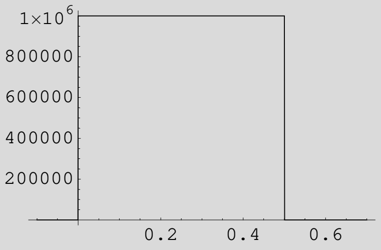
Appendx C

Using Shape Functions for Temperature Distribution Problem

The use of the general shape functions will be demonstrated for a simle example. A long balk with the dimensions (w, h, l) at a constant temrature To is subjected to a heat flow input q0(t) over the area at one end while the other end is put in the environment with constant temperature Tc. We analyse the temperature flow in the balk using the Ritz procedure.

Let us define the heat input function first:

```
q0[t_] := If[t < 0, 0, If[t < Ti, q0max, 0]];
Ti = 0.5; q0max=10^6;
Plot[q0[t], {t, -0.1, 0.7}]
```



```
- Graphics -
```

Let us assume a 3 D shape matrix representing the temperature distribution

```
S[x_,y_,z_]:={{1,x,y,z/l,-1+(z/l)^2,(z/l)(-3+ 4 (z/l)^2), 1
- 8 (z/l)^2 + 8 (z/l)^4}};
```

Number of state variables

```
Length[S[x,y,z][[1]]]

7
```

Necessary material and interface constants: k, ρ, c, ht. The complete equation set that we are solving:
-Kt θ + Q0 q0(t) - Qt θ + Qc = Ct $\partial\theta/\partial$t

The integrals to be precomputed:

```
Kel[x_, y_, z_] :=
  D[S[x, y, z], x] + D[S[x, y, z], y] + D[S[x, y, z], z];
```

$$\texttt{Kt = k}\int_0^{l}\int_{-w/2}^{w/2}\int_{-h/2}^{h/2}\texttt{Transpose[Kel[x, y, z]].Kel[x, y, z]}\,dx\,dy\,dz;$$

```
Kt // MatrixForm
```

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & h\,k\,l\,w & h\,k\,l\,w & h\,k\,w & h\,k\,w & h\,k\,w & 0 \\
0 & h\,k\,l\,w & h\,k\,l\,w & h\,k\,w & h\,k\,w & h\,k\,w & 0 \\
0 & h\,k\,w & h\,k\,w & \frac{h\,k\,w}{l} & \frac{h\,k\,w}{l} & \frac{h\,k\,w}{l} & 0 \\
0 & h\,k\,w & h\,k\,w & \frac{h\,k\,w}{l} & \frac{4\,h\,k\,w}{3\,l} & \frac{3\,h\,k\,w}{l} & \frac{32\,h\,k\,w}{15\,l} \\
0 & h\,k\,w & h\,k\,w & \frac{h\,k\,w}{l} & \frac{3\,h\,k\,w}{l} & \frac{69\,h\,k\,w}{5\,l} & \frac{16\,h\,k\,w}{l} \\
0 & 0 & 0 & 0 & \frac{32\,h\,k\,w}{15\,l} & \frac{16\,h\,k\,w}{l} & \frac{2816\,h\,k\,w}{105\,l}
\end{pmatrix}$$

$$\texttt{Ct = }\rho\,\texttt{c}\int_0^{l}\int_{-w/2}^{w/2}\int_{-h/2}^{h/2}\texttt{Transpose[S[x, y, z]].S[x, y, z]}\,dx\,dy\,dz;$$

```
Ct // MatrixForm
```

$$\begin{pmatrix}
c\,h\,l\,w\,\rho & 0 & 0 & \frac{1}{2}c\,h\,l\,w\,\rho & -\frac{2}{3}c\,h\,l\,w\,\rho & -\frac{1}{2}c\,h\,l\,w\,\rho & -\frac{1}{15}c\,h\,l\,w\,\rho \\
0 & \frac{1}{12}c\,h^3\,l\,w\,\rho & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{12}c\,h\,l\,w^3\,\rho & 0 & 0 & 0 & 0 \\
\frac{1}{2}c\,h\,l\,w\,\rho & 0 & 0 & \frac{1}{3}c\,h\,l\,w\,\rho & -\frac{1}{4}c\,h\,l\,w\,\rho & -\frac{1}{5}c\,h\,l\,w\,\rho & -\frac{1}{6}c\,h\,l\,w\,\rho \\
-\frac{2}{3}c\,h\,l\,w\,\rho & 0 & 0 & -\frac{1}{4}c\,h\,l\,w\,\rho & \frac{8}{15}c\,h\,l\,w\,\rho & \frac{5}{12}c\,h\,l\,w\,\rho & -\frac{2}{35}c\,h\,l\,w\,\rho \\
-\frac{1}{2}c\,h\,l\,w\,\rho & 0 & 0 & -\frac{1}{5}c\,h\,l\,w\,\rho & \frac{5}{12}c\,h\,l\,w\,\rho & \frac{17}{35}c\,h\,l\,w\,\rho & \frac{1}{6}c\,h\,l\,w\,\rho \\
-\frac{1}{15}c\,h\,l\,w\,\rho & 0 & 0 & -\frac{1}{6}c\,h\,l\,w\,\rho & -\frac{2}{35}c\,h\,l\,w\,\rho & \frac{1}{6}c\,h\,l\,w\,\rho & \frac{31}{63}c\,h\,l\,w\,\rho
\end{pmatrix}$$

$$\texttt{Q0 = }\int_{-w/2}^{w/2}\int_{-h/2}^{h/2}\texttt{Transpose[S[x, y, 0]]}\,dx\,dy;$$

```
Transpose[Q0] // MatrixForm
```

$$\begin{pmatrix} h\,w & 0 & 0 & 0 & -h\,w & 0 & h\,w \end{pmatrix}$$

$$Qt = ht \int_{-w/2}^{w/2}\int_{-h/2}^{h/2} \text{Transpose}[S[x, y, l]].S[x, y, l]\, dx\, dy;$$

```
Qt // MatrixForm
```

$$\begin{pmatrix} h\,ht\,w & 0 & 0 & h\,ht\,w & 0 & h\,ht\,w & h\,ht\,w \\ 0 & \frac{1}{12} h^3\,ht\,w & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{12} h\,ht\,w^3 & 0 & 0 & 0 & 0 \\ h\,ht\,w & 0 & 0 & h\,ht\,w & 0 & h\,ht\,w & h\,ht\,w \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h\,ht\,w & 0 & 0 & h\,ht\,w & 0 & h\,ht\,w & h\,ht\,w \\ h\,ht\,w & 0 & 0 & h\,ht\,w & 0 & h\,ht\,w & h\,ht\,w \end{pmatrix}$$

$$Qc = ht\, Tc \int_{-w/2}^{w/2}\int_{-h/2}^{h/2} \text{Transpose}[S[x, y, l]]\, dx\, dy;$$

```
Transpose[Qc] // MatrixForm
```

$$\begin{pmatrix} h\,ht\,Tc\,w & 0 & 0 & h\,ht\,Tc\,w & 0 & h\,ht\,Tc\,w & h\,ht\,Tc\,w \end{pmatrix}$$

Setting the parameters:

```
l=0.02; h = 0.01; w = 0.01;
To = 50; Tc = 50;
k = 36; ρ=7800; c=460; ht=10^6;
Tstart = 0;
Tend = 1;
Kt;Q0;Qt;Qc;Ct;
```

Initial values for the state variables:

```
θ0=Table[0,{i,Length[S[x,y,z][[1]]]}];
θ0[[1]] = To;
θ0

{50, 0, 0, 0, 0, 0, 0}
```

Solving differential equation (-Kt *θ* + Q0 q0(t) + Qt *θ* - Qc = Ct ∂*θ*/∂t) using simple method

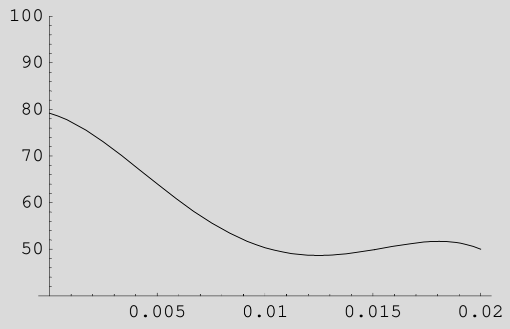
```
δt=0.001;
```

```
t = Tstart;
θ=θ0;
Tleft={};
While[t < Tend,
θ=N[θ+δt(LinearSolve[Ct,-Kt.θ+ Q0 q0[t] - Qt.θ + Qc])];
t=t+δt;
Tleft=Append[Tleft,S[0,0,0].θ];
];
θ
```

```
 {{-406.283}, {113.855}, {113.855},
  {360.507}, {-517.77}, {128.107}, {-32.315}}
```

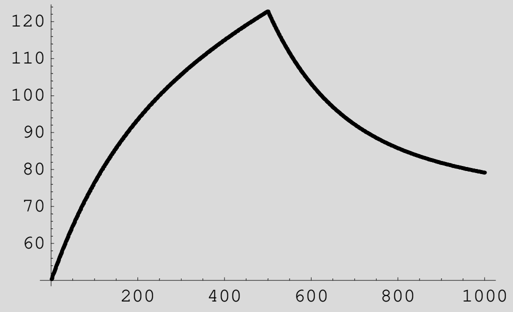
Final temprature distribution on the cenral line:

```
Plot[S[0,0,z].θ,{z,0,l},PlotRange->{40,100}];
```



Temperature at the left end of the balk as a function of time.

```
ListPlot[Flatten[Tleft]]
```



```
- Graphics -
```

Concluson

The complete procedure necessary for a transient heat transfer simulation has been presented. The small number of shape functions limits the accuracy of the results. This is particularly noticable on the plot of the temperature distribution over the balk at end of the simulation. Here the low order of the approximation results in non-monotonous distribution curve as the selected temperature shapes cannot ideally resolve the boundary conditions. The general qualitative behaviour is however captured correctly.

Part II

Hybrid Parallelization of Multibody Simulation with Detailed Contacts

Chapter 16

Introduction

16.1 Parallelization of Multibody Dynamics Simulations

The work presented in this thesis is focused on the problems rising in parallelization of multibody dynamics simulation codes with advanced analysis of contacts. Multibody dynamic simulation packages are designed to perform transient simulation of mechanical systems. The major players on the market are `MSC.ADAMS`[20], LMS's `Virtual.Lab Motion`[18] and `SimPack` from INTEC GmbH [25].

In this thesis we focus on the parallelization of SKF's proprietary bearing simulation framework BEAST (see Chapter 1).

The toolbox development in terms of design generalization has widened the application area of the program. Later versions of BEAST include models of grinding machines, experimental engines, gear transmissions, etc. BEAST can now be seen as a general multibody simulation tool specialized in detailed contact analysis.

The most computationally expensive part of BEAST can be described with the Algorithm 16-1. We can see that essentially three potentially parallel nested loops are used. The outer loops of the BEAST code (over pairs of bodies and contacts) were earlier parallelized by using SPMD style programming with message-passing by means of PVM/MPI libraries [9, 10]. However, the innermost loop over the contact slices had never been parallelized before. This is due to the tight coupling between the slices of a single contact that result in very irregular data access patterns rendering the distributed memory parallelization infeasible.

The distributed memory parallelization works efficiently for most models. Unfortunately, there are cases where the computations attributed to a single contact dominate an entire simulation run. One particular example is the simulation of an SRB (Spherical Roller Bearing) bearing presented in Section 20.

For such situations a more fine grained approach needs to be developed. The additional driving force for the presented work was the availability of a cluster of high-performance SMP computers as well as expectations for the appearance of sim-

Algorithm 16-1 Pseudocode for the computationally expensive part of BEAST

```
for each time step $t_i$ in serial do
    Obtain system state from ODE solver
    for each pair of bodies do
        Compute relative motion between the bodies.
        for each contact between the two bodies do
            Initialize local output variables.
            Perform a rough estimation of contact size (boxing).
            for each slice of the contacting surfaces do
                Construct the grid and refine if needed.
            end for
            for each slice of the contact do
                Integrate the contact (tribological calculations).
                Accumulate output.
            end for
            Finalization of contact calculations.
        end for
        Accumulate output.
    end for
    Send state derivatives to ODE solver
end for
```

ilar hardware in the near future.

The important requirement for the parallelization approach was strait-forward application to the legacy simulation code. This forced the choice of standard *C*++ to be the implementation language. Support of any compliant standard *C*++ compiler was also essential.

16.2 Utilizing Clusters of Shared Memory Computers

During the last decade shared memory machines have become much more cost effective. Assuming a fixed price, we can also note that the size of a parallel machine or a cluster counted as the number of processor nodes has grown as well. Already today we have a cluster of 48 two-way SMPs running at SKF. One can expect that a cluster with 4-processor nodes will be available in the near future. Multiprocessor professional workstations are also on their way to users of high-end simulation tools. Simultaneous multi-threading technology [7] is promising even more - possibly virtual - processors per-box in the coming years. Therefore the problem of effective use of computers with shared memory becomes more and more important.

An application can use shared memory machines and clusters of such machines in several ways:

- Rely on efficient implementations of PVM/MPI libraries.
- Implement a hybrid parallelization approach.
- Use pure shared memory execution (on a single SMP node).
- Use virtual shared memory programming.

Parallel programs using explicit message passing with a PVM/MPI library can obviously run on shared memory computers. Shared memory contributes to faster communication between processes within each node since data is not sent over the network but copied inside the local memory. This approach is attractive since legacy schedulers and other software can be used directly without any modifications. Unfortunately, there are some problems with such a system design.

The dynamic nature of the contact analysis calculation makes it very hard to predict which data is needed by a processor. On the other hand, network latency forces us to use larger messages to amortize the fixed part of the transferring cost over many bytes. Therefore, more data than necessary is sent between processors when explicit message-passing is used.

Another problem is the possible task granularity. With message-passing approach only relatively coarse grain tasks are feasible. Low level loops with many dependencies between iterations (e.g., different slices of the same contact) are very hard to separate. On a cluster with many nodes the largest task very soon becomes a bottleneck. For some of the cases we see that task granularity limits the maximum speedup to be between 1.7 and 4.

The existing schedulers for the message-passing code views the parallel computer as symmetric. However, a processor normally has shorter communication latency and larger bandwidth for messages sent to other processors in the same node than for the messages to the processors in other nodes. This difference is currently not taken into account. A more advanced scheduler might therefore gain some advantage by using a more detailed model for the communication hardware. This will not, however, solve the problem of the task granularity.

Multithreading is also advantageous if we consider processors with simultaneous multithreading. Such processors are most efficient when using different threads and not separate processes due to the shared cache and other resources of the virtual sub-processors.

Hybrid or dual-level parallelization that explicitly uses message-passing for communication between the nodes and a shared memory approach within the nodes is the second alternative to consider. The top level scheduler then works as before but assigns groups of tasks to compute nodes and not to individual processors. A single PVM/MPI process is run on each node. Each such process executes several concurrent computation threads within a node.

This may result in better load balancing and raise the potential for parallel execution since smaller tasks (e.g., different slices of a single contact) can be run in parallel. The hybrid approach also simplifies the scheduling of the message passing part of the problem. Two reasons contribute to this simplicity:

- Scheduling is done for nodes and not processors. Therefore the scheduling problem is reduced.

- There is no need to consider differences in the communication speeds between the processors belonging to the same and to different nodes.

This approach is probably most feasible for larger clusters where task granularity is the limiting factor for achieving good speed up. Simulation problems with much data sent between the processes can be candidates as well. The benefit here comes from explicit and more efficient shared memory communications.

The pure shared memory programming approach can be used on multiprocessor workstations. A shared memory scheduler operating on the complete problem can achieve almost ideal load balancing using dynamic methods for scheduling fine grain tasks. The shared memory approach can also take into account data layout in the memory and optimize locality. Note, however, that excessive synchronization and various cache effects may lead to low parallel efficiency of the shared memory approach.

A virtual shared memory assumes existence of a run-time system that guarantees a common and coherent view on the memory from all the nodes. Even though such systems exist they still require very careful data partitioning when programming, in order to achieve acceptable performance.

Chapter 17

Applicability of Shared Memory and Hybrid Parallelization

At this point we would like to discuss some articles on hybrid parallelization published during the last few years. Some of them report that using hybrid parallelization degrades the performance as compared to pure message passing. Fortunately, there are also reports indicating success in application of hybrid parallelization strategies. Our intention is to survey the previous work and identify the general parallelization strategies that may be applicable for most applications.

Most of the authors of the papers mentioned below rely on the OpenMP standard [2] for the shared memory parallelization. OpenMP advertises the incremental parallelization approach - only some computationally expensive loops are parallelized and rest of the code stays intact. Unfortunately, it soon became obvious that the naive use of OpenMP to create a hybrid parallel code from an already parallel application does not provide performance improvements.

The authors of [5] tried to implement OpenMP parallelization in NAS benchmarks. They found out that a unified MPI approach is better for most of the benchmarks. The hybrid approach becomes better only when fast processors make the communication performance significant and the level of parallelization is sufficient.

In [13] a hybrid approach was tried for a discrete element modeling simulation. The authors observed that their hybrid implementation was not efficient enough to outperform pure message-passing on an SMP cluster. The authors mention a large overhead for multi-threaded parallelization at fine grain level most likely due to excessive synchronizations and cache effects. This overhead is not very critical when running on a single SMP machine, but it makes hybrid implementation slower than the pure MPI one on a larger cluster. Another important consideration in the paper is the potential to decrease multithreading overhead, which would however require significant code restructuring.

The authors of [1] tried to analyze the sources of performance differences in OpenMP and MPI codes. They found out that basically no difference is attributable to the way the two models exchange data between processors. However, there were interesting differences in individual code sections. It was found that the OpenMP version incurs more code overhead (e.g., the code executes more instructions) than the MPI version, which becomes more critical as the number of processors grows.

Similar conclusions were made by the authors in [24]. The results for their test applications indicated that shared memory parallelization can achieve about half the parallel efficiency of MPI in most cases, while being competitive in some others. A hybrid strategy could show only a small performance advantage over pure MPI in some cases. The authors explain such results by the tradeoffs between the two approaches. While shared-memory programming can potentially reduce the intra-process communications compared to MPI, it often has additional overhead of explicit synchronizations. The authors also mention increased code complexity as a disadvantage of the hybrid approach. This was true for the NBODY application presented in the paper.

The authors concluded the paper stating that a pure MPI implementation is a more effective strategy than hybrid programming on SMP clusters. It should however be noticed that only a small cluster was used for the test runs.

An analysis of different hybrid parallelization strategies together with a survey and an in-depth study of shared memory programming with OpenMP vs message-passing programming with MPI can be found in [16]. The authors demonstrated that careful program analysis followed by data layout restructuring is needed for an efficient OpenMP program. Only with this strong programming effort OpenMP program will become competitive with its MPI counterpart. The authors report that the naive loop level OpenMP is simply not competitive and, currently, only SPMD style programming can provide good performance consistently. The authors of [6] came to the same conclusions regarding the data layout rearrangement. They also proposed extra data locality directives for OpenMP to facilitate data redistribution between threads.

Summarizing the results presented in the papers noticed above one can list the cases where hybrid parallelization is likely *not* to be efficient:

- Pure substitution of communication strategy from message-passing to shared memory does not improve performance.
- Cases where coarse-grain distributed memory parallelism can efficiently utilize all the available processors.
- Naive fine-grain loop parallelization does not normally improve the performance.

The reasons behind inefficiency are additional synchronization overhead and cache utilization issues.

Despite such strong arguments against shared memory parallelization some cases are still open for efficient hybrid approaches. We believe that the presented results do not discredit the idea of hybrid parallelization *per se*, but rather raise the question of applicability of the approach to the particular architecture and software combination.

The situation that will be further explored in this paper occurs when the number of processors available for a parallel application is higher than the maximum speedup reachable by a message-passing algorithm due to task granularity. The granularity can become the limiting factor either due to a really large number of processors or due to some application specific issues.

The following few references present some examples of such applications. The authors of [8] showed the feasibility of hybrid parallelization for the solution of CFD problems. In this case thread level parallelism facilitates a specific refinement algorithm which is hard to parallelize with traditional message-passing.

One of the first successful uses of hybrid parallelization was presented in [12]. For the presented algorithm pure OpenMP outperforms pure MPI on a single node. Across entire cluster of SMP nodes, the hybrid MPI/OpenMP implementation outperforms pure MPI.

Another important advantage of the hybrid approach when applied to large clusters has been demonstrated in [15]. The authors used a hybrid approach to map a large molecular dynamics simulation on an IBM pSeries 690 cluster with 1024 processors. Such a simulation would not be possible at all without the hybrid parallelization approach.

In a more general context the designers of ARMI communication library [23] found the need to exploit finer grain parallelism than what the message-passing programming style is suitable for. They showed that hybrid parallelization can yield advantages in some cases.

Let us now come back to the problems around parallelization of multibody dynamics simulation codes with advanced analysis of contacts. The case when the computations attributed to a single contact dominate an entire simulation run can become a granularity problem. If the achieved speed up is significantly lower than the linear expectation then a hybrid approach becomes feasible. The strategy presented in the following sections can then be applied.

Chapter 18

Parallel Shared-Memory Framework

18.1 Choice of Multithreaded Framework

The simulation algorithm presented in Section 16 can be seen as an iterative algorithm with nested parallel regions. Nested parallel regions are typical for the applications with nested loops with independent (or weakly dependent) iterations and for codes utilizing functional decompositions. Algorithm 18-1 presents a general pseudocode of such codes. Note that the number of iterations of the inner loops may be different for different iterations of the outer loops.

Algorithm 18-1 A pseudocode for a parallel program with nested parallel regions.

```
for i = F0, G0 in parallel do
    Some code dependent on i
    for j = F1(i), G1(i) in parallel do
        Some code dependent on i, j
        for k = F2(i, j), G2(i, j) in parallel do
            Some code dependent on i, j, k
        end for
        Some code dependent on i, j
        for k = F3(i, j), G3(i, j) in parallel do
            Some code dependent on i, j, k
        end for
    end for
end for
```

We have mentioned earlier that the outer loops of the algorithm could be parallelized in a SPMD style. However, the innermost loops could only be handled by a shared-memory approach because of the complex data dependencies between the loop iterations.

The purpose of this work was to complement the legacy message-passing scheduler with a shared memory parallelization subsystem. The message passing part is kept in touch and provided the coarse level scheduling of the outer loop iterations between the nodes. Within each node the newly developed shared memory parallelization is used.

Coarse-grained parallelization typically has higher computation to communication ratio than the fine-grained one and is therefore preferable. Hence, if several outer loops iterations are scheduled to the same compute node the shared-memory subsystem should run them in parallel before trying to parallelize the inner loops. The inner loops need to be parallelized only if significant load imbalance at the outer loop level is detected.

In the BEAST case, this means that if several contacts are scheduled to a compute node then different threads should start processing different contacts first. Only if significant load imbalance is detected separate contact should be considered for parallel execution.

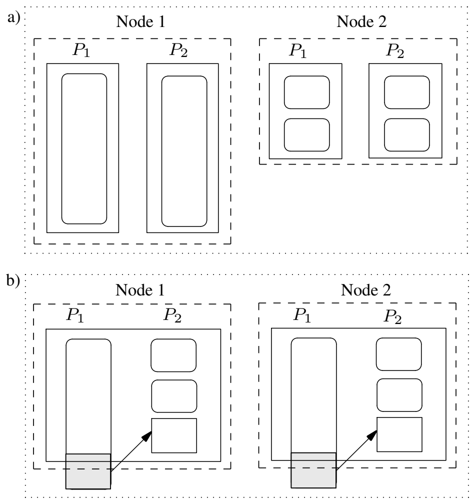


Figure 18-1: Load balancing between two dual processor nodes. (a) Imbalance for a pure message-passing code; (b) Balanced execution in hybrid code.

Figure 18-1 presents a situation where the hybrid code is expected to outperform the message-passing counterpart. In case (a) one-level message-passing task scheduling for four processors is done. Load imbalance occurs due to significant variation in task sizes. In case (b) the high level scheduler distribute tasks for two nodes. The shared-memory framework gets better load balancing by creating fine level tasks.

The most popular framework today for shared-memory programming in high-performance computing is certainly OpenMP. Unfortunately, still not all the compilers are supporting OpenMP extensions. Even fewer enable nested parallelism.

Besides, the number of features provided by OpenMP directives is limited and advanced scheduling algorithms and synchronization schemes are hard to implement.

Alternative frameworks like Cilk [27] and StackThreads/MP [17] are certainly more advanced. The work-stealing strategies (see below) implemented in the corresponding schedulers have proved to be efficient. The only disadvantage in using those frameworks is the limitation to a single compiler (*gcc*) which makes it limits the portability of the application. The necessity to manage the program stack for the parallel threads may cause implementation problems for building the frameworks for other compilers. It also makes it hard to apply the frameworks to a legacy *C++* code. Besides, it complicates the use of thread local storage, which may be utilized, e.g., for thread specific buffering in the application. Such a buffering requires that the ID of a thread executing a calculation does not change arbitrary. Another issue in both frameworks is the lack of explicit support for loop parallelization. The application programmer is therefore made responsible for the development of the necessary loop scheduling algorithms.

Even though we realized that the existing Cilk and StackThreads frameworks could not be directly utilized in our project the work-stealing strategy is certainly an attractive approach. The work-stealing strategy assumes two stages:

1. Initial scheduling of tasks to processors;

2. Possibility for a processor that have completed all its tasks early to choose a 'victim' among the other processors and 'steal' some work from it.

The work-stealing strategy makes it possible to significantly reduce the synchronization overhead since processors need to synchronize only when 'stealing' tasks and not when running their own tasks. The strategy also facilitates the implementation of the affinity principles as described in [19] and [26]. The affinity principle recommends scheduling consequent loop iterations to the same processor. In such a way cache contentions can be minimized. In a work-stealing implementation the initial tasks always contain chunks of consequent loop iterations.

Having looked at the existing multithreaded environments and our target application we have formulated the following requirements for the framework:

- Fork-join programming style for the whole application.

- Multilevel parallelism and special support for dynamically scheduled nested loops.

- The number of kernel level thread creation and termination operations should be minimized.

- The caller thread is responsible for task termination. No advanced and possibly architecture specific stack management should be used.

- Work-stealing and affinity methodologies should be used.

- Any standard C++ compiler should be usable.

To realize the requirements above we have decided to create our own framework implemented as a *C++* library on top of *pthreads* [22]. The algorithms presented in the following sections implement the strategies similar to the ones used in Cilk and StackThreads/MP but stack manipulations are avoided. This makes the framework portable at the price of lower parallel efficency in some cases. Additionally, the structures facilitating loop parallelization are introduced.

18.2 A Framework For Nested Parallel Execution

The scheduling sequence for the hybrid approach presented in this paper has two stages. First, the message-passing scheduler analyses coarse-grained tasks and distributes groups of them to different nodes (e.g., sets of contacts are distributed between nodes in our application). The legacy scheduler [10] was used in our case and is not further discussed in this paper.

At every compute node a thread pool with one thread per processor is running. The thread pool at every node has a single *master* thread responsible for communication with other nodes and a number of *worker* threads. The *master* threads receive the groups of tasks from the message-passing part of the framework and submit them to the shared-memory subsystem on each node. Second scheduling stage is then initiated. The shared-memory scheduler implements the work-stealing strategy allowing different threads within nodes to steal tasks from each other.

Some tasks running in the shared memory context contain computational work with potential for parallel execution (e.g., contact slices). Due to the application specific issues partitioning of this work is only feasible in shared-memory environment and therefore it is always scheduled as an atomic task by the message-passing scheduler. Such a task can be further partitioned in the shared memory context and a new group of tasks representing parts of the coarse-grain task can be created.

The smaller tasks can be used to improve load balancing within a node. Threads that complete their tasks (i.e., all the initially distributed complete contacts) early try to 'steal' the smaller sub-tasks from other processors within the same node. Fine grain parts of the heavier tasks can be moved to other processors at this point (e.g., parts of a contact calculation can be stolen).

Note that for the case of dual-processor nodes at most one coarse grain task will eventually be run with two threads.

In order to represent groups of concurrent tasks in our framework the concept of *task queues* is introduced. A task queue is an interface to an application specific code that has potential for parallel execution. It can be understood as a parallel region in an OpenMP program or a series of *spawn* requests in Cilk. In our framework it is represented by an abstract class. An object implementing the task queue is created by the application every time there is a set of tasks that can run in parallel. The task queues are then *submitted* to the framework for parallel execution. A new task queue can be created by any task including the tasks that are already running concurrently with others. The task queue class interface provides only one method: *run*. Different threads call this method concurrently until all the *tasks* in the queue are executed. The *submit* operation is blocking. It terminates when all the tasks in the submitted queue have completed. The *submit* operation is processed as a normal function call and always returns in the context of the calling thread. Therefore, individual tasks may safely store the references to the thread local storage on stack and utilize it for buffering data in a thread context.

Every task queue has a special scheduler responsible for the management of tasks within the queue. The scheduler decides how the tasks in the queue should be combined and ordered. The simplest implementation of a queue can be done with the following dynamic scheduling algorithm:

```
Routine Run
if no tasks left in the queue then
   return
end if
lock the queue
while there are tasks do
   take the next task from the queue;
   unlock the queue;
   run task;
   lock the queue;
end while
unlock the queue;
```

The task queue abstraction facilitates introduction of advanced loop schedules into the work stealing framework. Such schedulers may be application specific and take into account, e.g., some particular dependencies between the loop iterations. Alternatively, in case of simulation codes with a serial outer loop doing time stepping, feedback based scheduling approaches may be used. That is the information about the execution time of the iterations may be collected and utilized for scheduling during the consequent executions of the loop.

The code in Algorithm 18-1 needs to be modified by the application developer to match the structure expected by the framework. The potentially concurrent tasks have to be encapsulated into subroutine calls. Algorithm 18-2 presents an example code as accepted by the framework. Note that *submit* calls are nested and framework does not limit nesting depth.

A thread that encounters a *submit* call creates a special data structure *QueueNode*. The submitted task queue is recorded in this data structure and the thread becomes the *owner* of the associated parallel region. The nested parallel regions can, therefore,

Algorithm 18-2 Pseudocode for a parallel program as accepted by the framework.

```
Subroutine: work_ij(i,j)
    Some code dependent on i, j;
Subroutine: work_i(i)
    Some code dependent on i;
    Create a task queue queue_ij wrapping work_ij for j between from_j(i)
    and to_j(i);
    Submit queue_ij;
    Some code dependent on i;
Application entry:
    Create a task queue queue_i wrapping work_i for i between from_i and
    to_i;
    Submit queue_i;
```

be represented with a tree of the *QueueNode* structures. The tree is dynamically changing following creation and termination of tasks within the corresponding task queues.

Figure 18-2 presents an example for such a tree. Here the queue $Q1$ with two concurrent tasks was submitted to the framework. Two different processors started to work on the tasks in parallel. Within both tasks new parallel queues $Q2$ and $Q3$ were created. Two more processors got some work and started to execute tasks $Q2.T2$ and $Q3.T2$ respectively. Next level parallel queues were created in tasks $Q2.T1$, $Q2.T3$ and $Q3.T1$. Since all the four available processors were already busy, all the tasks in $Q4$ and $Q6$ could be executed by a single processor. The tasks in queue $Q5$ were eventually distributed between 3 processors.

Figure 18-3 presents the time-line and stack scoping for the described example.

The QueueNode structures keep the following information:

- The ID of the *owner-thread*.
- Parent QueueNode references.
- A *runnable* flag indicating if the associated queue has some tasks left.
- An array of pointers to concurrently running child QueueNodes.

The QueueNode is allocated on the stack of the *owner-thread* and marks the stack frame to be restored after the parallel region is done. Owner-thread is therefore responsible for the destruction of the QueueNode. The different child QueueNodes may appear when two different threads executing two different tasks from the same queue both create QueueNodes. This is a typical case of multilevel dynamic parallel execution. Note that different children always have different owner-threads and therefore the maximum number of siblings is limited to the total number of running threads. The *parent* QueueNode exists for all the tree QueueNodes except the root

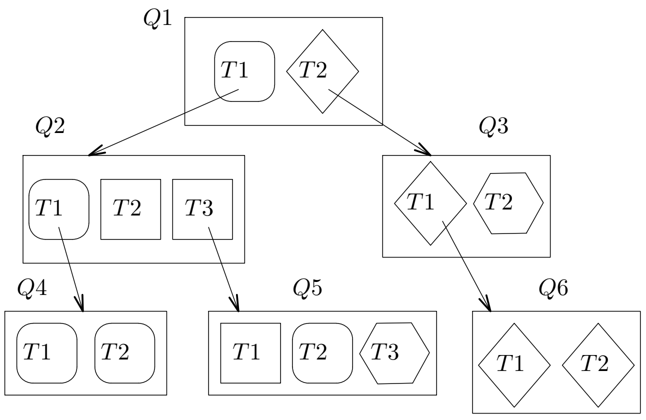


Figure 18-2: An example QueueNode tree executed on four processors. Different frames around individual tasks indicate different processors.

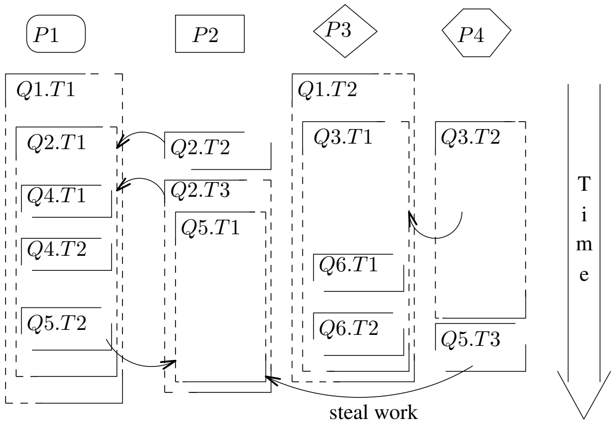


Figure 18-3: Execution sequence for the tasks depicted in Figure 18-2. Arrows indicate work stealing.

one. It provides a reference to the thread context that was active when a request for a new parallel region was encountered in the application.

The threads managed by the framework are arranged into a thread pool. The pool has a reference to the root QueueNode of the tree. The pool also contains an array of references to one QueueNode for every thread. The reference indicates the QueueNode currently processed by the corresponding thread. Initially all the references are empty.

A simplified pseudocode for the thread managing a newly created parallel region is presented in Figure 18-3. The application makes a request for a new parallel region by submitting a reference to a queue to the framework. A new QueueNode Q_n is created on the stack of the currently running thread (block 1). The reference to the submitted queue is stored in Q_n. If the new queue defines the top level parallel region the tree root pointer is set. Alternatively, the new QueueNode becomes a child to the thread's current QueueNode (block 2). The reference to the thread's current QueueNode is updated. The *run* method on the submitted queue is called (block 3). When the call returns the thread checks if some other thread had *stolen* some work from the queue. It then tries to search for the work available in the QueueNode subtree under Q_n and *steals* part of it (block 4). Breadth first search is utilized, i.e., all the QueueNodes at the higher level are checked before going down the tree. When no work is left the reference to the currently running QueueNode is restored (block 5). Finally, Q_n is destroyed and control returns to the application (block 6).

During an application run some worker threads are periodically *free*, i.e., they are not running application code. Those threads regularly look for a QueueNode with a runnable queue using breadth first search. If such a QueueNode is found, a thread tries to steal some work from the queue. It makes the QueueNode current and calls the *run* method on the queue. When the call returns the current QueueNode is marked as not runnable and the search is restarted.

Note that the reference to the current QueueNode is owned by each thread exclusively and no synchronization is necessary when updating it. Synchronization is however necessary for safe modification of QueueNodes. To avoid deadlocks the following locking protocol is used: a child QueueNode can only be locked if its parent QueueNode is locked. As soon as a child is locked the parent locking is released.

The breadth first search utilized by the algorithm ensures that the threads are processing the most coarse grained parallel region before starting with the fine level. This approach also favors the affinity of the tasks at the fine level since they are likely to be executed by a single thread.

No system level modifications to the call-stack of the threads are used. A call for a new parallel region made in the context of a thread always returns in the same context. This forces a thread that had some work stolen first steal some work back to ensure the fastest completion of the parallel region it owns. This can potentially result in extra synchronization when many threads are used. No big overhead is expected when the number of threads is low. Note that the presented solution is different from other work-stealing approaches where the 'victim' processor is chosen randomly.

Algorithm 18-3 An algorithm for an owner-thread running a parallel region (simplified pseudocode)

```
1 { Create a QueueNode $Q_n$ on stack;

2 { Get the current QueueNode $Q_p$ from the thread pool data;
    if there was no $Q_p$ then
        make $Q_n$ the root QueueNode for the thread pool;
    else
        set a child pointer in the $Q_p$ to point to $Q_n$;
    end if

3 { Set the new QueueNode $Q_n$ as currently running;
    Run the queue in the new QueueNode $Q_n$;
    Mark $Q_n$ as non-runnable;

4 { while other threads are running on the queue in $Q_n$ do
        while current QueueNode has child QueueNodes do
            Search a runnable queue in the QeueuNode tree from $Q_n$
            if a QueueNode $Q_c$ with runnable queue is found then
                Set $Q_c$ as currently running
                Run on the the queue associated with $Q_c$ (steal back)
                Set $Q_n$ as currently running
            end if
        end while
    end while

5 { if there exists a parent QueueNode $Q_p$ then
        Make the QueueNode $Q_p$ current;
        Remove the reference to $Q_n$ from $Q_p$;
    else
        Clear the root QueueNode reference for the thread
        pool;
    end if

6 { Destroy $Q_n$ and return;
```

Chapter 19

Feedback Based Loop Scheduling Strategies

19.1 Motivation

Most simulation codes for analysis of time-dependent dynamic processes have a serial outer loop doing time stepping. Pseudocode is presented in Algorithm 19-1.

Algorithm 19-1 Simplified pseudocode for a time-dependent process simulation.

```
i = 0; t_i = time_start;
while t_i < time_end do
    Process and update the system state at time t_i;
    Save output data if necessary;
    Evaluate time step length dt;
    i = i+1; t_i = t_i + dt;
end while
```

The system state typically completely defines the system configuration at time t_i. The only part of the loop that is feasible to parallelize for our application is the processing of the system state. This processing for mechanical systems consists of three stages. First, analysis of the relative motion between components is done. Then different interactions are processed and the resulting forces are calculated. Finally, the accelerations are computed and a new state is generated.

The important physical aspect for mechanical systems is handling of collisions between different bodies in the system and calculation of the contact forces. Very often detection of the collision and evaluation of the reaction force constitutes the most computationally expensive part of the simulation code.

Processing of contacts typically uses a multi-level approach. First a coarse grid is used to detect if bodies are in contact at all. If there is no contact the further processing is not necessary and zero forces are returned. Otherwise, a detailed geometry analysis with a finer grid is performed. At this point the exact geometry of the contact area is calculated. Finally, if the contact area is not zero, the reaction forces are computed by calculating the pressure and integrating it over the contact area.

Obviously, the same contact can take different time to compute depending on its area. The variations of computational time can reach several orders of magnitude depending on the analysis details and the algorithms used.

Fortunately, the changes in computational complexity between consequent time steps are likely to be small due to the physical nature of the model. If a contact was detected at time step t_i it is likely to be detected at the next time step as well. Hence, timing the contact calculation provides a good estimation of the computation efforts for the following iteration.

It is quite hard to partition a single contact calculation task into smaller tasks due to the nature of the mathematical models. For advanced contact models strong dependencies between different parts of the contact calculation algorithm can be identified. Another natural property of a contact calculation loop is the need to access the same geometry information in the iterations following after each other.

To summarize the discussion above we can identify the following computational properties of the calculation loops defined in Section 16 Algorithm 16-1:

- Irregularity and potentially strong variation of the computational time between inner loop iterations.
- Relatively small change in the computational time between two consequent time steps.
- Tight data dependencies between the most inner loop iterations.

Dynamic loop scheduling is a widely discussed research topic in programming of parallel computers. Any parallel programming conference has contributions discussing the scheduling schemes for different applications.

For the applications with an outer serial loop it is possible to record parameters of the parallel execution at time step i and use this information to optimize scheduling at step $i + 1$.

Many classical algorithms with a centralized task queue such as guided self-scheduling [21], factoring [14] and trapezoid self-scheduling [31] do not utilize this information.

Some approaches (see, e.g., [19, 32, 26]) utilize the fact that processor executing a chunk of iterations at time i is likely to have the data relevant for those iterations at the next time step $i + 1$ in the local cache. In such a case the iterations are said to have an *affinity* to the processor. Therefore scheduling that respects the affinity improves performance.

The designers of Feedback Guided Dynamic Loop Scheduling (FGDLS) [3, 4] demonstrated that explicit use of feedback information can significantly improve execution times. The further data on the performance of the algorithm was presented in [29].

The idea of the algorithm is to split the iteration space exactly into P *chunks*, where P is the number of processors. The parallel execution can then be performed as:

```
for $j$ = 1 to $P$ in parallel do
  Record start time $T_a$;
  work($x_j$, $x_{j+1} - 1$);
  Record stop time $T_b$;
  $t_j$ = $T_b$ - $T_a$;
end for
```

where `work`(x_j, $x_{j+1} - 1$) encapsulates the body of the original loop.

The scheduling tries to optimize the chunk sizes to balance computations between processors. The main idea is to approximate the real work-load distribution over iteration space with a piece-wise constant function. That is for any loop iteration `i` belonging to a chunk `j` its execution time is assumed to be $t_j/(x_{j+1} - x_j)$.

With such an assumption a scheduling algorithm that has complexity that depends only on the number of chunks and not on the number of loop iterations can be constructed.

Note, that in a traditional formulation of parallel loops for dynamic scheduling only the body of the original loop could be included into the `work` function. Hence the function needed to be called for each loop iteration (alternatively, each of the small chunks). Grouping variable number of iterations in a single function call reduces the required overhead making its linear in the number of chunks and not the number of iterations. Additionally, this improves the data locality within the chunks.

Let us now provide a pseudocode for the basic FGDLS algorithm (see Algorithm 19-2).

One important question regarding the FGDLS algorithm concerns its convergence. This was studied in a number of papers [30, 29, 28]. The basic condition for the global convergence of the algorithm may be formulated as a requirement for limited variation of the work-load between the iterations of the loop.

The original FGDLS algorithm and its modifications for nested loops (see, e.g., [11]) are not directly suitable for a work-stealing framework like the one presented here. The problem is that the algorithm assumes that all the processors are dedicated to the computational loop to be scheduled. In a work-stealing environment this is only the case for the most outer loop. An inner loop scheduler knows the maximum number of processors working, but it often happens that the higher level scheduler dedicates fewer processors to each particular inner loop. Further more, since the environment is completely dynamic different processors rarely enter a loop simultaneously. To cope with the specifics of work-stealing approach the original algorithm had to be modified. The modified algorithm is presented in Section 19.4.

Algorithm 19-2 Pseudocode for the basic Feedback Guided Dynamic Loop Scheduling (FGDLS)

Given:

A set of feedback records $\{x_j, t_j\}$, $j \subset [1, P]$, where:

P - is the number of processors running,

x_j - start iteration index for chunk j; sets $x_j, ..., x_{j+1} - 1$ for $j \subset [1, P]$ form a partitioning of the iteration set $1, ..., n$,

t_j - time to run the iterations $i \subset [x_j, x_{j+1})$.

Proceed:

Calculate the ideally balanced work load per processor as $w^* = \frac{1}{P}\sum_{j=1}^{P} t_j$.

Approximate the real workload distribution over the iterations $W^*(i)$ with a piecewise constant function $W(i)$:

$W(i) = W_j(i) = \frac{t_j}{x_{j+1} - x_j}$ for $i \subset [x_j, x_{j+1})$.

Find a new set of start iteration indices x_j^+ such as the work-load represented by $W(i)$ is distributed as evenly as possible between the processors:

```
$x_1^+$ = 1;
**for** each chunk $j \subset [2, P]$ **do**
    Find $k$, such as $\sum_{i=1}^{k} t_i \leq j * w^* < \sum_{i=1}^{k+1} t_i$;
    $x_j^+ = x_k + \lfloor \frac{1}{t_k}(x_{k+1} - x_k)(w^* * j - \sum_{i=1}^{k} t_i) \rfloor$.
**end for**
```

The algorithms presented in the following sections use the feedback information to reschedule loop iterations into large chunks. The scheduling is done once per iteration of the serial outer loop. The chunks are then wrapped in a *queue* interface and submitted to the framework as described in Section 18.2.

19.2 Minimal Feasible Parallel Region

In our experience one important consideration when designing a dynamic strategy was elimination of the short parallel regions. Running in parallel at a very fine grain level often leads to performance degradation due to synchronization overhead and caching effects. The basic requirement for a parallel code is to improve the wall-clock time when compared to the original serial code. Therefore, it is very important to reduce the possibility for such performance degradation.

The basic criterion can be formulated as follows:

$$\max(k_{\text{slow}}\frac{t_{\text{tot}}}{N_{\text{PE}}} + t_{\text{sched}}, t_{\text{min}}) < t_{\text{tot}} \tag{19-1}$$

where

- k_{slow} - a coefficient reflecting non ideal speed up when running the loop in parallel.
- t_{tot} - estimated total serial run-time for the loop.
- N_{PE} - number of physical processors on every node indicating the maximum possible speedup.
- t_{sched} - time for scheduling (often negligible).
- t_{min} - minimal feasible size of a parallel region, architecture dependent.

The k_{slow} coefficient was introduced to cope with fact that the linear speedup is rarely reached. The ideal value for the coefficient is obviously 1.0. Simple profiling showed that the best speedup achieved for the computation expensive loops for our application on two processors was around 1.8. This is primarily due to the communication overhead. Therefore, for our application the value of k_{slow} was chosen to be 1.2. Profiling is necessary to determine the optimal coefficient for the particular architecture and application. This profiling can be automated by varying the coefficient between 1.0 and N_{PE} and choosing the value of k_{slow} that gives the best wall-clock time.

Similar profiling strategy may be used to identify t_{min}.

19.3 A Simple Feedback Based Scheduling

The scheduling strategy presented here is suitable for loops with the following properties:

- The number of iterations is moderate.
- The order of loop iterations is not important (two consequent iterations access different data structures), i.e., affinity does not give any performance gain.
- Each loop iteration is heavy enough to be timed separately.
- Good load balancing is important, i.e., computation to communication rate is high.

In our application this strategy was used for scheduling over the outer loops: over the pairs of bodies and over the contacts.

Note that having timing information for each iteration effectively converts the original loop scheduling strategy into scheduling of a simple task graph with known times for each task.

Algorithm 19-3 presents the pseudocode for the strategy.

Algorithm 19-3 A simple feedback based scheduling.

```
Given:
    Execution time for each loop iteration.
Proceed:
    Sort iterations in descending order according to the execution time.
    For each iteration $i$, starting from the slowest one:
    Assign the iteration to the chunk that currently has the lowest execution time.
```

19.4 Nested Feedback Guided Parallel Loop Scheduling

We have discussed in Section 19.1 that the basic FGDLS algorithm cannot be used in a work-stealing environment. Here we present a generalization to the basic algorithm that eliminates this limitation.

In addition to the timing information for the execution of the loop iterations we require tracking of the number of processors actually used to run the loop. For each such processor the time when it enters T_j^e the loop is recorded and compared to the

Algorithm 19-4 Pseudocode for the FGDLS with variable processors entry time.

```
Given:
    Two sets of feedback records: $\{x_j, t_j\}, j \subset [1, P_o]$ and $\{\Delta t_j\}, j \subset [1, P_n - 1]$,
    where:
        $P_o$ - number of chunks prepared for parallel run of the loop,
        $P_n$ - number of processors that actually had resources to run the loop,
        $x_j$ - start iteration index for a chunk $j$,
        $t_j$ - time to run the iterations in chunk $j$,
        $\Delta t_j$ - the time delay between the moments when processors $j$ and $j + 1$
        actually became available for processing the loop. Note that for this algo-
        rithm the processors are numbered in the order they enter the particular
        loop.
Proceed:
    Calculate the total amount of work in the loop as $W = \sum_{j=1}^{P_o} t_j$
    Find the ideal work load for each processor $w_j^*, j \subset [1, P_n]$ using the available
    set of records ($\Delta t_j$):
        Find the ideal load for the last processor:
        $w_{P_n}^* = \frac{1}{P_n}(W - \sum_{j=1}^{P_n-1} j\Delta t_j)$
        for $j = P_n - 1$ down to 1 do
            $w_j^* = w_{j+1}^* + \Delta t_j$
        end for
    Approximate the real workload distribution over the iterations $W^*(i)$ with a
    piecewise constant function $W(i)$:
        $W(i) = W_j(i) = \frac{t_j}{x_{j+1} - x_j}$ for $i \subset [x_j, x_{j+1}),\ j \subset [1, P_o]$.
    Find a new set of start iteration indices $\{x_j^+\}, j \subset [1, P_n]$ such as the work-
    load represented by $W(i)$ is distributed as close as possible to the ideal load
    for each processor:
        $x_1^+$ = 1;
        for each chunk $j \subset [2, P_n]$ do
            Find $k$, such as $\sum_{m=1}^{k} t_m <= \sum_{m=1}^{j} w_m^* < \sum_{m=1}^{k+1} t_m$
            $x_j^+ = x_k + \lfloor \frac{1}{t_k}(x_{k+1} - x_k)(\sum_{m=1}^{j+1} w_m^* - \sum_{j=1}^{k} t_j) \rfloor$
        end for
```

entry time of the previous processor T^e_{j-1}. The difference in the entry times is stored and used for scheduling. Note that for this algorithm the processors are numbered in the order they enter the particular loop. That is the numbering is loop specific.

The original FGDLS algorithm tries to make execution time for each processor equal. The generalized algorithm tries to make the load of each processor proportional to the time the particular processors actually spends working on the loop. Pseudocode for the algorithm is provided in Algorithm 19-4.

Note that when all the processors enter the loop simultaneously the modified algorithm generates the same schedules as the original FGDLS algorithm.

19.5 Implementing Feedback Based Scheduling in OpenMP

OpenMP does not provide facilities for custom loop schedulers. In order to realize the presented feedback strategies the following code could be used:

```
#pragma omp parallel for
for(int i = 0; i < scheduler->get_n_buckets(); i++){
        scheduler->get_bucket_from_to(i, &from, &to);
        scheduler->start_time(i);
        work(from,to);
        scheduler->stop_time(i);
}
```

Here we assumed that a special `scheduler` class has been designed to implement the presented scheduling strategies.

Since the feedback base strategy is general and can be used in many applications it might be possible to extend OpenMP to support it. The following syntax can for instance be accepted:

```
#pragma omp parallel for schedule(feedback(fr))
```

where the `fr` argument provides a reference to an `OMP_FEEDBACK_RECORD` data structure stored by the application code.

The OpenMP subsystem could then provide a function:

```
OMP_FEEDBACK_RECORD* omp_init_feedback_record(<arguments>);
```

that would initialize the structure. The function could also accept additional arguments specifying the method for storing timing records. With the algorithms presented here two methods would be available: `FGDLS` and `SIMPLE`.

Chapter 20

Test Simulation Results

The results presented here were obtained on a 40-nodes Linux cluster (SuSE 9.2 Linux distribution). Every compute node of the cluster has two 2.4 GHz AMD Opteron 250 processors and 2GB primary memory. The nodes are connected via a GigabitEthernet network.

Short simulations of an SRB (spherical roller bearing) and a DGBB (deep groove ball bearing) bearings were chosen to make test runs of the presented hybrid algorithms. Profiling the simulation for the SRB case showed that calculations attributed to a single one out of total 275 contacts were dominating. Granularity in this case directly becomes the key speedup limiting factor and reduces the maximum possible speedup to be lower than 2. For the DGBB case there are 9 relatively expensive contacts out of the total 27. In this case granularity becomes an important issue only when the number of processors used is greater than 10.

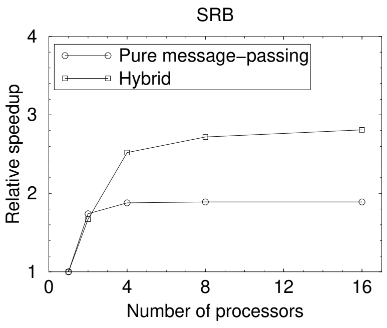
Table 20-1 presents the execution times for the test simulations on a cluster with dual-processor nodes. The hybrid algorithm was run with two threads per-node. Corresponding relative speedup curves are presented in Figure 20-1.

We can see that the hybrid approach achieves about the same performance as the distributed memory one when running on a few nodes. Hybrid code gains obvious advantage when the load imbalance caused by the task granularity limits the speedup of the pure message-passing code.

In fact, this result could be formulated into a simple rule for the mechanical engineers using the simulation tool. They were asked to identify the number of potentially large contacts N_c. It was then suggested to use the hybrid code on $N_c + 1$ dual-processor nodes. Users reported the decrease of wall-clock run time by a factor 1.5-1.8 as compared to the original pure message-passing code.

N_P	Pure message-passing on N_P processors		Hybrid ($N_P/2$ dual-processor nodes)	
	SRB	DGBB	SRB	DGBB
1	42270	737		
2	24304	367	25307	431
4	22419	202	16757	231
8	22256	148	15505	145
16		148	15039	111
32				100

Table 20-1: Execution times (seconds) for the test runs.



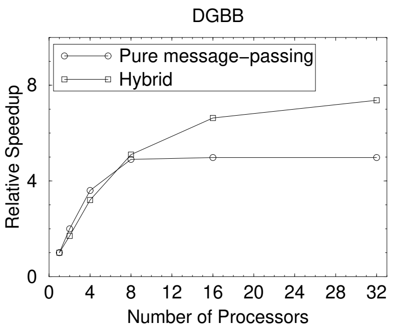


Figure 20-1: Relative speedup for the test runs.

Chapter 21

Conclusions

We have analyzed the applicability of the hybrid approach for different applications and identified its feasibility for the solution of task granularity related problems.

Then we presented an approach to hybrid parallelization of multibody dynamics simulation code with detailed contact models.

The original message-passing code of the simulation toolbox has been complemented with an object-oriented shared-memory framework that includes a multilevel scheduler implementing work-stealing strategy and two feedback based loop schedulers. The main scheduling algorithm of the framework is designed to be easily portable and can be implemented without any system level coding or compiler modifications. Of the two described loop scheduling algorithms one algorithm requires timing data for each loop iteration and is suitable for loops with minimal dependencies between the iterations. If the iterations have more significant dependencies the second algorithm, which is a generalization of the Feedback Guided Dynamic Loop Scheduling (FGDLS) [3] should be used. The generalized algorithm makes FGDLS suitable for a work-stealing framework by tracking the time that different processors spend working on the loop.

The presented test results demonstrated that the hybrid code outperforms the pure message-passing implementation increasing the speedup limit by 1.8 times on a cluster of two-processor SMP nodes for some cases. The theoretically maximum improvement, in this case 2, could not be archived since only parts of the application are actually parallelized at fine-grain level. Further studies of different shared memory scheduling algorithms and their combinations at different levels are necessary to archive further speedup improvements.

Another question that needs further investigation is an automatic detection of the best parallelization approach to be used on a particular compute node without *a priori* knowledge of the computational problem and, possibly, dynamic change of the strategy. Certainly, one can periodically try different approaches during the simulation run and choose the better to be used for a number of simulation steps (an approach similar to that presented in [10]). However, it is currently technically complicated to switch from the multi-thread enabled code to a pure message-passing code. Ideally, such switching requires two versions of the code to be available simultaneously. One version should be optimized for single-thread execution within the message-passing framework, while the other should implement the shared-memory parallelization.

Bibliography

[1] Brian Armstrong, Seon Wook Kim, and Rudolf Eigenmann. Quantifying Differences between OpenMP and MPI Using a Large-Scale Application Suite. In *ISHPC '00: Proceedings of the Third International Symposium on High Performance Computing*, pages 482–493, London, UK, 2000. Springer-Verlag.

[2] OpenMP Architecture Review Board. OpenMP - C and C++ Application Program Interface . In *www.openmp.org*, 1998.

[3] J. M. Bull. Feedback guided loop scheduling: Algorithms and experiments. In *Proceedings of Euro-Par'98*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[4] J. M. Bull, R. W. Ford, T. L. Freeman, and D. J. Hancock. A theoretical investigation of feedback guided loop scheduling. In *Proceedings of 9-th SIAM Conference on Parallel Processing for Scientific Computing*. SIAM Press, 1999.

[5] Franck Cappello and Daniel Etiemble. MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 12, Washington, DC, USA, 2000. IEEE Computer Society.

[6] Barbara M. Chapman, Amit Patil, and Achal Prabhakar. Performance Oriented Programming for NUMA Architectures. In *WOMPAT '01: Proceedings of the International Workshop on OpenMP Applications and Tools*, pages 137–154, London, UK, 2001. Springer-Verlag.

[7] Computer Science and Engineering, University of Washington. Simultaneous Multithreading Project. *http://www.cs.washington.edu/research/smt/*.

[8] Suchuan Dong and George Em. Karniadakis. Dual-level parallelism for deterministic and stochastic CFD problems. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–17, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[9] D. Fritzson, P. Fritzson, P. Nordling, and T. Persson. Rolling Bearing Simulation on MIMD Computers. *The International Journal of Supercomputer Application and High Performance Computing*, 11(4), 1997.

[10] Dag Fritzson and Patrik Nordling. Adaptive scheduling strategy optimizer for parallel rolling bearing simulation. *Future Gener. Comput. Syst.*, 16(5):563–570, 2000.

[11] D. J. Hancock, R. W. Ford, T. L. Freeman, and J. M. Bull. An Investigation of Feedback Guided Dynamic Scheduling of Nested Loops. In *ICPP '00: Proceedings of the 2000 International Workshop on Parallel Processing*, page 315, Washington, DC, USA, 2000. IEEE Computer Society.

[12] Yun He and Chris H. Q. Ding. MPI and OpenMP paradigms on cluster of SMP architectures: the vacancy tracking algorithm for multi-dimensional array transposition. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–14, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[13] D. S. Henty. Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 10, Washington, DC, USA, 2000. IEEE Computer Society.

[14] Susan Flynn Hummel, Edith Schonberg, and Lawrence E. Flynn. Factoring: a method for scheduling parallel loops. *Commun. ACM*, 35(8):90–101, 1992.

[15] Jürg Hutter and Alessandro Curioni. Dual-level parallelism for ab initio molecular dynamics: reaching teraflop performance with the CPMD code. *Parallel Comput.*, 31(1):1–17, 2005.

[16] Géraud Krawezik. Performance comparison of MPI and three OpenMP programming styles on shared memory multiprocessors. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 118–127, New York, NY, USA, 2003. ACM Press.

[17] Yonezawa Laboratory. StackThreads/MP Home Page. *http://www.yl.is.s.u-tokyo.ac.jp/sthreads/*.

[18] LMS International homepage. *http://www.lmsintl.com/*.

[19] E. P. Markatos and T. J. LeBlanc. Using processor affinity in loop scheduling on shared-memory multiprocessors. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 104–113, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[20] MSC Software Corporation homepage. *http://www.mscsoftware.com/*.

[21] C. D. Polychronopoulos and D. J. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Trans. Comput.*, 36(12):1425–1439, 1987.

[22] IEEE. Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application: Program Interface [C Language]. 9945-1:1996 (ISO/IEC) . In *IEEE/ANSI Std 1003.1 1996 Edition*, 1996.

[23] Steven Saunders and Lawrence Rauchwerger. ARMI: an adaptive, platform independent communication library. *SIGPLAN Not.*, 38(10):230–241, 2003.

[24] Hongzhang Shan, Jaswinder P. Singh, Leonid Oliker, and Rupak Biswas. Message passing and shared address space parallelism on an SMP cluster. *Parallel Comput.*, 29(2):167–186, 2003.

[25] SIMPACK homepage. *http://www.simpack.de*.

[26] Srikant Subramaniam and Derek L. Eager. Affinity scheduling of unbalanced workloads. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 214–226, New York, NY, USA, 1994. ACM Press.

[27] MIT Supercomputing Technologies Group. The Cilk Project. *http://supertech.lcs.mit.edu/cilk/*.

[28] Sabin Tabirca, Tatiana Tabirca, and Laurence T. Yang. Convergence of the Discrete FGDLS Algorithm. In *Proceeding of the International Symposium on Parallel and Distributed Computing*, volume 3726 of *Lecture Notes in Computer Science*, pages 233–244. Springer-Verlag, Oct 2005.

[29] T. Tabirca, S. Tabirca, L. Freeman, and T. Yang. Feedback guided dynamic loop scheduling: A theoretical approach. In *Proceedings of 3rd Workshop on High Performance Scientific and Engineering Computing with Applications (HPSECA 2001)*. IEEE Computer Society Press, 2001.

[30] T. Tabirca, S. Tabirca, L. Freeman, and T. Yang. Feedback Guided Dynamic Loop Scheduling: Convergence of the Continuous Case. *Journal of Supercomputing*, 30:151–178, 2004.

[31] T. H. Tzen and L. M. Ni. Trapezoid self-scheduling scheme for parallel computers. *IEEE Transaction on Parallel and Distributed Systems*, 4(1):87–98, 1993.

[32] Yong Yan and Xiaodong Zhang. Adaptively Scheduling Parallel Loops in Distributed Shared Memory Systems. *IEEE Transaction on Parallel and Distributed Systems*, 8(1), Jan 1997.

Part III

TLM Based Coupled Simulation Framework

Chapter 22

Introduction

22.1 Motivation

In the area of modeling and simulation of mechanical systems one can identify many different classes of models and corresponding tools. Specialization leads to different focus for different tools. For instance, consider the differences in focus for the equation-based multi-physics Modelica models [16], general multibody models in MSC.ADAMS [17], models with detailed contact definitions in SKF's BEAST, flexible components as modeled in FE tools [2, 1], etc.

There is no single universal tool that can be used to analyze all kinds of problems with maximum efficiency and accuracy. One might say that every tool is optimized for a certain kind of task. This has lead to the creation of a large number of different specialized simulation frameworks. Large industrial models often use some special features of the particular framework which renders the translation of models between different frameworks very complicated and costly.

In reality the different mechanical components modeled in different tools are dependent on each other. Two components that are in physical interaction, form boundary conditions for each other and some interface can often be defined.

Unfortunately it is often the case that the different classes of tools are used independently. Every class of tool uses approximations of the components it has interface with, that is, simplified models of the boundary conditions. Several time consuming iterations are often necessary to make the components converge to similar values on on the common interface. The limitations on modeling accuracy are thus fundamental.

The need to bring different components into a complete more tightly coupled simulation is therefore justified. This allows higher accuracy and preserves component investments.

Co-simulation is one of the possible techniques of joining existing sub-models into a more complete model. The joining technique is very different depending on the particular application area (see, e.g, surveys in [7, 10, 21]).

One method used earlier to enable closer interaction between such sub-models in a coupled simulation is *transmission lines modeling* (TLM). The TLM uses physically motivated time delays to separate the components in time and enable efficient co-simulation. The technique has proven to be stable and was implemented for coupling of different sub-systems [11, 18, 19, 20].

The TLM system should allow two or more different tools to perform a coupled transient simulation of complex systems. The idea is to use different tools to model and simulate different subsystems (e.g., different parts of a mechanical product) and TLM technique to allow dynamic information exchange between the tools. The choice of a tool for a particular subsystem is determined by user needs and tools features.

To give a concrete example, let us consider modeling of a vehicle. MSC.ADAMS seems to be a 'standard' tool for this field. However, ADAMS models are often quite coarse and mostly handle interactions between larger components. Specialized tools are better suited when detailed modeling of some specific components is of interest. Wheel hub units modeling is important for SKF as the manufacturer of such components. Proprietary SKF's BEAST allows much more accurate modeling of the hub units than ADAMS, while rebuilding the complete vehicle model in the specialized tool is certainly an overkill. So far, the only possibility for the tools to cooperate has been to record the load data during ADAMS simulation and then use it as input to the BEAST simulation of the component. The response of the hub can then be recorded and submitted back to ADAMS. Such iterative approach is very time consuming and introduces additional modeling error since data is transferred only after a complete simulation. In a TLM co-simulation information is transferred during the simulation run giving more and earlier accurate results.

22.2 Related Work

So far, co-simulations have gained most acceptance for mechatronic applications. Developers of control systems routinely utilize co-simulations to test control algorithms. The most common environment that serves as a coupling framework is SimuLink [14]. Simulation systems that support multibody dynamics simulations like MSC.ADAMS [17] or Dymola [4] provide special modules for running such co-simulations. Other competitive environments like CosiMate [3] are also available. The central issue in such simulations is often real-time performance since the models are eventually used for the hardware-in-the-loop simulations.

High Level Architecture (HLA) standard [12] is designed with the vision of distributed simulations in mind. The goal here is to simulate a complete environment with many different actors and tools. The main concerns are not simulation methods but protocols that allow individual actors to communicate with the environment.

The focus of this paper is different. Our intention is to couple transient simulation of mechanical components. The main issues in this case are stability and accuracy of the coupling method.

Let us provide a brief survey of the methods used for coupling of dynamics simulations. Our intent is to further motivate the choice of the TLM technique for the framework. A wider recent survey can be found in [10]. The same paper provides a classification of some of the co-simulation strategies depending on the interface variables exchanged between the sub-models and the co-simulation coordinator. The problem with setting kinematic variables into existing simulation codes is identified. This motivates the preference for approaches that only compute reaction forces. The authors also classify the time stepping methods typically used in co-simulations. The parallel time-stepping where the compatibility between the sub-systems is achieved at every global time step is identified as the most attractive.

Glue code [5, 6] is a co-simulation framework developed for connecting MSC.ADAMS to other codes with discrete or continuous time stepping. It uses quadratic interpolation to pass information about the ADAMS model to the other code and quadratic extrapolation for the force coming into the ADAMS model.

The latter article introduces a concept of *interface mass*. The interface mass is simulated in both coupled codes and gets different reaction forces in different simulations. The reaction forces are communicated between the tools.

The coupling framework was successfully used in coupling an ADAMS model to a flight simulator. The problem with this coupling approach is the lack of study on the numerical stability of the method which leads to the necessity for additional 'tweaking' for each particular case.

The paper [15] is mostly focused on fluid-structure interaction problems but the results can be applied to other kinds of co-simulations as well. The paper analyses different numerical coupling methods and a new block-Newton method is proposed. The discussion here is focused on global solution of implicitly coupled differential equations. The introduction of the global iterative solver requires close interaction with the numerical solvers in the sub-models and may be hard to implement for existing simulation codes.

A similar problem arises for the gluing algorithm presented in [10] which relies on information available at the subsystem interfaces but requires a special evaluation of the coupling matrices. This, again, may be hard to implement in an existing environment.

In [9] a control block, called Boundary Condition Coordinator, is used to minimize the difference in the coupled variables. The mismatch in the conditions does not have any physical meaning and should be seen as a numerical artifact. Additionally the paper does not address the problems related to the use of multistep implicit solvers for the sub-systems.

The TLM technique combines the following attractive properties:

- Very clear and small interface to the sub-models where kinematic data is only read and the response is introduced via an external reaction force and torque.
- Proven unconditional stability of the coupling method.
- Coupling parameters are physically motivated and no additional numerical error is introduced.

The outlined combination of characteristics makes the technique an attractive candidate for a co-simulation framework. The mentioned properties of TLM coupling are further discussed in Chapter 23.

Chapter 23

TLM background theory

The TLM (*Transmission Line Modeling*) method, also called Bilateral Delay Line Method [11], exploites the fact that all physical interactions in nature have finite propagation speed. The properties of the delay lines were studied in [13]. The method is briefly described below.

A basic one-dimensional transmission line is shown in Figure 23. For the mechanical case the line is basically an ideal elastic medium with force waves c_1 and c_2 going between it ends. The input disturbances are velocities v_1 and v_2 and the forces from the transmission line F_1 and F_2.

Note that the springs in our implementation are assumed to be isotropic. That is no cross-term waves are generated when working in 2D and 3D. See [18] for further discussions.

If the line delay is set to T_{TLM} and its impedance to Z_F then the govering equations are:

$$\begin{aligned} c_1(t) &= F_2(t - T_{TLM}) + Z_F\ v_2(t - T_{TLM}) \\ c_2(t) &= F_1(t - T_{TLM}) + Z_F\ v_1(t - T_{TLM}) \\ \\ F_1(t) &= Z_F\ v_1(t) + c_1(t) \\ F_2(t) &= Z_F\ v_2(t) + c_2(t) \end{aligned} \tag{23-1}$$

The equations show that the two simulation systems are decoupled with the delay

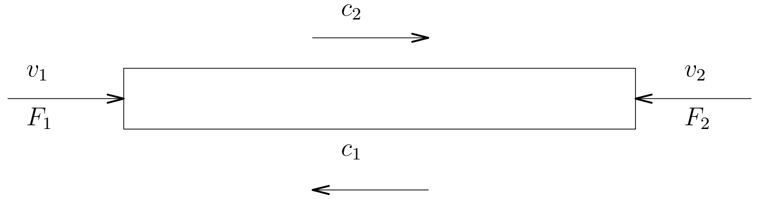


Figure 23-1: Delay line with the passing wave variables c_1 and c_2 and velocity variables v_1 and v_2.

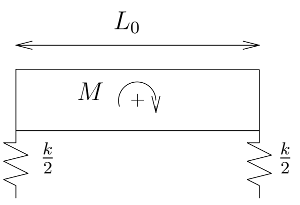


Figure 23-2: Estimating the rotational stiffness.

time T_{TLM}. Simulation framework can utilize this decoupling to enable efficient communications during co-simulation.

Representing the TLM connection with a simple model of a steel beam, the stiffness coefficient can be computed as (see [18]):

$$k = \frac{EA}{L_0} \tag{23-2}$$

where E is Young's modulus, A is the cross section area and L_0 the length of the beam.

The impedance Z_F has a relation to the spring constant k, $Z_F = kT_{TLM}$. The impedance factor can then be formulated as a function of the area and length of the steel rod according to

$$Z_F = \frac{EAT_{TLM}}{L_0} \tag{23-3}$$

To get the stiffness and impedance for the rotational degrees of freedom one can use the already computed stiffness k. If the arrangement depicted in Figure 23 is assumed, then:

$$k_\phi = \frac{M}{\delta_\phi} = 2\frac{(k/2)\delta_\phi(L_0/2)^2}{\delta_\phi} = \frac{kL_0^2}{4} \tag{23-4}$$

and the impedance for the rotation:

$$Z_{FR} = \frac{1}{4}Z_F L_0^2 \tag{23-5}$$

The time constant T_{TLM} can be computed using the speed of sound for the medium:

$$T_{TLM} = \frac{L_0}{v_{medium}} \tag{23-6}$$

It can be shown that the TLM element also introduces a (*parasitic mass*) that can be viewed to be outside the simulated system [18]. The total mass for the combined systems must therefore also include the parasitic mass of the TLM element in order to make, e.g., the energy conservation formulas correct. This mass depends on the impedance factor and the time delay factor

$$m_p = Z_F T_{TLM} \tag{23-7}$$

This implies that if the impedance factor Z_F is increased, the parasitic mass will increase if the synchronization delay T_{TLM} is not decreased. If the parasitic mass is large it may influence the system behavior and can not be neglected. Note, that for the simple beam case when TLM parameters are computed according to Equations 23-3 and 23-6 the parasitic mass is equivalent to the mass of the beam ($\rho AL0$, where ρ is the material density).

For practical purposes (see [8]) one can use the parameters of a material cube with an edge given by *characteristic distance* L_0. Equations 23-2 and 23-4 can then be used to compute the translational and rotational stiffnesses:

$$\begin{aligned} k &= \tfrac{EL_0^2}{L_0} = EL_0 \\ k_\phi &= \tfrac{kL_0^2}{4} = \tfrac{EL_0^3}{4} \end{aligned} \tag{23-8}$$

To give a concrete example, let us assume that connection medium is steel and the characteristic length is $L_0 = 0.1$. Steel has Young's modulus $E = 210GPa$ and the speed of sound in steel is $v_{steel} = 5180m/s$. The TLM parameters then can be computed:

$$\begin{aligned} T_{TLM} &= L_0/v_{steel} \approx 2 * 10^{-5} \\ Z_F &= EL_0T_{TLM} \approx 2 * 10^5 \\ Z_{FR} &= \tfrac{1}{4}Z_F L_0^2 \approx 500 \end{aligned} \tag{23-9}$$

Calculations like these give approximate values of the stiffness and the time delay of the TLM element. This gives a background for selecting the TLM line delay and impedance parameters. In cases when required T_{TLM} becomes a limiting factor, while the TLM link stiffness is much higher than the stiffnesses used in the sub-models, a lower stiffness and larger T_{TLM} may be considered.

The elastic medium that is modelled with the TLM element introduces oscillation frequencies (standing waves) given by:

$$f_{TLM,i} = \frac{i}{2\ T_{TLM}}, \quad i = 1, 2, 3, ... \tag{23-10}$$

The basic TLM model has no damping which can result in unwanted vibrations during simulation. In [18] a low pass filtering of the TLM charateristics is recommended:

$$c_{\text{filtered}}(t) = c_{\text{filtered}}(t - T)\ \alpha + c(t)\ (1 - \alpha) \tag{23-11}$$

The filtering is controlled by a damping constant α. The recommended value according to the paper is 0.2.

Chapter 24

The TLM Co-simulation Framework

24.1 Component Based Meta-Modeling and Simulation

In the area of modeling and simulation of mechanical systems one can identify many different classes of models and corresponding tools. It terms of meta-modeling every tool can be seen as a black-box handling a particular *component*. Where a component is a model defined in some specific language together with some modeling and simulation tool that can perform a transient simulation of it. The meta-model then is used to define the interconnections between the component.

When discussing co-simulation environment it is important to consider qualifications required for running a simulation. Simulation components belong to the specialized tools. Even though all tools have interfaces to external functions (see survey in the Appendix) the interfaces differ between tools. Therefore it is first necessary to have a software developer familiar with the particular tool architecture design and implement the meta-modeling interface for each tool. That is create a tool specific adaptor for the framework. The design of the framework should therefore be general and provide a clear interface that is easy to use from any specific environment. This work needs to be done only once and the results can be used in all the following simulations.

Next step is preparation of the isolated model for use in the co-simulation framework. At this point an expert user familiar with the particular tool is likely to be asked to modify the model so that it can be seen as a component. Basically the interface points in the model should be identified and named.

Once the components are ready other users should be able to include them in meta-models without considering the particular tools' specifics. The framework user have to be familiar with the meta-modeling notations and be able to define connection between components' interfaces. The most realistic assumption is that the meta-model user is an expert in one of the participating tools and has limited knowledge about the other tools. Installation of tools is done by a system administrator.

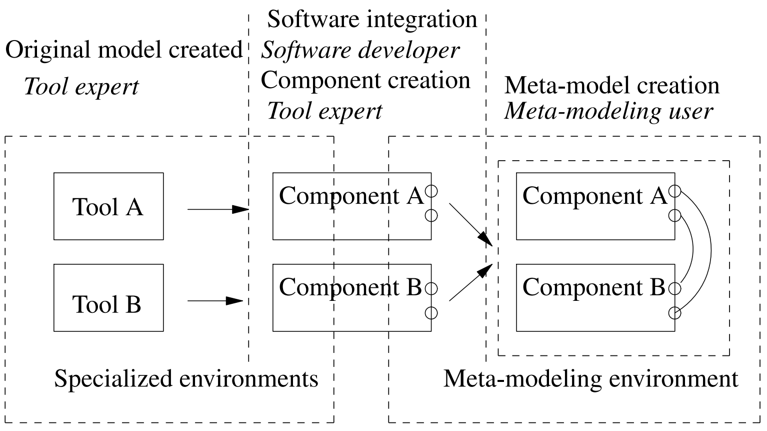


Figure 24-1: Modeling phases and necessary competences

Let us now summarize the roles and qualifications that are assumed in the architecture presented in this chapter:

- Software developer creates a specific tool adaptor for the framework. This work needs to be done once for each of the tools. Familiarity with *C*++ and specific tool's external interfaces is required.
- Expert user of a tool converts an isolated model into component with predefined and named interfaces. This needs to be done once for every component. Experience with the particular tool is required.
- Meta-modeling user creates a meta-model by connecting the interfaces of different components. Familiarity with meta-modeling is necessary. No specific tools knowledge is required.

Figure 24-1 illustrates the described modeling stages that are necessary for a co-simulation.

24.2 Simulation Framework Terminology

Two different system architectures were considered for the TLM co-simulation system: centralized one with an independent co-simulation manager tool and a decentralized, where different participating simulation tools communicate with each other in a peer-to-peer fashion. Both approaches have advantages and disadvantages. Centralized approach is easier to implement, gives more control over the co-simulation, makes it possible to create a separate meta-model defining the connection between the tools. It is, however, a bit less efficient from the performance point of view since the co-simulation manager might create a bottleneck in a large model.

Since currently co-simulations involving more than three tools are not expected (though supported), centralized approach was chosen.

To describe a model for a TLM co-simulation with several tools several compatible models need to be built. First, every tool should have a model of the components it is responsible for and define the interface points where the TLM connections are attached. After that the connections between tools need to be defined that is a *meta-model* should be created.

A terminology needs to be introduced to define concepts used in a TLM co-simulation context. With the background given in previous sections we can define:

- **TLM interface.** A named point on a mechanical object where position and velocity can be evaluated and a reaction force applied.

- **TLM manager.** The central simulation engine. It is a stand alone program that reads in a *Meta-model definition*. It then starts *Simulation components* and provides the communication bridge between the running simulations. That is the components only communicate with the TLM manager which acts as a broker marshalling information between the components as required by TLM theory. TLM manager sees every simulation component as a black box having one or several TLM interfaces. The information is then forwarded between TLM interfaces belonging different components.

- **TLM plug-in.** A *C++* library having a single abstract class representing TLM interface for a specific simulation tool. The TLM plugin can be seen by a simulation component as an external force/torque that depends on position, velocity and time. The implementation of the plugin handles the necessary communications with TLM manager.

- **Simulation component.** Any simulation program that has incorporated TLM plugin as a part of its code. A small script that takes the general parameters as input and starts the specific component is an additional requirement. This intermediate step is necessary since TLM manager needs a common way to start all the components and each tool might have some specific start procedures.

- **Meta-model definition** An XML-file describing

  - the *Simulation components* with there *TLM interfaces*;
  - TLM connections between the interfaces of different components;
  - simulation start, stop times and TLM manager port.

  XML syntax was chosen to describe meta-models since it allows many different general XML tools to be used directly to create and modify meta-models. An example meta-model definition is presented in Figure 24-3.

- **TLM data** Time-stamped force wave data and delayed position and orientation data that is communicated between *Simulation components*.

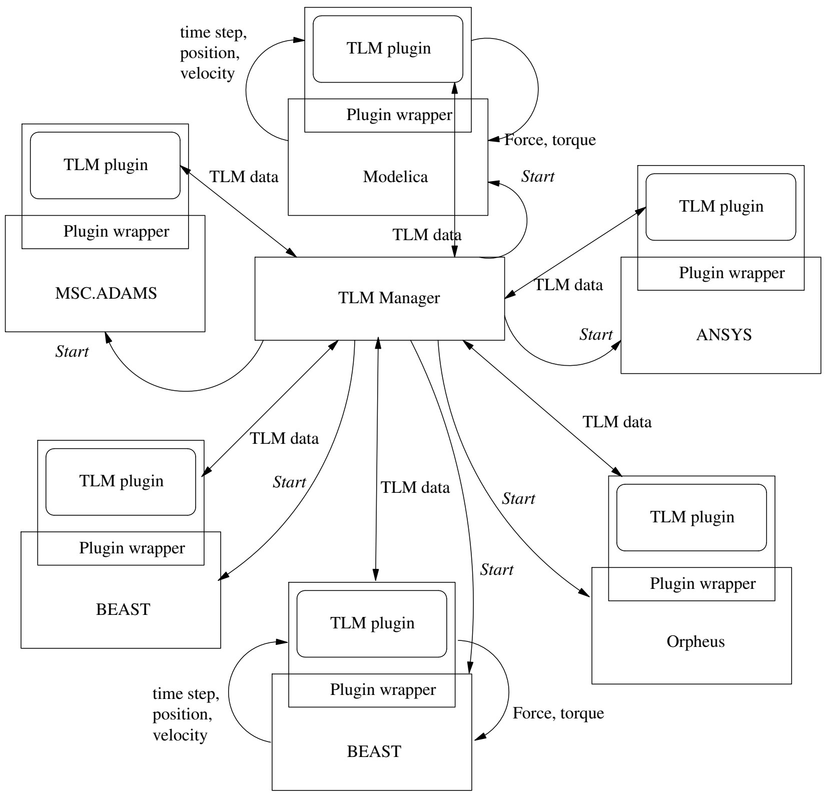


Figure 24-2: TLM system architecture. Currently Beast and ADAMS components are implemented and tested. Several instances of the same component can be running simultaneously.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- The meta-model definition file -->
<Model Name="SimpleAB">

    <!-- List of connected sub-models -->
    <SubModels>
        <SubModel Name="Dev_DGBB_NoC"
                StartCommand="StartTLMBeast"
                ExactStep="0">
            <!-- TLM interface points for SubModel A -->
                <InterfacePoint Name="TLMIR"/>
        </SubModel>
        <SubModel Name="Pendulum"
                StartCommand="startadams"
                ExactStep="0">
            <!-- TLM interface points for SubModel B -->
                <InterfacePoint Name="M9"/>
        </SubModel>
    </SubModels>

    <!-- List of TLM connections -->
    <Connections>
        <Connection From="Dev_DGBB_NoC.TLMIR" To="Pendulum.M9"
                Delay="1e-5" Zf="0.5e5" Zfr="500" alpha="0"/>
    </Connections>

    <!-- Parameters for the simulation -->
    <SimulationParams ManagerPort="11111"
                StartTime="0" StopTime="1e-3"/>

</Model>
```

Figure 24-3: An example meta-model definition in XML.

24.3 TLM co-simulation scenario

TLM co-simulation is initiated by TLM manager that reads in the meta-model defining the different sub-models and the connection between them. TLM manager tool then creates a server socket and start all the collaborating tools (see Figure 24-2). Note that multiple instances of the same tool can be started.

A start script or program should be written for every tool that needs to accept the following arguments:

```
StartToolX <Model> <from_t> <to_t> <Max_step> <Server>
```

where

- Model - typically gives the name of an input file for the sub-model;
- from_t and to_t specify the start and end time for the simulation;
- Max_step defines the maximum time step allowed for the tool. It is determined by the TLM delays in different connections to the tool;
- Server - gives the TCP/IP server name and port address that the TLM plugin should contact.

The start script should start its respective tool for the simulation in a batch mode. The tool needs then to initialize the TLM plugin (typically done via some tool specific wrapper code). It should then register the TLM interface points that are defined in the model. For every interface point TLM plugin will send a registration request to TLM manager to verify that the interface is defined in the meta-model and to get the TLM parameters (delay, impedance and damping) for this interface.

24.4 TLM Communication Modes

TLM communication mode depends on the properties of the numerical solvers used in the communicating tools. Some solvers have an option to produce output (make an integration step) with a predefined constant time frequency used as communication interval T_{comm}. Maximum allowed time step for such solvers can be set equal to the TLM delay. Note that this doesn't require a fixed time step solver. Solvers are allowed to change the step length within T_{comm} but no integration step should cross the time-equidistant synchronization points.

Lack of the time-equidistant output option makes it necessary to limit the maximum time step to half TLM delay. The following analysis explains the reasons for such a limitation.

Let us consider communications between two solvers A and B. The basic reason for choosing the maximum time step and communication interval is the necessity for solver A calculating forces at time t_A to have information about solver B at time

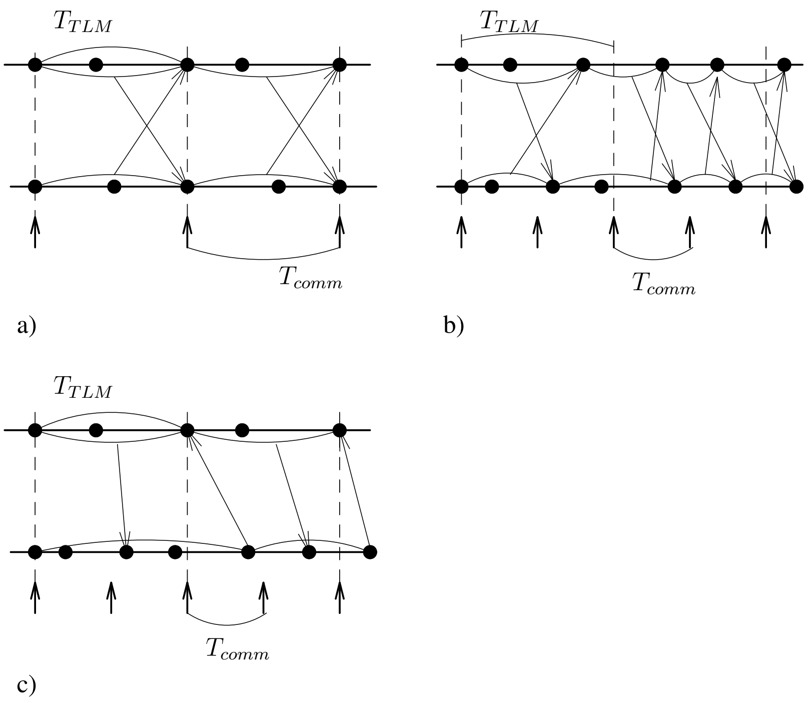


Figure 24-4: TLM communication modes: a) Two solvers with time-equidistant output; b) Two solvers with non-regular output; c) First solver with regular output, second one - non-regular.

$t_A - T_{TLM}$, where T_{TLM} is the TLM connection time delay. Since communication is symmetric the solver A data for time $t_A - T_{TLM}$ should be sent before t_A is reachable within a time step. Therefore the solvers with time-equidistant output can have maximum time step equal to T_{TLM}. They are expected to always respect this stepping and send information as soon as the next communication time point is reached. The solvers that do not support time-equidistant output, on the other hand, should have maximum time steps halved. Such solvers send out the information as soon as they cross the $T_{TLM}/2$ boundary.

Figure 24-4 shows the time lines for two collaborating solvers working in different modes. The filled points indicate the time steps taken. Arrows between the time lines present the information flow. That is the time instances when the computed time-stamped data is sent between the two simulations.

T_{TLM} and T_{comm} indicate the TLM delay in the connection and communication delay respectively.

24.5 TLM co-simulation interface requirements

The TLM co-simulation meta-modeling system requires every cooperating tool to provide the following functionality:

- Start simulation externally, specifying start and end time and maximal time step. This requirement basically corresponds to a request for batch mode of the simulation tool. No tool-specific graphical user interface should be necessary.

  The developer of the tool specific adapter should provide a start-up program that accepts the following command line parameters:

  - *Model* - the name of the sub-model as presented in the meta-model definition. This name typically corresponds to the component specific input file name.
  - FromTime - start time for the simulation.
  - ToTime - end time for the simulation.
  - Step - maximum time step allowed for the simulation. This depends on the minimum TLM delay associated with one of the TLM links connected to the sub-model.
  - Server:port - name of the host machine running TLM manager application and the TCP/IP port where TLM server is listening. This information is required for TLM-plugin initialization. It is provided by the TLM manager as the last argument to the start script.

- Communicate position, orientation and velocity of a point to the collaborating tools and receive the reaction force and torque to be applied at this point. We will call this point as TLM interface point in this report.

  A requirement to communicate point motion data poses some difficulties for the case of variable time-step numerical solvers. Such solvers can take trial time steps that are then rejected based on the some error criteria. Besides, implicit methods require Jacobian matrix to be evaluated which in turn leads to multiple evaluation of forces for small variations of the position and velocity state variables. All such calls are irrelevant for the coupled components and should not be send over the TLM link.

  Therefore the client code must be able to detect successful (completed) steps and communicate point motion *only* for such time instances. Note that it is, however, allowed to evaluate the force multiple times during the iterations within a step.

- Additional technical requirement is the possibility to implement the required functionality in *C++*. This is due to the fact that the standard plug-in component is implemented in *C++*.

Fortunately, the survey presented in the appendix shows that the outlined requirements are easily met by all the modern simulation tool considered.

Chapter 25

System Design

This chapter outlines the main aspects of the design of the framework.

25.1 Communication Protocol

The protocol is designed to be as minimal as possible to keep it easily portable and fast. Conceptually, the following stages can be outlined:

1. *Simulation Component* connects to *TLM manager*.
2. *Simulation Component* sends a component registration message, reporting its name.
3. *TLM manager* replies with the component ID or error code.
4. *Simulation Component* sends a registration request for each of the *TLM Interfaces*, reporting the names.
5. *TLM manager* replies with the interface ID or -1 if the interface is not used in the meta-model.
6. *Simulation Component* completes its initialization and sends a *CheckModel* request.
7. *TLM manager* waits for the *CheckModel* requests from all the components, verifies that all the interfaces defined in meta-model are connected, and sends back confirmation to all the components. Initialization phase ends at this point.
8. *Simulation Component* regularly sends time-stamped data for its *TLM interfaces* to the *TLM manager*
9. *TLM manager* marshals the received messages to the right components. Note that the manager does not modify message content, it just changes the source TLM interfaces ID into the destination TLM interface ID in the message header according to the TLM connection map.

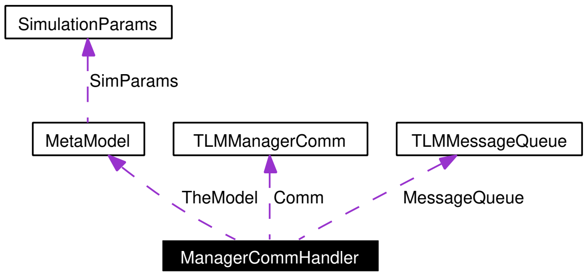


Figure 25-1: *Design of TLM manager application.*

Stages 9 and 10 are repeated until all the *Simulation Components* terminate.

25.2 TLM Manager

The central classes for the TLM manager and the relation between them are presented in Figure 25-1.

The `MetaModel` class is created and filled in by the `MetaModelReader` class (not shown) when it processes a meta-model definition file. The `MetaModel` holds the TLM manager's representation of the model. That is 'proxy' classes (i.e., light-weight manager objects providing a bridge to the corresponding component objects) representing the TLM components and interfaces and `TLMConnection` classes representing TLM links.

`TLMManagerComm` encapsulates the lower level operations on TCP/IP sockets.

`TLMMessageQueue` is a dynamic storage of message buffers. The intention is minimization of the necessary memory allocation/deallocation operations.

`ManagerCommHandler` implements the server side of the TLM simulation protocol as defined in the previous section. Separate threads are used for sending and receiving messages in order to reach high responsiveness of the application.

25.3 TLM Plugin

The functionality of the TLM plugin is accessible for the client applications via the abstract class `TLMPlugin`.

Special interface is implemented for the distributed application that use the master-slave paradigm. For such applications the master is responsible for requesting time stamped data from and to the TLM Plugin, that is for communication with TLM manager. The data can then be sent to the slave nodes that can use the static

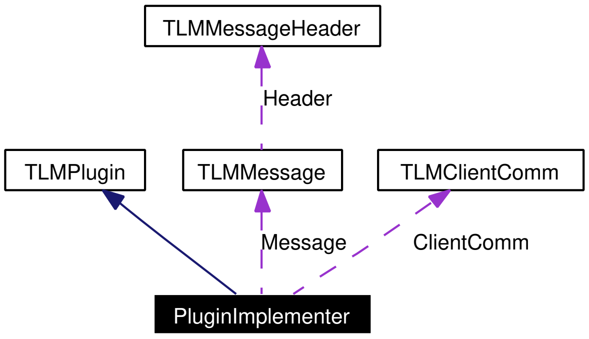


Figure 25-2: *Design of TLM plugin library.*

`GetForce` method to evaluate the reaction force and torque. The computed motion information should be reported to TLM manager from the master node only.

The `PluginImplementer` realizes the TLM interface main functionality, including data interpolation in time and the communication protocol. `TLMClientComm` class implements a wrapper for lower level TCP/IP client sockets. `TLMMessage` class is used to encapsulate the messages exchanged with the TLM manager.

25.4 Beast wrapper

The specific of the Beast wrapper is the necessity to support parallel simulation. The instance of the `TLMInterface` class is created only on the master node. The master process is therefore responsible for the communications with the TLM manager. Slave nodes are limited to the use of static functions defined in the `TLMInterface`. The dynamic data required for force calculations is sent to the slaves and the controlled motion is sent back to the master node.

25.5 ADAMS wrapper

The implementation of the wrapper is straight-forward. The basic call sequence includes:

- `sysary` function is used to obtain motion data;
- `units` function is used to get the model units and scale to the SI-units is necessary;

- `timget` function is used to identify the converged solver time steps.

One additional concern was the necessity to handle repeated calls for the same time instance.

Chapter 26

Test Cases

26.1 Simple Pendulum

The first model is classical for ADAMS. It is a simple pendulum shown in Figure 26-1. The pendulum will oscillate due to the gravity.

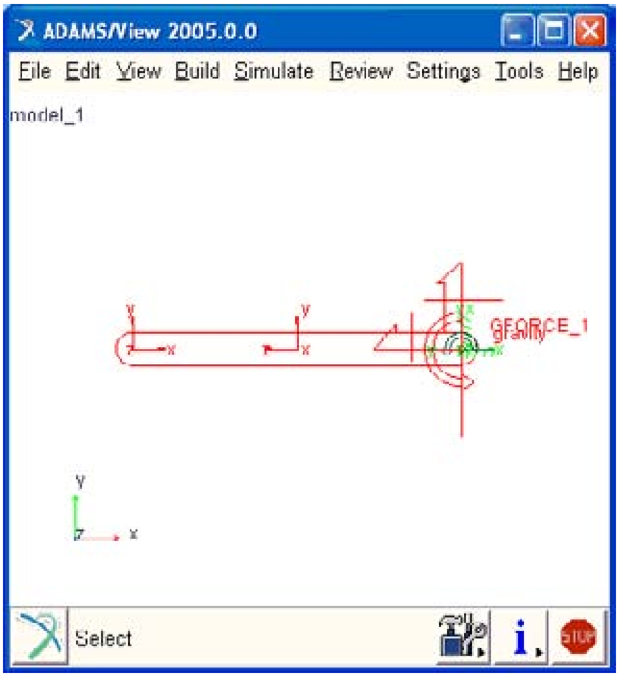


Figure 26-1: Simple pendulum model in ADAMS

In the co-simulation case the revolute joint that connects the pendulum arm to the ground was substituted with a DGBB bearing modelled in BEAST (see Figure 26-2).

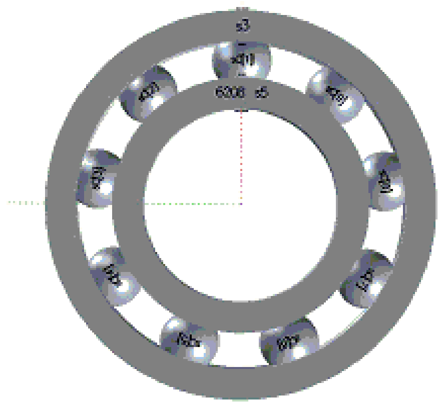


Figure 26-2: DGBB bearing in BEAST

The graphs showing angular velocity of the pendulum arm are shown in Figure 26-3. One can see some differences caused by the friction and damping in the DGBB.

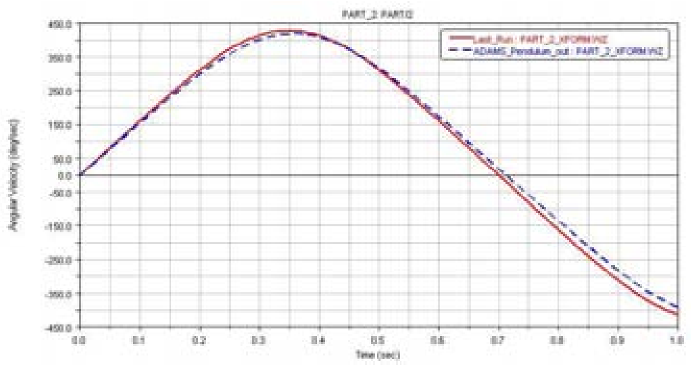


Figure 26-3: Comparison of pendulum’s rotational velocity (red solid line - model with ADAMS joint; blue dashed - TLM)

26.2 Double Pendulum

An ADAMS model of a double pendulum is presented in Figure 26-4. The revolute joint connecting the two segments of the pendulum was substituted with the same DGBB as in the previous example. Connection to ground uses an ADAMS joint.

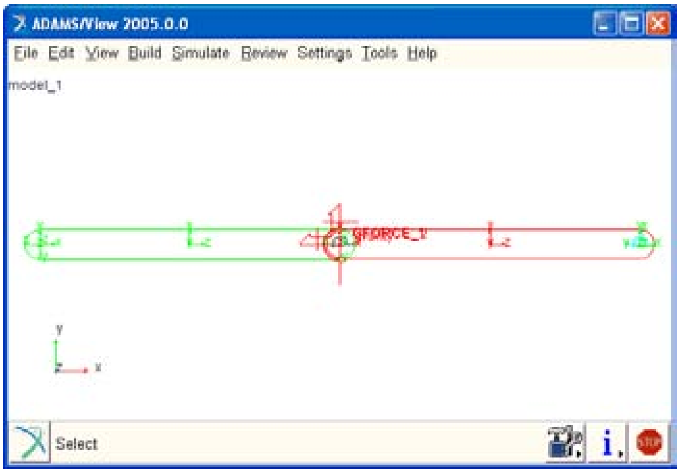


Figure 26-4: Double pendulum model in ADAMS. Arms are connected via TLM

Figure 26-5 presents the graphs of angles between the two pendulum arms and Figure 26-6 presents the calculated forces in the supporting joint. One can see that the friction and finite stiffness of the bearing causes some differences.

At the same time an example of typical output from a Beast simulation - contact power losses is presented in Figure 26-7.

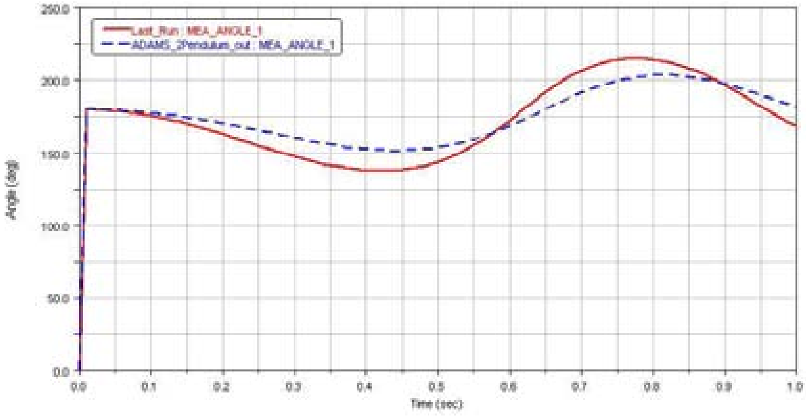


Figure 26-5: Comparison of angle graphs (red solid line - model with ADAMS joint; blue dashed - TLM)

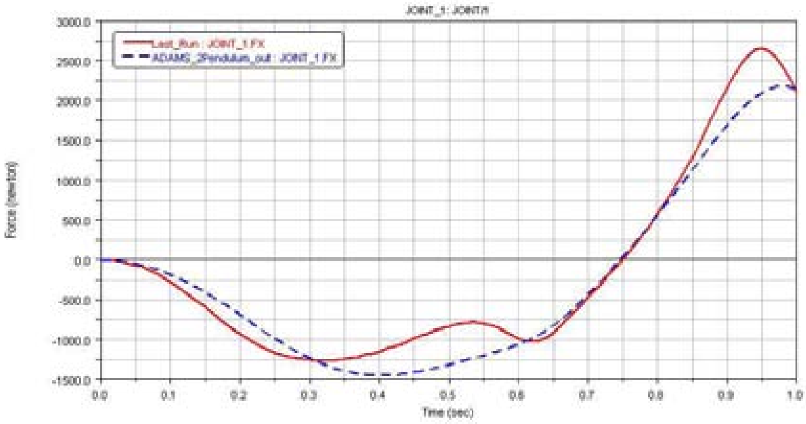


Figure 26-6: Comparison of the reaction forces in the supporting joint (red solid line - model with ADAMS joint; blue dashed - TLM)

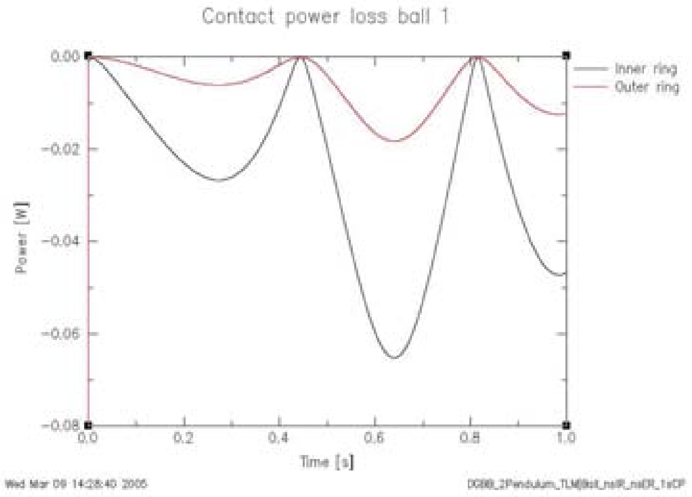


Figure 26-7: Power loss in the ring contacts of DGBB for Ball 1

26.3 A Car Model

The model shown in Figure 26-8 was originally developed at SKF. With minimal changes to the model, the revolute joint representing the hub unit in the front left wheel was substituted with a BEAST model presented in Figure 26-9.

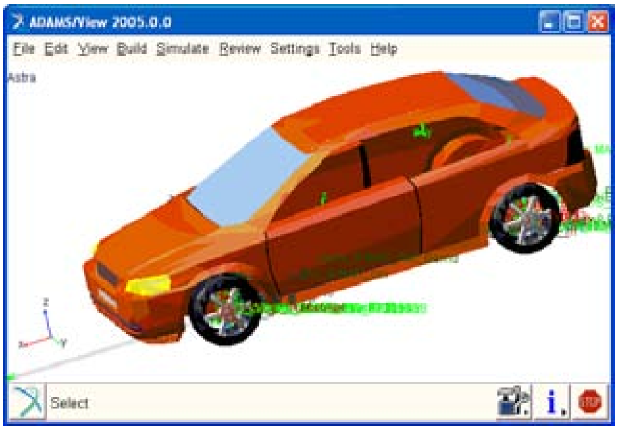


Figure 26-8: A car model in MSC.ADAMS. Left front wheel is connected via TLM

The wheel in this artificial case was positioned off-center, which can be seen in the comparison in Figure 26-10. Figure 26-11 presents a curve that would typically be a simulation output for an analysis engineer.

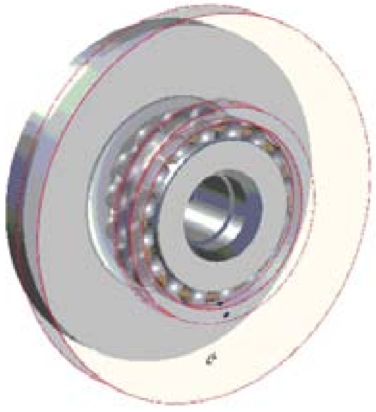


Figure 26-9: Wheel hub unit model in BEAST

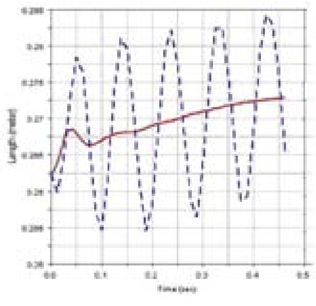


Figure 26-10: Comparison of wheel center position graphs showing off-center in co-simulation (red solid line - model with ADAMS joint; blue dashed - TLM)

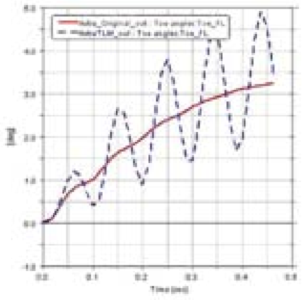


Figure 26-11: Comparison of the toe angle graphs (red solid line - model with ADAMS joint; blue dashed - TLM)

It should be pointed out that the initial conditions for the original car model specifying initial translational velocity of $20m/s$ were not suitable for co-simulation. Modification to enable zero velocity start with a fast acceleration to the desired speed had to be done. This is due to the practical difficulties in specifying correct velocity for all bearing components given the translational speed only. Attempts to start a simulation with inconsistent velocities on the interfaces have lead to numerical problems for the ODE solvers of the components.

Our experience with the car model has proved the importance of the use of right competences as discussed in Section 24.1. Particularly, our attempts to modify the complex car model without expert knowledge of the tool has lead to significant time delays and a number of unsuccessful simulations caused by modeling errors. As soon as the components were prepared the meta-model setup could be done within one hour by an engineer with minimal user knowledge of the meta-modeling framework.

The simulation was performed in a distributed environment. The MSC.ADAMS simulation together with the TLM manager were running on a PC workstation and the BEAST simulation was running on a Linux cluster.

No significant delay due to the network communications were noticed. The simulation time was completely dominated by the BEAST calculations. As the MSC.ADAMS simulation was advancing much faster the BEAST process never needed to wait for the data. Therefore the total simulation time did not increase as compared to a typical pure BEAST simulation. This proved the efficiency of the designed communication protocol.

The wall-clock time for the simulation reached 48 hours which can be considered as a demonstration of the stability of the framework implementation.

Chapter 27

Conclusions

A coupled simulation framework based on the TLM approach has been presented. The advantages of the TLM technique such as its proven unconditional stability and minimal requirements on the interfaces were discussed.

We have underlined that coupling of simulations with the TLM line has physical meaning. The practical calculations that may be utilized to decide the characteristic parameters of the coupling were also presented.

The object-oriented design of the co-simulation framework was described. This includes a communication protocol specially developed to support coupling of variable time step differential equations solvers.

The presented framework design is general with minimal requirements on the participating simulation tools. Integration of a new tool into the framework is straight forward for an experienced software developer.

The framework was successfully used for connecting MSC.ADAMS and SKF BEAST simulation models. Some of the test runs were presented in the text.

The approach to the coupling between co-simulations that utilizes physical properties of the real connecting media is very promising. The system users are afforded a practical way to estimate the effects introduced by the co-simulation and relate it to the properties of the connected components.

Further development of the technique may introduce additional physical parameters, such as, e.g., connection damping, providing a way to model more advanced connections directly in the co-simulation. It may also be interesting to apply TLM technique for coupling other kinds of physical processes, e.g., heat transfer.

Bibliography

[1] ABAQUS homepage. *http://www.abaqus.com/*.

[2] ANSYS solutions homepage. *http://www.ansys.com/*.

[3] Cosimate homepage. *http://www.tni-world.com/cosimate.asp*.

[4] Dynasim and Dymola home page. *http://www.dynasim.com*.

[5] Andrew S. Elliot. A Highly Efficient, General-Purpose Approach for Co-Simulation with ADAMS. MDI Inc, Nov 2000.

[6] Andrew S. Elliot. Status Update on Advanced, General-Purpose Co-Simulation with ADAMS. MDI Inc, May 2002.

[7] C. A. Felippa, K. C. Park, and C Farhat. Partitioned analysis of coupled mechanical systems. Invited Plenary Lecture, 4th World Congress in Computational Mechanics, Buenos Aires, Argentina, July 1998, expanded version in Comp.Meth.Appl. Mechanics and Engineering.,190, 3247-3270, 2001.

[8] D. Fritzson, J. Ståhl, and I. Nakhimovski. Transmission line co-simulation of rolling bearing applications. To be submitted, Journal of Multibody Dynamics, 2006.

[9] Ben Gum H., Harry Asada, and Xiangdong He. Software Development of Co-Simulation of Algebraically Coupled Dynamic Sussystems without Disclosure of Proprietary Subsystem Models. In *IMECE*, New Orleans, Louisiana, Nov 2002.

[10] Gregory M. Hulbert, Zheng-Dong Ma, and Jinzhong Wang. Gluing for Dynamic Simulation of Distributed Mechanical Systems. In Jorge A.C. Ambrosio, editor, *Advances in Computational Multibody Systems*, pages 69–94. Springer, 2005.

[11] Peter B. Johns and Mark O'Brien. Use of the transmission-line modelling (t.l.m.) method to solve non-linear lumped networks. *The Radio and Electronic Engineer*, 50(1/2):59–70, Jan/Feb 1980.

[12] Frederick Kuhl, Richard Weatherly, and Judith Dahmann. *Creating Computer Simulation Systems. An Introduction to the High Level Architecture.* Prentice Hall PTR, 2000.

[13] J Larsson. *Interoperability in Modelling and Simulation.* PhD thesis, Linköping University, 2003.

[14] Mathworks homepage. *http://www.mathworks.com*.

[15] Hermann G. Matthies and Jan Steindorf. Strong coupling methods (revised version). In *Stuttgart Multifield Conference*, Stuttgart, Germany, April 2002.

[16] Modelica standard. *http://www.modelica.org*.

[17] MSC Software Corporation homepage. *http://www.mscsoftware.com/*.

[18] Krus P. Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints. *Transactions of ASME, Journal of Dynamic Systems Measurement and Control.*, Dec 1999.

[19] Krus P. and Jansson A. Distributed simulation of hydromechanical systems. In *Third Bath International Fluid Power Workshop*, Bath, UK, 1990.

[20] S.H. Pulko, A. Mallik, R. Allen, and P.B. Johns. Automatic Timestepping in TLM Routines for the Modelling of Thermal Diffusion Processes. *Int. Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 3:127–136, 1990.

[21] A. Veitl, T. Gordon, A. Van De Sand, M. Howell, M. Valasek, O. Vaculin, and P. Steinbauer. Methodologies for couling simulation models and codes in mechatronic system ananlysis and design. *Vehicle System Dynamics Supplement*, 33:231–243, 1999.

Appendix: Review of Available External Function Interfaces

MSC.ADAMS interfaces

MSC.ADAMS is a collection of several tools where the essential ones for co-simulation purposes are ADAMS/View (pre- and post-processor program) and ADAMS/Solver (the simulation subsystem). The typical scenario for meta-modeling would be as following:

- The model is created (or loaded) in ADAMS/View and TLM interface point is chosen. Points in terms of ADAMS are called *markers*. A general 6-component force is defined at this marker and its value is set to `Subroutine` with `Routine` field set to a custom library. This indicates that a user-written function from the specified library will be used for actual calculation. The only arguments required for a co-simulation user function are the action and reaction markers ID.
- The model is exported to an ADAMS/Solver dataset by using `File-Export` menu and choosing `ADAMS/Solver Data Set` as output file type.
- Now the TLM manager tool can be started and the ADAMS model can be specified as one of the collaborating tools. The tool then initializes the simulation that generates dynamic output as a normal ADAMS simulation would do. The ADAMS/View can be used to analyzed the output of the simulation.

The essential interface features used by the TLM module are:

- [GFOSUB] interface implementation in the custom dynamic library;
- [SYSARY] is utilized for getting kinematic data from ADAMS;
- [TIMGET] is used for interfacing the ADAMS DAE solver.

In order to start an ADAMS simulation externally the top level batch file for all ADAMS products (`mdi`) can be used. The external code should issue the following system command:

```
mdi ru-s  <ADAMS command file>
```

The ADAMS command file (`.acf`) can be automatically generated. Its content should look as following:

```
model-to-simulate
output-files-base-name
sim/dyn, end=2.0, dtout=0.01
stop
```

The `acf` file gives the following information:

- model-to-simulate tells the name of ADAMS/Solver dataset file ('adm') containing the model definition
- output-files-base-name gives the base name for constructing the simulation output file names ('req', 'gra', 'res')
- 'sim/dyn, end=2.0, dtout=0.01' tells the solver to perform a dynamic simulation with the end time 2.0 using data output step 0.01. Note that for the case of TLM the TLM communication time step should be a multiple of data output step.
- 'stop' command marks the end of the simulation and forces ADAMS/Solver to terminate.

Modelica External Function Interface

Modelica standard [16] allows to call external functions written in C from inside the Modelica code. External functions are typically written and compiled separately from the Modelica code. The following steps are required to integrate the external function into the Modelica simulation:

1. Define an external function in Modelica
2. Compile the Modelica code into C-code
3. Compile and link the Modelica C-code and the external function code

   We might also compile the external function C-code into a library first and link it to the Modelica C-code

The most widely used implementation of the Modelica standard is Dymola [4]. Dymola supports two ways of including C-code into a model:

- with the inline functions that are written directly in the model code;
- linking the external function as a static library.

The Dymola simulation executable is called `dynosim.exe` and can be started by the TLM manager process. Input parameters such as simulation start and end time can be read from a simulation input file. This file can be specified on the command line, e.g., `dynosim.exe input_file`. If no input file is specified the default input file called `dsin.txt` is read.

The external function interface can be used to integrate a TLM component into the Modelica simulation. To achieve this we need to define a TLM Modelica class which in turn calls an external function in the TLM component. The TLM component can be linked to the Modelica simulation to connect the Modelica simulation to the the co-simulation manager application.

MATLAB external functions interfaces

External simulation control: Start, Stop, Maximum timestep

Batch mode MATLAB can be started externally in a number of ways. The most common option is to run MATLAB in batch mode.

```
matlab -nodesktop -nosplash -minimize -r "m-file"
```

There are also a number of ways to pass additional input parameters:

1. Variables values can be specifed at the command line.
2. The M-file can be converted to a function. Input parameters can then be passed into the function from the command line..
3. Input parameters can be stroed in a file that the script M-file can open.

The Engine library The MATLAB engine library is a set of routines that allows you to call MATLAB from your own programs, thereby employing MATLAB as a computation engine. MATLAB engine programs are C or Fortran programs that communicate with a separate MATLAB process via pipes, on UNIX, and through a Component Object Model (COM) interface, on Windows. There is a library of functions provided with MATLAB that allows you to start and end the MATLAB process, send data to and from MATLAB, and send commands to be processed in MATLAB.

Communication with TLM plug-in

It is possible to call external C or Fortran subroutines from MATLAB as if they were built-in functions. MATLAB callable C and Fortran programs are referred to as MEX-files. MEX-files are dynamically linked subroutines that the MATLAB interpreter can automatically load and execute.

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani**: Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms,1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund**: Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld**: Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén**: Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Langugaes from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationsystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Emperical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004. ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linquistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modellling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005. ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.