

# **Efficient remeshing and analysis views for integration of design and analysis**



# Efficient remeshing and analysis views for integration of design and analysis

PROEFSCHRIFT

ter verkrijging van de graad van doctor  
aan de Technische Universiteit Delft,  
op gezag van de Rector Magnificus prof. ir. K. C. A. M. Luyben,  
voorzitter van het College voor Promoties,  
in het openbaar te verdedigen op donderdag 17 februari 2011 om 15.00 uur

door

**Matthijs SYPKENS SMIT**

doctorandus in de wiskunde  
geboren te Utrecht.

Dit proefschrift is goedgekeurd door de promotor:  
Prof. dr. ir. F. W. Jansen

Copromotor:  
Dr. W. F. Bronsvoot

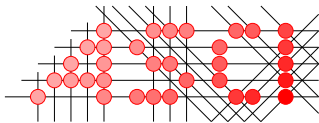
Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. F. W. Jansen	Technische Universiteit Delft, promotor
Dr. W. F. Bronsvoot	Technische Universiteit Delft, copromotor
Prof. dr. ir. C. Vuik	Technische Universiteit Delft
Prof. dr. R. C. Veltkamp	Universiteit Utrecht
Prof. dr. L. Kobbelt	RWTH Aachen University
Prof. dr. G. M. Turkiyyah	American University of Beirut
Dr. J. S. M. Vergeest	Technische Universiteit Delft



Netherlands Organisation for Scientific Research

This research was financially supported by the Netherlands Organisation for Scientific Research (NWO).



Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.  
ASCI dissertation series number 225.

ISBN 978-90-5335-368-4  
© 2011, Matthijs Sypkens Smit

*in memory of  
my grandfather, J. H. Sypkens Smit  
and my father, M. J. Sypkens Smit*



Alcanzar alguno a ser eminente en letras le cuesta tiempo,  
vigilias, hambre, desnudez, váguidos de cabeza,  
indigestiones de estómago, y otras cosas a éstas adherentes.  
*(For a man to attain to an eminent degree in learning costs him  
time, watching, hunger, nakedness, dizziness in the head, weakness  
in the stomach, and other inconveniences.)*

---

*Don Quijote de la Mancha*  
MIGUEL DE CERVANTES SAAVEDRA





# Contents

<b>Preface</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Modern design	1
1.2 Analysis in design	3
1.3 Research objectives	6
1.4 Structure of this thesis	9
1.5 Contributions	10
<b>2 Design and analysis models</b>	<b>11</b>
2.1 Historical perspective	11
2.2 Design models	14
2.3 Analysis models	17
2.4 Model integration in the product development cycle	22
<b>3 Analysis and meshing</b>	<b>27</b>
3.1 Meshes for finite element analysis	27
3.2 Meshing methods	31
3.3 The prospect of remeshing	40
<b>4 Variational tetrahedral meshing of mechanical models</b>	<b>45</b>
4.1 Variational tetrahedral meshing	46
4.2 Enhancing VTM for meshing mechanical models	53
4.3 Constructing the boundary	56
4.4 Enhanced mesh extraction	62
4.5 Practical considerations	64
4.6 Examples	66
4.7 Conclusions	69
<b>5 The feature difference</b>	<b>71</b>
5.1 Background	72
5.2 The feature model	74

5.3	The difference between two feature models	78
5.4	Representing and constructing the difference model	85
5.5	Conclusions	89
<b>6</b>	<b>Efficient tetrahedral remeshing</b>	<b>91</b>
6.1	Remeshing based on the difference model	92
6.2	The combined model	94
6.3	The remeshing procedure	97
6.4	Copying mesh nodes	98
6.5	Adding new nodes, and the free/fixed distinction	101
6.6	Efficiently constructing a quality mesh	103
6.7	Results and discussion	105
6.8	Conclusions	109
<b>7</b>	<b>Integration of design and analysis models</b>	<b>117</b>
7.1	Approaches to integration of design and analysis	119
7.2	Multiple-view feature modelling	127
7.3	Analysis views	129
7.4	Conclusions	133
<b>8</b>	<b>Relating an analysis view to a design view</b>	<b>135</b>
8.1	A prototype for an analysis view	137
8.2	Procedure for automated abstraction	142
8.3	Linking the design and analysis views	147
8.4	Open issues	151
8.5	Conclusions	154
<b>9</b>	<b>Conclusions and future research</b>	<b>157</b>
9.1	Conclusions	158
9.2	Future research	160
	<b>Bibliography</b>	<b>163</b>
	<b>Summary</b>	<b>171</b>
	<b>Samenvatting</b>	<b>175</b>
	<b>Curriculum Vitae</b>	<b>179</b>

# Preface

During the summer of 2005 I finished my study in Computational Science at the University of Utrecht, which had focussed on solving large scale numerical problems and scientific modelling. It was part of the mathematics curriculum, but many of the required courses were given by the physics and computer science departments. The diversity in subjects appealed to me. Therefore, I felt compelled to continue along the path of scientific study, but something with less focus on mathematics.

I interviewed for a Ph.D. position at Delft University of Technology, concerning an NWO-sponsored project on the integration of design and analysis models. At the time I believed that the proposal was primarily concerned with meshing of analysis models, a field that I was reasonably familiar with. Only after being hired, and having worked on the project for some months, it became apparent to me that the design and modelling aspects played a much larger role than I had initially understood. It was a pleasant surprise that broadened my perspective on the subject of research.

Looking back, I suppose my misconception is highly demonstrative of the schismatic relation between design on the one hand, and analysis on the other hand. Coming from a background of mathematical modelling, I only saw the analysis side in the research proposal. I did not properly understand what concepts such as *semantic feature modelling* really meant. I have been fortunate to study both design and analysis, and their integration in this project, in an academic context. To achieve proper integration between these two disciplines, mutual understanding is undoubtedly a prerequisite. I hope this thesis to be a step in that direction.

This thesis could not have existed without the contribution and support of many. I am thankful to all of them, with the following people deserving to be mentioned explicitly.

Wim Bronsvort, my advisor and copromotor, to whom I am very grateful for giving excellent guidance throughout my Ph.D. study by frequently exchanging ideas, steering me in the right direction when necessary, and correcting almost all my texts and presentations. I very much enjoyed our

regular conversations on bureaucracy, politics, culture, sports, and current events. Erik Jansen, my promotor, for reading my thesis and having me as a Ph.D. student in his group. Rick van der Meiden, for writing software that I used in my research, answering questions on SPIFF, general exchange of ideas, and sharing a room with me for the larger part of my stay at the university. Rafael Bidarra, for exchanging ideas, his stimulating views on Dutch society and language, and being close to a mentor. Paulos Nyirenda, for always stimulating my ideas, his boundless and contagious enthusiasm, and answering so many of my questions concerning feature modelling.

Ruud de Jong and Bart Vastenhouw, for administering all the systems, and always being there to help, whatever you need. Toos Brussee, Bianca Abrahamsz, and Helen de Roo, for their excellent support and care in quickly handling a wide range of administrative tasks. The colleagues with whom I shared a room and many conversations: Rudolf ter Haar, Peter Kok, Thomas Kroes, Tim Tutenel, Ruben Smelik, Ricardo Lopes, and Stef Busking. And all other colleagues whose company and knowledge I enjoyed during lunches, coffee breaks, cake gatherings, colloquia, and other group activities: Jorik Blaas, Charl Botha, Eric Griffith, Gerwin de Haan, Peter Krekel, François Malan, Peter van Nieuwenhuizen, Frits Post and Lingxiao Zhao.

I thank Magali de Walick for her critical remarks on my designs of the thesis cover.

I am very grateful to my closest friends, who regularly inquired about the progress of my thesis, stimulate my thoughts on diverse topics outside my work, and make life truly pleasant just by knowing them: Karin Brakkee, Eric Broersma, Arnoud Breunese, Stella Evers, Jes Jørgensen, Suzanne Jørgensen-Bisschop, Jorge de Pedro del Pozo, Wouter Prummel, Cecilia Schouten, Fiona Schouten, Famke de Vries-Wildeman, Jitze Jan de Vries, Joost Wery, and Thessa Wery-van Zoeren.

Special thanks to my paranympths Josien Braas and Thijs Kinkhorst for standing by me on the day of the defence, and the many memories I share with them.

My family for their love and trust in me. I am thankful to Matthé, my father, for all that he has given me as a parent, which includes his love for art and architecture. I thank my mother Margriet and my brother Michiel for their love and strength, their support, and their great company at countless times. And lastly Saskia, the love of my life: you mean the world to me.

# Chapter 1

## Introduction

The result of product design is everywhere around us; cars, air conditioners, coffee cups, flower pots, telephones, desk chairs, waste bins, and door hinges, are just a tiny sample of all the objects around us that have passed through the process of design. Every man-made object is, to some degree, the result of a design process.

Design in the broadest sense is therefore nothing new. Our earliest ancestors who hunted with spears were already designing their weapons. A spear must have a shape and weight that are favourable for throwing it, its head must be sharp and strong, and as a whole it should be durable. Through trial and error they discovered how to build their spears, with the materials that they disposed of, such that the utility of the weapon was optimised.

Modern design still shares conceptual similarity with prehistoric design, as it continues to revolve around optimising utility, and progresses through a cycle of design stages that bears resemblance to the practice of trial and error. The main difference is the use of scientific knowledge and tools to support the design process. Although trial and error still exists, the choices we make in designing our products are driven largely by scientific principles, mostly from the domain of physics. Also, over the second half of the 20<sup>th</sup> century, the use of computers has become integral to the design process.

### 1.1 Modern design

Many of the products in our lives are complex, constructed by advanced technological processes, and they need to satisfy many demands and constraints in addition to just fulfilling their primary purpose. Besides serving their purpose, products must also be better than those of the competitor,

and they must be cheaper and quicker to produce. Meeting all those requirements, can only be achieved through a continuous process of adapting and improving designs.

Computers have played a key role in the immense advances that the design process has experienced since the second half of the 20<sup>th</sup> century. Computer-aided design (CAD) [Lee, 1999] clearly offers advantages over design with ruler, pencil, and paper. It enables the specification of complete three-dimensional (3D) models, which can be viewed from arbitrary angles. Mistakes are easier to correct in a digital environment, and errors of consistency between several drawings with different viewpoints is far less an issue than it used to be. Highly complex shapes can be created with precision and a high degree of automation. Geometric calculations are quick and easy to perform. Multiple designs can be combined into a design of a more complex product, called an *assembly*. The models can be given colour and texture, and thus be rendered realistically. This allows to get a proper visual impression of the product, before it has been actually built. The model can be inspected and shown in ways that are impossible even with a physical prototype of the product.

Obviously, the usefulness of flexibility in changing a digital design goes beyond correcting mistakes. Models are parametrised with the intent of varying several aspects of the design. Nowadays, we have the ability to quickly manufacture products in large quantities within a short time of completing the design. This has pushed specialised and even on-demand designs, where a product is built according to the specific wishes and requirements of the customer. To enable this efficient construction and adaptation of models, modelling software has evolved significantly. *Feature modelling* is the current norm, which consists of building models by combining high level constructs with a generalised shape, which can often be associated with some aspect of functionality [Bronsvoort et al., 2006]. Such semantics of models is to some degree being incorporated in design methodologies, as the software can be used to specify semantics, as well as automatically check that the semantics is not violated.

Product design involves more than just specifying the shape, materials and semantics. Large-scale manufacturing should not start until there is sufficient reason to believe that the product lives up to expectations, thus after the specification of the product, a phase follows wherein this is checked. Both individual parts, and the product as a whole, have to fulfil a range of requirements. These requirements often concern the physical behaviour of the product under certain circumstances. They are either called for by law, or by the fact that violating the requirements would imply malfunctioning of the product. The most obvious way to test the product and its parts is by

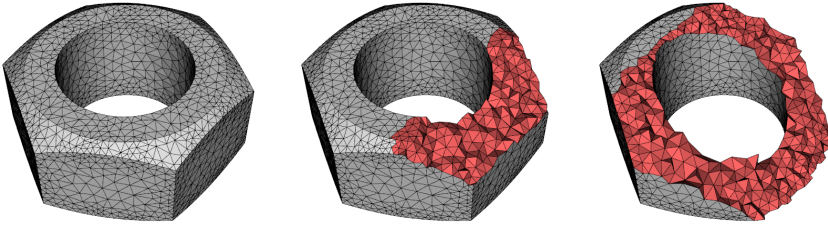
manufacturing a test version of the product and investigating its behaviour. This, however, is expensive and takes a lot of time. Manufacturing a single copy of a product is clearly more expensive than having it manufactured in a mass production process. And making a product for the first time takes longer because the manufacturing process has less automation than in mass production, or parts of the process even have to be developed.

Product testing can largely be replaced by *analysis*, also commonly referred to as *computer-aided engineering* (CAE). This is the simulation of the physical behaviour through numerical approximation of physical models, or virtual product testing. In the past, at various stages in the design process a test model needed to be built. For example, in the design of a seemingly simple product such as an air cooling system for a car, it could require the production of more than ten test models, until the design satisfied all demands. Analysis does not absolve us from the necessity to build test models, but it can strongly reduce the number of real-world models that has to be built during the product design cycle. For example, for the design of the air cooling system in an actual industrial setting, the number of test models that was built, could be reduced to just two. Performing analysis has therefore become an integral aspect of modern design.

Product design is not a linear process, but rather we speak of the *product development cycle*. Models are iteratively adapted, and new insights, gained during some phase of the development, can lead us back to an earlier phase. After each adaptation of the design, many tasks that concern the testing of a product before it goes into manufacturing have to be repeated. In case of analysis, it needs to be verified that the changes have resulted in the desired improvement, or conversely, that the changes have not resulted in violation of the requirements. Some of the tasks that have to be repeated at each iteration can be performed relatively quickly, as they require only limited attention after adaptation of the model. Analysis, however, rarely falls in this category.

## 1.2 Analysis in design

The role of analysis is to assess the product before it is actually built. Analysis can in principle be applied for investigating any aspect of the (physical) behaviour for which a mathematical model exists. Some examples of questions that are looked into are: can the product withstand low or high temperatures, how does it behave in a crash/overload scenario, how is air circulation around the object, where do weak spots develop during the production process, etc. The kind of properties that are verified and the rigour that is required in this phase highly depend on the kind of product. Nonetheless,



**Figure 1.1:** Tetrahedral mesh covering the volume of a nut model

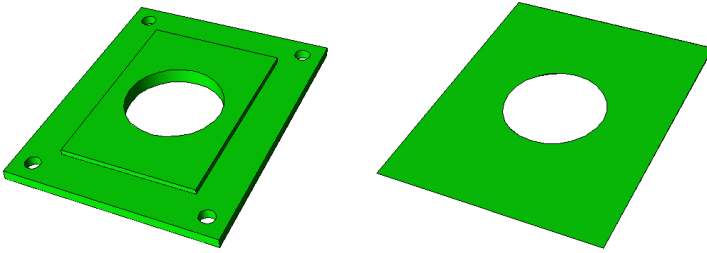
virtually all products, ranging from complete cars to mobile phone casings, need to have some aspects analysed before they can go into production.

Analysis is a broad concept; any numerical simulation of some aspect of the product could be deemed a type of analysis. The most commonly applied method is that of *finite element analysis* (FEA) [Brenner and Scott, 2007]. With this approach, the geometry of the product is decomposed into simple geometric shapes such as triangles, quadrilaterals, tetrahedrons, or hexahedrons. This process is called *meshing* and the decomposition of the model is the (finite element) *mesh* [Owen, 1998; Shimada, 2006]. Figure 1.1 shows an example of a mesh and two slices through its volume. The elementary shapes are the support over which polynomial basis functions, commonly piecewise linear functions, are blended together to represent a continuous solution over the whole domain composed by the elements. The solution is found by discretising the model of the aspect that is being studied, i.e. a set of equations, such that it becomes a formulation in terms of the nodal values by which the basis functions are supported. This leads to a large system of equations, which can be solved by applying an appropriate numerical algorithm.

Analysis has significantly lowered the time between conception and mass production of a product by largely eliminating the lengthy and expensive process of creating test models. It has increased the reliability and performance of products as the barrier to performing numerical analysis is lower than to carrying out experiments with test models. Also, it has allowed for increased creativity in designs, as it is easier to experiment with imaginative designs that test the limits of our engineering capabilities. Bridges and skyscrapers, for example, are rarely built as a test model, but through numerical analysis a range of more challenging designs becomes accessible.

Despite the clear advantages that analysis has to offer to the design process, it is by no means quick and simple to perform. The calculation itself can take more than a week in extreme cases, but it requires little human





**Figure 1.2:** Design model and its abstract analysis model

intervention. The preparation stage of the analysis model, on the other hand, mainly consists of interactive tasks that are performed by engineers. The analysis model should not be confused with the analysis mesh, but rather represents a suitable geometry to derive a mesh from. It also contains all the information to drive an analysis regardless of a specific mesh, such as boundary conditions, boundary values, and mesh element size requirements. The engineers either have to create the analysis models from scratch, or they have to fix and adapt an imported geometric model. A suitable model for analysis is normally an abstraction of the design model. Small details, unless they are deemed important to the outcome of the analysis, are left out to improve the speed and accuracy of the calculation. One step further is dimensional abstraction, where the object of analysis is (partly) represented by a two-dimensional (2D) or one-dimensional (1D) geometry. An example of this is shown in Figure 1.2. Commonly, the decision on what the shape of an analysis model should be is mostly a task for experienced humans, in which the role of automation is minor. Once a geometric model has been obtained that is suitable for analysis, boundary conditions have to be assigned, and often the meshing process has to be guided to some extent. Preparation for analysis can easily take multiple weeks, and surpassing even a month is not uncommon.

Many of the tasks performed during preparation of the analysis models involve copying information that was already captured in the design model, e.g. recreating part of the geometric shape. Unfortunately, this information is often not captured explicitly, i.e. it is not specified which parts of the geometry are part of a particular analysis model. This is due to the high degree of expert knowledge and experience-based intuition that is commonly involved in describing the analysis geometry. Without automated methods, it is only natural to delay the description of the analysis geometry until the basic design has been finished, and rarely is this description given in relation to the design model. Also, during design the need to later perform analysis

is not optimally taken into account, or at least the design software creates models that are not ideally suited as a take off point for analysis. The design software can, for instance, produce models that are not watertight, i.e. the individual geometric entities do not accurately connect as the topology indicates, or models that have unnecessary geometric elements in their description. This can frustrate the meshing process.

In particular, after the completion of an analysis step and subsequent changes to the design, the work of creating the analysis model is repeated again, sometimes completely from scratch. Of course the engineers have gained experience with the model and can get the work done more quickly. Most likely they will only need to adapt previous models, but it still takes valuable human time. If there are multiple analysis models, then the engineer has to propagate the changes to all these models. If the geometry has been changed for some models, then these need to be meshed again.

We observe that preparing analysis models is currently a labour intensive process, but at the same time an engineer performing the tasks might not feel that he is really doing challenging or interesting work. The principal cause for this is *lack of automation* in the process, which in turn is mainly caused by a *lack of integration* between the (geometric) design phase and the analysis phase.

### 1.3 Research objectives

Both geometric design and analysis are highly evolved and technologically advanced processes, but they evolved in their own separate context. Over the decades they have matured and have brought large gains in efficiency and versatility. In particular the enormous advancements in computing performance per unit cost have driven innovation.

For design this means that more and more complex calculations could be performed interactively, allowing the designer to work more intuitively. Instead of thinking out the effect of certain operations in advance, the designer can directly see the effect of an operation, and fine-tune it with the aid of interactive visual feedback. Also, the software has become 'smarter', taking over simple tasks that were once in the hand of the designer. With many diverse applications of design, each application has different tasks that are suitable for automation, which naturally leads to specialised environments for solving particular design problems. Currently, specialisation seems to be the primary way of adding more knowledge in an automated fashion to the design process.

The innovations in analysis are very diverse, but virtually all of them serve to increase the dimensionality, complexity, and accuracy of the compu-

tations. A large part of the innovations in analysis is of mathematical nature, consisting of new algorithms that optimise some aspect of the calculation. Handling ever larger data sets, increasingly through parallel computations, is another aspect that has been central to the advances in analysis. Lastly, with analysis models getting more and more complex, interactive control over the analysis process, including preparation of the geometry, meshing, and setting up all the relevant details for the computational process, has become very important.

This preparation step is necessary since the models on which analysis is performed have become more and more complex. In the early days of analysis, the responsibility for creating the models laid with the engineers performing the analysis. Commonly, their task was to describe the perimeter of the 2D domain/model, either with spline curves or an approximation by means of straight line segments. The analysis was much less an integral part of the product development cycle as it is today. Of course, knowledge gained from the analysis would certainly be used to improve a product, but it stood more on its own, as an independent tool for research to improve engineering knowledge, develop accurate models, and reliable analysis techniques. With the rising complexity of models, it became common to link the analysis model more closely to the design model, and with this progression, analysis became a genuine part of the development cycle.

This practice of actively incorporating analysis into the design process, has made it increasingly apparent that there is lack of integration between design and analysis, as the preparation of the analysis model and, in general, going back and forth between the design model, the analysis model, and the mesh, is fraught with many tedious and partially redundant tasks. Recognising that there is a need to improve the efficiency of product development here, we concern ourselves in this thesis with the improvement of the integration of design and analysis.

Improving this integration, in our opinion, requires making transitions between the design model, the analysis model, and the mesh more efficient. This should be accomplished by remedying the lack of reuse of information captured in different phases of the process, and will lead to cheaper, more reliable, and more efficient product development.

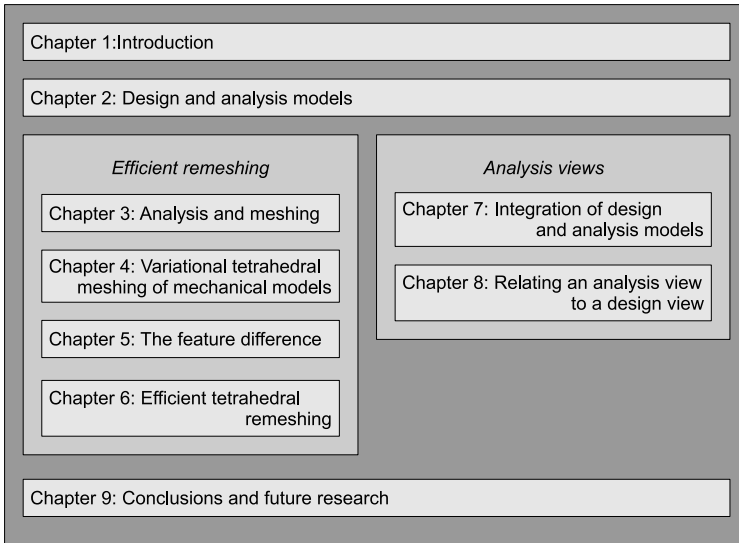
We do not strive for a complete and universal solution. That would be impossible, considering the very broad range of techniques and specialised software that is used in both design and analysis. Instead, we focus on two aspects within the design-analysis loop, and describe how improvements can be made here. The first aspect concerns efficient meshing of analysis models, whereas the second aspect concerns the integration of design and analysis models. These two aspects are phrased into the following research

questions:

1. How can we reduce the time spent on meshing an analysis model after a change in the design?
2. How can we relate analysis models of various levels of abstraction with the design model?

The first aspect is the efficiency related to the meshing of geometric designs for finite element analysis. Before analysis can be performed, one needs to create a mesh, i.e. a discrete break-up of the geometry of the product into simple elements, such as tetrahedrons or hexahedrons for a 3D model. This process of creating a finite element mesh from a suitable geometry has been largely automated, but because of the complexity of the problem it can still take a considerable amount of computation time, depending on the specific characteristics that are desired from the mesh, and the meshing method. In many cases, the mesh is completely recreated in each iteration of the design process. In particular when the modifications are small, it might be more efficient to reuse parts of the mesh from the previous model. We propose a new method for efficiently remeshing an analysis model based on an earlier mesh, and show the gains that can be achieved. The advantage of this method is that it can produce high quality meshes in less time. At the basis of this remeshing approach stands our novel method for meshing mechanical models, which in turn is based on an existing meshing method, namely variational tetrahedral meshing (VTM) [Alliez et al., 2005]. We propose extensions which make VTM suitable for meshing mechanical models. The resulting method works in a highly flexible manner with the mesh, and is therefore appropriate to base our remeshing approach on.

The second aspect we consider is the integration of abstract analysis models and the design model. Analysis is regularly performed on the exact product geometry. Details can be left out, or properties can be computed more efficiently on lower-dimensional models. Working with a range of such abstracted models and the design model, causes a lot of work on reanalysis when the design model has been modified. Changes in the design should preferably be propagated automatically to the abstract analysis models. We can even imagine a need for modifying an abstract model itself after an analysis, and propagating changes back to the original geometry and other abstract models. This would be useful when the outcome of the analysis prompts for changes in the design, which in particular happens when the analysis concerns shape optimisation. We discuss and demonstrate a way to achieve such an integration of models by means of *analysis views* on a product, in addition to a design view and possibly other views, in the context of multiple-view feature modelling, which allows for propagation of changes



**Figure 1.3:** Outline of the thesis

between all these views. The views basically are different feature models of the same product, and together they form the product model.

A more detailed description of the context wherein we approach each of the aspects, will be given in the next chapter. But first, we give a brief overview of the contents of this thesis.

## 1.4 Structure of this thesis

The outline of this thesis is presented in Figure 1.3. It can be used as a guide in understanding the logical structure of the thesis.

We start by discussing the context of the problem in more detail in Chapter 2. Here we describe the essential characteristics of both design and analysis, the characteristics of their respective models, how people wish to combine the two, and the problems this poses.

With a clear idea of the context, we lift out two aspects of the integration of these two disciplines:

1. efficiently remeshing analysis models (Chapter 3–6),
2. maintenance of abstract analysis models in conjunction with the design model (Chapter 7–8).

Focussing on these two aspects, we propose and discuss methods to improve the efficiency of the design and analysis process.

The first improvement is essentially the efficient reuse of earlier generated high quality analysis meshes, after a model has been modified and analysis has to be repeated. In Chapter 3 we give some background on meshing for (finite element) analysis, in particular with tetrahedral elements. In Chapter 4 we introduce our novel meshing method for this research, which is based on a method for generating high quality meshes of non-mechanical models. We discuss the techniques that we have employed to make the method suitable for meshing mechanical models. In order to reuse parts of the mesh between models that share similarities, we have to identify what exactly the differences and similarities between these two models are; we present the model that we have developed to capture this information in Chapter 5. In Chapter 6 this all comes together, where we present a method to efficiently remesh analysis models, and the gains that we have observed.

The second improvement results in better maintenance of abstract models. We propose to integrate the design and analysis models more closely in a fundamental manner in a multiple-view feature modelling approach. We call these analysis models that are linked to a design model *analysis views*, and the basic idea is presented in Chapter 7, after a discussion of some other approaches to the integration of design and analysis models. In Chapter 8 we discuss a method that we have developed to implement some of the ideas that were put forward in the previous chapter.

We end with conclusions on what has been achieved in Chapter 9, and reflect critically upon issues that remain. We also look ahead which further improvements we might expect in the future with respect to the integration of design and analysis.

## 1.5 Contributions

This thesis includes several contributions that have been published as peer-reviewed articles. The contents of Chapters 4, 5, 6 and 7 correspond, respectively, to the work described in [Sypkens Smit and Bronsvort, 2008], [Sypkens Smit and Bronsvort, 2007], [Sypkens Smit and Bronsvort, 2009a], and [Sypkens Smit and Bronsvort, 2009b]. The content of Chapter 8 consists of recent work, and has not yet been published.

## Chapter 2

# Design and analysis models

This chapter serves to provide some background on the subjects that are central to this thesis. We start, in Section 2.1, by placing computer-aided design (CAD) and computer-aided engineering (CAE) in a historical context. Next, in Section 2.2, we characterise design models and their common representations in more detail, and give a brief overview of feature modelling. In Section 2.3 we look at the characteristics of analysis models, and explain the essence of finite element analysis (FEA). We conclude with a discussion of how design and analysis are commonly used side-by-side in the product development cycle, and what the resulting bottlenecks are, in Section 2.4.

### 2.1 Historical perspective

Computer-aided design and computer-aided engineering started out as two separate fields of both practice and research [Hughes et al., 2005]. The first was to replace the practice of drawing designs by hand. The computer could offer higher precision in drawing, assist the designer with simple calculations for dimensioning, make it easier to experiment with variations of a design, and quickly undo mistakes. It offered a more efficient and versatile drawing tool. The *Sketchpad* project developed by Ivan Sutherland in the early 1960s [Sutherland, 1963] is widely regarded as the first instance of interactive CAD [Bozdoc, 2010].

From that point, major companies such as General Motors, Lockheed and McDonnell Douglas started directing a lot of research into CAD and related technologies. The trend of internal development was set from early on. The high commercial value of CAD technology made that a large part of the advancements were made outside of academic institutions. From within the large companies that had so much to gain from digital design techniques,

emerged the CAD industry. The software that they initially developed for themselves, quickly turned out to be a profitable asset that could sell, and thus it was generalised into tools and libraries that were marketable.

Promising technologies would quickly leave the academic realm to be exploited commercially. Academic institutions did deliver major contributions to the CAD industry, but it seems to have been a somewhat one-sided affair. Still today, there is a tension between companies developing CAD software and academia trying to research this matter. The companies are generally unwilling to talk in detail about their technologies, or about the problems that they or their clients are facing. Also, it is nearly impossible for independent scientists to access the advanced models that large companies are working with. This attitude of secrecy results from the fear that giving any information to the competition, might make them lose part of their edge over that competition. Although perfectly understandable, it probably does hamper scientific research into CAD.

Another consequence of this closed attitude is poor interoperability. For example, closed file formats, different internal tolerances, and lack of standardisation can make it difficult to work with a heterogeneous set of modelling programs, both for normal users and researchers. Fortunately, it seems that this is slowly changing. The call for open standards, interoperability and free control over users' data, which has gained traction over the last decade, seems to have some effect. Companies developing CAD software are getting more permissive with their source codes, and open source offerings such as Open CASCADE [Open CASCADE S.A.S., 2010] are trying to get a foothold in the market, creating a stimulus to compete less on the basis of lock-in, and more on merit.

The earliest instances of CAE occurred more than a decade before the start of CAD. They basically came down to the numerical approximation of physical problems that were difficult or impossible to solve by hand. Since computers were relatively slow and numerical algorithms not as refined as they are today, the geometry of the problems that could be practically investigated was simple. Comparatively little time was spent describing the geometry of the domain, and usually it would be implicitly part of the numerical code. Decoupling the geometry of the domain from the numerical routines to solve the physical problem requires more complex code, which only became feasible when computers arrived that were easier programmable than those of the late 1940s and 1950s. The finite element method, developed in the late 50s and early 60s, was one of the first methods that was able to handle arbitrary domains.

For almost any technology and application it holds that it could make tremendous advancements because of the quick progress in computer hard-



ware. For CAE the converse statement holds, as it could arguably be called the most important driver of the development of the computer. The need to perform complex calculations concerning, amongst others, ballistics, nuclear physics, and fluid dynamics, in a sense prompted the invention of the computer [SIAM, 2010]. The commercialisation of software for numerical analysis seems to have happened slower than for CAD software, and certainly resulted in much less division between the academic institutions and industry. This can be explained by the more extensive breadth and variation in analysis research. The development of CAD software was about creating a single, complex approach that enables modelling wherein all types of shapes and operations on these shapes can be combined, whereas the implementation of numerical analysis for a specific problem is relatively simple to do. The problems in numerical analysis, however, are very diverse, and the same holds for the solution methods. It is much harder to generalise all these in a single system, especially in such a way that the system can be used by someone who is not experienced with the technical details. The degree of expert knowledge that is required to perform analysis is high; meshing, choice of Krylov subspace method, parallelism, error analysis, application of multigrid, modelling of the physical problem, and time discretisation, are all elements that people need to be knowledgeable of when performing analysis. Analysis is to some extent an inherently academic exercise, at least more so than CAD.

The two fields of CAD and CAE matured separately over the decades and are now quintessential to product development. As both grew more complex, their underlying data structures evolved naturally. At the same time, however, the demand to perform analysis with CAD-developed geometries grew. Most products in development today are at some stage subjected to analysis. At that point, a geometric input suitable for the analysis is usually derived, largely by hand, from the CAD geometry. This is a laborious task that regularly has to be (partially) repeated. Although the design and analysis models in essence describe the same object, the models are different, and going back and forth between them is a clear bottleneck within the product development cycle. Trying to remove this bottleneck is, in effect, an effort to improve the integration of CAD and CAE.

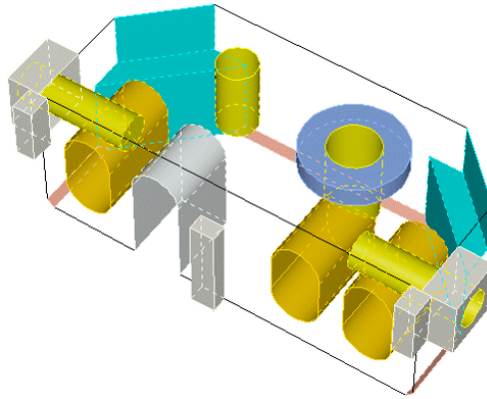
To better understand the issues concerning the integration of design and analysis, we continue by individually characterising each of their models and commonly used representations.

## 2.2 Design models

A design model is used in building a precise description of a product, most importantly its shape [Lee, 1999]. Most likely the designer will already have an idea of the shape when he starts to input the design model, but it is within the process of working with the design model that he makes the shape and all the dimensions exact. We further characterise design models by discussing the following two topics: 1) underlying geometry and topology, and 2) high-level elements for construction and interaction.

Design models need to be able to reflect arbitrary shapes and allow precise control over the shape, e.g. having a curve or surface pass through particular points, or the specification of derivatives of curves at certain points. The most common way to achieve this is by having curves and surfaces in the underlying representation that can be manipulated on an individual basis. To this end, the curves and surfaces are represented by means of simple splines or NURBS. The curves and surfaces are all stored in a network of vertices, edges and faces that records how the geometric elements relate topologically to each other. Commonly the only geometric elements stored in the design model are those of the boundary representation (BRep), mainly faces, edges, and vertices, but some models store more elements in order to enable a more advanced modelling approach. If only a BRep is stored, then the topology is often required to be *manifold*, which basically means that it can only represent a solid, three-dimensional shape. More complex models enable the construction of *non-manifold* topology [Weiler, 1988]. The topology describes where geometric elements meet each other, which generally involves a certain *tolerance*, e.g. the bounding curve segments of two adjacent surfaces might not be strictly identical. This is because exact geometrical computations are not feasible with computers. Since the boundary of an object can be divided into a set of surfaces in many ways, the underlying topology of the design model says as much about the choices that were made in modelling the object, as about the object itself: identical objects can have different underlying topologies.

The low-level geometry and topology are not the only elements to constitute the design model, but they are present in almost all design applications. The designer, however, does not normally work with the geometry on this level, but has higher-level entities at his disposal, such as features [Shah and Mäntylä, 1995]. Features represent parametrised shapes that are ultimately described by low-level geometric elements. The design model is thus manipulated through the addition and reparametrisation of features, and as such they are the primary constituents of the model, whereas the lower-level geometry that comprises the BRep is derived from the feature model by



**Figure 2.1:** A feature model with a variety of different features

boundary evaluation [Requicha and Voelcker, 1985]. Also, the designer can relate the parameters, such as size and relative position, of different features to each other by means of *constraints* [Hoffmann and Joan-Arinyo, 1997]. These are also part of the design model. Lastly, there can be a broad set of properties that do not directly relate to shape, such as material and colour, associated with (parts of) the model. These are usually implemented by means of *attributes* that are linked either to the features or directly to the low-level geometric elements. They are also part of the design model, as they have to be properly maintained when modifying the model. All together, these are the basic components that make up a design model.

### Feature modelling

We already mentioned the use of features to efficiently construct models consisting of low-level geometric primitives. A feature is a high-level building block for modelling with a shape that is usually common to more than one model. See Figure 2.1 for an example of a feature model. The concept of *feature modelling*, however, goes further than offering prefab combinations of low-level geometric primitives [Bronsvort et al., 2006]. Feature modelling as an approach to model construction, is not so much about quickly defining the geometry of a single, specific model, as it is about the commonalities to all models that we are constructing.

Thinking about commonalities inevitably leads to include meaning in the reasoning. Models have common shape characteristics because the shape has a certain meaning [Shah and Mäntylä, 1995]. Not all similar shapes have the

exact same meaning, and different shapes can represent a similar meaning, but no shape goes completely without meaning, even if it is principally aesthetic. The more scientifically common term for meaning is *semantics*. If the semantics of features is explicitly taken into account by the design system, we speak of *semantic feature modelling* [Bidarra and Bronsvort, 2000]. Storing semantics in addition to the geometry requires a more advanced representation than just a BRep. For example, a cellular model [Bidarra et al., 1998] has been used for this purpose.

If the modelling system knows of the semantics of features, then it will be able to better assist the designer. To this end, a feature modelling system must thus not offer common shapes, since these can have different semantics, but common semantics. For instance, a cylindrical shape could be used to represent the geometry of both a pin and a hole in a model. In a feature modelling environment, the difference between these two would be made explicit. But feature semantics goes beyond the distinction of a boolean union and subtraction. The cylindrical hole shape, for example, can be used for a feature that pierces through the model, i.e. a through hole, or it could result in a single opening, representing a blind hole. Also, not all holes, whether they are through holes or blind holes, are equal. Some holes should stay completely void of material, whereas for other holes this is not important. The hole might call for a certain clearance, not only at the entrance of the hole, but also in a larger area in front of the hole. It might seem a silly idea that a designer would accidentally fill up a hole with another feature, but when the set of features contained in the model gets larger, it does become harder to keep track of it all. There might be multiple designers working on the model, each focussing on different parts of the model, making it harder to keep track of all the semantics when combining the parts. With knowledge of the semantics, the modelling system can assist the designer through *validity maintenance* [Bidarra and Bronsvort, 2000]. When the semantics of a feature is threatened to be violated, the program can inform the designer of the violation. One step further is to use the semantics to restrict the freedom of the designer in interactive operations on the model, e.g. for positioning, orienting, or dimensioning a feature in the model.

Common design features are protrusions, slots, holes, ridges, and pockets. Although there have been several attempts at creating a complete feature taxonomy, this has proven difficult. Since both the need for certain shapes and their associated semantics are not universal, the ability to create new classes of features is an essential element of feature modelling. Commercial systems are already equipped with large libraries of features. Although they offer excellent tools for modelling, and are therefore used throughout

industries, they still have some way to go when it comes to semantics and creating an environment that has a higher-level understanding of a model than just in terms of its geometry.

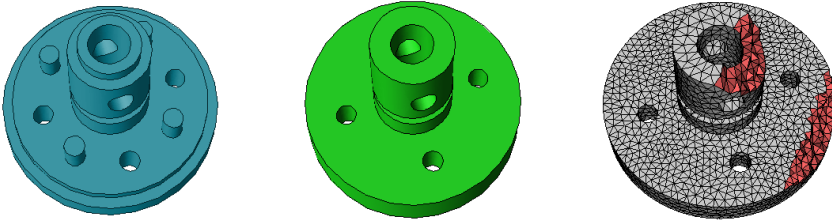
In particular challenging is the incorporation of freeform features [van den Berg et al., 2002; Pernot et al., 2008]. With the ability to manufacture smooth, curving, and elegant shapes, the demand for products with such shapes has been continuously increasing. Efficiency in designing such products, however, is comparatively poor, as their models are generally not constructed by means of features, but with more elementary, geometry-oriented operations. Generalising freeform shapes and the geometric interactions between them, such that they can be represented as features, is still a topic deserving further research in feature modelling [van den Berg and Bronsvort, 2010].

The use of features extends beyond its application in part design, as they can also be used, though with possibly different semantics, for other product development phases, such as analysis, manufacturing planning, and assembly modelling and planning. Each phase can have its own view on a product with different features, related to the intuitive decomposition of the product from the point of view of that phase [Bronsvort and Noort, 2004]. In an analysis view, for example, there could be features to indicate boundary values and conditions, mesh density requirements, and specifications of the maximum heat, stress, etc. that is admissible. The concept of multiple-view feature modelling is discussed in some more detail in Chapter 7.

## 2.3 Analysis models

An analysis model, in our treatment, is a geometry description on which an analysis is based. This is generally *not* the analysis mesh, as the mesh is generated from the analysis model. Instead, it represents a suitable geometry to be meshed and contains all the information to drive an analysis regardless of a specific mesh. The analysis mesh, often a finite element (FE) mesh, is a decomposition of the analysis model into simple geometric elements such as triangles and quadrilaterals in 2D, or tetrahedrons and hexahedrons in 3D, that are instrumental in the numerical analysis [Reddy, 2005]. The vertices that bound these elements, such as the four vertices at the corner of a tetrahedron, are called the (mesh) *nodes*. Depending on the particular solution method, it is possible that additional nodes are placed regularly in each element, e.g. at the midpoints of the edges, or the centre of the element. A wide range of *meshing software* exists to perform the construction of a mesh based on a given geometry, offering quite a lot of control over the desired characteristics of the mesh.

Figure 2.2 shows an example of a design model, together with a cor-



**Figure 2.2:** Design model (left), analysis model (middle), and tetrahedral mesh (right)

responding analysis model and a corresponding mesh. The distinction, however, between the analysis model and the mesh is not always clear in practice. In some cases, there is only a mesh and no true analysis model. In addition to providing the geometry, an analysis model can also provide information to steer the mesh generation, and all other inputs for the analysis, such as initial values and boundary conditions [Chand et al., 2008].

The geometry of the analysis model is almost always an abstraction of the design geometry, better suited for analysis, though the difference between the two can be small. Usually, not all detail of the design is needed for the simulation of the physical behaviour of the real product. Too much detail might even reduce the global accuracy of the result, in particular when the result needs to be obtained in a limited amount of computing time. With too many geometric details, computational resources are devoted to sections that are irrelevant for the central question that the analysis is supposed to answer, with an increased likelihood that the fidelity of the result is worse. Analysis models thus tend to have a notably simpler geometry. It is even common for the geometry of the analysis model to be of a lower dimension than the design model, or, in case of symmetries, to only represent a subset of the model. For example, thin plates can be modelled by *shell elements* that have 2D geometry (embedded in 3D space), and incorporate the thickness of the plate as a numerical parameter that affects the behaviour of the elements. This reduces the complexity of the calculation compared to full-dimensional analysis, without necessarily spoiling the fidelity.

The representation of the analysis model varies. In some cases it is a BRep composed of splines, similar to the common representation of the design model, but rarely can the geometry of the design model be used directly as the analysis model. Even if there is no need to reduce detail, then the meshing software might still be unable to deal with the geometry of the design model. The common alternative is a finely faceted (usually

triangulated) model that has been generated from the design model. This faceted data structure is straightforward, so that it can easily be transferred and manipulated, and has no issues with tolerance in joining the individual elements, as only the topology is explicit. Algorithms that operate on this kind of representation are relatively easy to implement. A faceted BRep of a design model, however, is by itself not yet a usable analysis model. The geometry most likely needs to be adapted to make it more suitable for meshing, and be prepared for the assignment of boundary values and conditions. More about this in Section 2.4.

The analysis model needs a topology that divides the surface into more or less smooth patches, i.e. regions of the surface where the first derivative in all directions is approximately continuous everywhere except on the boundary of the region. This is required by the meshing software for the creation of a good mesh, as most meshing algorithms treat edges in a special manner, e.g. by starting and forcing node allocation there. For example, edges that do not separate surfaces at an angle with each other, can cause unnecessary mesh nodes to be put into the mesh, unjustifiably forced at specific positions, which could needlessly lead to a lower quality result. In addition to a breakdown in sufficiently smooth regions, this topological description also contains the faces, edges and vertices on which boundary values or boundary conditions are specified. Also part of the analysis model is information for steering the generation of the analysis mesh, in particular the mesh size. This is most often stored in a regular 3D grid or a 3D mesh, possibly the mesh of a previous analysis, but no real standards seem to exist as these data structures often have close ties with the particular meshing method.

### **Finite element analysis**

The most widely used technique for performing analysis on a design is finite element analysis. The method was developed with an initial focus on engineering problems, in particular structural engineering. With a history going back to the late 50s and early 60s, it has a very well developed mathematical background [Brenner and Scott, 2007]. The theory is concerned with solving systems of partial differential equations (PDEs) or integral equations, which are the mathematical formulations for almost all physical phenomena, including engineering related ones. It does not constitute a single algorithm that can be used to solve arbitrary PDEs, but rather is a formalism for deriving such an algorithm, given a specific class of equations. Each problem class thus needs its own algorithm, and generally this derivation requires mathematical skill that warrants human involvement.

The popularity of finite element analysis is undoubtedly grounded in the huge versatility of its applications. The set of problems that can be subjected to FEA is very large, as is the number of choices that can be made in the solution process at different independent levels. A choice for finite element analysis does not imply a single specific algorithm, and what constitutes a good approach depends on many factors, such as the particular nature of the physical problem (domain deformation, extreme/near-singular inputs, crack propagation, etc.), the mathematical characteristics of the problem (order of the PDEs, non-linearity, elliptic/hyperbolic/parabolic nature), the discretisation of the domain, the required accuracy, and the available computer hardware (CPU and memory). It is normally not known beforehand what the best approach would be, though one can make an educated guess based on previous experiences. Of course, within a particular analysis environment, the number of choices is likely to be limited, as only a subset of the known approaches and combinations thereof will have been implemented. Therefore it is quite common that researchers of FEA, at least when it comes to the more fundamental aspects, rarely rely on commercial programs, but write their own code. With a history of over 50 years, it is still a highly researched topic.

Due to the broad range of mathematical intricacies that are part of the theory of FEA, we avoid a mathematical treatment here, but instead present only the basic ideas [Reddy, 2005]. A principal step in FEA is to convert the system of PDEs (or integral equations) into its *variational* or *weak* formulation. This formulation consists of integrals that, under the right assumptions, define an inner product on an infinite dimensional function space. The specific characteristics of the equation, foremost differentiability, require that this is a *Sobolev* space. In this weak formulation appear both the unknown function  $u$  that satisfies our original equation, and a function  $v$  that, apart from a small constraint, is arbitrary. This formulation can be discretised in space, such that there effectively is an individual equation for each segment of the domain. In case of a 3D mesh, there would be an equation that relates  $u$  and  $v$  over each individual element. Since the formulation should hold for arbitrary functions  $v$ , we can choose functions that are 0 everywhere except over a particular element. For each element there exists such a function, and these are called the basis functions. Simple polynomials are chosen as basis functions. This way, all the basis functions together span the entire discretised domain, but their influence is local. Together they form a space of piecewise polynomial functions. The solution  $u$  has now become approximated by  $u_h$  and will necessarily lie in this same polynomial space. Because of the derivatives of  $u$  that appear in the variational formulation, the equations over the elements all refer to their neighbouring elements,



except on the boundary, and thus we cannot solve the equations individually, but they have to be assembled into one big linear matrix equation with the values of  $u_h$  at particular nodes of the elements as the unknowns.

The solution of the matrix equation depends on the nature of the problem. If it is a steady state problem, then there is a single piecewise polynomial solution, and the matrix equation can be solved either by a direct solver or by a Krylov subspace method, which is the only feasible approach when the system of equations is large. If the problem is time-dependent, say the shape evolution of a metal object that under pressure deforms over time, then the matrix equation is itself again a system of ordinary differential equations (ODEs). This system can be solved by numerical integration, wherein the time  $t$  is discretised. This happens separately from the discretisation of space and is not specific to the finite element method. In that case, for each time step a complete system of equations, similar to the one for steady state problems, is solved. All these solutions together represent the approximation of the evolution of  $u$  over time.

The easiest choice for the polynomial functions that leads to a continuous solution is linear polynomials. However, if the PDE is of higher than second order, i.e. it contains a third derivative or higher, then higher degree polynomials such as quadratic ones are required. But regardless of the order of the PDE, higher degree polynomials can always be used. They can increase the accuracy of the solution, in particular a good accuracy can be obtained with a comparatively low number of elements. However, higher degree polynomials do not assuredly lead to higher accuracy, but rather there is a more complex relation that involves, amongst other aspects, the nature of the problem.

The other major choice of importance is the discretisation into elements, or the mesh. Generally, there is a relation that smaller elements lead to higher accuracy, but again, depending on the characteristics of the specific problem at hand, there are some cases where this does not apply. There has been a lot of research into meshes and the relation to the accuracy of the solution. The elements can be varied in size throughout the mesh, using larger elements where the domain does not have intricate geometric detail, or where no outcome of interest is expected, and using smaller elements in selected regions where the solution varies strongly, or the geometry of the domain has small details. In time-dependent problems, the mesh can even be adapted after each time step, resulting in a dynamically changing mesh. This can be done by moving the nodes of the existing mesh, or by introducing new elements into the mesh and removing old ones. More information on mesh generation follows in Chapter 3.

Another important aspect to finite element analysis is error estimation

[Ainsworth and Oden, 1997]. The solution to the system that is being solved is already an approximation to the real solution of the equations we started out with. The real solution is most likely not a piecewise polynomial, but in most cases the latter should offer a reasonably close approximation. On top of that comes the inaccuracy through the discretisation of the derivatives, and the inaccuracies that result from solving the linear system, and possibly integrating this in time. The easiest method to estimate the accuracy is to perform the analysis with various mesh sizes and time steps, and look at the convergence behaviour. Usually the calculations for comparison are performed with coarser meshes and time-stepping, but it takes time to perform the calculations nonetheless, and then the differences have to be turned into an estimate of the accuracy. There are other methods for error estimation based on theoretical arguments, but these often only apply to academic problems, or the bounds on the accuracy are too large to have practical value.

In this thesis we do not deal with the details of actual analysis, i.e. the heavy-duty calculations, but focus on the steps leading up to this. The aspect of FEA that is most important to us, is mesh quality of tetrahedral meshes. More on this in Chapter 3.

## 2.4 Model integration in the product development cycle

Originally, the design and analysis models stood more or less on their own, each within its own domain of application, but over time, with the turnaround of the product development cycle getting progressively shorter, this has changed. With a growing need to perform analysis on specific instances of a design, consequently grew the need for closer and increased interaction between the two models.

Design and analysis models are both part of the product development cycle, which encompasses all the tasks that are commonly performed from the first conception, up to the actual manufacturing and marketing of the product. When sequentially listing all the tasks, this gives the impression of a mostly linear process. But in fact it is more of a cycle, since most products are, at least partially, based or inspired upon a previous product; incremental improvements are the norm in product development. But a similar pattern appears when we zoom in to the development steps for an individual product. There is a certain interaction between all the steps, because the geometric design, the physical behaviour/performance, manufacturing, assembly, packaging, marketing, etc. of the product are all interrelated.

Obviously, the interaction between geometric design and analysis is of particular interest to us. Analysis is principally used to verify that the

manufactured product behaves to specification. If this turns out not to be the case, then the design has to be adapted, and afterwards the analysis repeated. In particular since we try to minimise material use, for both economic and ecological reasons, and want to build products cost-effectively, thus using the cheapest material that can do the job, the margin between the (limit) specifications and the actual behaviour is likely to be small. In other words, the circumstances under which a product fails, tend to lie relatively close to the specified conditions under which it should never fail. Also, analysis is explicitly used to optimise shape, so that the outcome of the analysis directly determines (a part of) the shape of the design model. During shape optimisation, commonly a whole range of calculations is performed, each for a slightly different model. All this calls for frequent and close interaction between design and analysis, and thus their models.

So how can we characterise this interaction of design and analysis, in particular between their models, within the product development cycle? Because the diversity in types of design and types of analysis is so large, there are several kinds of specialised environments that all operate (at least) slightly different [Beall et al., 2004]. There is thus no single, complete characterisation of the *combined workflow* of design and analysis. We can only describe those aspects that are common and of importance to the integration issue. Furthermore, the combined workflow in current practice concerns mostly the derivation of analysis models from design models, which is essentially a one-way model conversion executed with a wide range of tools and techniques.

We are not aware of any survey research on the combined workflow of design and analysis in relation to the size and complexity of the models, the type or level of abstraction of the analysis models, and the type of simulations, as used in practice. We can only go by what scholars are indirectly suggesting about current practice by means of their research efforts, and our personal experiences. We recall that design and analysis models normally have a different geometry, as the analysis model represents an abstraction that is (better) suited for analysis. The abstraction covers a wide range, from slightly tweaking the design model, to being a dimensional abstraction that at first sight has little resemblance to the design model. Starting with a design model, there are several ways to get to an analysis model:

- Build the analysis model from scratch.
- Adapt the design model into an analysis model, sticking to the representation of the design model [Lee et al., 2005; Foucault et al., 2008; Thakur et al., 2009].

- Create the analysis model starting from a faceted BRep of the design model [Chong et al., 2007].

The first option is attractive if the geometry of the analysis model to be constructed is simple, at least compared to the design model. In this case recreating the few elements that are similar in both models is manageable. Creating a model from scratch by hand does, however, carry the risk of introducing errors. The engineer can, for example, make a mistake with the dimensions. Since there is no link between the two models, whenever the original model changes, the analysis model has to be updated by hand. Either the analysis model is updated each time a change is made to the original model, which is tedious, or after a range of modifications, which is difficult and error-prone.

The second option for creating an analysis model is to adapt (a copy of) the design model in its existing representation. This is attractive if the changes between the geometry of the analysis model and the design model are relatively small, e.g. if just a few design features need to be removed. When this is done by hand on a separate copy, there is no link between the two models. To maintain the relation, a specialised environment for performing the abstraction would have to be used, which is not very common at present. Also, there is more to be dealt with in preparing the analysis model than abstraction by removing a subset of the features. The analysis model has to be suitable for mesh creation, which means that a topology has to be specified and, more importantly, that the model should not exhibit any geometric characteristics that are detrimental for the generation of a quality mesh, such as very sharp angles or very thin faces. This possibly requires a specialised modelling environment, as the design software that causes such nuisances is often not well suited for removing them. Exporting and importing the model, might then lead to tolerance issues with the definition of the geometry, e.g. resulting in a model that is not watertight.

The third option for constructing an analysis model is through manipulation of a faceted model generated from the design model, which is then adapted in steps to become a suitable analysis model. The model is made watertight, the surface is remeshed, feature size analysis is performed, and in a (partly) automated fashion the model is simplified. Although there is no real link between the design and analysis model with this approach, the advantage is that many steps can be automated. It is often noted though, in particular with respect to dimensional reduction, that the automated steps can give you a quick start, but frequently this will not give a result exactly as desired. Some manual tweaking seems unavoidable.

The distinction between these three approaches to obtaining an analysis model is not very sharp. There are approaches in use that are a mix of these,

where, for example, first the design model is somewhat simplified, and then exported as a faceted model for further automated processing.

It is obvious that the workflows just characterised, have deficiencies with respect to efficiency and maintaining consistency. In Chapter 7 and 8 we focus on the aspect of keeping design and analysis models consistent. But first, in Chapters 3 to 6, we focus on meshing of analysis models, which has to be performed each time before the actual calculation can start, and on how efficiency can be improved here.



## Chapter 3

# Analysis and meshing

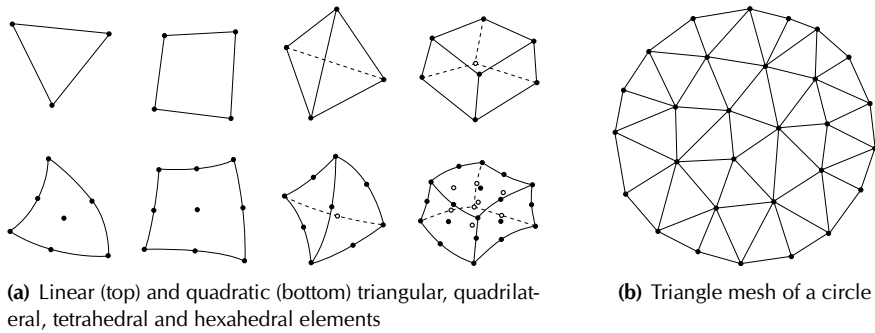
For finite element analysis, a mesh is required of the geometry of the analysis model. This mesh is a decomposition of the geometry into small, simple elements. The procedure for generating such a decomposition is known as *meshing*.

Meshing is not limited to the field of analysis, but it is also performed in, for example, computer graphics. There it commonly applies to just the surface of 3D objects, instead of the volume, and the elements are almost always triangles. In the context of surface meshes, the process of meshing is also called *tessellation*, and this field comes with a rich range of algorithms and techniques [Botsch et al., 2010]. Meshing for analysis, however, predates meshing for display of 3D graphical models by more than a decade, with the latter starting in the mid 70s, and the former in the late 50s.

Although there is some overlap between the meshes and meshing techniques from these two disciplines, their purpose is essentially different. Therefore they each have their own perspective on what determines the quality of a mesh and which are its favourable characteristics. We are mainly interested in analysis, and start this chapter, in Section 3.1, by looking at meshes for this purpose. Next, in Section 3.2, there is some background on mesh generation techniques. Finally, in Section 3.3, we argue that the efficiency of mesh generation in the context of the product development cycle, can be improved.

### 3.1 Meshes for finite element analysis

The most common type of elements in meshes for finite element analysis, are triangles and quadrilaterals for 2D geometries, and tetrahedrons and hexahedrons for 3D geometries. Figure 3.1 shows some examples of elements and a



**Figure 3.1:** Examples of mesh elements and a mesh

simple triangle mesh. The choice for a particular type of element depends on the geometry of the analysis model, and the capabilities of both the meshing and the analysis software. Hexahedral elements are generally perceived as superior due to better numerical stability and more efficient convergence [Tautges, 2001; Baker, 2005]. Meshing with hexahedral elements is, however, very difficult, and automated methods are scarce, with the performance depending strongly on the geometry of the model. Models with complex boundaries, and a topology that cannot be broken down easily into block-like shapes, can therefore lead to a choice for tetrahedral elements, even though hexahedrals would be numerically favourable. However, depending on the nature of the particular problem, and the quality of the tetrahedral mesh, the numerical stability does not have to be an issue [Cifuentes and Kalbag, 1992; Benzley et al., 1995]. In this thesis, we focus on tetrahedral meshes, because they can more easily be generated by autonomous procedures. This is the type of mesh that should be assumed in the sequel when it is not specified explicitly. Another way in which we denote either triangular or tetrahedral meshes is by the term (*2D* or *3D*) *triangulation*.

The role of mesh elements is to serve each as a domain on which basis functions are supported. The basis functions represent a continuously varying (solution) value over the element by interpolating values at the nodes. In a linear element, the nodes commonly lie at the vertices. For higher degree interpolation, additional nodes are placed on the boundary and in the interior of the element. This enables smoother variation of the solution, with higher degrees of continuity between the elements as well. Mesh generation is typically just concerned with generating the bare geometry of the elements, as the additional nodes are trivial to generate. Elements on the boundary, however, deserve special attention if higher degree elements are used, as it is



necessary that their geometry represents the model boundary as accurately as possible with the polynomial approximation, following the smoothly varying shape of the analysis model, which is derived from the geometry of the design model (see Section 2.3). For this purpose, some methods require access to the original spline surfaces, instead of a fine approximation with triangles.

In general there is a distinction between *boundary conforming* and *non-boundary conforming* meshes. The latter type does not represent the boundary, or at best makes a crude attempt. Generally these meshes are grids that overlap with the analysis model, with the cells that fall inside the model, according to some measure or heuristic, being used to represent the object of analysis. Non-boundary conforming meshes are easier to generate, in particular when quadrilateral or hexahedral elements are used, and because of this, they were more popular in the earlier days of analysis.

A conforming boundary does explicitly represent the boundary of the analysis model, by having all nodes on the mesh boundary at coordinates that lie on the actual surface. Additionally, the element faces that form the mesh boundary must accurately follow the actual surface [George et al., 1991]. Clearly, it requires some kind of measure to determine what this means. To this end, the Hausdorff distance and the maximal variation of the surface normal over the element, are commonly applied. It depends on the problem and the desired accuracy, whether a non-boundary conforming mesh can be used. This is more likely the case if the need for accuracy is limited and the physical behaviour on or near the boundary, or the effect of variations in shape, are of relatively minor importance. Our interest is directed towards boundary conforming meshes, because they are more commonly used nowadays, even though they are more challenging to generate.

When generating the mesh, the curvature over the surface of the analysis model can serve as a guideline for the number of elements that needs to be placed. In areas of higher curvature, more elements are needed to accurately capture the boundary. Consequently, the mesh elements will vary in size, with smaller elements near areas of the boundary with small details or high curvature. Variation in element size is also useful in relation to the physical problem. At places where the derivatives are high, i.e. the solution varies strongly from point to point, more elements will most likely improve the accuracy. For example, this could be around areas where a high concentration of stress, or high turbulence is expected. The desired size of the elements is given by the *mesh sizing function* [Quadros et al., 2004]. This function can specify either an absolute or a relative size for the elements. Since the elements are generally not perfectly regular, their size is defined by some measure such as the average edge length, or the radius of the circumcircle

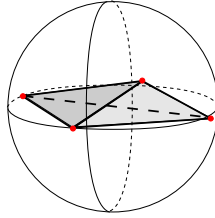
or circumsphere.

Generally, an absolute size requirement  $\mu_x$ , where  $x$  denotes position, is considered to be satisfied if it holds for the actual size of element  $T$ , denoted by  $l_{T(x)}$ , that  $\frac{1}{2}\sqrt{2}\mu_x \leq l_{T(x)} \leq \sqrt{2}\mu_x$  for all  $x$  contained in the element. This means a considerable margin for deviation from the requested size, but without such a margin it would be virtually impossible to make every single element adhere to its sizing requirement.

However, smaller elements, and thus more elements, are not the only factor in the accuracy that is accounted for by the mesh. The shape of the elements is also important. Elements with small or large angles, or short edges in comparison to the average edge length of their element, are bad for the accuracy of the analysis. Small in this context means about  $5^\circ$  or smaller, and large means  $175^\circ$  to  $180^\circ$ . There is no single threshold that separates good and bad, but as an angle approaches the limit, the potential for causing problems rises very quickly. For the 2D problem, i.e. filling a planar shape with triangles, it is known that the angle and relative edge length can be systematically bounded in the mesh generation, such that quality triangles will always result. The only exception is if the boundary of the model contains sharp angles, as small angles are then unavoidable. Filling 3D domains with tetrahedrons is, however, harder. Procedures with guaranteed bounds on the angles exist, but the bounds are so low that it has little practical value over other approaches without such guarantees.

But even if all the angles and the edge lengths are good, there can still be bad elements in 3D meshes, in the form of *slivers*. A sliver is a flattened tetrahedron with its four vertices nearly in a single plane, positioned such that their projection to the plane forms a convex quadrilateral with no short edge. Such a tetrahedron has virtually no volume. See Figure 3.2 for an example of a sliver. Many measures for element quality, such as the radius-edge ratio, which is defined as the ratio between the circumradius of the tetrahedron and its shortest edge, do not account for the bad quality of slivers. Several measures have been introduced that do account for the bad quality of slivers as well, mostly by introducing some term relating to the volume. An example is the volume-length ratio, which is defined as  $V/l_{\text{rms}}^3$ , with  $V$  the volume of the tetrahedron and  $l_{\text{rms}}$  the root mean square of the edge lengths [Parthasarathy et al., 1994; Klingner and Shewchuk, 2007]. These measures generally reach optimality for the regular tetrahedron, i.e. a tetrahedron with six edges of equal length.

We can also consider optimality from the point of view of the analysis. Small and large angles can give trouble due to the interpolation of the basis functions and their derivatives. Shewchuk [2002b] has expressed the quality of linear elements in terms of interpolation error of a function



**Figure 3.2:** Example of a sliver

and its first derivative. These formulas could be used to optimise a mesh, resulting in the minimization of these interpolation errors, but explicitly optimising for these quantities does not seem common. However, it can be seen from the formulas that the regular tetrahedron is optimal in the scale-invariant measure, i.e. without looking at the influence of element size on the interpolation error. So striving for near-regularity in all elements, which basically all measures do, seems a sensible approach.

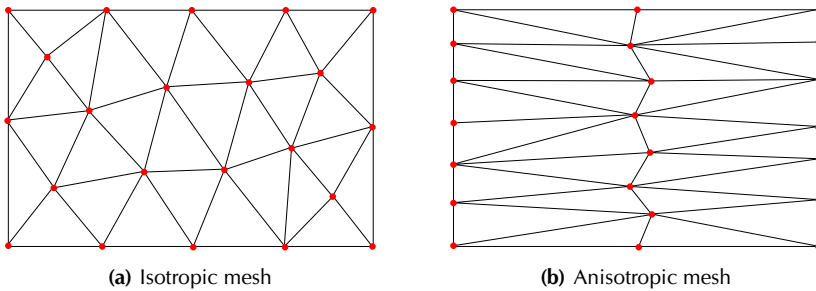
We must be careful though to distinguish individual element quality and overall mesh quality. A mesh with many very high quality elements, and a few really bad ones, could perform worse than a mesh that consists entirely of mediocre elements, but without a single element that is really bad. A single bad element can badly interfere with the outcome of a calculation [Shewchuk, 2002b]. Therefore, it is not only important to have high quality elements at average, but to avoid bad quality elements all together.

For some problems, however, very thin elements with very small angles, but positioned along a particular direction, can be desirable [Apel and Dobrowolski, 1992]. The solutions of these problems contain so-called *anisotropic features*, and the associated meshes are called *anisotropic meshes*, as opposed to the regular meshes that are known as *isotropic*. Anisotropic meshing is essentially an extension of varying element size over a mesh, but with a mesh sizing function that is a metric tensor, i.e. the required size varies depending on the direction. Figure 3.3 shows an isotropic and an anisotropic mesh side by side. In this thesis, we limit ourselves to isotropic meshes.

In the next section, we present an overview of the techniques and algorithms that are used in generating triangle and tetrahedral meshes.

## 3.2 Meshing methods

The collection of all meshing methods is far too large to describe here in full. For an extensive overview we refer to the literature. A lengthy mesh-

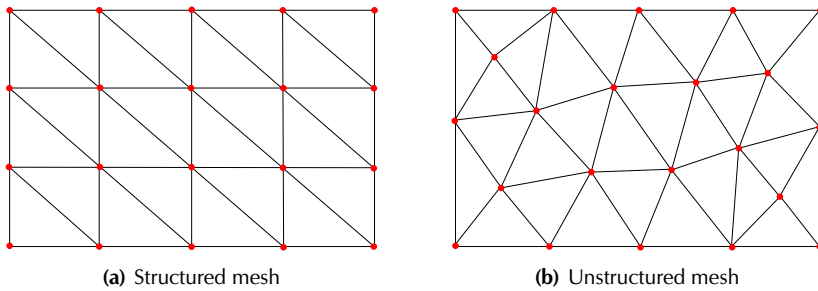


**Figure 3.3:** Isotropic and anisotropic meshes for a similar geometry and number of nodes

ing bibliography has been compiled in [Mackerle, 2001]. Two well-known meshing resources with broad content on the web are [Owen, 2006] and [Schneiders, 2006]. Comprehensive surveys include [Owen, 1998], [Teng and Wong, 2000], and [Shimada, 2006]. Eppstein [2001] has given a tutorial on the global optimisation of mesh quality. Du and Wang [2006] discuss recent developments in robust and quality Delaunay meshing. Frey and George [2000] cover many aspects of mesh generation in their book, which can be considered a standard work on the topic. We give a short overview of the basic techniques, with a focus on tetrahedral meshing. Many topics, such as surface meshing, variation in element size, theoretical guarantees, and anisotropic meshing techniques, are not discussed here.

Meshing algorithms can be compared on the basis of their characteristics. Some of these characteristics have been touched upon in the previous section, such as the type of mesh elements the algorithm produces, whether a boundary conforming mesh is constructed, the quality of elements, variation in element size, and how well the boundary is represented. Other characteristics are robustness, required input, speed, ease of implementation, and whether the approach is structured or not.

The robustness of the algorithm indicates the likelihood that the process fails in some sense, e.g. because of numerical instabilities or unexpected or unfortunate cases, such as degeneracy. Can it mesh arbitrary geometries, or is it likely to fail on some inputs? The required input specifies the kind of data sets the algorithm expects as input. For tetrahedral meshing algorithms this is commonly a *piecewise linear complex* or a *boundary representation* consisting of connected vertices, (non-linear) edges, and (non-planar) faces. Meshing a piecewise linear complex is less complex, since the curved faces and edges have already been discretised, but this might mean a concession to quality, over having the original faces and edges available to the algo-



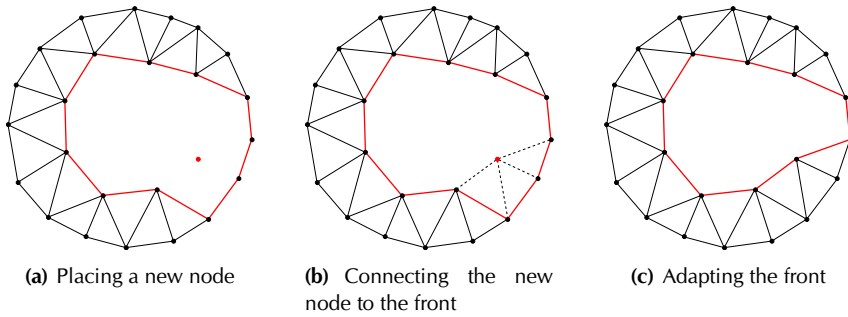
**Figure 3.4:** Structured and unstructured meshes for a similar geometry and number of nodes

rithm. The speed of the algorithm is important too. Generally, there is a relation that slower algorithms result in higher quality meshes, but this certainly does not hold for all cases. Longer running algorithms often follow some iterative procedure to optimise the quality of the elements, but as the running time increases, the improvements in mesh quality asymptotically approach the optimum, i.e. the longer the algorithm runs, the smaller the further increases in quality. Ease of implementation is not really a concern for the user, once the algorithm has been implemented and is available for use, but it is a characteristic that matters to developers. Some algorithms, in particular those that come with theoretical guarantees, can be very complex to implement, which makes their application impracticable.

There are two classes of meshing methods: structured and unstructured. The structured approach gives rise to very regular meshes, as it maps regular mesh patterns with a fixed connectivity to (parts of) the geometry of the analysis model. Unstructured methods do not work with a fixed mesh connectivity between the nodes, but rather place each node or element individually, or at least determine connectivity based on the node placements, e.g. through the Delaunay criterion, which will be discussed shortly. Figure 3.4 shows an example of both types. Since it is relatively easy to fill a geometry with triangles or tetrahedrons, unstructured methods are popular and dominant for these element types. Quadrilaterals and hexahedrons, on the other hand, are mostly used in structured meshing approaches, though there has been quite a bit of research into unstructured methods for these element types as well.

For tetrahedral mesh generation there are basically two major classes of algorithms: advancing front and Delaunay based methods.

Advancing front methods create a mesh by working from the boundary towards the interior. Because of this, the mesh quality near the boundary is



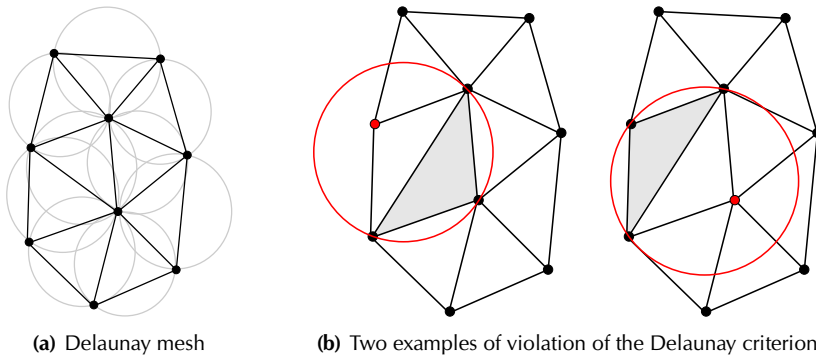
**Figure 3.5:** Mesh construction with the advancing front technique

at average very high, whereas the quality in the interior is lower, since the fronts of elements that move inwards meet there. Filling the last remaining voids with quality elements is more difficult, as the space might not fit the element shapes that are desired.

The first step in a typical advancing front method is to discretise the boundary into edge segments for a 2D model, or triangles for a 3D model. All these edges or triangles together constitute the *front*, or multiple fronts in case of holes (2D) or voids (3D) in the geometry. Starting with the elements from the front, which have dimension one lower than the mesh to be constructed, a new node is placed in the interior, such that it forms one or more new elements by combining it with the front, and the front is adapted. Adding a new node continues until the whole model has been triangulated. The technique is illustrated in Figure 3.5. This is the core idea, but many variations on this theme exist, mainly consisting of additional rules for node placement, for deciding where to advance the front, and for adding connectivity for multiple adjacent elements at once. These rules aim to improve the quality of the individual elements and the overall mesh, and help to fulfil additional requirements such as variation in element size.

The second class of meshing techniques consists of Delaunay based methods. This class is not completely disjunct from the advancing front methods, as the Delaunay principle is used in combination with that approach as well. The range of publications on this topic is very broad, as the subject has been widely researched. The majority of triangular and tetrahedral meshes for analysis is likely generated by an algorithm from this class.

At the core of all these techniques lies the *Delaunay criterion*, which states that the circumcircle of a triangle, or the circumsphere in case of a tetrahedron, must not contain any nodes other than the nodes of the element itself [Delaunay, 1934]. An element that satisfies this criterion is said to



**Figure 3.6:** The Delaunay criterion: the circumcircle of each triangle may not contain other nodes than those of the triangle itself.

have the Delaunay property, or simply to *be* Delaunay. See Figure 3.6 for an illustration of the Delaunay criterion.

It is known that any set of nodes with distinct coordinates has at least one triangulation for which *all* the elements are Delaunay [Delaunay, 1934]. This is called a *Delaunay triangulation* of the nodes. This triangulation is unique, except when there is degeneracy, which means that four nodes (five in case of a circumsphere) or more lie exactly on the same circle that is empty of other nodes. This obviously leads to multiple triangulations that are Delaunay. The simplest example of this is the triangulation of the four corners of a square, which can be done in two ways, and they are both Delaunay.

An interesting property of the Delaunay triangulation in a 2D plane is that it maximizes the minimum angle, i.e. of all possible complete triangulations of a particular node set, there is no triangulation where the smallest angle is larger than the smallest angle contained in the Delaunay triangulation [Lawson, 1977]. This seems attractive, considering that we associate very small angles with bad quality elements. In general, the Delaunay triangulation results in fairly regular meshes, in particular when the nodes are spaced regularly. Unfortunately, this minimisation property does not extend to triangulations in higher dimensions [Rajan, 1994], but the method results in fairly balanced meshes nonetheless. Because of this, and the ease of having a fixed principle defining the connectivity, the Delaunay criterion has become so popular.

Almost all methods for constructing a Delaunay triangulation work by building it incrementally. We explain a simple approach for a triangulation in the 2D plane: one starts out with a single triangle, which is large enough to contain all the nodes that are to be triangulated. Then a single node is added.

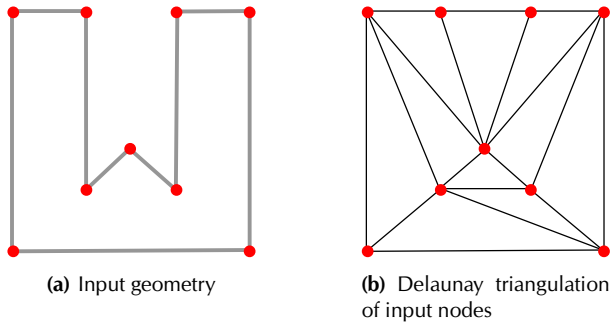
This new node falls inside the triangle, and here three new triangles are created by connecting the new node with the corner nodes of the containing triangle. This process is executed iteratively with all the nodes. However, after the first node, we need to take care that the Delaunay property is maintained for all triangles. This is done by finding all triangles of which the circumcircle contains the new node. These are the triangles that violate the Delaunay criterion, and therefore they are removed. The remaining cavity is triangulated by connecting the newly added node with all the nodes bordering the cavity. There are many variations on this approach, which, for example, introduce smart data structures to quickly locate the triangles/circumcircles that a node is contained in, or sort the nodes spatially and base the insertion order on this. Also, robustness is an import issue in these algorithms. For instance, it is necessary to determine with high precision whether a node falls inside or outside a given circle, which requires appropriate calculation techniques [Shewchuk, 1996]. Triangulation in 3D is more complex, but the ideas are similar.

We have just discussed the triangulation of a given node set. Another problem is the triangulation of a given geometry. In that case, the triangulation algorithm itself can generate the location of the nodes. Of course, with analysis in mind, building a mesh with high quality elements is the principle aim. A wide range of techniques has been published on this topic, many of which fall in the category of *Delaunay refinement*. This means that a coarse initial mesh is used as a starting point, which is then iteratively improved by inserting nodes. Many techniques are variations on the work of Chew [1989] to insert circumcentres, i.e. the centre of the circumcircle, of bad quality triangles. In particular when the triangles are very thin, the circumcentre is likely to fall outside the triangle itself, but because the new node is lying inside the circumcircle it violates the Delaunay criterion for the bad quality element and that element is thus removed from the triangulation. Another technique is the terminal-edge Delaunay algorithm, which inserts nodes on certain long edges of the triangulation [Rivara et al., 2001].

Delaunay triangulations always cover the convex hull of a node set. This means that for concave models, part of the triangulation has to be discarded as it lies outside the model, and, more importantly, that the boundary will not necessarily be represented by the Delaunay triangulation of the nodes. In other words, it might not be possible to only discard the elements that lie completely outside the model, as some elements are partly on the inside and partly on the outside. A simple example of this is shown in Figure 3.7, where two edges of the input geometry are missing in the triangulation of the input nodes. There are two solutions to this.

The first is adding nodes, or *Steiner points*, such that a Delaunay mesh





**Figure 3.7:** Boundary of input geometry not represented by Delaunay triangulation

develops that *does* contain the complete boundary. Intuitively it is easy to see that this can work as we imagine increasing the node density on the boundary. This makes it more likely that the edges between all nodes adjacent on the boundary (or triangles in case of a 3D model) are contained in the Delaunay triangulation. However, this can lead to very bad meshes, and some sophistication is needed to make sure that the algorithm finishes. A Delaunay triangulation that completely represents the boundary of an input model is called a *conforming* Delaunay triangulation. A practical algorithm to construct such a triangulation is described by Cohen-Steiner et al. [2002].

The principal alternative to a conforming Delaunay triangulation, is a *constrained* Delaunay triangulation. A constrained Delaunay triangulation basically uses a modified version of the Delaunay criterion. Edges and faces that lie on the boundary of the model, including edges and faces on internal boundaries that have to be explicitly represented in the triangulation, play a special role, as opposed to the standard Delaunay criterion that only deals with nodes. These edges and faces can affect the ‘visibility’ of nodes that might violate the Delaunay property of an element. Effectively, a node is invisible for a particular element if no straight path exists between that node and any point in the element’s interior such that the path does not cross any boundary edge or face. If a node is invisible for a particular element, this means that it does not violate the Delaunay criterion for that element. Because of this, some triangles (tetrahedrons) that could normally not exist in a Delaunay mesh, due to a node that is contained in their circumcircle (circumsphere), now are allowed in the triangulation since they are constrained Delaunay. Quite some work has been carried out on this topic, amongst others, by Shewchuk [2002a] who has presented an algorithm that provably results in a constrained Delaunay triangulation. In particular in

3D this problem is harder than it might seem, as some polyhedrons cannot be triangulated at all without the addition of Steiner points [Schönhardt, 1928; Bagemihl, 1948].

The algorithmic step that modifies a (Delaunay) triangulation such that afterwards the complete boundary is contained in the mesh, is called *boundary recovery*. Algorithms for creating conforming and constrained Delaunay triangulations usually start from a Delaunay mesh, in which case we say that the algorithms perform boundary recovery. After the boundary recovery the model boundary is completely represented by elements in the mesh, but for convex shapes there are still elements in the mesh that fall outside the model. These elements need to be removed in a process that is called *mesh extraction*.

With the ability to perform boundary recovery, the next point of attention is *mesh optimisation*. As we have already mentioned, creating a quality mesh with tetrahedrons is notably harder than with triangles. Once we have obtained some mesh that is valid, it is only natural to work from there and try to further improve it. Mesh optimisation is therefore a widely explored topic. There are basically two ways to approach the optimisation: maintain the (constrained) Delaunay property, or drop it. And then in either case we can differentiate between methods that only change the connectivity, that change the node positions, and that add new nodes to the mesh. The last option of adding nodes, is normally only used in conjunction with the Delaunay criterion to update the connectivity. In general, adding large numbers of nodes is not desirable since it has a big impact on the running time of the analysis and the memory requirements. Adding a few nodes should not be a problem, as long as explicitly given characteristics of the mesh, such as mesh sizing, are not affected.

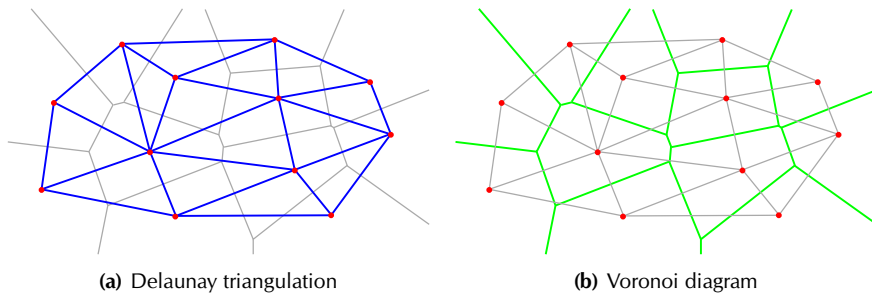
One of the earlier research efforts into mesh optimisation, has been carried out by Joe [1995]. The idea here is to use *local transformations* of the mesh, which only change the connectivity in the region surrounding a bad quality tetrahedron. One of the bad quality elements that is specifically targeted, is the sliver tetrahedron. Various techniques have been explored to remove this type of element, of which the approach of *sliver exudation* stands out [Cheng et al., 2000; Edelsbrunner and Guoy, 2002]. The basis of this approach is to use a so-called *weighted Delaunay triangulation*, which has assigned to all the nodes weights that affect the connectivity of the triangulation. With all weights equal, the triangulation is the normal Delaunay triangulation, but by changing the weights, the connectivity can locally change, effectively giving nodes a wider range of influence, enabling them to violate the empty circumsphere condition of some elements, without actually having their coordinates inside. Sliver exudation aims to make a mesh free of slivers by

changing the weights of the nodes.

Another method of mesh optimisation is *mesh smoothing*, which is essentially changing the node positions such that they are spaced more regularly, which results in higher quality elements. Mesh smoothing by itself does not avoid slivers, but it does lead to meshes that are more regular. The earliest and one of the most used smoothing schemes is *Laplacian smoothing* [Field, 1988]. In this approach, all node positions are in turn updated by the arithmetic mean of the incident nodes, i.e. the nodes that share an edge with the node that is being updated. The range of variations and improvements of this approach is extensive, but one direction deserves to be mentioned in particular, namely Voronoi-based optimisation.

The Voronoi diagram is the dual of the Delaunay triangulation, which means that fundamentally the structures are identical, but they provide a different perspective [Voronoi, 1907]. The Voronoi diagram divides space surrounding a node set into regions, such that each region consists of all the points that are closest to a particular node. The centres of the Voronoi regions correspond to the nodes of the Delaunay triangulation, and the circumcentres of the Delaunay triangles correspond to the points where three Voronoi regions meet. Every pair of adjacent Voronoi regions corresponds to an edge in the Delaunay triangulation. See Figure 3.8 for an illustration of a Delaunay triangulation and its corresponding Voronoi diagram side by side. The latter perspective leads to a particular smoothing scheme, namely one that creates a *centroidal Voronoi tessellation* (CVT) [Du et al., 1999]. This is a Voronoi diagram in which the nodes lie (as close as possible) on the geometric centres of their corresponding Voronoi regions. It turns out that this approach of adapting the node locations results in very smooth meshes [Du and Wang, 2003]. A further improvement on this is mesh smoothing based on the *optimal Delaunay triangulation* (ODT) [Chen, 2004]. The ODT is closely related to the CVT, as the optimised quantities are very similar. The principal difference is that the ODT optimises a quality of the Delaunay triangulation, whereas the CVT optimises a quality of the Voronoi diagram. For 2D it has been proven that these schemes are asymptotically each other's dual, but for 3D and higher dimensions this is still a conjecture. More theory on the ODT can be found in [Chen and Xu, 2004].

Although the common approach is to perform mesh smoothing and general optimisation after an initial meshing stage, there are methods that explicitly incorporate the optimisation as part of the meshing method itself. An example of this is *variational tetrahedral meshing* (VTM) [Alliez et al., 2005]. This method produces essentially a conforming Delaunay triangulation that is iteratively optimised to approximate an ODT. It takes a comparatively long runtime, but results in high quality elements. However, the method



**Figure 3.8:** A Delaunay triangulation and its corresponding Voronoi diagram

is not really suitable for mechanical models, which is why we developed extensions, to be discussed in Chapter 4, such that we can use the method for meshing such models.

Another recent approach that combines many techniques for optimisation and achieves very good quality meshes, but at the cost of a long runtime, is aggressive tetrahedral mesh improvement [Klingner and Shewchuk, 2007]. Yet other examples that illustrate that the highest mesh quality comes at a computational price, are the work of Dittmer et al. [2006], Acikgoz and Bottasso [2007], and Branets and Carey [2005].

### 3.3 The prospect of remeshing

The purpose of a mesh is to perform analysis with it, commonly finite element analysis. This is performed from early on in the design cycle to keep a check on global compliance. Later on the analysis becomes more detailed, as the design gains detail. The results help to steer the design process. Not only must the product effectively fulfil its purpose, but also the cost of material and production must be kept low. Analysis, and therefore the meshes that are used, are thus an integral part of the product development cycle.

An engineer that is offered a choice of meshes for his analysis, with everything else being equal, would certainly choose the highest quality mesh. What constitutes quality will depend on the particular context, but it will be the mesh that gives the highest expected accuracy within the hardware and time constraints. In practice, however, *ceteris paribus* rarely holds, as each meshing method bears its particular costs. Meshing algorithms that strive to optimise some quality measure on the mesh are often of variational nature, minimising an energy functional related to the quality measure, just like the smoothing for an ODT does. This makes high quality meshes costly

to produce in terms of runtime.

Two principal measures in analysis are *computation time* and *accuracy*. A certain minimal accuracy is required for most applications. The accuracy of an analysis is generally not known, but can be estimated, with *a posteriori* error estimates being more precise than *a priori* error estimates. The engineer chooses the parameters of the analysis such that he can be reasonable sure that the required accuracy is attained. With the analysis parameters set for a certain accuracy, computational time is effectively fixed. The primary ways to reduce it are better algorithms and more computing power.

In general, for a fixed number of nodes, the use of high quality meshes decreases the time spent on analysis. Therefore, a more expensive meshing method can still decrease the total time of analysis if it provides a higher quality mesh. The application of FEA can thus benefit from using high quality meshes, even though they come at a higher computational cost. Any approach that could bring down this time spent on meshing, however, would be a welcome improvement.

In this light, we look at the possibility of cutting the time spent on high quality meshing, within the context of the product development cycle. Since during the design process, analysis is performed routinely on models with more or less similar geometry, our idea is to base the construction of a new mesh on a previous mesh, which has been used in the previous design iteration. Iterative improvements to a model, in particular during the latter design stages, often have a local scope, i.e. change the geometry in a relatively limited way. For a computationally expensive meshing procedure, we expect to save time by adapting the previous mesh, instead of meshing the modified model from scratch. Figure 3.9 gives an example of the evolution of a design model. For this example, it seems intuitively clear that parts of the mesh from one model could be used for the other mesh as well.

Figure 3.10 shows how this approach fits into the product design cycle in comparison with the common design cycle. It shows an abstract depiction of the design cycle that focusses on the relation of the three steps of modelling, meshing, and analysis. The diagrams are very similar, except that with the incorporation of remeshing, a new relation arises that links the meshing step of an earlier to that of a later stage. No longer is the mesh created from scratch each time, but instead it is based on the mesh from the previous iteration.

In addition to the efficiency gains, there is another benefit of remeshing over meshing from scratch, namely that points and connectivity in certain areas can remain identical. This can aid in the comparison between analysis results. If nodes and elements can be mapped as locally corresponding entities between two meshes, then their values can be compared without

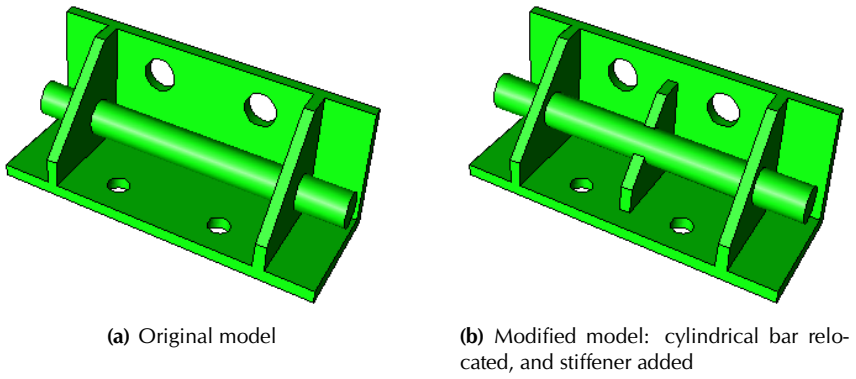


Figure 3.9: Model modification

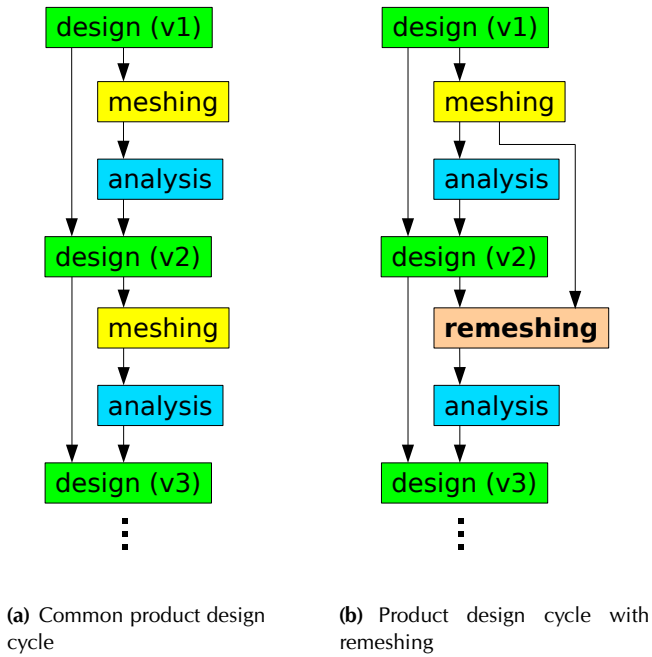


Figure 3.10: Incorporation of remeshing into the product design cycle with analysis

interpolation.

We thus propose to improve the process and speed of automated quality mesh generation, by reusing the mesh from the previous iteration in the development cycle. Little has been published on the topic of efficient remeshing after model modification. One way to avoid remeshing at each step of shape optimisation is to use mesh morphing/deformation, which in principle maintains mesh connectivity by only changing the node positions. Two recent expositions of this approach can be found in [Liu and Yang, 2007] and [Date and Onosato, 2008]. Both, however, work with surface meshes, though the latter intends to extend the method to volumetric meshes for the purpose of analysis. This approach is most effective when the changes in the model are subtle. In fact, the range of shapes that can be dealt with by this approach is limited. In particular changes in topology pose a problem.

Remeshing of 2D and 3D triangular meshes has also been discussed by François et al. [1999] and François and Cuillière [2000]. Herein two strategies are proposed of which the first is akin to mesh morphing. Changes in topology cannot be handled this way. The other strategy removes tetrahedrons around modified features, and locally reconstructs the mesh for those features. With the first strategy, the quality of the elements decreases with the impact of the modification. The second strategy can handle changes in topology but is only effective for changes with a local geometric scope. A somewhat broader view that includes hexahedral meshes is presented by Sheffer and Ungor [2001]. This work offers no single, generic approach, but discusses a range of techniques that can be applied in several different situations. The focus is specifically on parametric model modification. For tetrahedral meshes, the principal ideas are mesh morphing combined with selective quality improvement of bad elements that have appeared. Topological changes are not dealt with here either. Lastly, some techniques for interactive modification of finite element meshes are offered by Bidmon et al. [2004].

We propose an approach to remeshing that can handle changes in topology, and ensures a high quality mesh, in contrast to many of the methods based on mesh morphing. We base our remeshing approach on the VTM method, as it is flexible in nature and delivers high quality meshes. Since this method is not really suitable for meshing mechanical models, we first propose some extensions for this to the method. These are discussed in Chapter 4. In order to identify which parts of a previous mesh can be suitably reused in the creation of a new mesh, we need to describe the correspondences and differences between the geometry of two feature models. The model we have developed to describe this, is presented in Chapter 5. In Chapter 6, we present our novel approach to remeshing, based on the mod-

els and techniques from the two preceding chapters. So in the next chapter, we continue with a discussion of our extensions to the VTM method on which we base our approach to remeshing.



## Chapter 4

# Variational tetrahedral meshing of mechanical models

We are looking for ways to improve the product development cycle, in particular the interaction between design and analysis. A vital step in this process is meshing the analysis model. There are many meshing methods available; see Chapter 3 for an overview. We have developed a new method for meshing mechanical models, based on the existing, more general Variational Tetrahedral Meshing (VTM) method [Alliez et al., 2005]. This new method can produce high quality tetrahedral meshes for mechanical models, and is very suitable to implement the remeshing approach presented in subsequent chapters of this thesis.

As we have discussed in Chapter 3, the quality of the mesh is very important when meshing for FEA, as even a single bad element in a mesh can degrade the quality of the analysis. Quality is foremost determined by the size and the angles of the elements. More specifically, in tetrahedral meshes, such as in Figure 4.1(a), large dihedral angles (close to  $180^\circ$ ) cause problems with the interpolation, whereas small dihedral angles (close to  $0^\circ$ ) cause bad conditioning of the stiffness matrix. From this perspective, the elements in a quality tetrahedral mesh should all have a shape similar to a regular tetrahedron. Figure 4.1(b) shows a histogram of the volume-length ratio for the elements of the mesh in Figure 4.1(a). This ratio (defined in Section 3.1), is 1 for a perfectly regular tetrahedron, and very close to 0 for a sliver. The histogram shows that the majority of the elements has a shape similar to the regular tetrahedron, and that there are virtually no bad elements. This is the histogram of a high quality mesh.

The sizes of the elements are also of influence on the quality of the mesh. Smaller elements lower the discretisation error, but large differences in size

between the smallest and the largest element are again bad for matrix conditioning [Shewchuk, 2002b]. A quality mesh thus balances the variation in element size with the need for mesh sizing and the quality of the individual elements.

VTM achieves to construct a quality mesh, and does so with a fixed number of nodes. This is a useful characteristic for analysis, since based on the number of nodes the memory requirements and run time of the analysis can be estimated. The demonstrated results of VTM for smooth models are impressive, but little has been remarked on its applicability to meshing of mechanical models for FEA. Meshes for analysis, in particular, need to represent the boundary accurately, including all geometric traits such as (sharp) edges and corners. While acknowledging that their method approximates the boundary, Alliez et al. [2005] do mention the possibility of meshing models with sharp geometric traits and show images of two examples, but a real discussion is lacking.

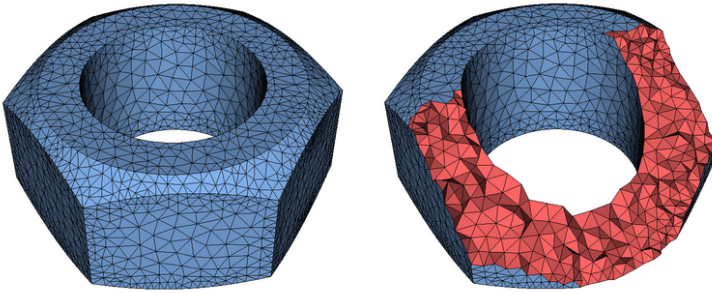
We have investigated the application of VTM to mechanical models for FEA. In particular, we have looked at the representation of the model boundary by the mesh. We have developed enhancements geared towards obtaining an accurate representation of the boundary, and demonstrated that with these adaptations this approach is feasible for generating high quality meshes of mechanical models that are suitable for analysis. We also offer some practical considerations for applying the method.

We start with a review of the original variational tetrahedral meshing algorithm in Section 4.1, followed by a discussion of the aspects where the method is lacking with respect to the meshing of mechanical models in Section 4.2. In Section 4.3 and Section 4.4 we propose enhancements to the algorithm, followed by some practical considerations for applying the method in Section 4.5. Finally, we show several meshes generated by this approach in Section 4.6, and conclude the chapter in Section 4.7.

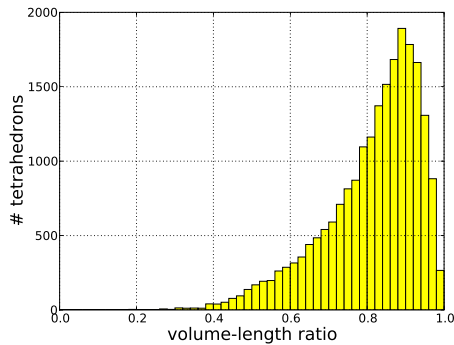
The larger part of this chapter has already been published in [Sypkens Smit and Bronsvort, 2008].

## 4.1 Variational tetrahedral meshing

The quality of systematically constructed meshes in 3D is generally far from optimal. Since long, people have tried to improve the mesh quality after the initial construction (see Section 3.2). Laplacian smoothing has been a popular method since the early days, moving nodes to relax the mesh for the given connectivity. However, this simple procedure can improve tetrahedral meshes only to a certain degree, leaving many elements of bad quality in existence. A significant step forward for the optimisation of tetrahedral



(a) Tetrahedral mesh of a nut model

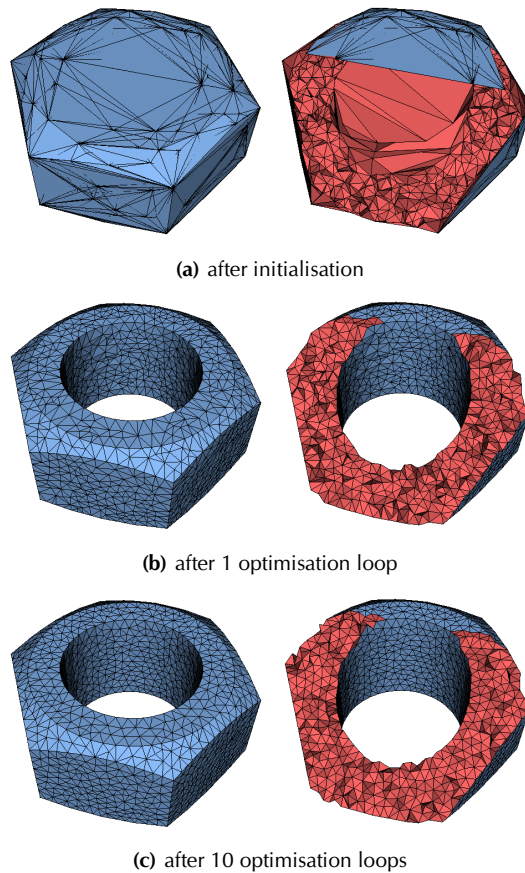


(b) Distribution of volume-length ratio of elements in the mesh in (a)

**Figure 4.1:** Mesh and quality of a nut model

meshes has been the focus on centroidal Voronoi triangulations (CVTs) [Du and Wang, 2003], where nodes are moved in order to approach the centroid of their Voronoi cell. Later Chen and Xu in [Chen, 2004] and [Chen and Xu, 2004] introduced the optimal Delaunay triangulation (ODT), further improving the quality of the primary mesh elements.

From these ideas, variational tetrahedral meshing (VTM) was conceived, a method that does not optimise mesh quality as an after-thought, but integrates the optimisation in the mesh construction procedure. The pivotal idea of VTM is to optimise the quality of the elements and to work on boundary conformance in a single iterative process. This is a clear deviation from the common practice of first meshing the boundary, and subsequently the interior, such that it conforms to the boundary mesh. With each iteration of VTM, the quality of both the interior and the boundary elements is im-



**Figure 4.2:** Evolution of the mesh during VTM of a nut model

proved. Figure 4.2 illustrates the evolution of a mesh during VTM. The connectivity of the mesh is governed by the Delaunay criterion and thus, unless the model is convex, the final mesh has to be extracted from the Delaunay mesh of the nodes. Also standing out, is the use of an indicative number of nodes to construct the mesh. When performing an analysis, the number of elements or nodes that can be handled is often roughly known. In those cases, a method that can generate a quality mesh with the specified maximum number of nodes, can offer an advantage.

We summarise the working of the algorithm. For a more detailed treatment, we refer to the original presentation of the algorithm [Alliez et al., 2005]. In the description of the algorithm appear a couple of terms that

are important to distinguish. The *nodes* are points connected to form the elements that compose the resulting mesh. All nodes that lie on the boundary of the mesh are called *boundary nodes*, and all others belong to the set of *internal nodes*. For a balanced positioning of the boundary nodes, we use *boundary samples*, also known as *quadrature samples*, which result from a dense sampling of the boundary. For each boundary node, there have to be about 10 boundary samples, or possibly more. To decide whether a point lies inside the model, we use a *control mesh*, which is a mesh that accurately represents the boundary. Lastly, there can be a mesh sizing function  $\mu$  applied, which indicates the (relative) desired density of the nodes both on the boundary and in the interior. Next, we continue by describing the four steps that the algorithm roughly consists of:

1. Initialise data structures, and construct the mesh sizing function.
2. Distribute the nodes.
3. Iteratively optimise the nodes.
4. Extract the mesh.

### Initialise data structures, sizing function

At the start of the algorithm, supporting data structures are built and initialised. An efficient point location test is needed, both for the distribution of nodes and for the extraction of the final mesh from the resulting Delaunay mesh that covers the convex hull of the nodes. For the latter, we need to decide for tetrahedrons whether they fall inside or outside the model boundary. The control mesh is used for this, which is a constrained or conforming Delaunay mesh of the input model, and it is constructed upfront. Also during initialisation, the boundary samples are created and categorised into sets corresponding to the edges or the faces they belong to. They can be created as the nodes of a fine-sampled surface mesh.

Also, for the generation of a mesh with varying element size, alternatively known as a *graded* mesh, a mesh sizing function  $\mu$  needs to be constructed. For this we use an approximation of the *local feature size* (lfs) constructed with a procedure based on the work of Amenta and Bern [1999]. The lfs captures the shortest distance of a point to the nearest geometric trait of the model that the point itself is not a part of. Effectively, this means that the desired element size depends on the local level of geometric detail. Other measures, mostly similar in spirit, could also be used, such as [Quadros et al., 2004] or [Persson, 2006]. VTM uses the mesh sizing function as a relative measure, since it works with a fixed number of nodes. For the generation of a uniform mesh,  $\mu$  would have to be set to 1 everywhere.

### Distribute nodes

The requested number of nodes is spread out in accordance with the sizing function. This is done by iterating the cells of a grid that covers the bounding box of the model. In a first iteration, for each cell that has its centre inside the control mesh, a number determined by the sizing requirements is added to a running total for all cells. Based on the resulting total, in a second iteration, a fair proportion of the nodes are placed randomly inside the grid cells that have their centre inside the model. The cells are traversed in serpentine order, spilling over any non-integer number of nodes that are called for into the next cell. For example, if 0.2 nodes are called for in a particular cell, then no node is placed there, but the 0.2 is added to the number of nodes that, in accordance with the sizing function, are called for in the next cell. After this process, we end up with a cloud of nodes that covers approximately the volume of the original model, with the node density varying relative to the sizing function. Figure 4.2(a) shows an example of what the mesh is like at this stage.

### Iteratively optimise nodes

During the optimisation process, the nodes fall in two categories: boundary nodes and interior nodes. Every node starts as an interior node, but can become a boundary node when it is selected as such during the determination and repositioning of the boundary nodes. Each iteration of the optimisation loop starts with the identification and positioning of the boundary nodes. After that, the rest of the nodes, deemed part of the interior set, is optimised.

So at the start of each iteration it is determined which nodes are part of the boundary. These boundary nodes are then (re)positioned in accordance with the sizing function, aiming for balanced node spacing on the boundary. This is achieved by employing the boundary samples, generated at initialisation time, by associating a quadrature value to each of them. These values depend on the local value of the mesh sizing function. The samples will have a connectivity defined between them, as they are generated as the nodes of a fine-sampled surface mesh.

For each boundary sample, at position  $\mathbf{x}$ , we locate its nearest node and have the sample exert a virtual pull on that node proportional to the area that the sample covers ( $ds$ ) and consistent with the sizing function:  $ds/\mu^4(\mathbf{x})$ . This value represents the weight of the pull, or quadrature value, of the sample. The nodes that have at least one sample pulling on them, are now considered part of the boundary set. They are moved to the average location of the pulling samples, weighted by the quadrature values. The rest of the nodes belongs to the interior.

To ensure that nodes end up at the edges where a clear angular separation between faces exists and at the corners, we treat their samples differently from the surface samples. The quadrature value for samples on edges is computed as  $dl/\mu^3(\mathbf{x})$ , with  $dl$  the length that the sample covers, and to corner samples an infinite quadrature value is assigned, to ensure the assignment of nodes there. The procedure starts with the regular surface samples pulling in and repositioning nodes, then the edges, and finally the corners are taken care of.

After dealing with the boundary, the Delaunay mesh is reconstructed as nodes have moved. Then each interior node, i.e. each node  $x_i$  that was not selected as a boundary node, is moved according to the formula:

$$\mathbf{x}_i^{\text{new}} = \frac{1}{s_i} \sum_{T_j \in \Omega_i} \frac{|T_j|}{\mu^3(\mathbf{g}_j)} \mathbf{c}_j, \quad \text{with} \quad s_i = \sum_{T_k \in \Omega_i} \frac{|T_k|}{\mu^3(\mathbf{g}_k)}. \quad (4.1)$$

Here  $\Omega_i$  denotes the one-ring of tetrahedrons that share node  $x_i$ ,  $|T_j|$  denotes the volume of tetrahedron  $T_j$ ,  $\mathbf{c}_j$  is the circumcentre of tetrahedron  $T_j$ , and  $\mathbf{g}_j$  is the centroid of tetrahedron  $T_j$ . The effect of this relocation is similar in idea to the relocation of a node towards the centre of its Voronoi cell. Instead of optimising the compactness of the Voronoi cell, this operation aims at improving the compactness of the tetrahedrons in the one-ring around the node. For a more detailed motivation we refer to [Alliez et al., 2005]. If the new location would invalidate the Delaunay property of the mesh, then the connectivity is changed to keep it a Delaunay mesh.

The optimisation loop alternates between these two phases of 1) determining and repositioning the boundary nodes, and 2) optimising the location of the interior nodes. Either a fixed number of iterations is performed, or a condition on the evolution of the quality improvement is used. Figures 4.2(b) and (c) show how the mesh of the nut model looks after 1 and 10 iterations, respectively.

### Extract mesh

After the optimisation ends, the mesh representing the model has to be extracted from the resulting Delaunay triangulation, which covers the convex hull of the nodes. As the model is usually not convex, it needs to be decided which tetrahedrons contribute to the model, i.e. are inside, and which tetrahedrons fall outside the model. This process is called *peeling*, as it can be envisioned as the one by one removal from the mesh of those tetrahedrons that fall outside the real model boundary. It is worthwhile to ask why this is possible in the first place. Normally, the Delaunay triangulation of a node set of a model does not contain the complete boundary; some faces and

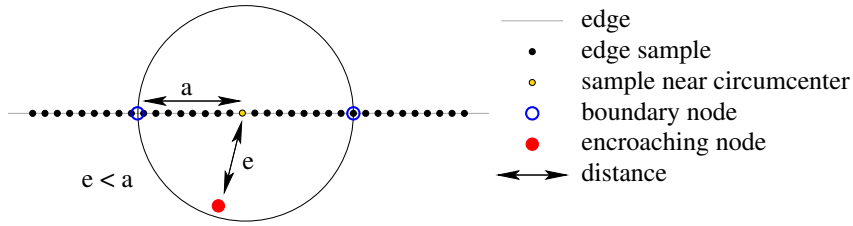


Figure 4.3: Encroaching of part of edge

edges have to be recovered. Why can we expect the model boundary to be present in the triangulation after the optimisation procedure in VTM?

There are no theoretical guarantees that the boundary will be present, but for denser sets of boundary samples, the chance of success increases; see Figure 4.3. If a node encroaches upon the minimal circumsphere of two nodes that currently represent part of a boundary edge, the node is likely to be drawn to the boundary, since a sample near the centre of the edge between the two nodes will most likely have the encroaching node as its nearest node. Similarly for the surface area of the boundary, a node that encroaches upon the minimal circumsphere of a triangle that needs to represent part of the boundary, is likely to be drawn to the boundary. This effectively results in a mesh for which all boundary elements have empty minimal circumspheres. With a higher number of boundary samples, the likelihood that this ad hoc protection procedure works, increases. An edge or face with its minimal circumsphere empty of other nodes is called *Gabriel* and is guaranteed to be present in the Delaunay mesh [Matula and Sokal, 1980]. We expect that all edges from the model boundary can be (almost) completely covered by Gabriel segments, and thus that the edges are found in the Delaunay mesh. Segments that are not Gabriel, are not guaranteed to be in the mesh, but it is still highly likely that they are. The reasoning holds similarly for the presence of triangles that represent the faces of the model boundary. The procedure might, however, fail near small angles. We will come back to this in Sections 4.4 and 4.5.

With the expectation that an accurate representation of the boundary is present, the final mesh can easily be extracted from the Delaunay mesh. The procedure for mesh extraction that is followed in [Alliez et al., 2005] is quite simple: a tetrahedron  $T_i$  is definitely part of the model if its circumcentre falls inside the control mesh. If the circumcentre falls outside the control mesh, tetrahedron  $T_i$  is still considered to represent part of the model if

$$\frac{d(\mathbf{c}_i, \partial\Omega)}{r_i} < 0.4. \quad (4.2)$$



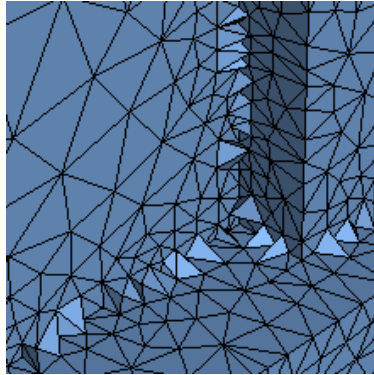
Here  $r_i$  denotes the circumradius of tetrahedron  $T_i$ , and  $d(\mathbf{c}_i, \partial\Omega)$  the distance of the centre of the circumsphere to the boundary. This causes tetrahedrons that have their circumcentre only just outside the control mesh, relative to the size of the circumradius, to be considered as part of the model as well.

## 4.2 Enhancing VTM for meshing mechanical models

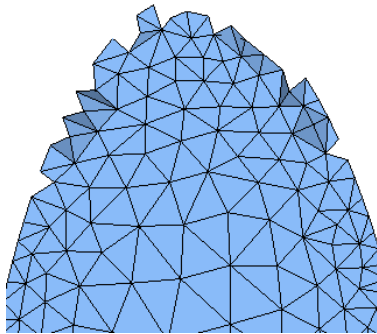
The VTM algorithm has deficiencies when meshing for FEA. In particular, the representation of the boundary and the extraction of the right mesh are issues. The algorithm only aims at approximating the boundary, but for mechanical models it is essential that the boundary is accurately represented by the mesh. This means that all essential features, such as edges and corners, should be present with the right position, shape and size. VTM often fails in this respect.

Even assuming that the model boundary is accurately represented in the mesh, the procedure for mesh extraction proposed in [Alliez et al., 2005] does not consistently determine the correct boundary. Elements that should be considered outside are classified to be inside and vice versa. This is in particular clear near edges, where a failure to extract the right mesh causes (part of) an edge to be missing or hidden behind excess elements. See Figure 4.4(a), which shows excess elements near concave edges. These elements can actually be removed (they do not lie partly inside the model), but they are considered part of the model nonetheless. Figure 4.4(b) shows an example where the same procedure has removed elements that are necessary to accurately represent the model boundary. An acceptable boundary was present in that case. Figure 4.4(c) shows the model corresponding to this mesh from the side. The dark face on the right is the face in view in Figure 4.4(b).

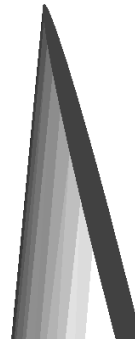
Also, there can be sliver-like elements left flat on the boundary. These are not true slivers in the sense that not all of them have six similar edge lengths, but they are extremely flat, so on this basis we call them slivers as well. This happens in particular in concave areas, as can be seen in Figure 4.5(a) and its zoom-in in Figure 4.5(b). The blue sphere is the circumsphere of the selected tetrahedron. All four corner nodes are in view. The only edge that is not visible runs at the back between the bottom and top node. In addition, when working with a robust Delaunay implementation, slivers can easily arise in flat faces if not all vertices lie exactly in the same plane. This is rarely a problem when following the procedure we described earlier to extract the mesh; however, when using alternative criteria in the procedure for mesh extraction, such as the ones we propose further on, results such as



(a) Excess elements along edges in concave areas

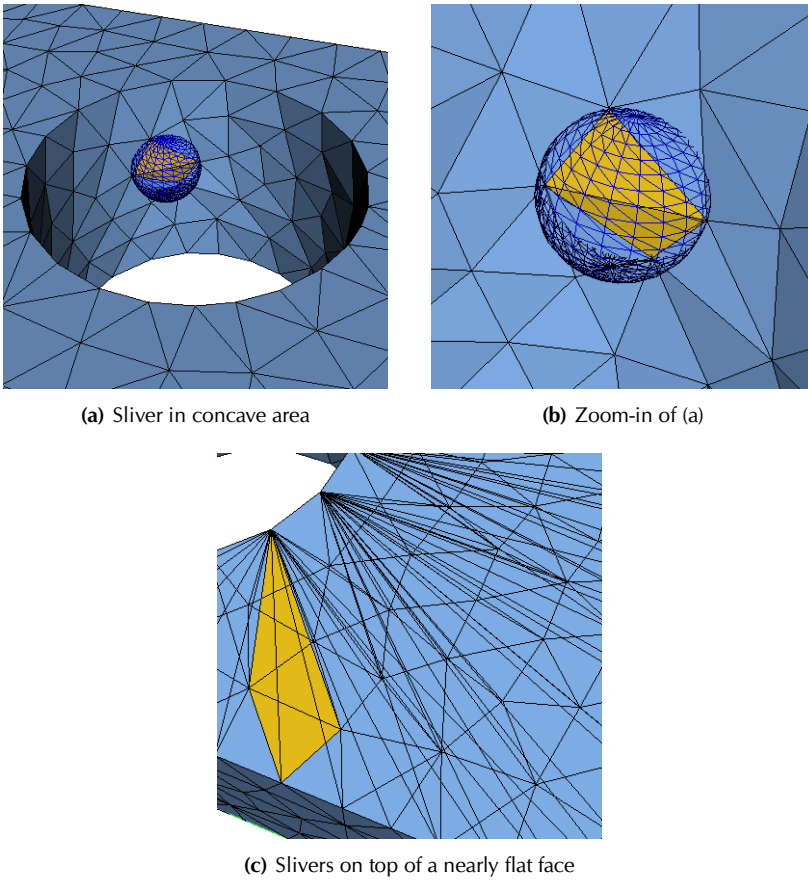


(b) Missing elements near small dihedral angle



(c) Object in (b) from the side

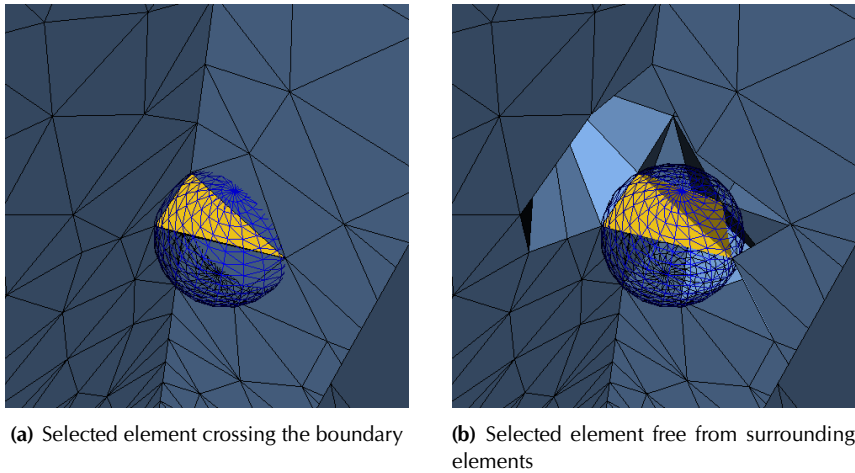
**Figure 4.4:** Failing mesh extraction



**Figure 4.5:** Slivers on the boundary

in Figure 4.5(c) can arise. Here a handful of very flat tetrahedrons lie on top of each other and on top of quality tetrahedrons. We should be careful to avoid including these tetrahedrons in the extracted mesh.

Another problem is that an acceptable representation of the boundary of the model is not always present in the Delaunay mesh that results from the optimisation step. One can end up with elements that cross the boundary, such as illustrated in Figure 4.6. Figure 4.6(a) shows the boundary of the extracted mesh; in Figure 4.6(b) some adjacent tetrahedrons have been removed from the view. The selected tetrahedron has one vertex in the interior of the mesh and three on the boundary, but not all on the same boundary face. One of its faces is outside the boundary, one is inside, and the



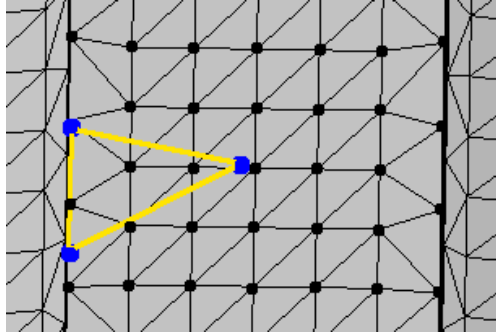
**Figure 4.6:** Elements crossing the boundary

remaining two cross the boundary. This was not a problem for the original VTM algorithm, since it was primarily intended for meshing models for which an approximate boundary is acceptable. However, a better boundary representation is required for the analysis of mechanical models.

The problems mentioned can, to a large extent, be eliminated. We will first discuss, in Section 4.3, the presence of an accurate representation of the boundary in the mesh and describe some enhancements to the algorithm that aim to enforce such a representation. Thereafter, in Section 4.4, we will discuss an alternative approach to mesh extraction that performs better at extracting the intended mesh.

### 4.3 Constructing the boundary

As indicated in the previous section, we should take precautions to prevent ending up after the optimisation step with a Delaunay triangulation from which the boundary cannot be recovered. This problem can mostly be avoided by making a good choice upfront for the number of boundary samples and the number of nodes. During the optimisation step, we pay close attention to signs that indicate a potential problem. In such a case, we can apply *node splitting*. Finally, after the optimisation loop, we make sure that the mesh is suitably prepared for the extraction procedure. We discuss each of these four aspects individually.



**Figure 4.7:** Quadrature samples on the boundary of the region shown in Figure 4.6

### Number of boundary samples

We discussed in Section 4.1 the procedure that aims to protect the boundary by pulling a node towards the boundary if it encroaches upon the minimal circumsphere of a boundary element. With sufficient boundary samples, practically every boundary edge and face will consist of elements with an empty minimal circumsphere. The problem in Figure 4.6 is indeed caused by a lack of boundary samples. Figure 4.7 shows a mesh of the boundary samples for the model; the outline of the boundary face that should have been present to avoid the problem is included in yellow. For the selected tetrahedron in Figure 4.6(a), the corner node in the interior of the mesh is inside the minimal circumsphere of the missing face. If there would have been a boundary sample close enough to the centroid of the missing face, then the interior node would have been pulled to the boundary, effectively emptying the minimal circumsphere of the missing triangle, and increasing the density of the nodes on the boundary relative to those in the interior. An accurate representation of the boundary through a triangulation of the boundary nodes, has smaller minimal circumspheres, which are less likely to be encroached by interior nodes.

In our experience, applying this protection procedure with roughly 10 boundary samples locally available around each node on the boundary, makes the occurrence of an element crossing the boundary extremely rare. We remind the reader that an empty minimal circumsphere of a boundary face is a stronger restriction than necessary for it to appear in the Delaunay triangulation. Thus even if an encroaching node fails to be pulled to the boundary to guarantee protection, the boundary can still be present in the Delaunay triangulation. This is even likely if the number of samples is high.

However, some models are inherently difficult to mesh with a conforming Delaunay triangulation, mostly due to small dihedral angles. The risk of boundary encroachment near such an angle cannot be completely eliminated by our heuristics. Failure to correctly represent the boundary should be detected during or after the mesh extraction.

### **Number of nodes**

The boundary protection procedure works by locally making the density of the nodes on the boundary slightly higher than in the interior. This procedure can only work if there are enough nodes in the area. The other factor of importance for an accurate representation of the boundary is thus the number of nodes relative to the complexity of the geometry. A certain number of nodes is necessary to be able to capture the boundary. Even constrained Delaunay triangulations can need extra nodes (Steiner nodes) just to be able to construct a triangulation in the first place, without any consideration of quality (see Section 3.2). The number of nodes obviously depends on the complexity of the boundary, with a more complex boundary calling for more nodes. The number of nodes used in a constrained triangulation, such as produced by TetGen [Si, 2006], is of course the bare minimum. As we build a conforming, instead of a constrained, triangulation and we strive for quality elements, we need many more nodes. The required number depends on the local feature size. The average edge length in the triangulation cannot be longer than the lfs in a successful capture of the boundary. Since we need some flexibility, the number of nodes should be such that the average edge length is smaller than the lfs by at least a factor two. As we want all corners to be represented by the mesh, we add all corner samples explicitly to the mesh as nodes, and we effectively ignore them during the boundary procedure.

### **Node splitting**

Since we are working with heuristics, the use of a reasonable number of nodes at the start is still no guarantee that locally always the right number of nodes is available for capturing the boundary accurately. We can, however, during the boundary phase of the optimisation loop detect symptoms of failure to represent the boundary. As we have detailed already, each boundary node is normally pulled upon by multiple boundary samples. This helps to balance their spatial distribution on the edges and the faces of the model. However, nodes should only be balanced within a single face or edge. Whenever a node is pulled on by boundary samples belonging to multiple edges or multiple faces, the node will end up hanging between

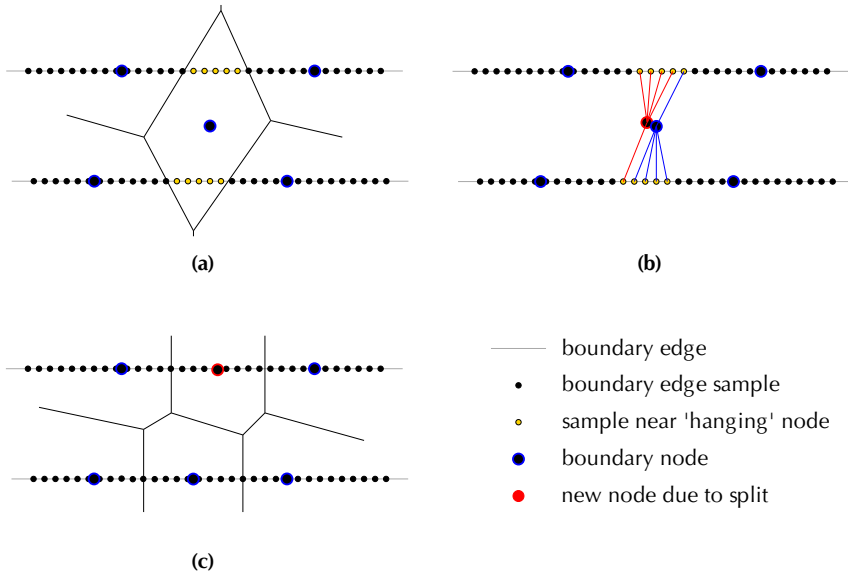
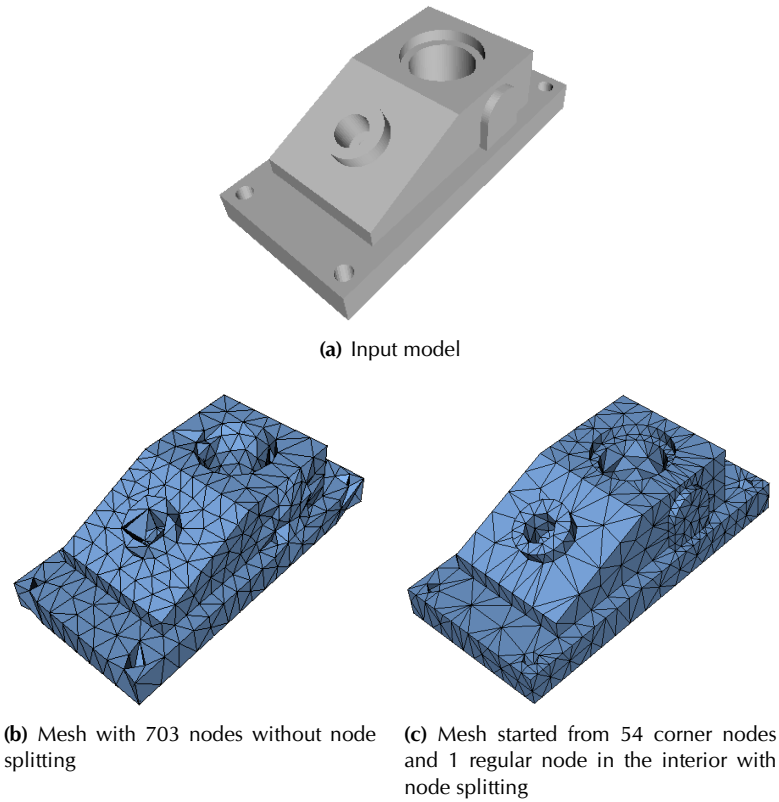


Figure 4.8: Split example

the two, and thus contribute nothing to the representation of the boundary. Categorising all the boundary samples into sets that belong to the same edge or face, we can count the number of sets that are pulling on a specific node. If more than one set is pulling on the node, this indicates that there are locally too few nodes to represent the boundary such that we can be reasonably sure that it can be recovered from a Delaunay triangulation.

If this is the case, we can add a second node right next to that node, effectively splitting it, leaving one node for one edge or face, and one for the other. This process is illustrated in Figure 4.8. We see in Figure 4.8(a) a node that is pulled on by samples of two different edges, in (b) the node split; the new node is placed randomly at a small distance from the original node, and in (c) the final situation wherein each node is only pulled on by samples from a single set of edge samples and thus balanced within that edge. In order to carry out this procedure, we must have categorised all samples according to their edge or face of origin. It takes a couple of iterations to reposition the nodes in the vicinity of the split. That is why we split nodes only at certain iterations of the optimisation loop, and always continue after this with a couple of iterations in which no splits are performed.

The effectiveness of the approach is illustrated by Figure 4.9. Figure 4.9(a) shows the model that was used as input, and Figure 4.9(b) shows the result-

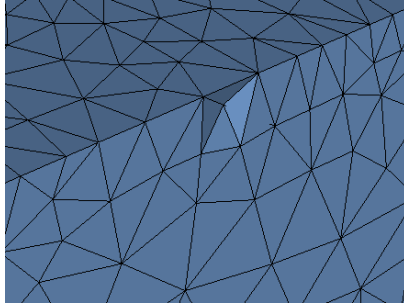


**Figure 4.9:** Node splitting for model based on the ANC101 model

ing mesh when no node splitting is employed. This mesh has 703 nodes. The mesh in Figure 4.9(c) also has 703 nodes, but it was started with just 55 nodes (54 corner nodes and 1 regular node in the interior). Through subsequent node splits, due to the pulling of edge samples from multiple sets, the number of nodes was increased to 703. After reaching this number of nodes, no more splits were called for. These nodes all lie on the boundary. The mesh consists of 1858 tetrahedrons. Comparing Figure 4.9(b) and Figure 4.9(c) it is clear that the node splitting considerably improves the quality of the representation of the boundary.

This example stresses the effectiveness of the splitting procedure, but we do not recommend the construction of meshes with the use of relatively many splits. A small number of splits is acceptable and can further strengthen the likelihood of a successful recovery of the boundary, but a





**Figure 4.10:** Boundary node not on boundary

large number of splits should be interpreted as an indication that we might have distributed too few nodes, or that the geometry poses difficulties in the creation of a conforming Delaunay triangulation. We therefore only split nodes hanging between edge samples from different sets, as splitting nodes on their proximity to different surfaces rarely leads to just a few splits. It turns out that a small fraction of the nodes can be closest to boundary samples from two different surfaces, without leading to any problems.

### Preparation for mesh extraction

Before starting with the actual mesh extraction, which will be described in the next section, we want to be sure that all boundary nodes actually lie on the boundary of the control mesh. The boundary nodes are those nodes that were identified as such during the last iteration of the optimisation loop, which means that they were the closest node to at least one of the boundary samples. Given that we split a node if samples from multiple edges are pulling on it, and that edge samples are handled after the surface samples, all nodes that are attracted by an edge actually end up on that edge. Since we do not split nodes pulled on by samples from multiple surfaces, it might occasionally happen that a node is caught between two surfaces. An example of such a node is illustrated in Figure 4.10. The node in the centre of the 'dent' is a boundary node. Two relatively flat tetrahedrons cover it, but they have been removed for the illustration. This node is being pulled on by a boundary sample from both the horizontal and the vertical face. The edge samples all had other nodes closer. If the covering tetrahedrons are not removed, this particular example does not yield a wrong mesh, but it does affect our reasoning about the mesh extraction, where we expect all boundary nodes to actually lie on the boundary of the control mesh.

Therefore we project all boundary nodes onto the boundary of the control

mesh, before starting the mesh extraction. For all nodes lying in flat faces, the projection distance will be zero or negligible. Boundary nodes caught between faces, as in Figure 4.10, will have a larger projection distance. In our experiments, we noticed no problems related to this kind of projection. In particular if the number of such nodes is large, we could alternatively decide to project only if the projection distance is small; if a node is not close enough to the boundary, we change its status from a boundary node to an interior node.

This projection of boundary nodes to the boundary of the control mesh, also affects another set of nodes, namely those in curved faces. Since each node is placed where it is being pulled to at average, it will not always be placed exactly on the boundary of the control mesh. In concave areas, the node will be located slightly outside the boundary of the control mesh, whereas in convex areas the node will be located slightly to the inside of the boundary of the control mesh. The deviation, and thus the projection distance, depends on the local number of nodes that is used, relative to the local curvature.

If necessary, after projection the mesh connectivity will be updated to maintain the Delaunay property.

#### 4.4 Enhanced mesh extraction

In the previous section we have discussed how to increase the likelihood that a correct boundary is present in the Delaunay mesh. Now we will discuss the enhanced mesh extraction procedure.

As we have seen in Section 4.2, the original mesh extraction procedure does not always yield correct results. Leaning on the knowledge that all boundary nodes actually lie on the boundary, our adapted version of the decision procedure is as follows:

1. All tetrahedrons are assumed to be inside at the start. On inspection we can decide to remove one.
2. All tetrahedrons that have at least one interior node as one of its four nodes are inside, and therefore will never be removed. We thus only consider removing those tetrahedrons that have four boundary nodes.
3. If the centroid of a tetrahedron that has four boundary nodes falls outside the control mesh, then the tetrahedron is directly considered outside.
4. The only tetrahedrons we are left with, are those that have four boundary nodes and the centroid inside the control mesh. Assuming that a

correct boundary is present, we must conclude that these tetrahedrons are inside; a tetrahedron that has its centroid inside the mesh is either completely inside or it intersects the boundary. Only if a tetrahedron has a volume-length ratio smaller than  $10^{-1}$ , we add it to a list to be considered for a *clean-up peeling*.

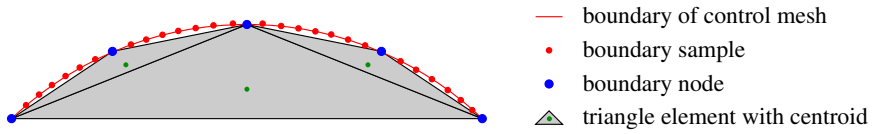
5. In the clean-up peeling, we want to remove very flat tetrahedrons that might lie on the boundary and can be removed without negatively affecting the quality of the boundary representation. We try to remove the tetrahedrons iteratively, by inspecting for removal only those tetrahedrons that are currently considered on the boundary. If such a tetrahedron has two or more faces on the current boundary, it is removed. If it has only one face on the current boundary, it is only removed if its volume-length ratio is smaller than  $10^{-4}$ . This process is continued until no more tetrahedrons can be removed.

The thresholds of  $10^{-1}$  and  $10^{-4}$  for the volume-length ratio are chosen based on our experience. They might not be optimal to clean-up the flat tetrahedrons of every mesh. Lower values of these parameters indicate a more conservative approach. This procedure is justified as follows.

We know that all boundary nodes actually lie on the boundary of the control mesh and the remaining nodes in the interior. All tetrahedrons that have at least one interior node must obviously be inside.

For the remaining tetrahedrons, with four boundary nodes, the decision is firstly based on the location of the centroid of the tetrahedron. We consider two cases: a convex area and a concave area. In a convex area, if the centroid of such a tetrahedron is inside, the tetrahedron must be inside. In a concave area, if the centroid is outside, the tetrahedron must be outside. See Figure 4.11 for an illustration in 2D near a convex part of the boundary. The grey triangle elements are considered inside, because their centroid is inside the control mesh (marked in red). The concave case is identical, except for the reversed orientation of the boundary. In that case, the grey triangle elements are considered outside, because their centroid is outside the control mesh. This reasoning holds as long as the number of nodes is not larger than the number of samples from the control mesh, a requirement that is of course met.

In strictly flat areas there should be no elements formed between four boundary nodes. However, because of finite precision arithmetic, such elements sometimes *do* appear, such as depicted in Figure 4.5(c). This is the kind of element that we aim to remove during the clean-up peeling, together with very flat tetrahedrons in convex areas. The elements inside a convex area that have only boundary nodes tend to be flat. If they are very



**Figure 4.11:** Triangles and their centroids near a convex part of the boundary

flat, it is better to remove them. In case the flat element does not lie on the boundary, we could attempt to remove it by inserting the circumcentre of the tetrahedron as an interior node of the mesh.

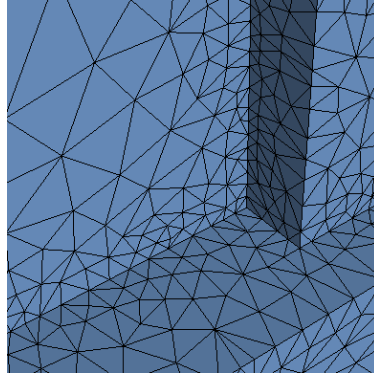
Figure 4.12 shows the meshes of the models that were not recovered well with the original extraction procedure, as they are extracted by the procedure just described. Figure 4.12(a) and (b) correspond to Figure 4.4(a) and (b), respectively. Figure 4.12(c) shows the mesh of the model of Figure 4.4(c) from the curved backside. Exactly the same triangulations were used for comparing the effectiveness of the recovery. The described procedure handles the extraction of these meshes correctly.

Despite the methods described in Section 4.3 to achieve this, there is still no guarantee that the correct boundary is present in a mesh. We can try to detect this after the extraction procedure. A simple way to do this is to traverse all the boundary faces of the mesh after the extraction, and compare the normal at the centroid of each face with the normal of the control mesh for the projection of the centroid to the boundary of the control mesh. If the deviation between the normals supersedes a certain threshold, depending on the problem at hand, a problem with the boundary is likely. A relatively large distance between the centroid of a face on the boundary and its projection to the boundary of the control mesh is also an indication of a potential problem. We have not investigated the robustness of using these two measures to detect problems with the boundary recovery, but given that the correct representation of boundaries is currently much less of an issue than in the past, we feel that this is a surmountable problem. Most of these cases can be solved by the insertion of a single extra node.

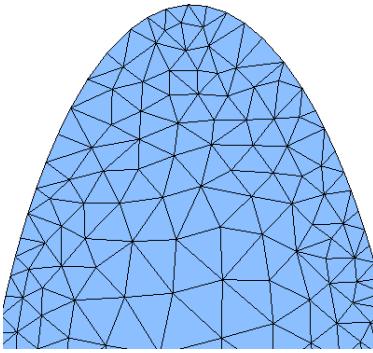
## 4.5 Practical considerations

Various aspects influence the effectiveness of the enhanced VTM method and the quality of the resulting elements. We discuss the most prominent considerations for applying the method.

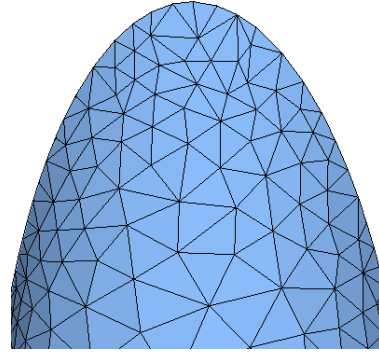
As explained in Section 4.2, having more boundary samples, relative to the number of nodes, increases the likelihood that the boundary is well



(a) Near edges in concave areas



(b) Near small dihedral angle; front



(c) Near small dihedral angle; curved backside

**Figure 4.12:** Successful boundary recovery

represented by the mesh. With 10 samples locally around each node, problems with the boundary are rare in our experience. In case of a uniform sizing function, this amounts to 10 times as many samples as the number of nodes that end up on the boundary. If mesh sizing is employed, either the density of the samples needs to follow the sizing function, or the ratio between the number of samples and number of boundary nodes needs to be higher. For models with a simple geometry, less boundary samples can suffice. Concave regions usually require a higher number of samples with respect to the number of nodes than convex regions.

Since the final mesh is a Delaunay triangulation that approximately conforms to the boundary of the control mesh, it is a requirement that

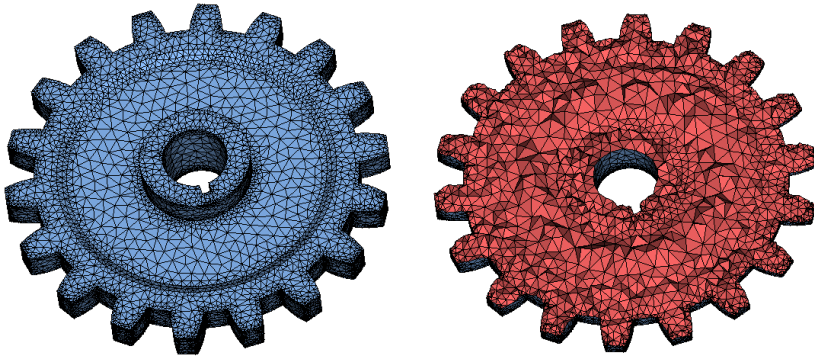
the model admits a conforming Delaunay triangulation with a reasonable number of nodes. Models that require many nodes for their conforming Delaunay triangulation, will be hard to mesh with VTM, or require at least as many nodes, possibly more than is acceptable for the analysis. Sharp dihedral angles and thin slits are the most common causes of a failure to represent the boundary or of a prohibitive number of nodes.

Most realistic and useful mechanical models do not exhibit geometric features that prevent the successful construction of a good quality conforming Delaunay mesh. All these models can be meshed with excellent quality elements by application of the presented approach. The optimisation of element quality is for a substantial part effectuated by the relocation of interior nodes. Therefore the method is most suitable for creating meshes that have a substantial number of nodes in the interior or, more generally, many nodes relative to the complexity of the geometry.

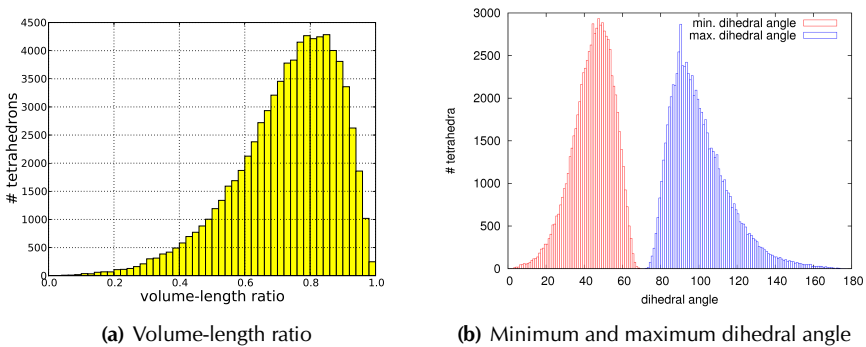
The method optimises the elements for quality. However, we have made no attempt to specifically optimise the quality of the worst elements. The elements with the worst quality make up only a small percentage of all tetrahedrons, and most of them have more than one node on the boundary. By means of simple flipping operations, which transform a set of tetrahedrons into a different set by rearranging the internal faces of the initial set, we can increase the volume-length ratio of the worst elements in the mesh substantially. In our experiments, we always succeeded to get it above 0.1, but higher values, depending on the complexity of the geometry, are not uncommon. If desired, the quality can be further improved by applying aggressive tetrahedral mesh improvement [Klingner and Shewchuk, 2007]. Since the number of poor quality tetrahedrons is low, we expect that large improvements to the lower bound of the quality can be achieved in a short time.

## 4.6 Examples

To test the validity of our enhancements to the VTM algorithm, we have meshed many models. We show some results here. Figure 4.13 shows the mesh of a gear model and Figure 4.14 its mesh quality. The mesh has 20179 nodes and 77027 tetrahedrons, and a sizing function was used for its construction. Figure 4.14(a) shows the distribution of the volume-length ratio for all tetrahedrons, and Figure 4.14(b) the distribution of the minimum and the maximum dihedral angle for all tetrahedrons. The regular tetrahedron has a dihedral angle of approximately  $70.53^\circ$  between all faces. Both small (close to  $0^\circ$ ) and large (close to  $180^\circ$ ) angles are detrimental to the quality of the analysis. From both graphs we gather that the majority of the elements



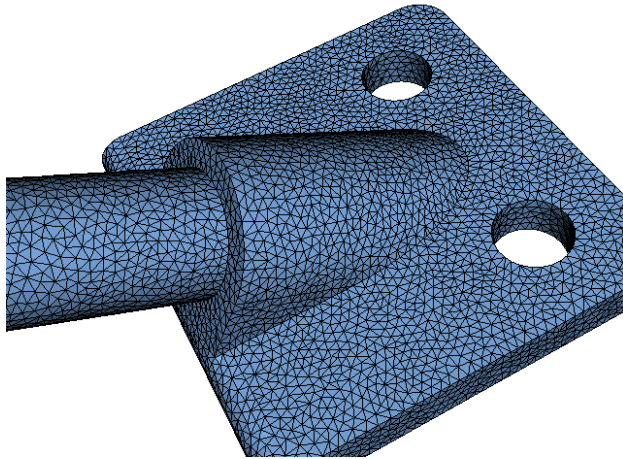
**Figure 4.13:** Gear example. Left: exterior mesh, right: slice exposing interior mesh.



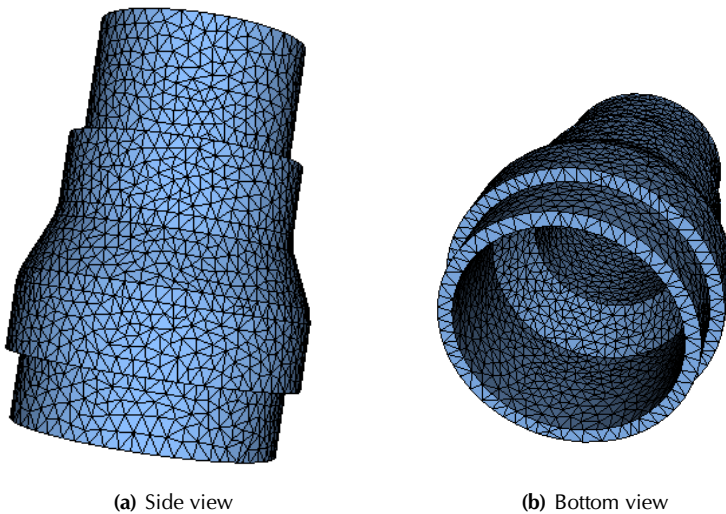
**Figure 4.14:** Quality of gear mesh

has a shape close to that of a regular tetrahedron. The model boundary is correctly represented.

Figures 4.15 and 4.16 show two more examples, both created with a uniform sizing function. Both models have a boundary with several curved areas. The mesh of the construction piece in Figure 4.15 has a relatively large volume with many interior nodes, whereas the mesh of the tube in Figure 4.16 has virtually no interior nodes. The quality distribution of both these meshes is similar to that of the gear, with the one of the tube being slightly worse as it has few interior nodes. The large majority of the elements, however, is excellent and the boundary is correctly represented.



**Figure 4.15:** Mesh of a construction piece



(a) Side view

(b) Bottom view

**Figure 4.16:** Mesh of a tube



## 4.7 Conclusions

Our findings are supported by an implementation of the described algorithm in C++. The program takes a finely triangulated model boundary as input, supports mesh sizing, and is controlled by several parameters such as the approximate number of nodes and the number of iterations in the optimisation loop.

With variational tetrahedral meshing, high quality meshes can be generated. This is achieved through an optimisation procedure that makes changes to both the boundary and the interior nodes, alternating between the two. The method as proposed originally has deficiencies that make it unsuitable for the generation of meshes of mechanical models for finite element analysis. This is mainly due to the boundary being represented incorrectly by the mesh. We have presented several enhancements to overcome the deficiencies. With these enhancements, variational tetrahedral meshing becomes an attractive method to create good quality tetrahedral meshes of mechanical models.

Some models, however, remain difficult to handle with this method. Since the resulting mesh is an approximate conforming Delaunay mesh of the control mesh, many elements can be needed to correctly represent areas where different parts of the boundary are close together. A constrained Delaunay mesh would use less elements in such cases. If a dihedral angle is too small, in particular in a concave region of the model, then we get either a cascade of node splits, or the boundary is likely to be incorrectly represented somewhere near the edge of the dihedral angle.

The enhancements to VTM come at some computational cost, but the running time of the new algorithm is still of the same order as that of the original algorithm; both are dominated by the cost of continuously updating the Delaunay triangulation, which scales very well. The exact running time depends on many factors. The most important ones are the numbers of nodes, boundary samples and iterations of optimisation. Using a sizing function can increase the setup-time substantially. During the first couple of iterations, the overall quality of the mesh increases quickly, and then the improvements become smaller. Whether extra iterations are worth the time is thus a highly subjective matter. Most of the models we tested, with 20 iterations and at least 5000 nodes, could be meshed in a matter of minutes, with none taking more than an hour. In our view, the higher computational cost will not be prohibitive for most models in practice, since better meshes will often lower the time spent on analysis. The method has element quality as its principal objective, and this comes at the cost of a higher running time. We are not aware of any meshing method that achieves

the quality distribution that VTM offers, certainly not with significantly less computational effort.

The importance of high quality meshes for analysis is evidenced by the attention that this topic has been receiving in the meshing community for many years now, but is also supported by theory [Shewchuk, 2002b]. We have not verified the claimed benefits through performance of actual analyses. Such a verification would be a substantial and complicated task, as there are many factors involved, such as the geometry of the domain, the particular physical problem, and the choice of parameters controlling mesh generation. Moreover, it would be necessary to compare meshes generated by different meshing methods, which inevitably is subjective as it highly depends on the choice of methods to compare against. Also, the control over the mesh generation process can differ strongly between methods, e.g. some methods have virtually no control over the number of elements. Lastly, the relative likelihood of a breakdown of the analysis due to the mesh is of clear importance, but is difficult to establish. Performing such a rigid comparison would certainly be useful, but falls outside the scope of our work.

We now continue in the next chapter with a discussion of the difference between two feature models, and present a method for giving a precise description of this difference. Based on this description, we aim to adapt and further extend the meshing procedure that we have just described, such that it can partially reuse the mesh from the previous design iteration, for the creation of a new mesh.

## Acknowledgements

For the implementation, CGAL [CGAL Editorial Board, 2006] and TetGen [Si, 2006] have been used. CGAL was mainly used for its robust geometric predicates and Delaunay triangulation. TetGen was used to create the control mesh. The meshes of the gear (Figure 4.13), the construction piece (Figure 4.15) and the tube (Figure 4.16) were based on example models included with HOOPS [Tech Soft 3D, 2006].

## Chapter 5

# The feature difference

We humans are fairly good at identifying patterns, certainly in comparison with computer algorithms. When given two related models, a human can quickly identify similar parts within the two models, in spite of differences in size, position, or even shape characteristics of these parts. See, for example, the two related models in Figure 5.1; we automatically assume a correspondence between each of the four cylindrical pockets and the holes in their interior, even though the dimensions and positions are different.

Although humans do have a talent for this kind of identification, they easily make errors, and they have difficulty with keeping track of everything for larger and more complex models. With our aim of remeshing analysis models in mind, we need to find similarities and differences between models. However, given the mentioned shortcomings of humans, and our goal of improving the efficiency of the product development cycle, requiring additional human input for this task of relating the geometry between two models, seems out of the question. We thus require an automated approach

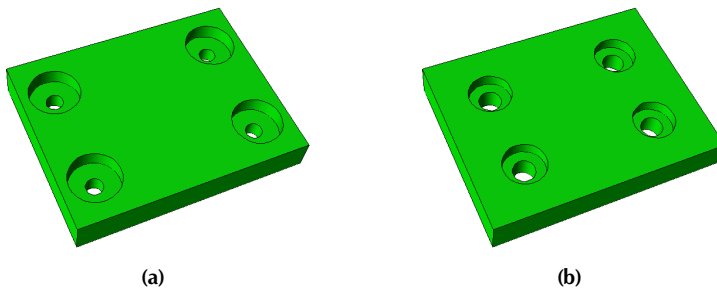


Figure 5.1: Two related models

for finding the similarities and differences between two analysis models. In this chapter, we describe such an automated method, which additionally is geometrically precise. We call the resulting output the *difference model*.

We begin the chapter, in Section 5.1, with some background and motivation on which the ideas of the difference model are founded. After that, in Section 5.2, follows a discussion of the properties of the feature model assumed in this work. Then, in Section 5.3, we explain what we mean by the geometric difference *in terms of features*. In Section 5.4 we discuss the representation of the difference and the outline of a method to construct such a representation. We end with conclusions in Section 5.5.

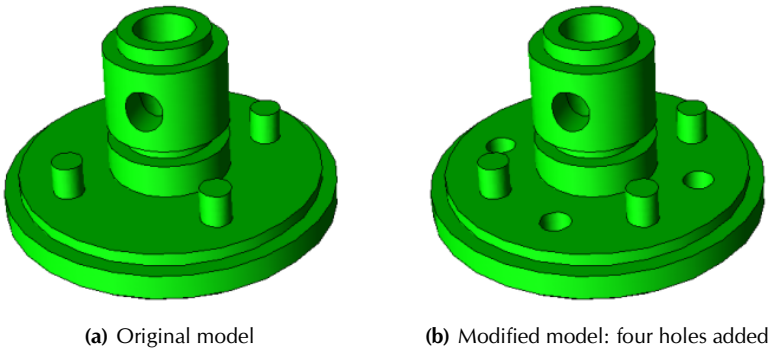
The larger part of this chapter has already been published in [Sypkens Smit and Bronsvort, 2007], and some other parts in [Sypkens Smit and Bronsvort, 2009a].

## 5.1 Background

Feature modelling, as we have discussed in Chapter 2, is nowadays the prevalent approach to product modelling. Any system that, through shapes that compose the geometry, adds more information than geometry to the model, can be considered a feature modeller. Some examples of data commonly attached to the model through features are design intent, properties of the material, and machining data. Our effort to describe or quantify the differences between two models, will assume that they are feature models, and that we have access to the underlying data structures.

A straightforward way to find the difference between two geometric shapes is with a boolean (symmetric) difference operation. This operation only takes the BReps of the models into account. Features, however, can describe more than just the geometry of the boundary. They can also overlap and interact in regions internal to the volume bounded by a BRep. Therefore the boolean difference is not powerful enough to handle feature models in a more general way than just in terms of the geometry of their BRep. Furthermore, when considering the geometric difference of two models with a single boolean difference, the result does not properly capture differences and similarities as we intuitively perceive it: we can regard an individual, translated feature as identical in two models, as illustrated by Figure 5.1, whereas the global boolean difference will result in both the disappearance and creation of new geometry.

Recently there has been a lot of interest to identify geometric similarity between models for 3D shape retrieval [Regli and Spagnuolo, 2006; Tangelder and Veltkamp, 2008]. Methods for this roughly aim to identify geometric characteristics and their relations, and compare the resulting *shape signatures*



**Figure 5.2:** Model modification

of the models to assess similarity. They often succeed in yielding a result that connects more closely to our intuition of similarity. However, typically the result of a similarity analysis does not result in a map that relates specific parts of the geometry directly between the models. The methods that are based on an explicit break-down of the geometry, such as the ones presented in [Biasotti et al., 2006] and [Bespalov et al., 2006], do help to relate specific parts or regions between models, but in general the relation does not allow for a one-to-one mapping between geometric elements. Moreover, their conception of a feature is purely geometric, with little correspondence to the broader feature concept that attaches meaning or knowledge to parts of the model.

To better understand why these solutions do not satisfy our requirements for our purpose of remeshing, we take a look at these requirements by means of a concrete example. For the remesh procedure, a description is needed of the difference between two models, such as those in Figure 5.2. The model that was made first is being referred to as the *original model*. When the original model has been adapted, the resulting model is being referred to as the *modified model*. Our goal is to determine for each part of the geometry of the modified model, both of BRep elements and the volume, whether it relates to some part of the geometry of the previous model or not. Those parts that can be related might carry the same mesh, whereas for the remaining geometry new mesh elements have to be constructed. Of course, if we reuse a mesh subset for a part of the geometry of the model, it must be connected to other subsets of the mesh, either also reused or newly constructed. We must take specific care that the mesh quality in those regions is on par with the overall quality.

The analysis mesh conforms to the geometry of the model, which in turn

depends on the features. Changes in the geometry result from manipulation of features. The mesh thus depends indirectly on the geometry of the features.

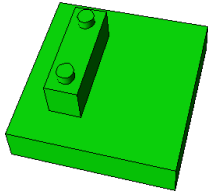
If, after a model modification, the geometry of a feature that adds material, including its interactions with other features, has not changed, then the mesh corresponding to that feature does not need to change either, and can thus be reused in the mesh for the modified model. See, for example, Figure 5.3, which shows the meshes of two variants of a model with a base block feature and a rib feature, including two cylindrical protrusions, on top: the mesh section of the rib feature on top could be identical in both meshes, as the geometry of this feature is identical in both models. The geometry of the base block, on the other hand, is subtly different between the two models, as its top surface connects in different locations to the rib feature on top (see Figures 5.3 and 5.4). Therefore the mesh corresponding to the base block feature cannot be identical for both models. However, as the difference is subtle, the meshes could for the larger part still be the same. For this we need a description of how the geometry of the base blocks differs between the two models.

If we know for each feature whether its geometry has remained the same, or, alternatively, how it is different from its original geometry, we have enough information to reuse subsets of the original mesh in the mesh for the modified model. All details of the remeshing procedure are discussed in Chapter 6. We proceed here with the description of how the geometry of a feature relates between two models. This description can be constructed for any feature of the two models, and is called its *feature difference*.

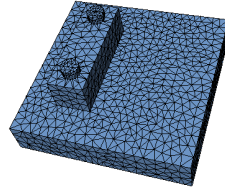
We describe the feature difference in a general fashion. Although the ideas were conceived based on our motivation of remeshing, we present it as a general concept that holds for all feature models in which the features have a clearly defined geometry. Also, the meaning of the features, or any other attached concept, can be taken into account for the resulting difference. This description is geometrically precise, and enables the mapping of similar feature geometry between two models. It might also be of use to other applications than remeshing that need to map data that is attached or related to features between models.

## 5.2 The feature model

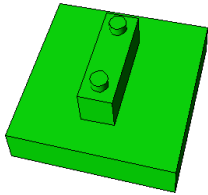
We have already given an introduction to feature modelling in Section 2.2. There we discussed how a feature model connects closely with how the user intuitively sees an object as a combination of shape aspects. We now discuss it from a more technical point of view, and specify properties of the feature



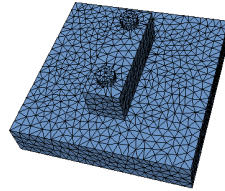
(a) Original model: base block with a rib feature on top



(b) Mesh of original model

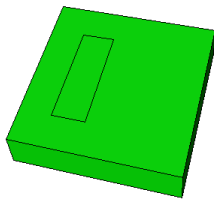


(c) Modified model: base block with rib feature translated

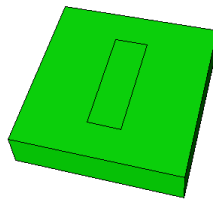


(d) Mesh of modified model

**Figure 5.3:** Original and modified model and their meshes



(a) Geometry of base block in original model



(b) Geometry of base block in modified model

**Figure 5.4:** Subtle difference in geometry of base block

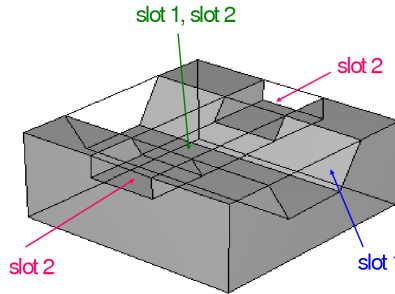
model assumed here.

In feature modelling in general, the geometry of a feature is parametrised. The parameter values can be given explicitly by the designer or be derived from constraints on the model. For a specific model instantiation, all the values of feature parameters have to be known, together with a set of relations that uniquely defines the relative positioning of the features. Additionally, we discern whether a feature is *additive* or *subtractive*, indicating respectively that it adds material to the model, or that material is absent. This, however, is not enough for a complete description of the model. There also needs to be some information, either implicit or explicit, on how to resolve overlapping features. In the case of features that define the model geometry: if two features of additive and subtractive nature overlap, it needs to be resolved what the result in the overlapping areas is. A common solution is that the most recently added or modified feature determines the interpretation of the interaction domain. A variation, applied in semantic feature modelling, is the use of a feature dependency graph, whereby dependent features determine the interpretation when interacting with the features on which they depend [Bidarra and Bronsvort, 2000]. Neither of these solutions works for all possible cases [van der Meiden and Bronsvort, 2007]. For our work, it is not relevant how exactly the interactions are resolved, so we just assume that there is a method that takes care of this.

The problem of interpreting interacting features extends beyond the case of additive and subtractive nature, to interacting feature aspects in general. An artificial example would be a feature with a 'colour attribute', which gives both the boundary and the volume a colour value. This is not a simple boolean value, like in the case of nature. In areas where features of different colour overlap, it must be resolved which colour, possibly a 'mix', is assigned. The need for interaction resolution is common to all entities, including edges and vertices. In some cases, such as distilling the BRep from a set of interacting features, the contribution of lower dimensional entities depends trivially on the volumetric contributions, i.e. only entities that are adjacent to volumetric contributions of different nature, are part of the BRep. It cannot be assumed though that a trivial solution for resolving the interpretation for entities of lower dimensions applies to feature interaction in general.

Various types of representation are used for feature models. The BRep is a straightforward choice, but it has deficiencies in the context of advanced feature models that need to store information on the overlap of features or, more in general, that need to maintain feature semantics [Bidarra and Bronsvort, 2000]. The cellular model seems to be a more suitable representation in this context [Bidarra et al., 1998]. We therefore use the cellular





**Figure 5.5:** Example of a cellular model

model for the representation of the feature models.

The cellular model captures all interactions of features, including interactions that do not affect the geometry of the boundary of the model. It is a non-manifold geometric representation of a feature model. For a model that consists of a single volume, its cellular model is a connected set of quasi-disjoint cells such that the geometry of each feature is represented by a set of cells. The subdivision into cells is determined by the property that no two cells can overlap volumetrically. Whenever two features overlap, their geometry is split into cells such that each cell either completely describes an overlapping region or the volume covered by the cell is exclusive to a single feature. A feature that covers a cell, is said to *own* that cell. Each cell is owned by at least one feature; where features overlap, the cell has multiple owners. The cells that contribute material to the model are said to have *additive* nature, whereas the other cells have *subtractive* nature. Figure 5.5 shows a simple cellular model. In this figure, there are three features: a base block, and two slots. For each cell with subtractive nature, it is indicated which feature(s) own(s) it.

Each face in the cellular model delimits either two adjacent cells, or a cell and the outside of the model. For each face we can thus discern two sides or *cell faces*. Each cell can be described in terms of its set of cell faces. This is the basic cellular model. The model can be extended such that its edges and vertices can also be accessed as entities belonging to a particular cell. In that case, however, we obviously cannot use the concept of ‘sides’, since the number of cells that can share an edge or vertex is not fixed.

For each cell, cell face, and optionally cell edge and cell vertex, it is recorded in an *ownerlist* which features own that entity, i.e. to which features it belongs. This is essential for manipulation of and reasoning with the features. The meaning or interpretation of each entity is also stored. In

the simplest case, considered here, this is the nature of the entity, which effectively indicates for 3D cells whether the volume lies inside or outside the model. The concept of nature can be extended to lower dimensional entities, where it indicates whether the entities are part of the model boundary. We refer to such entities that are part of the model boundary as entities with *boundary* nature, and to those entities that are not part of the model boundary, as having *non-boundary* nature. The cellular model can be built and modified by the addition and removal of individual features, and supports a range of methods for querying the ownership and interpretation of entities and their relation to other entities in the model. To keep track of the cell geometry, and ownership, we partly rely on the *cellular topology* component of ACIS [Spatial Corporation, 2006]. A thorough description and discussion of the cellular model can be found in [Bidarra et al., 1998].

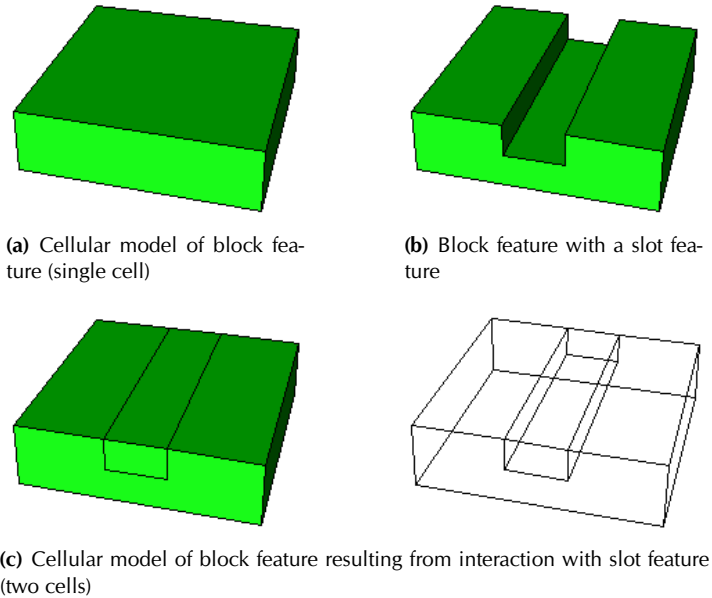
In the sequel of the chapter, we thus assume that the feature models under consideration are represented by a cellular model. Our representation of the difference between two feature models relies on a structure resembling a cellular model.

For our work it is additionally important that the models are *complete* feature models. This means that a model is completely determined by the aggregation of features, and that each feature has its individual geometry explicitly defined, instead of implicitly defined based on possibly non-persistent BRep entities pertaining to other features. Blends in current commercial systems, for instance, are not implemented as features with a geometric definition of their own, but rather are BRep modifying features [Nyirenda et al., 2007]. We, however, need to track the geometry of *each* feature and how it evolves between models. If a feature that has a blend attached, is moved together with the blend to a different location in the model, we should be able to relate the geometry of the blend at its new location, to the geometry on its previous location.

Lastly, we assume that the differences between two feature models that we describe are relatively small, in particular, that the models are variations in the evolution of a single design. It should be possible to relate corresponding features between the two models.

### 5.3 The difference between two feature models

An important aspect in the feature difference is that a feature, placed into a model, is affected by its interaction with other features. One of the clearest examples hereof is the interaction of features having different nature (adding material or removing material): when a slot feature is added to a block, then the geometry of the block changes, as does the volume the block occupies.



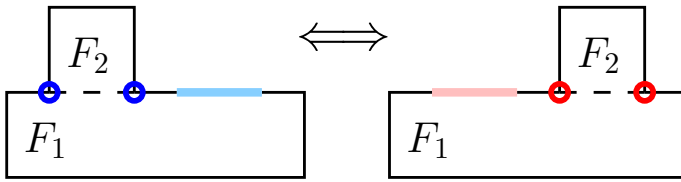
**Figure 5.6:** Block feature interacting with a slot feature

This is illustrated in Figure 5.6. More subtle is the change when an additive feature is attached to another additive feature. Figure 5.4 illustrates this, showing the imprint that the rib feature causes on the geometry of the base block by its attachment. When looking at the geometry of a feature in a particular model, we also take this additional geometry, resulting from its interaction with other features, into account.

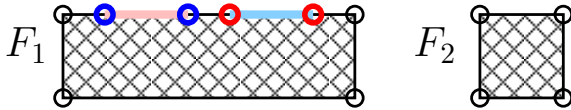
We now explain in some detail what the feature difference is. For a particular feature, it is the comparison of the feature as it was in the original model, with the feature as it is in the modified model. This comparison includes the whole of the geometry of the feature, as it is stored in the two corresponding cellular models, which is more than what appears in a BRep. In a BRep only those elements appear that are part of the boundary of the model, whereas in the cellular model the complete boundary of each individual feature is stored. We refer to those elements that are part of the cellular model, but do not appear in the BRep, as *non-boundary geometry*. The feature difference is defined for each feature, and consists of both BRep and non-boundary geometry of the feature as part of the original model, combined with the corresponding geometric information of the feature as part of the modified model. Additionally, the structure consists of cells that

represent regions of volume. The feature difference thus relates the complete geometric information of a single feature, including geometry emerged from interaction with other features, between the two models. We recall that the collection of all feature differences for two models is called the *difference model*. Since features overlap –if not with their volumes, then with their faces–, some geometric elements are part of the feature difference for more than one feature. Depending on the feature for which the feature difference was constructed, it can vary how that element is interpreted (as unchanged geometry, new or old). We call this interpretation of the difference by a particular feature, its *perspective* on the difference. We will now illustrate the feature difference, including this aspect of multiple perspectives, by looking at a couple of examples.

To illustrate our ideas, we primarily use simple, 2D examples. However, the principal application in mind is 3D modelling, to which the concepts readily extend. In Figure 5.7(a) there are two models. On the left is the original model and on the right is the modified model. Both models consist of the same two features  $F_1$  and  $F_2$ , but their relative positioning is different, as the location of feature  $F_2$  has been modified. In Figure 5.7(b) we see the feature difference for each of the features. We discern four classifications of geometry in the feature difference: 1) *persistent-identical*, 2) *persistent-different*, 3) *new*, and 4) *old*. Persistent-identical geometry is, from the perspective of a particular feature, identical in every aspect in both models. The complete geometry of feature  $F_2$  is an example of this, as both its shape and its interactions with other features are the same in the original and the modified model. The feature difference for  $F_1$  carries two examples of persistent-different geometry: the separation between feature  $F_1$  and feature  $F_2$  as it was in the original model (pink), and the corresponding separation in the modified model (light blue). Note that the light blue edge already existed as part of the BRep in the original model. In the modified model, however, it is not part of the BRep, but it *does* exist in the cellular model, since it is part of the geometry of the two individual features. We classify this in the feature difference as persistent-different geometry, which thus indicates that the element was part of the BRep in one of the two models, but not in the other. More specifically, we call the example of the light blue edge persistent-different: *modified to non-boundary*, since it was part of the BRep in the original model, but not in the modified model. The reverse holds for the pink edge, which we classify as persistent-different: *modified to boundary*. The red vertices at the ends of the light blue edge are *new* to feature  $F_1$ , as they were not present in the original model. The blue vertices at the ends of the pink edge are *old* to feature  $F_1$ , as they were present in the original model, but not in the modified model.



(a) Original model (left) and modified model (right)



(b) Difference model, consisting of the feature difference for both features

**Figure 5.7:** Difference model: relocation of a feature









**Table 5.1:** Classification of the feature difference for cells; ‘+’ = additive nature, ‘-’ = subtractive nature, ‘ ’ = not in model.

original	modified	classification
+	+	⊗ persistent-identical: additive nature
-	-	⊗ persistent-identical: subtractive nature
-	+	⊗ persistent-different: modified to add. nature
+	-	⊗ persistent-different: modified to sub. nature
	+	⊗ new: additive nature
	-	⊗ new: subtractive nature
+		⊗ old: additive nature
-		⊗ old: subtractive nature

Table 5.1 lists all possible classifications of geometry that can be encountered in the feature difference for cells, whereas Table 5.2 does this for faces, edges and vertices. Except for the terminology, the two classification schemes are very similar, as the ‘boundary / non-boundary’ and ‘additive / subtractive nature’ classifications fulfil analogous roles. The distinction in the names helps to reason more intuitively with the concepts. In our 2D examples, there are no faces, and the interior regions of the features constitute the cells, which we classify by nature. The tables also serve as a legend for the colours in our illustrations. Note that some of the classifications are indicated by the same colour, but it should be clear from the context which is indicated.

The first example, in Figure 5.7, showed a case of change in relative positioning of features. This is the first of the three types of model modification

**Table 5.2:** Classification of the feature difference for faces, edges and vertices; b = boundary geometry, n = non-boundary geometry, ' ' = not in model.

original	modified	classification
b	b	 persistent-identical: boundary
n	n	 persistent-identical: non-boundary
n	b	 persistent-diff.: modified to boundary
b	n	 persistent-diff.: modified to non-boundary
	b	 new: boundary
	n	 new: non-boundary
b		 old: boundary
n		 old: non-boundary

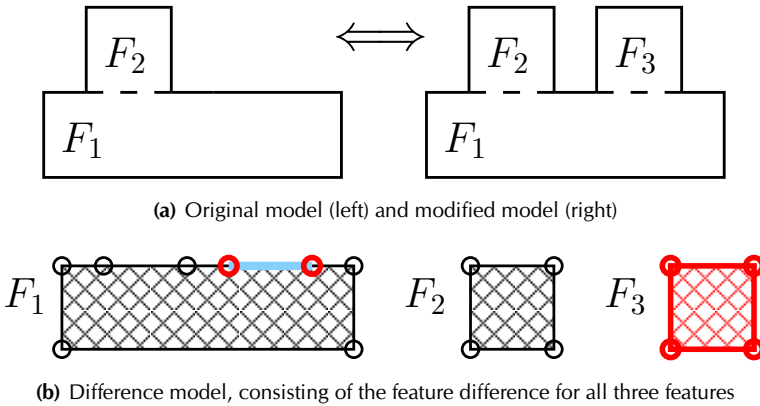
that we discern for our examples:

- change in relative positioning of features
- addition or removal of features
- change in feature shapes.

A combination of these operations can cover any modification that can be made to a feature model. We now look at examples of the other two types of modification.

Figure 5.8(a) contains an example of feature addition. This example is quite similar to the previous example in Figure 5.7, only here feature  $F_2$  stays in its place and a new feature  $F_3$  is added to the model. The original model thus consists of two features and the modified model of three. In Figure 5.8(b) we see the feature difference for each of the features. The complete geometry of feature  $F_2$  is once again persistent-identical, as from its perspective everything stays the same. The geometry of feature  $F_3$  is completely new, since none of it existed in the original model. In the feature difference for feature  $F_1$ , the separation between feature  $F_1$  and feature  $F_3$  is classified as persistent-different: modified to non-boundary, as indicated by the light blue edge. The reasoning is similar to the previous example: the edge was part of the boundary geometry in the original model, whereas it exists as non-boundary geometry in the modified model.

Removal of a feature is essentially the same as addition, only with the order of the two models reversed. If we reverse the roles of original and modified model for the example in Figure 5.8(a), then the difference result would be identical to the one presented in Figure 5.8(b), except that the red entities of  $F_3$  and the red vertices of  $F_1$  would be blue, and the light blue edge of  $F_1$  would be pink. In general, the feature difference is symmetric, meaning



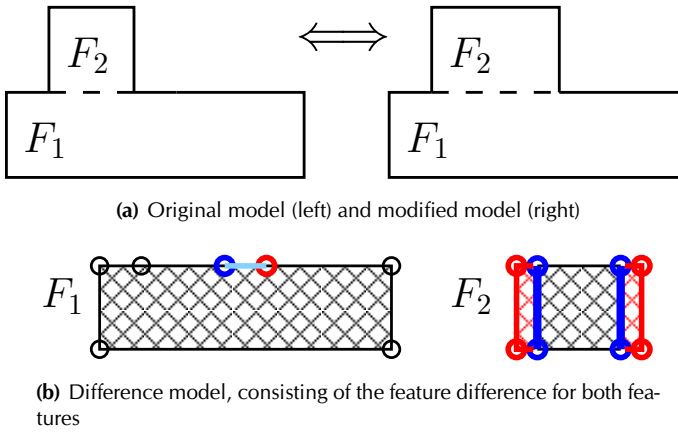
**Figure 5.8:** Difference model: addition of a feature

that it effectively makes no difference which model is the original model and which one the modified model. Only the interpretation of new /old and the direction of modification will be switched. There is no preference for either of the models when describing their difference. Each feature that exists in either of the compared models will thus have to be represented in the difference model.

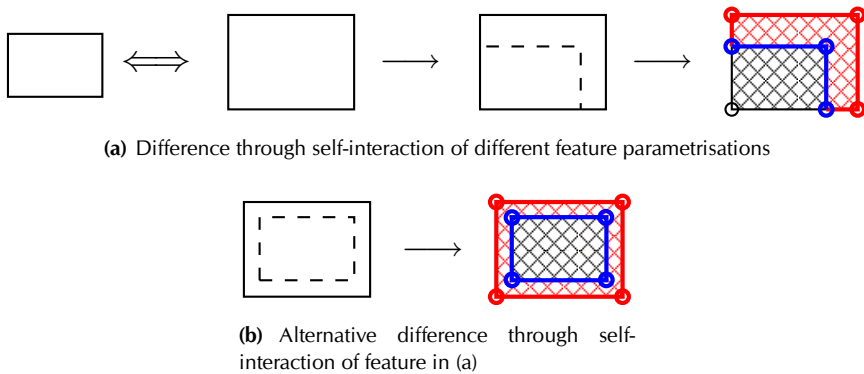
In the previous two examples, none of the individual features change shape. In Figure 5.9, we have an example where a feature does change shape. We could handle such a case as a feature removal and addition, but this precludes the feature's geometry from being mapped. Instead, we let the two different versions of the feature interact with each other and combine their geometries. In this case, part of the geometry of the feature will not be persistent, since the shape, i.e. geometry, of the feature differs between the two models. This is illustrated by feature  $F_2$  in the difference model of Figure 5.9. The feature difference for feature  $F_2$  may seem surprising: in overlaying the geometries, the feature centre has been used as the point of reference to align on, but other alignments might have been chosen too.

In general, there is no unique way to compute the interaction of two versions of a single feature. The result depends on how the two geometries are overlaid. Typically we would use some internal coordinate system of the feature to align both geometries, but the choice of this coordinate system is arbitrary. In many cases there will be a natural preference for a certain system, but in particular for features with symmetries the choice can be argued.

Figure 5.10 illustrates how a variation of the point of reference for the



**Figure 5.9:** Difference model: reshaping a feature



**Figure 5.10:** Variation in point of reference in self-interaction between features

self-interaction of two different versions of the same feature affects how we classify the entities for the feature difference. In Figure 5.10(a) the volume and edge-segments that overlap, together with the single lower left vertex, are considered persistent. All entities that do not overlap with geometry of the same dimension are marked as new or old. If instead of a corner vertex, the centre of the feature in Figure 5.10(a) is used as the point of reference, then the resulting difference will be as in Figure 5.10(b).

In some situations, the way in which the designer modified the feature might indicate a preference for how the different versions ought to self-interact. For instance, he might drag a face along one of the feature axes to



elongate the feature, which is a hint that the designer sees the opposite face as a fixed reference. However, this does not apply in general. The shape of a feature might be modified in multiple steps, the feature might be translated, or it might be modified as a consequence of dependency relations. We cannot deduce a single, natural point of reference for all these cases.

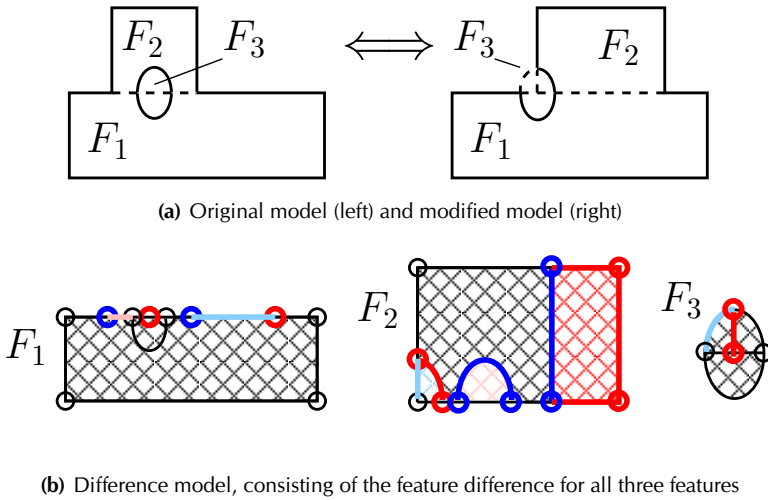
We will now look at a more involved example that shows how reshaping a feature is handled in the context of a model with multiple features and a change in relative positioning of these features. Figure 5.11(a) shows two models built from the same three features. Feature  $F_2$  has been translated and reshaped in the evolution from the left to the right model. Features  $F_1$  and  $F_2$  have additive nature, whereas  $F_3$  has subtractive nature, which is also the resulting nature in areas of interaction with  $F_1$  and  $F_2$ . In Figure 5.11(b) the difference model for these two models is shown. The feature differences for  $F_2$  and  $F_3$  have been enlarged for clarity. All blue and red coloured entities are again old and new, respectively, when regarding the left model as the original model, and the right one as the modified model. This is, as always, caused by a change in interaction with other features or with itself through position or shape change. Pink and light blue again represent the persistent-different entities, with pink standing for geometry that was modified to boundary geometry, and light blue modified to non-boundary geometry. Note that the difference for  $F_2$  contains two such areas caused by a change in the interaction with  $F_3$ , partially enclosed by, respectively, a red and a blue arc coming from part of the geometry of  $F_3$ .

Figure 5.12 shows a 3D example of the feature difference for two related models that consist of a base block, a through hole, and a rib on top of the base block. In the absence of a visualisation tool for the feature difference, we have coloured this example by hand for the faces, edges and vertices. Due to the constraints on the visualisation, the classification of the volume of the cells has not been indicated in the illustration, but it is an essential part of the complete feature difference.

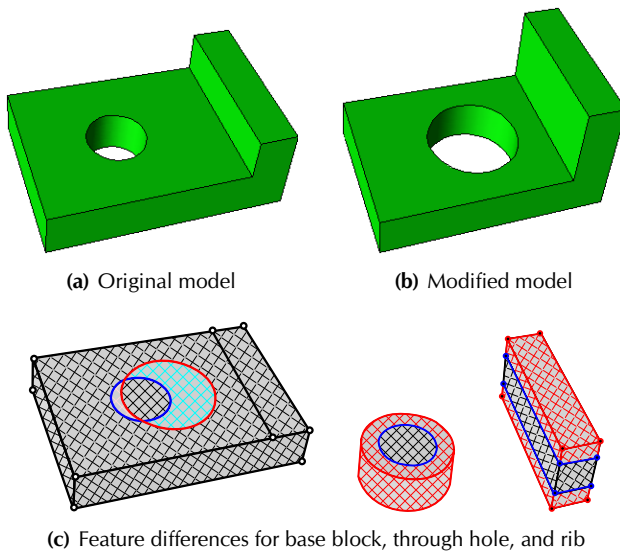
## 5.4 Representing and constructing the difference model

For application of the feature difference, we need to represent the geometry and its classifications, and be able to systematically derive these representations.

The difference between two feature models is represented in the difference model. This model is conceptually close to the schemes used to illustrate the ideas. It is composed of the feature differences for all individual features. Since each feature difference is actually an aggregate of a feature's interactions with two different models, it has little meaning to



**Figure 5.11:** Difference model: relocation and reshaping of a feature



**Figure 5.12:** Feature differences for a simple 3D model

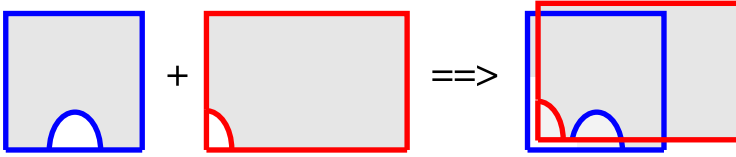
geometrically combine all feature differences into a single model. As it turns out in Chapter 6, the geometric combination of differences can be useful within a specific context, but this combination is not instrumental to the representation of the differences and similarities.

The data structure of the cellular model can be used to represent the feature difference for a single feature, i.e. each feature difference is represented by a separate cellular model. Its cells arise from interactions of the feature in both models with other features, and possibly between different versions of the feature itself. Attached to each entity is its classification of the difference: persistent, new or old. If the geometric entity is persistent, then additionally it is stored whether it is persistent-identical, i.e. its interpretation has remained identical, or persistent-different, i.e. its interpretation differs between the two models. In the latter case it is indicated as well how the interpretation has changed. For the new and old entities, it is clear to which of the two models it belongs. For each entity, the interpretation in the models to which the entity can be mapped, can be accessed. Each entity also has an ownerlist with the features the entity belongs to.

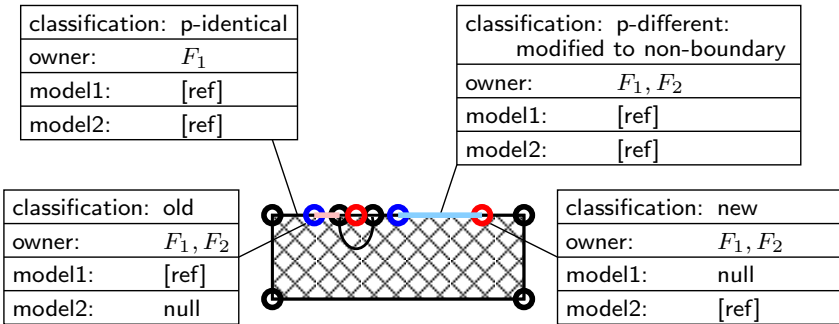
From the difference model of a feature, the contribution of that feature to both feature models can be derived. For efficient use of the model, additional data structures may be used, e.g. to support methods to access all entities of a particular difference classification, per feature or globally. The requirements of the data structures will vary per application. In particular we note that, in common applications, it is not necessary to explicitly construct the difference model for features that have an identical shape and interaction with surrounding features in both models. In those cases we just record that, from the perspective of these features, every entity is persistent-identical.

One way to construct the difference model is based on the two cellular representations of the feature models under comparison. The feature difference for a particular feature is the result of a non-regular union operation between the two cellular models that cover the shape extent of the feature in, respectively, the original and the modified model. Each shape extent is a subset of the cellular model that represents a complete model, but is by itself again a cellular model. In a non-regular union, all geometry of the combining objects is kept, e.g. faces that are part of another face are not combined into a single face, but remain explicitly available in the description as separate faces. The operation is supported by ACIS [Spatial Corporation, 2006]. Through application of the non-regular union, the difference attributes are also determined.

The process is illustrated in Figure 5.13 for feature  $F_2$  from Figure 5.11(a). The geometry of the first shape extent is indicated in blue and the geometry of the second shape extent in red, i.e. only the edges are coloured. When the



**Figure 5.13:** Merging two cellular models in the construction of the feature difference



**Figure 5.14:** Description of entities in data structure

geometries are combined, all vertices, edges, faces and cells are collected in a single geometric description. By default all entities will be marked old or new, depending on the model they originate from. During the combination, the attributes of merging entities of the same dimension are compared and from this follows the difference attribute. All these merging entities, since they overlap with an entity of the same dimension, will be marked persistent. Additionally, the interpretation is compared for the overlapping entities, and on that basis it is decided whether the classification is persistent-identical or persistent-different, which, for the latter case, includes a specification of how it is different. The other entities, which are either new or old, will not take part in a merge and thus keep the default classification, which will give the correct result.

Figure 5.14 illustrates part of the resulting data structure for four entities in the feature difference for feature  $F_1$  from the models in Figure 5.11(a). The references model1 and/or model2 point to the entity in its originating cellular model(s). If an entity is not persistent, then the reference to the entity in one of the two models being compared, will not exist.

The actual implementation is not as simple as the concept of the operation.

In particular the tracking and merging of cells is a reasonably complex problem. The cellular topology component of the ACIS geometric kernel [Spatial Corporation, 2006] provides basic support to work with cellular models, but propagation and merging of entities, in particular of 3D cells and their attributes, requires substantial effort to make it work correctly and efficiently.

## 5.5 Conclusions

The feature difference between two models categorises, through geometric entities, the complete geometry of both models as similar, thus mappable, or exclusive to either of the models. In addition to the differences in feature geometry, it records the differences in interpretation. We have shown how this works out for the case where the interpretation is simply a question of whether the entity contributes to the boundary or, in the case of cells, to the enclosed volume. However, the difference concept extends to a broad class of features where the interpretation might not be mappable to a boolean value. The only requirements are that the features have a clearly defined geometry, and that the interpretation resulting from feature interaction is determinate.

Computation and representation of the feature difference have been implemented for volume and surface elements as part of the remeshing approach described in Chapter 6.

It is probably only possible to consider an exact feature difference between two models if the features between the models can be mapped. This limits the application to models that are descendants from the same model or a modification of each other. For feature models wherein the features have, for example, been identified by a feature recognition algorithm, creating this mapping is likely to be a problem.

The information contained in the feature difference model is virtually unobtainable from a basic BRep structure. In general, any modelling activity that deals with volumetric interactions of shapes needs a data structure similar to the cellular model. Without such a data structure, everything would need to be calculated on the fly, which for large and complex models will be more costly than maintaining the more complex cellular data structure, and thus needlessly hampering interactive performance.

With the feature difference we can identify, in particular, the differences and similarities between two analysis models, such that parts of the previous analysis mesh can be reused in the construction of a new mesh. How this procedure works, and the role therein of the feature difference, is discussed in the next chapter.

Since the feature difference is geometrically exact and close to our intuitive idea of how two models differ, because it is in terms of the features that compose the models, we suppose that there are more applications than just remeshing. Any application that has complex data attached to its features, or represented by specific features, and needs to map this to a similar model, might benefit from using the approach introduced here.

## Chapter 6

# Efficient tetrahedral remeshing

Finite element analysis (FEA) is nowadays widely used by industry to perform product tests. These tests reduce the number of real world test models that have to be built. This is beneficial as building prototypes is costly in terms of both time and money. In Section 2.3, some background information on FEA has been given.

Although FEA saves time and money in comparison to traditional product testing, it is nonetheless a time-consuming operation by itself. For complex models, the analysis process can take multiple months from start to finish, with the actual numerical analysis taking far less time than the work in preparation of the analysis. This is partly due to lack of automation and tool integration, poor data conversion, and (manual) repetition of tasks. One of the pivotal steps that precedes the simulation is mesh generation, the decomposition of the virtual product model into a mesh of simple geometric elements. In Chapter 3, some background information on meshes and mesh generation has been given.

The computation time and accuracy of the analysis depend, amongst many factors, on the mesh and the quality of its elements. In general, the use of higher quality meshes, decreases the time spent on analysis. Higher quality meshes, however, take longer to generate. As we have argued in Section 3.3, the efficiency of the product design cycle might well be improved by means of remeshing. This should enable faster generation of meshes, without making concessions to the high quality of the meshes. In this chapter, we present an approach for efficient remeshing of tetrahedral meshes.

We start, in Section 6.1, with some background and a conceptual discussion of our remeshing approach and its relation to the difference model. This is followed, in Section 6.2, with the description of a data structure that extends the difference model, and plays a central role in the remeshing algo-

rithm. Then, in Section 6.3, we present the basic idea of the actual algorithm. In Section 6.4 we describe how, based on the feature difference, node subsets of an earlier generated mesh are copied, in Section 6.5 how we fill the areas of the model that have remained void of nodes with new nodes, and in Section 6.6 how to complete the new mesh by combining the copied nodes and the new nodes, and efficiently constructing a quality mesh from all these nodes. This is followed by Section 6.7 with a presentation of some results that demonstrate the gain in efficiency and the quality of the meshes. Finally, we conclude the chapter in Section 6.8.

The larger part of this chapter has already been published in [Sypkens Smit and Bronsvort, 2009a].

## 6.1 Remeshing based on the difference model

With more sophisticated algorithms for quality mesh generation coming at our disposal, more CPU time is being spent on meshing. Meshing algorithms that strive to optimise some quality measure on the mesh are often of variational nature, minimising an energy functional related to the quality measure. An example of this is the variational tetrahedral meshing algorithm (VTM) for mechanical models, which has been introduced in Section 3.2 and elaborated in Chapter 4. We use VTM in the sequel of this chapter to refer to the extended method we have described in Chapter 4.

In Section 3.3, we have explained how this focus on quality meshes has motivated us to make meshing more efficient by basing the construction of a new mesh on a previous mesh, which has been used in an earlier design iteration. We base our remeshing approach on VTM, because it produces high quality tetrahedral meshes. We prefer tetrahedral meshes since they are easier to deal with in completely automated methods than the alternative of hexahedral meshes. The concept of our approach to remeshing is, however, not strictly bound to our particular meshing method, nor to the type of mesh element.

In our work, the analysis model is assumed to be a feature model. In current practice, the design model and the analysis model are usually not the same. In Chapters 7 and 8, methods are discussed that aim to improve the integration of the analysis model with the design model. We assume here that the design model is the analysis model, or that changes to the design model can be automatically propagated to the analysis model, analogous to change propagation in the multiple-view feature modelling paradigm, which will be described in Chapters 7 and 8.

We look at model modification either in the context of a humanly controlled design cycle or an automatic shape optimisation process. In particular



in the latter case, many meshes can be created. In either case, we expect the iterative model improvements, in particular during the later stages when analysis becomes more important, to have a local scope, i.e. to change the geometry in a relatively limited way.

Looking at these changes between models in the context of efficient remeshing of a feature model, it seems natural to regard the geometry of the model and the changes therein from the point of view of the features. Instead of relating parts of two models in a global sense, such as by way of the boolean difference of the two models as a whole, we relate the two models on a feature basis. This is the most natural approach, since the models themselves are composed and modified by means of features. When, for example, a feature is relocated, we can still identify it as the same feature, and as such we can relate the geometry pertaining to this feature between the two models. This principle of relating models based on their individual features lies at the heart of the difference model, as discussed in Chapter 5. This is a significant difference with the conventional approaches to model comparison, referenced in Section 5.1, where there is only a single perspective for the comparison, namely the complete model. In our approach, a geometric element can be related to the other model from the perspective of one feature, whereas from the perspective of another feature there is no relation. Multiple perspectives only occur in the feature difference where features overlap each other (see Chapter 5).

Building upon the ideas that underlie the difference model, we have conceived a new, efficient remeshing approach that copies parts of the mesh, exploiting the relations of the features between the two models. This allows for a mesh correspondence that is intuitive. The concept of associating mesh elements with individual features has been suggested in the past [Unruh and Anderson, 1992], but this was in the context of complete mesh generation. Another reason that makes the difference model more attractive than other methods for model comparison, is that these deal exclusively with the BRep, whereas the feature difference explicitly maps parts of volume between the original model and the modified model as well. We need this for the purpose of copying mesh elements between corresponding parts of volume.

The novelty of our method is that we can handle more complex shape modifications than the approaches referenced in Section 3.3, including changes in topology caused by parameter modifications and feature addition/removal. It also tends to conserve larger portions of the mesh, and the mesh correspondences match well with the intuitive correspondences we see. Subtractive features and overlapping features are not an obstacle to our approach. Maintaining the quality of the existing elements and delivering a complete quality mesh as a result are our objectives. To this end we

determine exactly how the geometry differs between the two models.

For the remesh procedure, thus a description is used of the difference between two models, such as the models in Figure 5.2 on page 73. In line with the terminology used in Chapter 5, the model that was made first is being referred to as the *original model*, whereas the model resulting from adaptation is being referred to as the *modified model*. The description of the difference should help us to find those parts of the geometry that appear identically in both models, as for these parts we might reuse mesh elements from the original model for the new mesh of the modified model. For those parts of the geometry that have no relation to the other model, we need to construct new mesh elements. Identifying all those parts of the geometry is an essential aspect of the remeshing procedure.

In the next section, we describe a data structure that extends the difference model, and plays a central role in the remeshing algorithm in carrying out the aforementioned tasks.

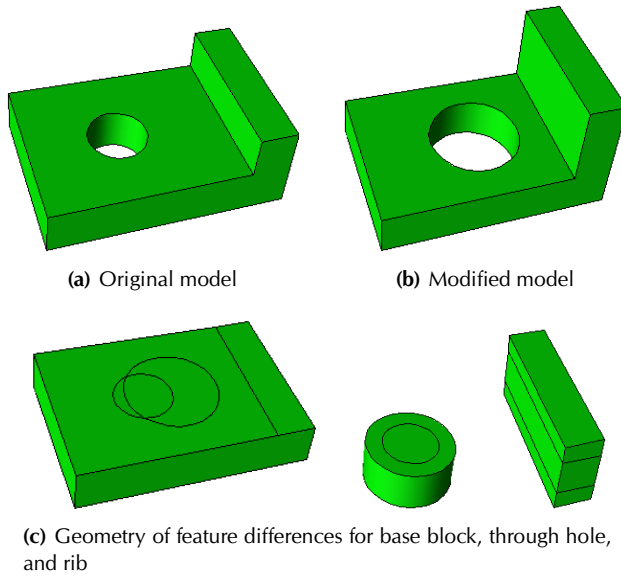
## 6.2 The combined model

We use cellular models to store and query the original and the modified feature model. The feature differences are cellular models as well, as they are constructed by copying and combining subsets of the original and modified models.

The feature difference for a particular feature contains the combined geometrical description of the feature for the two models that are being compared. All the geometric entities carry a feature difference classification (see Tables 5.1 and 5.2 on page 81), and a list of the features that own the entity. Together with the classification, it is also stored to which model an entity belongs. This is actually enough information to reconstruct the geometry of a feature in either the original or the modified model from the feature difference. To recover the geometry of a particular feature in the original model, we can basically remove all geometric entities classified as new; the geometry of the feature in the modified model can be obtained by removing all old entities.

Each feature in a model is associated with a transformation, which is tied to a global point of reference by which the positions of all features are related. We regard the transformation as part of the feature parametrisation. When the feature parametrisation is known for all features in a model, then its cellular model can be (re)constructed.

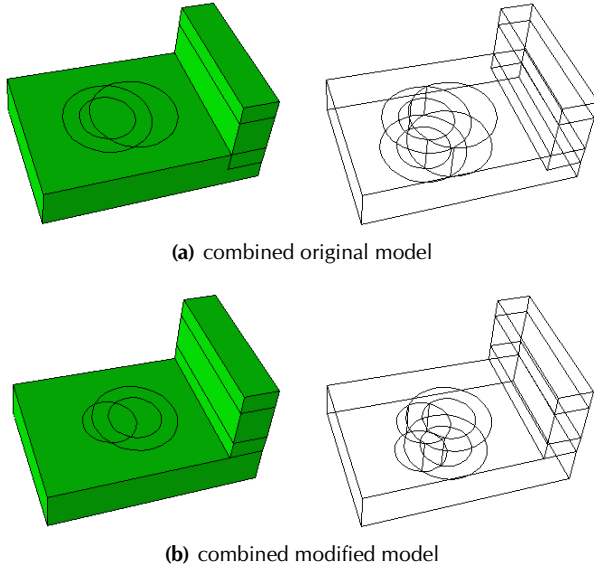
It follows from the previous two paragraphs that from the feature differences, both the original and the modified model might be obtained. Of course we do not actually do this, as we already have those models, but



**Figure 6.1:** Feature differences for a simple 3D model

in a similar vein we can also combine the feature differences by using the feature transformations of either the original model or the modified model. Figure 6.1 reproduces the example of model modification from Figure 5.12 (page 86), except that the features differences are represented in Figure 6.1(c) with their bare geometry without classification. The model consists of three features: a base block, a through hole and a rib, with each a corresponding feature difference. Figure 6.2 shows how the three feature differences can be combined into two different structures: (a) the *combined original model* (following the positioning of the features in the original model), and (b) the *combined modified model* (following the positioning of the features in the modified model). Notice how the two structures differ. For the combined original model, the feature difference for the through hole and the rib are positioned w.r.t. the base block such that the original feature geometry lines up. For the combined modified model, on the other hand, the feature differences are positioned such that the modified feature geometry lines up.

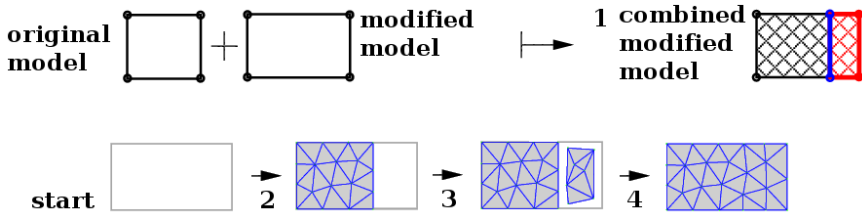
A combined model captures the interaction of feature differences, and by this means enables us to relate the original geometry and the modified geometry to each other. For our remeshing algorithm, we are only interested in the combined modified model. We will briefly explain how this particular structure is useful to our remeshing algorithm.



**Figure 6.2:** Combining feature differences following the feature configuration of either the original or the modified model

By selecting the right subset of cells from the model in Figure 6.2(b), we can construct the model of Figure 6.1(b). The complete geometry of the model we need to mesh, without use of transformations because the cells are already connected in the right way, is thus represented by a subset of the combined modified model. To each of its cells, at least one difference classification is associated, but multiple classifications per cell are possible. This is because the feature differences in the combined model can overlap, e.g. the feature differences for the through hole and the base block overlap in both examples of Figure 6.2. From the difference classifications, we know whether the geometry, from the perspective of some feature(s), is persistent (or not) and thus can be related to geometry of the original model (or not). This is why the combined modified model is essential to our goal of remeshing, as through these relations we find the subsets to copy from the mesh of the original model, the parts of the geometry where new mesh elements need to be constructed, and all other parts of the geometry where attention to the mesh is required, e.g. the places where mesh subsets of different origin meet.

The geometry of a combined modified model can get fairly complex, as the feature differences, which already contain the geometry of two different



**Figure 6.3:** Schematic representation of the four steps of the remeshing procedure

versions of a feature, can intersect once more amongst each other. Inspection of this structure, however, is essential for our remeshing procedure.

However, before delving into the details of using the information in the combined modified model, we take a step back, and discuss the overall remeshing procedure.

### 6.3 The remeshing procedure

The remeshing procedure is conceptually simple. We wish to generate a mesh for a model that is a modified version of an earlier design model (the modified and the original model, respectively). Part of this mesh is going to be supplied by the mesh of the original model. The procedure for this is:

1. Analyse the difference between the two models.
2. Initialise the new mesh with all mesh sections that can be copied.
3. Construct new mesh elements in remaining void areas.
4. Perform VTM with an appropriate subset of the nodes and boundary samples.

This procedure is schematically represented in Figure 6.3.

The first step consists of the construction of the feature difference and the combined modified model, which have been discussed in Chapter 5 and the previous section, respectively.

In the second step, we map subsets of the previous mesh to the modified model. The idea is that we determine from the combined modified model which (parts of) features from the original model can serve as a mesh source, and from this we decide which mesh elements to copy to the new mesh. Here we aim to copy large and continuous mesh subsets, such that extra work to improve connections between separately copied mesh subsets is

minimised. Any part of the geometry of the new model that is not assigned a mesh by copying, remains without mesh.

In the third step, we fill up these voids. We now have the basis for the new mesh.

In the fourth step, the VTM algorithm is executed, with the result of the previous two steps as initialisation of the mesh. Since copied subsets of the mesh do not need optimisation, we adapt the VTM algorithm to work only with a subset of the nodes and the boundary samples. The attention is focussed on the new mesh subsets and the other places where the mesh needs improvement, such as between adjacent mesh subsets where a good connection is lacking.

Until now, we have consistently talked about copying “subsets of the mesh”. Since the VTM algorithm generates a Delaunay mesh, we can suffice with copying just the mesh nodes, as the connectivity is handled in the VTM algorithm by the Delaunay criterion. This makes the procedure considerably easier to implement. The input for the VTM algorithm, here applied in the context of remeshing, will be the set of initial nodes to start with, accompanied by a flag that indicates which nodes are *free* and thus are in need of optimisation. The new nodes and the nodes in areas of transition between copied mesh subsets, are the free nodes. The other nodes are *fixed*; their relative positioning can remain untouched. We now continue to discuss the procedure for copying nodes.

## 6.4 Copying mesh nodes

The idea behind the copying procedure is that the modified model can receive nodes through the relation of persistent feature volume with the original model. Each feature whose feature difference contains some persistent volume of additive nature, can copy nodes that belong to that volume in the original model, to the modified model.

Since features can overlap, we cannot simply copy the nodes for each cell of persistent volume in each feature. Where features overlap in volume, too many nodes would be assigned. Also, we want to copy mesh subsets in one go from large and continuous regions, such that the need for optimisation between copied mesh subsets is minimised. To this end, we sort the features by the size of their persistent-identical volume of additive nature, and start with copying nodes from the feature for which this volume is largest. Next comes the feature with the second largest persistent volume, etc.

The complete procedure is as follows:

- [ 1. Assign nodes to features in the original model. ]

2. Find cells with persistent volume of additive nature (*copycells*) and cells with new volume (*newcells*) in the combined modified model.
3. Determine which features correspond to which copycells and calculate for each feature the volume of copycells that it covers.
4. Copy nodes of features to empty copycells, starting with the feature with the largest volume to copy.

The last step is finished when all copycells have nodes copied to them. Due to overlap of features, it is not required that all features have contributed nodes to the new mesh at the end of this step.

#### **Assign nodes to features in the original model**

This step is independent of the modified model, and as such it is not strictly part of the remeshing procedure. It can be performed in advance.

The original model is stored as a cellular model. All nodes of the corresponding mesh are assigned to cells of the cellular model. To speed up the operation, the nodes are first tested for inclusion with the bounding boxes of the cells. Then, for each node on or inside a bounding box, an accurate inclusion test (an internal function of ACIS [Spatial Corporation, 2006]) with the corresponding cell is performed. A few nodes might not be assigned to any cells at all, due to tiny differences between the coordinates of the mesh and the geometry of the model. In such a case, the projection distance of the node to all cells is calculated and the node is assigned to the closest cell, assuring that every node is assigned to at least one cell.

For each cell it is known to which features it corresponds. After the assignment of the nodes to the cells, we can thus retrieve all nodes inside or on the boundary of each feature.

#### **Determine copycells and newcells**

In the combined modified model (cf. Figure 6.2(b)), all cells have classifications attached, indicating for each feature that (co)owns the cell how its geometry relates to the original model (persistent-identical, persistent-different, new or old). This classification can differ between the owning features. We limit our attention to the cells of additive nature that contribute to the representation of the modified model, since these are the cells that need to be covered by mesh elements. Of those cells, we mark each cell that is persistent-identical with additive nature, according to at least one classification, as a copycell. The volume of such a cell can be related, in at least one way, to the original model, as the persistent classification indicates

that a similar counterpart exists in the original model. All remaining cells of additive nature belong to the set of newcells. These also need to be covered by mesh elements, but they cannot be provided by the mesh of the original model.

### **Create map from features to copycells**

At this point, we know which cells in the combined modified model are copycells, and which nodes are contained in each cell of the original model. Unfortunately, there is no one-to-one map between the cells in the modified combined model and those in the original model. A cell from the original model can relate to multiple cells in the modified combined model. It should be possible to identify those relations, but instead we have opted for another approach.

Since we want to copy sets of cells that are as much as possible adjacent, to avoid having to perform mesh improvements between nodes copied from different origins, we instead copy the nodes on a feature by feature basis. All nodes copied from a single feature, obviously have the same origin. An additional benefit of this approach is that we avoid the need to explicitly establish relations between the copycells and the cells of the original model.

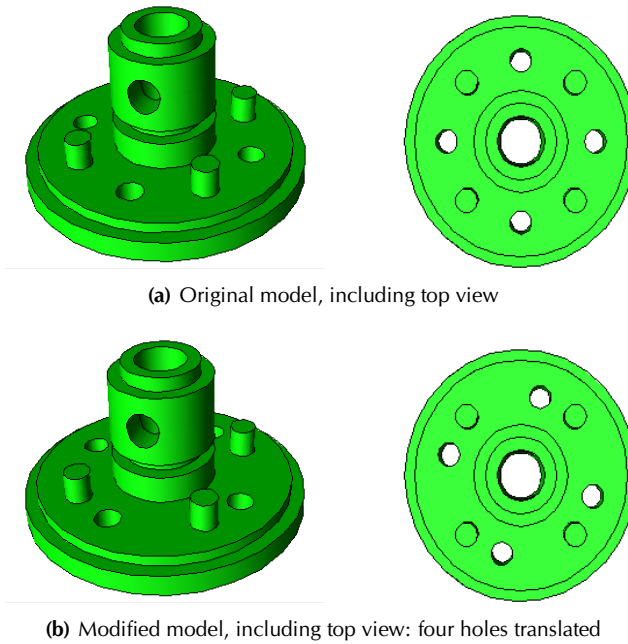
We want to start copying nodes from the feature that occupies the largest volume of copycells. We thus need to calculate the total copy volume for each feature. This calculation is combined with the creation of a map that relates the features to their copycells.

### **Copy nodes from features to copycells**

Starting with the feature that has the largest volume of copycells, nodes are copied to the corresponding copycells. For efficient copying, the copycells are joined into a single body (which can consist of multiple lumps). Then for each node in the feature, an inclusion test is performed against the copycell body. The nodes that belong to the body are included in the mesh of the new model. Before performing the inclusion test, the coordinates of the nodes have to be transformed from their location in the original model to their location in the modified model. This transformation is the compound of the inverse transformation of the feature's position in the original model and the transformation that gives the position in the modified model. We avoid copying multiple times to the same copycell, by keeping a list of copycells that have been taken care of.

Once nodes have been copied to each copycell, the model still lacks nodes in those regions of the model that could not be mapped to the original model, i.e. the newcells. See for an example Figure 6.5(a), where the copied nodes





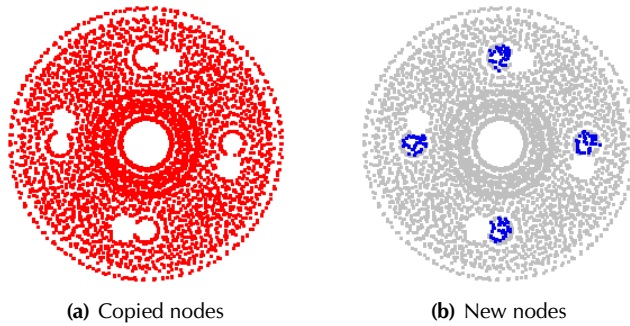
**Figure 6.4:** Two variations of a model

are shown for remeshing the model of Figure 6.4(b) based on the mesh of Figure 6.4(a). At the original location of the four hole features there were no nodes to be copied to the mesh of the modified model. These regions are covered by newcells, and these need to be filled by nodes as well (see Figure 6.5(b)). This is elaborated in the next section.

## 6.5 Adding new nodes, and the free/fixed distinction

The previous section dealt with filling the set of copycells, coming from the combined model, with nodes from the mesh that corresponds to the original model. The newcells set comprises those cells that have additive nature, but cannot receive nodes from the original model. This is either due to a newly added feature with additive nature, an enlarged feature with additive nature, or the removal/translation of a feature with subtractive nature.

The procedure for filling the set of newcells with nodes is basically the same as the node initialisation procedure of VTM described in Section 4.1. Before the nodes are spread out, the average node density in the earlier mesh is measured. Then the cells of a grid that covers the newcells' volume



**Figure 6.5:** Top view of copied and new nodes for the original and modified model of Figure 6.4

are traversed, to calculate the average number of nodes that each grid cell should receive. Only grid cells of which the centre lies inside the newcells' volume are counted. Finally the nodes are spread out by traversing the grid cells once more. Figure 6.5(b) shows an example of a set a new nodes, with in the background the copied nodes.

We now have all the nodes that are to be used for the initialisation of the new mesh. However, to efficiently process these nodes, the nodes need to be divided into a *fixed* and a *free* set. The free nodes will be actively involved in the optimisation process, whereas the fixed nodes will be left untouched. It is clear that all new nodes belong to the free set. The majority of the copied nodes should belong to the fixed set —otherwise there would be little gain in remeshing—, but not all of them.

There are two cases in which attention to copied nodes is required:

1. adjacent cells that have nodes copied to them from different origins
2. persistent-different faces on copycells.

Both cases are illustrated by the example of Figures 5.3 and 5.4 (page 75): 1) in the mesh of the modified model, the nodes of the base block do not match with the nodes of the rib feature on top, as the node sets connect at a place different from before; 2) the nodes of the base block near the place where the rib feature was previously connected, do not properly represent the shape of the boundary; previously those nodes were in the vicinity of a non-boundary face, whereas in the modified model that face does represent boundary.

The faces from the cellular model affected by the first case are determined during the node copy operation. If the copied nodes in two adjacent copycells

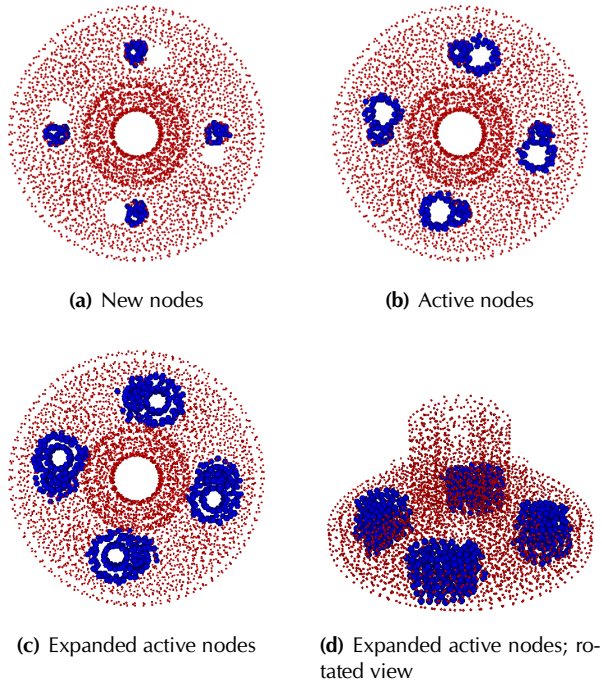
are transferred from their original coordinates by the same transformation, then they have the same origin; otherwise, attention is required to assure the quality of the mesh around the separating face. The second case can be inferred from the combined modified model: for each face of a copycell it is determined 1) whether it has nodes lying in that face, i.e. whether the face was part of the BRep of the original model, and 2) whether nodes need to lie in that face, i.e. the face is part of the BRep of the modified model. If these two results are different, then the nodes in and near that cell face need to be in the set of free nodes. This can be inferred by looking at the classification of the face, from the perspective of the feature that supplied the nodes to this cell. If this classification is persistent-different, then the above is the case. We call the faces from the combined modified model that require attention for the nodes in their vicinity, *active faces*.

For each active face, it is determined which nodes in its vicinity have to be transferred to the set of free nodes. This is done by means of regularly spaced sample points on the surface of the face. Each sample locates its nearest node. All nodes that are nearest to a sample, become part of the free set.

## 6.6 Efficiently constructing a quality mesh

The meshing procedure is basically the same as the one described in Chapter 4, but adapted to avoid unnecessary computations. Instead of following the procedure for creating the initial node distributing, we use the nodes as determined by the procedures described in Section 6.4 and Section 6.5.

These nodes have been classified into two sets, fixed and free. In our optimisation procedure, we only handle the free nodes and leave the fixed nodes untouched. However, nodes from the fixed set can be transferred to the free set. This happens twice during the optimisation, as we transfer the layer of fixed nodes adjacent to the current free set, to the free set. The underlying idea of this expansion of the set of free nodes, is to give the optimisation procedure more freedom to achieve a quality connection between the free and fixed nodes. It is done once directly after the first iteration of the optimisation loop, and three iterations later a second time. The delay in adding the second layer is to first give the smaller set of free nodes an opportunity to settle a bit, as the biggest variations in node locations occur during the first couple of iterations. Once the extent of the changes has diminished, we expand the set of free nodes one more time. Figure 6.6 compares the new nodes, the active nodes at initialisation, and the active nodes after the first expansion of the active set. The final set of active nodes covers a sizeable region, compared to the initial set of active nodes. This is



**Figure 6.6:** New, active and expanded active nodes, for the models of Figure 6.4

necessary to uphold the quality of the final mesh. When expanding the free node set, we must take care to keep the expansion local, as the Delaunay mesh covering the convex hull also connects nodes that are not adjacent in the final geometry. The risk of expanding the free set to a nonadjacent region of the model occurs nearly always when expanding from a boundary node to another boundary node. For that case, we have therefore added the precondition that the two nodes share a connection to an internal node.

The other reduction in computational cost is achieved by adapting the boundary procedure (see Section 4.1 and Section 4.3), wherein the boundary samples are pulling on their closest node to achieve a balanced positioning of the nodes on the boundary. Normally all boundary samples look for their closest node, but since a large part of the mesh does not change at all, this would be unnecessary work. Instead we only do this once for all boundary samples at the start. Those boundary samples that have a free node as their closest node are added to the set of *active* samples, the rest is *non-active*. The samples adjacent to an active sample are placed in the set of *border-active* samples. Only the active and the border-active samples

are used in the boundary procedure to pull on nodes, to possibly change their positions. Most of the time the border-active samples will have a fixed node as their closest node, but when one does have a free node closest, then this border-active sample is transferred to the set of active samples and its non-active adjacent samples turn to border-active samples. This way we remain confident that the boundary samples are doing their work where needed, but no more than that. When the set of active nodes is expanded, the set of active samples is also efficiently expanded where needed.

Summarising, the new meshing procedure is as follows:

1. Initialise the data structures.
2. Initialise the mesh with the free and fixed nodes.
3. Run the optimisation loop:
  - use adjusted boundary procedure based on only the active samples
  - optimise node positions of free nodes
  - if iteration-step = 1 or iteration-step = 4: expand set of free nodes.
4. Extract the mesh.

The number of iterations performed in the optimisation loop will typically be the same number as would be used for a complete meshing of the model. We will now show some of the results achieved by the complete remeshing procedure.

## 6.7 Results and discussion

To study and compare the effectiveness of our remeshing procedure, we present six cases of model modification, all shown in Figure 6.7:

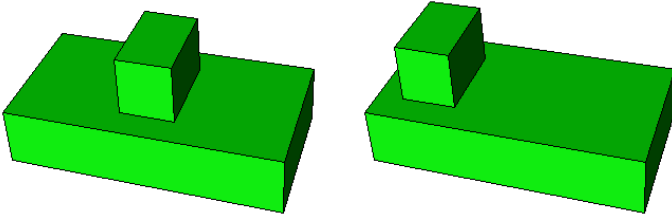
- (a) **simple** translation of block on top
- (b) **tool1-ab** enlargement of pins
- (c) **tool1-ac** translation of holes
- (d) **tool2-ab** addition of stiffener
- (e) **tool2-bc** translation of pipe
- (f) **tool3** reparametrisation of base block.

These cases cover a range of situations encountered in model modification. The first case (a) is a simple example that serves as a reference. The second case (b) demonstrates a change where only reparametrised features are involved. The third case (c) involves the creation of new holes and filling the remaining voids. The fourth case (d) demonstrates the addition of a new feature and its interaction with existing features. The fifth case (e) deals with changes in feature interaction and topology. Finally, case (f) demonstrates the capability of the approach to even handle changes in the shape of the base feature. We do not consider this last case to be typical for the application of remeshing.

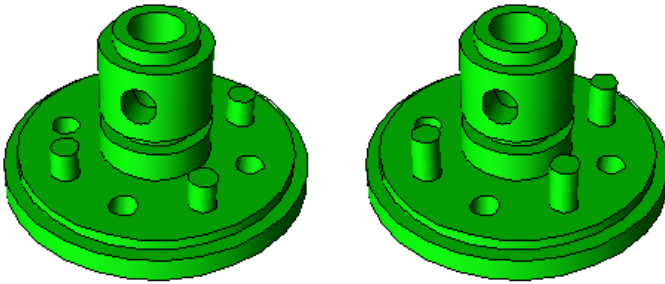
The first aspect we consider is runtime. The results are shown in Table 6.1 (page 113). We compare the time that it takes to mesh the modified model with the regular meshing procedure ( $total_1$ ), with the time needed for the remeshing procedure ( $total_2$ ). A breakdown of the total runtimes  $total_1$  and  $total_2$  is given in Table 6.2 and Table 6.3, respectively. Because the final number of nodes of a remeshed model is hard to control, we first perform the remeshing. After this, the regular meshing procedure is performed, aiming for the same number of nodes. Since occasionally nodes can be removed or added during the algorithm, this match is not perfect. The column *#nodes* in Table 6.1 and Table 6.2 lists the number of nodes in the final mesh for the regular meshing procedure, whereas Table 6.3 shows the number of nodes in the remeshing result. The numbers are generally very close. We have roughly aimed at 15000 or 30000 nodes. The number of boundary samples is, by means of a heuristic formula, set at roughly 8 times the number of nodes that ends up on the boundary. This is a reasonable lower bound for this ratio, since one would rarely want to use less boundary samples per boundary node. The ACIS faceter component is used to generate the samples. Each experiment is performed with either 5 or 10 iterations. The final column of Table 6.1 lists the runtime of the remeshing procedure as a percentage of the runtime of the regular meshing procedure.

We observe that in general:

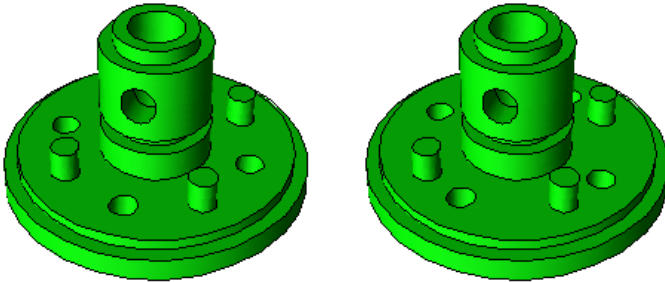
- By remeshing, the runtime is brought down to between 10% and 45% of the time for the regular meshing procedure.
- For a larger number of nodes, the efficiency gain is higher. In such cases, the percentage of internal nodes is higher. Since the optimisation of the internal nodes takes the bulk of the cpu-time, the realised savings by remeshing are higher. A secondary factor is that for a higher number of nodes, the expansion of the free node set by two layers affects a smaller percentage of the nodes.
- The optimisation loop scales roughly with the number of iterations,



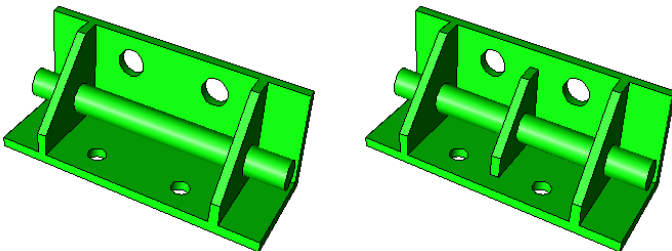
(a) simple-a and simple-b



(b) tool1-a and tool1-b

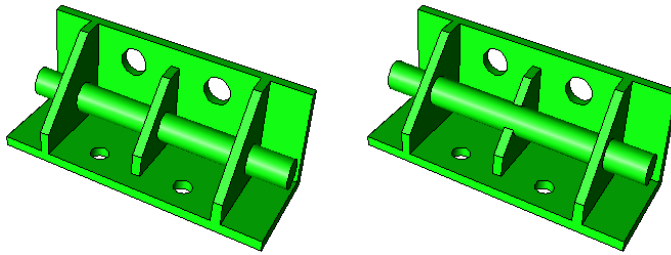


(c) tool1-a and tool1-c

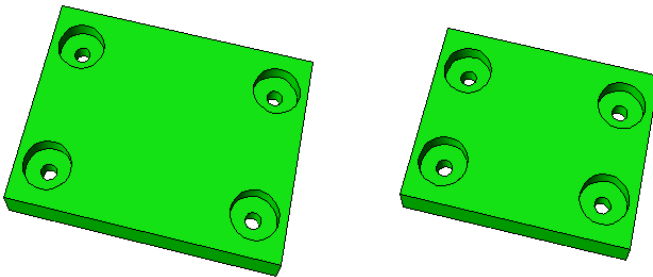


(d) tool2-a and tool2-b

Figure 6.7: Six cases of model modification



(e) tool2-b and tool2-c



(f) tool3-a and tool3-b

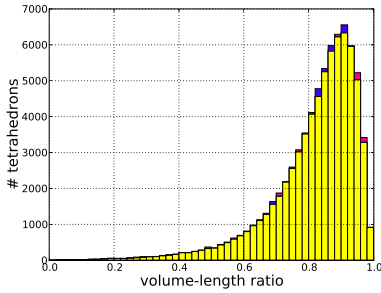
**Figure 6.7:** [continued] Six cases of model modification

as this constitutes the bulk of the computations, and the other steps are independent of the number of iterations.

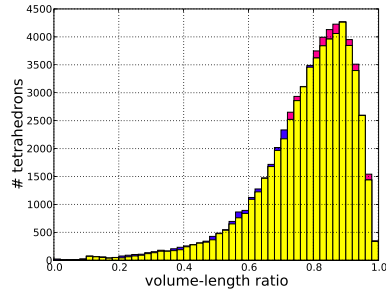
- The time spent analysing the models and copying the nodes ( $t_4$  and  $t_5$ ; Table 6.3) is smaller than the setup time of the standard meshing procedure ( $t_1$ ; Table 6.2).

We measure the quality of a mesh by calculating the volume-length ratio of each tetrahedron (see Section 3.1), and display these values in a histogram. The value 1 corresponds to a perfectly regular tetrahedron. The mesh qualities corresponding to the results of meshing from scratch and of remeshing, respectively, are very similar. The most salient differences are shown in Figure 6.8. Here Figures 6.8(a), 6.8(b) and 6.8(c) show that the quality of the regular meshing is slightly better than the remeshing results, as the magenta bars stick out on the highest end of the quality spectrum, and the blue bars on the lowest end. Figure 6.8(d) shows an example of the opposite case, where the remeshing result is slightly better. The difference in quality for all the other test cases is either similar or even less pronounced.

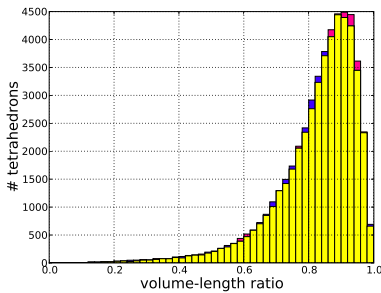




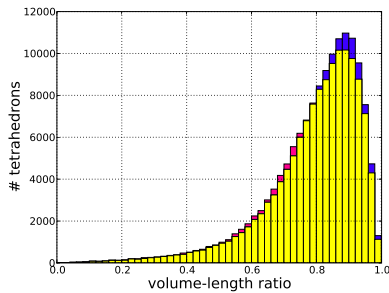
(a) tool1-ac, 10 iterations with close to 15000 nodes



(b) tool2-bc, 10 iterations with close to 15000 nodes



(c) tool3, 10 iterations with close to 15000 nodes



(d) tool2-ab, 5 iterations with close to 30000 nodes

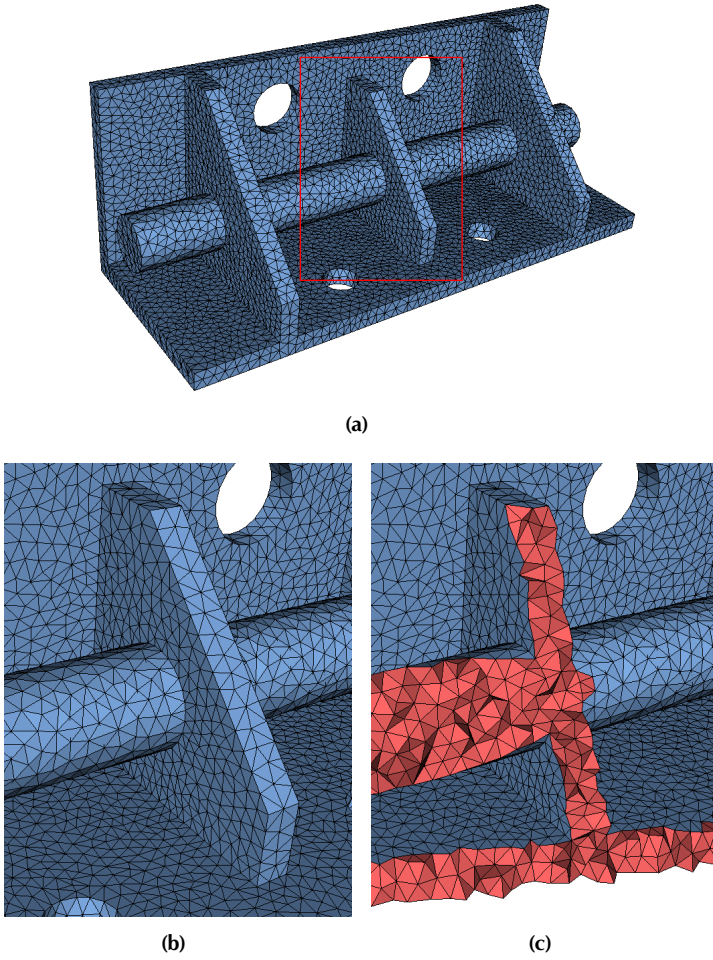
**Figure 6.8:** Quality comparisons between result of regular meshing (magenta) and remeshing (blue). Yellow denotes overlapping histogram bars.

The thin ‘tails’ of lowest quality elements in the volume-length histograms, can often be further improved with little effort. By application of simple operations, such as flipping, most, if not all, of the remaining low quality tetrahedrons can be eliminated.

Figure 6.9 shows the result of remeshing for model tool2, including a close-up and a cut of the close-up, showing part of the interior of the mesh. There are virtually no visible signs that the mesh for the stiffener in the middle was added later by means of remeshing.

## 6.8 Conclusions

We have presented a viable technique for tetrahedral remeshing of feature models for finite element analysis. It is based on the feature difference, and



**Figure 6.9:** Remesh result for tool2-ab

approaches the issue from the point of view of individual features. This is sensible since incremental changes to feature models are made in terms of addition, removal, and reparametrisation of features.

The approach has been actually implemented confirming the validity of the approach, and enabling an evaluation of the performance. The implementation consists of two separate C++ programs. The first analyses the two models and the previous mesh, and outputs the initial fixed and free node set. The second is a variant of the meshing program implemented to test the algorithm described in Chapter 4, adapted to perform the remeshing algorithm.

A reduction of meshing time between 55% and 90% is achieved for our test models. The efficiency gain depends on several factors. Models that have a large volume relative to the surface area, generally result in the most substantial improvements, as they have relatively many elements that remain fixed during the optimisation phase of the remeshing procedure. For a particular model, a larger node set tends to show bigger gains in efficiency, as the percentage of internal nodes, which are the most costly, is higher. The time to construct the combined modified model can increase quickly when many features are overlapping or interacting. In most feature models, however, we expect the number of simultaneous interactions to be low, as many overlapping features are an indication of a bad feature model. Furthermore, for the analysis of the difference between the original and the modified models, resulting in the output of the free and fixed nodes set, we have implemented one particular approach. We feel that this approach is a good compromise between complexity and efficiency. Other approaches might improve the efficiency of this step, but this would have to be researched.

The quality of the remeshing result is consistently on par with the high quality of the normal meshing approach. A key factor in upholding the quality is the expansion of the free node set, which incorporates nodes from the immediate neighbourhood of the areas where the attention is directed to, into the optimisation process. The optimisation procedure can achieve a higher quality result when it can move more nodes. Not expanding the node set leads to a visible disparity between the fixed and free nodes in the final result, and it might even result in failure to properly represent the boundary.

In our remeshing approach, the Delaunay property defines the connectivity. Obviously, when the connectivity had been changed to further optimise the mesh, e.g. to remove near-slivers by flipping, this will not be transferred to the new mesh by merely copying the points. In that case, we would need to copy the connectivity too, or repeat the final optimisation step.

The node density is currently assumed to be uniform over the model. Obviously, when the geometry of the model changes, then the demands for the mesh can change as well. In particular the required mesh density may change. With the type of changes between models that we aim for, we do not expect the need for model-wide adaptation of the density, but the change in density requirements does need to be taken into account. Some regions will have to be adapted. These regions can be identified either by comparing the copied mesh with the new global requirements or, alternatively, through the use of analysis features that specify the density requirements throughout the model. In the latter case, the difference between the density requirements between the two models can be determined exactly. The feasibility of adding density requirements through analysis features is, however, open for research.

The procedure described in this chapter makes some assumptions on how the analysis model is constructed as a feature model, and thus can be related to previous iterations of the analysis model. The ability to relate features between analysis models, implies a consistent relation between the design and the analysis model, since the latter is an abstraction of the former. Unless all changes to the analysis model are performed manually, we would thus also assume propagation of changes between the design and the analysis model. These assumptions usually do not hold in practice. Chapters 7 and 8 further address this issue, by exploring methods to integrate design and analysis models, such that propagation of changes between the models does become possible.

**Table 6.1:** Runtime measurements comparing the regular meshing procedure with the remeshing procedure for the six cases of model modification.  $N_{exp}$ : experiment number, #samples: number of boundary samples, #nodes: number of nodes in the mesh for the modified model after meshing, #iter: number of iterations of the optimisation loop. All times,  $total_1$  and  $total_2$ , are in seconds. Column  $total_1$  corresponds to the regular meshing procedure, and column  $total_2$  to the remeshing procedure. % time =  $100.0(total_2/total_1)$ .

$N_{exp}$	model	#samples	#nodes	#iter	$total_1$	$total_2$	% time
1	simple	56232	15058	5	863.29	91.74	10.62
2	"	56232	15058	10	1634.37	163.80	10.02
3	"	111532	30087	5	3145.85	236.34	7.51
4	"	111532	30088	10	6030.42	433.20	7.18
5	tool1-ab	89290	15195	5	472.46	90.97	19.25
6	"	89290	15195	10	932.37	113.61	12.18
7	"	165814	30341	5	1535.33	177.54	11.56
8	"	165814	30341	10	2870.37	240.57	8.38
9	tool1-ac	87352	14997	5	469.40	108.08	23.02
10	"	87352	14999	10	922.29	165.83	17.98
11	"	163136	30089	5	1528.41	243.93	15.95
12	"	163136	30075	10	2848.63	378.99	13.30
13	tool2-ab	121788	15637	5	196.89	77.86	39.54
14	"	121788	15637	10	332.76	98.10	29.48
15	"	242711	31275	5	505.77	161.64	31.95
16	"	242711	31275	10	874.61	202.19	23.11
17	tool2-bc	117216	15047	5	195.35	83.57	42.77
18	"	117216	15038	10	327.27	103.91	31.75
19	"	233187	30100	5	515.15	154.20	29.93
20	"	232111	30087	10	889.17	190.08	21.37
21	tool3	51818	10248	5	193.81	83.86	43.26
22	"	51580	10194	10	330.39	144.72	43.80
23	"	101220	20480	5	532.60	162.72	30.55
23	"	101216	20420	10	984.91	271.79	27.59

**Table 6.2:** Breakdown of runtime measurements for the regular meshing procedure for the experiments listed in Table 6.1.  $N_{exp}$ : experiment number, #nodes: number of nodes in the mesh for the modified model after regular meshing. All times,  $t_1$ – $t_3$  and  $total_1$ , are in seconds.  $t_1$ : setup,  $t_2$ : optimisation loop,  $t_3$ : mesh extraction and other post-processing,  $total_1 = t_1 + t_2 + t_3$ .

$N_{exp}$	#nodes	$t_1$	$t_2$	$t_3$	$total_1$
1	15058	24.85	835.32	3.12	863.29
2	15058	25.01	1606.25	3.11	1634.37
3	30087	87.54	3051.84	6.47	3145.85
4	30088	87.37	5936.73	6.32	6030.42
5	15195	42.90	422.75	6.80	472.46
6	15195	43.07	882.48	6.81	932.37
7	30341	84.99	1437.65	12.69	1535.33
8	30341	85.74	2772.13	12.50	2870.37
9	14997	41.46	422.27	5.67	469.40
10	14999	41.50	875.08	5.71	922.29
11	30089	82.97	1434.56	10.88	1528.41
12	30075	83.59	2754.29	10.75	2848.63
13	15637	28.65	161.59	6.65	196.89
14	15637	28.83	297.40	6.53	332.76
15	31275	65.89	427.40	12.48	505.77
16	31275	67.10	795.11	12.41	874.61
17	15047	35.81	149.42	10.12	195.35
18	15038	35.91	281.37	10.00	327.27
19	30100	70.66	427.90	16.59	515.15
20	30087	70.84	801.57	16.76	889.17
21	10248	12.31	178.98	2.53	193.81
22	10194	12.40	315.54	2.46	330.39
23	20480	27.70	499.90	4.99	532.60
23	20420	28.28	951.69	4.94	984.91

**Table 6.3:** Breakdown of runtime measurements for the remeshing procedure for the experiments listed in Table 6.1.  $N_{exp}$ : experiment number, #nodes: number of nodes in the mesh for the modified model after remeshing. All times,  $t_4$ – $t_8$  and  $total_2$ , are in seconds.  $t_4$ : setup and construction of feature difference,  $t_5$ : construction of combined model, copying and creation of nodes, and analysis indicating free/fixed nodes,  $t_6$ : setup for meshing,  $t_7$ : optimisation loop,  $t_8$ : mesh extraction and other post-processing,  $total_2 = t_4 + t_5 + t_6 + t_7 + t_8$ .

$N_{exp}$	#nodes	$t_4$	$t_5$	$t_6$	$t_7$	$t_8$	$total_2$
1	15062	0.10	5.07	28.26	54.88	3.44	91.74
2	15062	0.10	5.03	29.82	125.47	3.38	163.80
3	30087	0.10	9.65	77.10	142.30	7.18	236.34
4	30088	0.10	9.75	79.93	336.41	7.01	433.20
5	15199	2.01	12.51	45.19	23.94	7.32	90.97
6	15199	1.97	12.46	45.39	46.55	7.24	113.61
7	30345	1.99	24.11	88.03	49.74	13.67	177.54
8	30345	2.00	24.08	89.09	111.83	13.57	240.57
9	15005	2.34	12.91	44.30	42.42	6.10	108.08
10	15007	2.33	12.89	44.49	100.02	6.10	165.83
11	30097	2.37	25.68	89.24	115.09	11.55	243.93
12	30083	2.28	25.66	87.24	252.19	11.62	378.99
13	15649	1.40	9.01	38.88	22.18	6.39	77.86
14	15649	1.35	8.98	38.97	42.45	6.35	98.10
15	31287	1.39	16.26	86.78	45.64	11.57	161.64
16	31287	1.38	15.99	87.88	85.55	11.39	202.19
17	15055	2.32	9.42	40.74	21.51	9.59	83.57
18	15046	2.30	9.23	40.48	42.23	9.66	103.91
19	30108	2.32	16.59	81.27	37.69	16.33	154.20
20	30095	2.36	16.81	81.12	73.78	16.01	190.08
21	10272	1.77	8.62	17.88	52.57	3.03	83.86
22	10218	1.79	8.56	18.12	113.31	2.95	144.72
23	20504	1.76	16.67	37.34	101.07	5.87	162.72
23	20444	1.76	16.84	37.26	210.09	5.82	271.79





## Chapter 7

# Integration of design and analysis models

We have argued in Chapter 2 that the integration of design and analysis models could be improved. In this chapter, we discuss previous approaches and our vision on how to accomplish this. We briefly recall the characterisation of these models and the issues surrounding the poor interaction between them. For a more detailed account, one should refer back to Chapter 2.

A design model usually consists of low-level curves and surfaces stored by means of NURBS or similar types of representation, a topological structure that describes how all geometric elements are related to each other, a set of parametrised features and constraints that implicitly describe the geometry of the model, and a set of attributes that is attached to certain features or low-level geometric entities.

An analysis model is a geometry description on which an analysis is based. Sometimes no distinction is made between the analysis model and the analysis mesh, but we do: the analysis mesh is derived from the geometry of the analysis model. In some analysis approaches alternative to FEA, the geometry of the analysis model can be used directly [Hughes et al., 2005], but this is not common in practice. Figure 2.2 showed an example of a design model, together with a corresponding analysis model and a corresponding mesh. The analysis model is used to track and store information on the analysis that is independent of the mesh, which includes information to steer the actual mesh generation, in particular mesh sizing.

The geometry of the analysis model is itself derived from the design model, as it usually is an abstraction of the design product geometry. This abstraction is better suited for analysis, generally because it has a simpler geometry, leaving out the details that are irrelevant for the result of the

analysis, and because its geometric description lends itself better for quality mesh generation.

The representation of the analysis model varies. It can be a BRep composed of splines, similar to the common representation of the design model, but since it typically has a geometry that differs from the design model, it has its own distinct representation. There are basically three ways to obtain the analysis model: 1) build it from scratch, 2) adapt the design model into an analysis model, and 3) create the analysis model starting from a faceted BRep of the design model. The choice for a particular approach has consequences for how the two models can interact. In current practice, this commonly results in a poor integration.

But what is it exactly that we mean by *integration* in this context? Integration implies that we have two separate identities, which both should be conserved. As soon as we can no longer distinguish between the two, we have gone beyond integration and ended up with the stronger concept of assimilation.

We could imagine a design environment where analysis would be assimilated into the workflow, i.e. a situation in which there would be virtually no distinction between the design and analysis aspect. In such a case, the analysis would most likely be performed completely transparently, without requiring the designer to activate a separate analysis process of which the only purpose is to support decision making in the design process. In other words, the designer would no longer intuitively experience that he is doing analysis.

Although it is a nice fantasy, few people feel that a true assimilation of design and analysis is within our current reach. Therefore, we aim for something more attainable, which is integration. Here the conceptual divide between the two tasks clearly remains, but the two should work together more smoothly, as any interaction between the two should be transparent. As it currently stands, the relation between design and analysis is not really mutual. The analysis phase clearly serves the design process, but not so much the other way around. In order to achieve integration, according to the definition we have just given, the relation between design and analysis models should become more symmetric, at least from a technical perspective.

Our solution is the *analysis view*, which builds on and extends the concept of *multiple-view feature modelling*. In this chapter, we explain the concept of the analysis view. In Chapter 8, we will discuss some techniques for the realisation of this concept. We start out in this chapter, in Section 7.1, with a comparison of five previously proposed approaches to improve the integration of design and analysis, which usually comes down to a closer integration of their respective models. We contrast these approaches with the

multiple-view approach, and in particular the analysis view, in Sections 7.2 and 7.3. The chapter is concluded in Section 7.4.

The larger part of this chapter has already been published in [Sypkens Smit and Bronsvoort, 2009b].

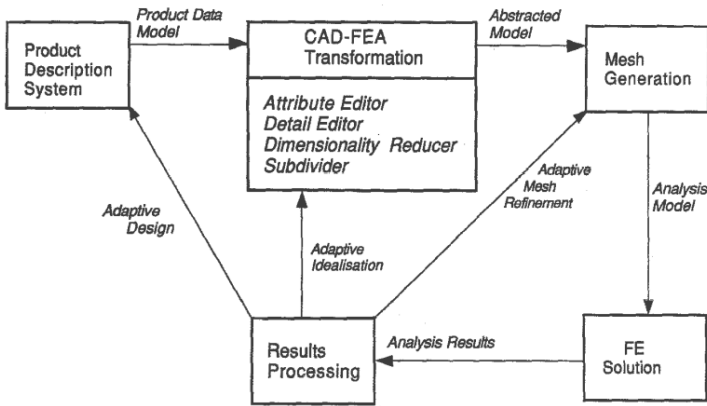
## 7.1 Approaches to integration of design and analysis

There have been several propositions, or *visions*, on how to make the combined workflow of design and analysis more efficient, or, in other words, to achieve integration between CAD and CAE. We have selected five of them to serve as a basis for comparison and discussion of approaches to this integration. The approaches differ in level of technical detail, emphasis and scope, but generally they represent attempts to integrate the design and the analysis models. Together they give a good impression of the spectrum of approaches to the integration of design and analysis.

### **An Early Vision**

The paper *Steps towards CAD-FEA integration* [Arabshahi et al., 1993] is one of the early publications to signal a need for more efficiency between the CAD and CAE process, and gives an overview of “a future system which would allow and encourage more automated CAD-FEA transformation using tools that operate directly on the solid model.” The authors remind us that CAE has not always been performed upfront to assist the design phase and to confirm that products will work as expected, but rather was used in the early days to investigate product failures a posteriori. Although there was effectively no integration at all at the time of writing, the authors already acknowledge the need to couple the design and analysis model: “For a truly integrated CAD-FEA system, attributes must be applied to the design model so that they can be taken into account during the different stages of the idealisation process. A two-way link between attributes and geometry is required to be able to query the geometry with regard to its attributes, and attributes with regard to their geometry.” They also remark that: “[...] it should be possible to relate analysis results back to the design geometry to allow for design modification/optimisation. Because of differences in the geometric domain [...], a simple reverse geometric transformation may not be possible. Formal links may therefore be required between design and analysis models [...] by the exchange of parameters.” These ideas are still topical today. In general, their view on CAD-CAE integration is similar in many ways to the current views expressed by scholars.

Their vision, however, is mostly conceptual. The need for various tasks, and for corresponding specialised tools, is recognised. They distinguish

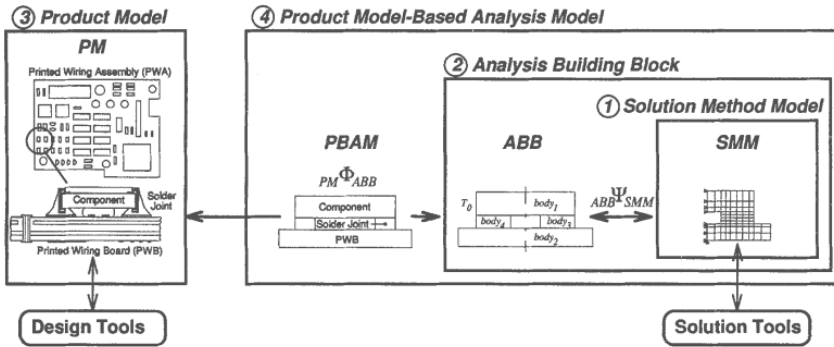


**Figure 7.1:** Overview of the system proposed by Arabshahi et al. Image source: Arabshahi et al. [1993].

functional components such as an attribute editor, detail editor, dimensional reduction aid, and tools to preprocess the geometry for meshing. Figure 7.1 illustrates how the components fit together. They mention requirements for data structures and representations, but details are, not surprisingly, absent. In fact, they primarily perceived a lack of tools to properly derive an analysis model from a design model, and properly prepare it for analysis. Although they recognise the need to link the two models, they may have underestimated how difficult this would prove to be.

### **Multiple Representations**

In *Integrating engineering design and analysis using a multi-representation approach* [Peak et al., 1998], a *multi-representation architecture* (MRA) is presented. The evaluation and validation of the ideas was done in the specialised domain of solder joint fatigue routine analysis, but the proposed architecture is generic and can also be applied in other domains. In many domains, *routine analysis* is regularly applied, i.e. an analysis process that is performed (almost) by default for each design, has a well-defined series of input parameters and modelling elements, and requires little human creativity. Such an analysis usually focuses on a single aspect of the physical behaviour. Specialisation obviously makes it easier to achieve integration and implement a system with richer semantics. This enables the MRA to address the “*information-intensive nature of CAD-CAE integration*”, which is the apt characterisation made by the authors.

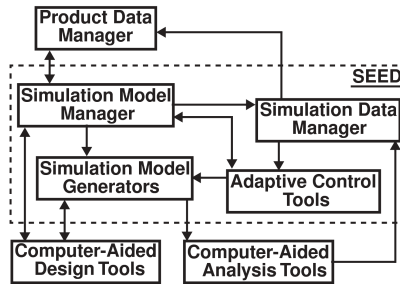


**Figure 7.2:** The multi-representation architecture. Image source: [Peak et al., 1998].

The MRA consists of four components: 1) solution method model (SMM), 2) analysis building block (ABB), 3) product model (PM), and 4) product model-based analysis model (PBAM). There are inter-representation mappings between the SMM and the ABB, and between the ABB and the PM. The PBAM helps to bridge the gap between the ABBs and the PM. See Figure 7.2 for a schema of the architecture.

The PM effectively represents the design model. The design-oriented attributes are related to idealisation attributes to support the information needs of potentially many analysis models. The analysis model, as we characterised it in Chapter 2, is here created by a particular PBAM, which mainly represents a particular physical model, through the combination of ABBs and creating fitting idealisations by means of the design-analysis associativity and the rules of the PBAM. The ABBs are elementary analysis primitives, such as a spring. They serve as product-independent elements of analysis models that can be instantiated and combined. Analogous to the design feature concept, we might consider them a kind of *analysis features*. The analysis model is thus an assembly of several ABBs as created by a PBAM. It can be converted into a solution method model, which can be a finite element mesh together with all the other inputs that an external, specialised analysis program might need. Identifiers are maintained along the path of mapping one representation to the next, such that the analysis results can be related back all the way to the design model, e.g. to identify in the design model for which components the maximal stress was exceeded.

The results demonstrate that within a specialised domain and with the modelling space sufficiently constrained, integration of design and analysis



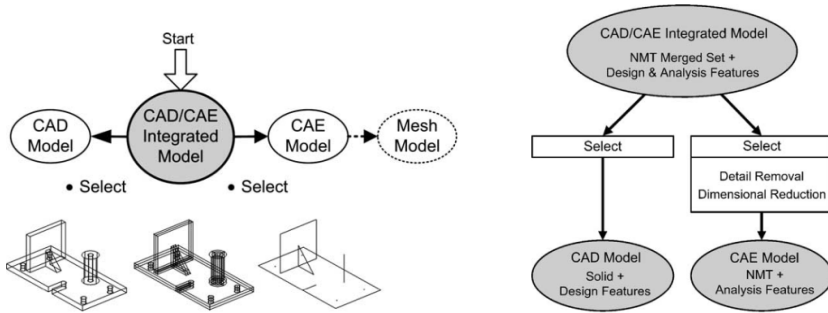
**Figure 7.3:** The Simulation Environment for Engineering Design (SEED) within the product design cycle. Image source: [Shephard et al., 2004].

models can be achieved to a reasonable degree.

### **Simulation Environment for Engineering Design**

In *Toward simulation-based design* [Shephard et al., 2004], the *Simulation Environment for Engineering Design* (SEED) is described, and how this integrates with CAD and product data management tools. This approach is quite generic, as it is not tied to particular types of analysis, modelling techniques or tools. There are four functional components of SEED: 1) simulation model manager, 2) simulation data manager, 3) adaptive control tools, and 4) simulation model generators. Figure 7.3 shows how the components of SEED interact with each other and with the other tools outside SEED.

The simulation model generators are responsible for the construction of the analysis model. They do this based on the input of the adaptive control tools, the simulation model manager, and the CAD system. The simulation model manager contains a high-level functional view of the design to support analysis of specific subsets of components. In conjunction with the CAD system and the adaptive control tools, which decide on the need for idealisation of geometry, a complete geometric representation for analysis can be generated. The simulation model manager also contains the topology associated with the geometric model. How the system deals with the creation and/or maintenance of this topology, in particular when subsets of components are analysed, seems to be an unresolved issue. The simulation model generators are also responsible for creating the mesh of the analysis model, based on cues from the adaptive control tools and the topology. The simulation data manager is responsible for the effective utilisation of simulation information within the design process. It does this



**Figure 7.4:** Full CAD/CAE integration by means of a single master model. Image source: [Lee, 2005].

by providing access to the analysis results, and communicating this to the user as a response that is effective and fitting in the context of the analysis problem.

The authors state that “[...] *the historic mistake made by both CAD and CAE developers has been to take the simplest view of this integration as a direct data transfer process.*” The objective of SEED is to bring the design model and the knowledge involved in the analysis process closer together. With SEED, the traditional way of supplying the input to the CAE tools is effectively replaced by a layer of closely cooperating tools between the CAD system and the CAE tools that generate the desired inputs for the latter. The interacting components contribute and deal with knowledge important to the analysis on the one hand, and knowledge of the design model on the other hand. In the interaction of these components, the design and analysis models are brought closer together. Although this generic approach results in a high-level of automation, the models are not actually linked. For instance, after a change to the design model, the process of preparing the analysis model has to be executed again.

### ***A Single Master Model***

Lee [2005] describes in *A CAD-CAE integration approach using feature-based multi-resolution and multi-abstraction modelling techniques* a feature-based system in which both CAD and CAE models reside within a single master model. This is referred to as *full integration of CAD and CAE models*; see Figure 7.4.

The master model is stored as a non-manifold topology (NMT). A design starts with the Boolean combination of form features. For each feature, an

idealisation feature is automatically added to the model. The geometry of all these features is stored together in the master model. When an analysis needs to be performed, the analysis model can be automatically extracted from the master model by the process of idealisation. This consists, firstly, of choosing a level of detail (LOD), by default eliminating features on the basis of the size of their volume. Secondly, for the resulting model the level of abstraction (LOA) is selected. The criteria for determining the LOA depend on the type of analysis. Due to the history-based modelling approach, to make the proper assembly of a subset of dimensionally reduced geometries possible, the features need to be rearranged, which is accomplished by the identification of the *effective zones* of the features.

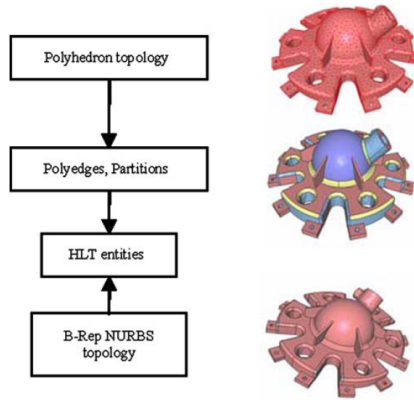
It is stated that in the master model the geometries of all features of all abstraction levels are stored in the *merged set*, a structure that seems similar to the cellular model discussed in Section 5.2. It is, however, unclear to what extent all geometric entities of the features at different levels of abstraction reside in the same data structure. Obviously, when working with multiple geometric representations for a single feature, care has to be taken that efficiency remains acceptable, even for more complex models.

The underlying assumption of this approach is that an analysis model can always be formed by the individual idealisations of a subset of the design features. This will not always hold, since, for example, the geometry of some analysis models is based on global shape characteristics such as symmetry. The author acknowledges that the range and flexibility in dealing with the analysis model could be improved. Although the proposal is conceptually powerful, due to its very tight integration, it might not be generic enough to scale to complex and realistic cases.

### ***A Mixed Shape Representation***

Hamri et al. [2008] and Drieux et al. [2007], the authors of *Interfacing product views through a mixed shape representation (Part 1 and 2)*, want to improve the interface between *product views* in general, by introducing a representation that can bring different data representations, basically CAD geometry and faceted geometry, closer together. They call this a *mixed shape representation*, and one of its applications can, obviously, be to bring the design and the analysis model closer together. The mixed shape representation mainly consists of a BRep NURBS topology and a faceted model that is organised by means of *Polyedges* and *Partitions*, and a *High Level Topology* (HLT) that links the CAD geometry and the polyedges and partitions. See Figure 7.5 for an illustration. The faceted model is a more natural data structure for the preparation of an analysis model/mesh, and by means of the HLT this





**Figure 7.5:** Mixed shape representation. Image source: [Hamri et al., 2008].

representation also has information from the design model at its disposal, making certain operations for the preparation of an analysis model/mesh easier to automate.

The model or *product view* (PV) that is actively being modified is denoted the *master* representation, whereas the other is the *slave*. The manipulation of the mixed shape representation works by the application of operators, which modify this representation from the point of view of either the CAD or the faceted representation in a structured way, and can have semantics associated with them. The link between the two representations makes it easier to apply reasoning based on geometric algorithms, e.g. feature size detection, and relate the faceted geometry to the NURBS model, and vice versa. Operations in one view can thus be aided by the representation of the other view. Although the link by means of the HLT is dynamically maintained under the operations that the authors define, in particular operations that prepare the geometry for analysis, it seems that the structure is not maintained throughout the modelling process, as not all operations that can be applied to a master representation can correspondingly be applied to the slave representation. The authors also acknowledge that linking dimensionally reduced models has not yet been dealt with.

## Conclusions

The five approaches that we have reviewed all contribute to the spectrum of solutions for integrating design and analysis models, and through valuable observations give us insight into what can and what needs to be done.

One of the principal tasks in bringing design and analysis models closer together, is to bridge the gap between their different geometries. This gap consists of different levels of abstraction and detail, distinct underlying representations, and requirements that the analysis model has to satisfy to make it a suitable model to be meshed. In many operations that are commonly performed in the process of turning a design model into an analysis model, knowledge from both models is helpful. Performing a straightforward conversion from the one representation to the other, means losing the benefits that information from the other representation has to offer. At least throughout the process of deriving an analysis model from a design model, there should be a link that relates the two models to each other.

Creating, and in particular maintaining such a link, has not been solved in a general fashion, and the lack of knowledge of the other model stands in the way of automation. In general, incorporation of knowledge is essential for effective automation, as the modelling system needs to replace or assist human actions that require some degree of 'understanding' of the analyses, model types and common problems. Basically, anything that decreases the number of tasks and input choices for the engineer can be considered knowledge, e.g. the software knowing the meshing requirements for a specific analysis context, such as the need for a boundary layer mesh. Other examples of knowledge incorporation would be features that help to steer the abstraction process of their geometry, and coupling a faceted representation to a traditional BRep geometry, such as in [Hamri et al., 2008; Drieux et al., 2007], which opens up new ways to reason with the analysis model. Obviously, it is easier to incorporate knowledge for specialised domains or tasks, as it is easier to identify common tasks and to enumerate special cases.

Although automation is improving, there is no true linking of models in the sense that they are dynamically maintained under modification of the design. This means that, in particular with many, quick iterations of the design cycle, some steps will have to be repeated again and again. Also, making modifications to the analysis model, as might happen, for example, during design optimisation, and propagating those changes back to the design model, seems infeasible to automate with the proposed approaches.

In essence, we need to integrate two separate specialisations, design and analysis, which share a product model. Such integration has already been realised with reasonable success for several tasks other than analysis within the product development cycle. Complete integration of all technologies and processes involved in product development, is called *Product Lifecycle Management* (PLM) [Stark, 2005]. An important requirement for PLM is that a designer working in any product development phase can work

with information that is relevant for that phase, without being diverted by information that is relevant only for other phases. However, the information for all development phases should be integrated, so that no inconsistencies arise. We believe that a promising way to support this is *multiple-view feature modelling*. After having discussed, in this section, other approaches to integrating design and analysis models, we now look at how various views on a product have been integrated in this approach.

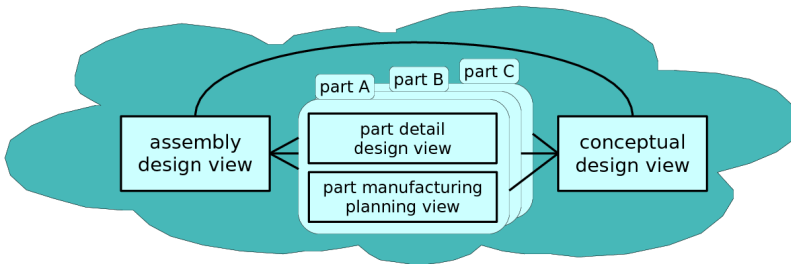
## 7.2 Multiple-view feature modelling

Current commercial feature modelling systems do not adequately support PLM. Some systems have specific models for different development phases, but these models are not very well integrated. Other systems do integrate information for, for example, assembly design and part detail design, but these systems typically do not have specific models for these development phases.

*Multiple-view feature modelling* can do better here, by providing a separate view on a product for each development phase, and integrating all views. Each view contains a feature model of the product specific for the corresponding phase. Since the feature models of all views represent the same product, they have to be kept *consistent*.

Quite a lot of research has been done on multiple-view feature modelling [de Kraker et al., 1997; Martino et al., 1998; Hoffmann and Joan-Arinyo, 1998]. These approaches all work on a single part of a product. A new view on a part can be derived from an existing view by feature conversion, the process of converting one feature model into another feature model. Several views on a part can be simultaneously maintained. Modifications made in one view, are automatically propagated to the other views on the part, also by feature conversion, although not all approaches propagate modifications in both directions. The approaches can be used to, for example, support design for manufacturing: while a part is being built with design features, these features can be immediately converted into features in a manufacturing planning view, which can be used to concurrently check the manufacturability of the product. Multiple-view feature modelling bears a certain conceptual resemblance to the *model-view-controller* paradigm from the field of software engineering [Buschmann et al., 1996]; within a single modelling system, there can be several views that each offers a different presentation of what is essentially a single underlying model.

To even better support the concept of PLM, an approach to multiple-view feature modelling has been developed in our group, and implemented in the SPIFF prototype feature modelling system, that supports more product



**Figure 7.6:** Multiple-view feature modelling as currently implemented

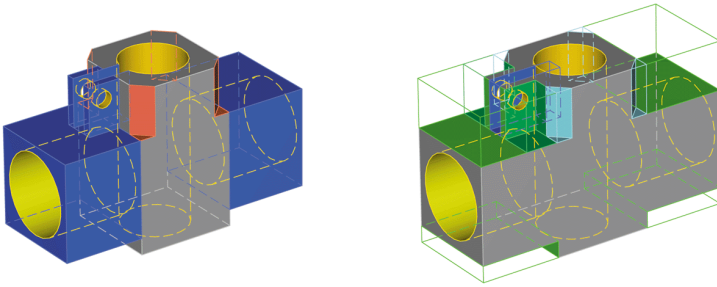
development phases, in particular conceptual design, assembly design, part detail design, and part manufacturing planning [Bronsvort and Noort, 2004]; this is illustrated in Figure 7.6.

In the *conceptual design view*, the product configuration can be modelled with conceptual components, which are to be implemented by one or more parts, and interfaces between these conceptual components, which are to be implemented by a connection. Components are built from a base shape, concepts, such as depressions and protrusions, and reference elements. Interfaces between components are characterised by means of degrees of freedom between the components. The complete geometry of the components does not have to be specified: for example, for some concept only certain properties, such as its maximum volume, might be specified.

In the *assembly design view*, the connections between the components in the product can be modelled with assembly features, in particular *connection features* [van Holland and Bronsvort, 2000]. A connection feature needs to be created for each interface in the conceptual design view, and linked to that interface. The interface and the connection feature should reduce the same freedom. In order to accommodate the connection feature, features may have to be created on the components in the assembly design view, e.g. a pin feature and a hole feature for a pin-hole connection feature.

In the *part detail design view*, the detail shape of the parts can be modelled, typically with features such as through holes and protrusions. It allows the designer to refine the parts that are represented by the components in the conceptual design view, and which may have been refined in the assembly design view to accommodate connection features. Features may be created for concepts in the conceptual design view, and linked to those concepts. A feature should satisfy the requirements specified for the corresponding concept, e.g. that the volume should be less than  $80 \text{ cm}^3$ .

In the *part manufacturing planning view*, the way each part is to be man-



**Figure 7.7:** Part detail design view (left) and part manufacturing planning view (right) on a part. Image source: [Bronsvoort and Noort, 2004].

ufactured is determined. Manufacturing planning features are similar to detail design features, but note that the set of features for the manufacturing planning view on a particular part can be quite different from the set of features that has been used to design the part, simply because a product can look different from the points of view of design and manufacturing planning. This view allows the designer to analyse the parts for manufacturability and to create a manufacturing plan for them. The feature model in a manufacturing planning view is automatically linked to the feature model in the corresponding part detail design view. See Figure 7.7 for the part detail design view and the part manufacturing planning view on a part.

*Consistency maintenance* integrates all views on a product, by ensuring that their feature models remain consistent. It checks the consistency of pairs of feature models, based on consistency definitions specified for these pairs. If an inconsistency is found, it recovers the consistency of the models. This involves, among other things, constraint checking and feature conversion in both directions [Bronsvoort and Noort, 2004].

In the multiple-view feature modelling approach described above, there are no analysis views yet, although analysis is an integral part of PLM. We therefore here propose to add analysis views to the multiple-view feature modelling approach, which leads to the integration of analysis with design, and other product development tasks, in a generic way.

### 7.3 Analysis views

An analysis view in the multiple-view feature modelling approach should be a view on the product that is suitable for an engineer to perform analysis with. The view does not stand by itself, but is an integral part of the product model, and is maintained as such. The engineer, however, should be able to control

the analysis as flexibly as in a completely separate analysis environment.

Putting the analysis model on equal footing with the design model is different from the approach of integration that considers analysis as a derived activity using a model with similar geometry. Making the analysis model an integral part of the product model, automatically entails the realisation of enduring consistency. This should ultimately be the most efficient, as every operation that is applied to a particular view and has an effect on other views as well, such as modifying shape dimensions, can be propagated automatically to the other views where the operation has effect. The principal idea is that no information that essentially appears identically in both models, should be entered more than once. Consistency has to be maintained just as well when employing several separate environments, only the burden then lies with humans, and mistakes are therefore much more likely.

A product can have multiple analysis contexts, and within each context several views. A particular analysis view presents a physical model of the product that is best suited for that analysis context, e.g. joint fatigue, injection mould flow, or stress analysis. The view allows interaction with the analysis model, assignment of boundary values and boundary conditions, and control over miscellaneous parameters, such as those within the physical model, for mesh generation, and for control over the solution methods.

Having specialised analysis views for different analysis contexts based on the underlying physical models, allows for specialisation. This helps to bring the knowledge from specific analysis domains into the product model space, and allows for reasoning with that knowledge to support the creation and maintenance of analysis views and associated meshes in a context-sensitive manner. In the work discussed in Section 7.1, it is nicely illustrated how knowledge in a specific context can help to streamline the automation.

Since analyses are performed at various stages of the design, also when the detailed geometry is not yet complete, it should be possible to associate a range of analysis views, at different levels of (dimensional) abstraction, to a product model. Each view in turn can have multiple simulation runs associated with it, with possibly different meshes.

In Figure 7.8 a schematic overview is given of how multiple analysis views fit in our multiple-view feature modelling approach. There are several analysis contexts that represent different physical problems. Within each context, multiple analysis views can be created and maintained. Each view essentially represents a single analysis model and can have its own particular geometry. An analysis view has a mesh associated to it, and possibly the results of earlier analyses, including the mesh that was used in the analysis, and all other relevant parameters. An analysis view can relate to a single

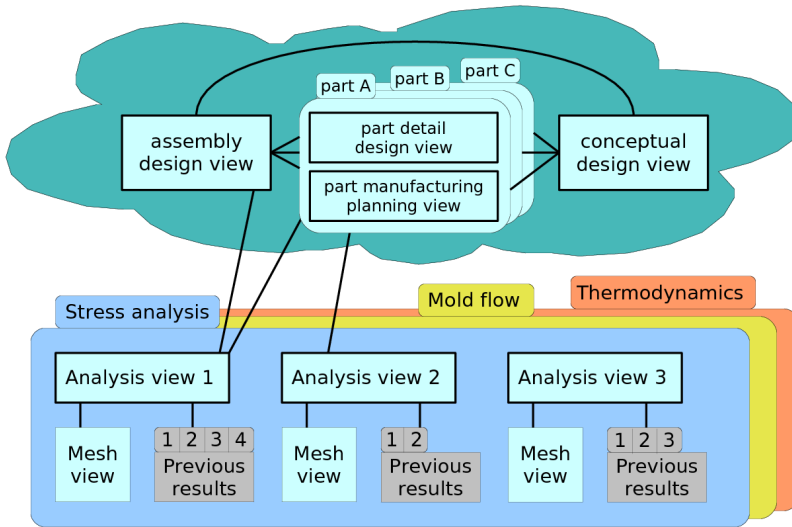


Figure 7.8: Multiple-view feature model with the addition of analysis views

part or an assembly of parts.

A central characteristic of the multiple-view feature modelling paradigm is multi-directional propagation of modifications. With multiple views available, modifications of the model should be made *within the view* that prompts for these modifications, as that view offers the most natural interface. Not every modification in a particular view has effects on the other views, but if it does, then the modelling system should either know how the effect should be propagated to the other views, or let the engineer assist the propagation at an appropriate time. This should also hold for the analysis view. When the engineer, based on the analysis result, concludes that the geometry of the model should be modified, he should be able to do this through the analysis view. In particular in the case of analysis for geometry optimisation, where the result is a shape that is optimal in certain respects, it should be possible to propagate the result to the design view and other views.

Integrating design and analysis models by means of the multiple-view feature modelling paradigm, will maximally exploit the similarities and relations between the models. However, since the geometry of the design and analysis models can have notable differences, we are confronted with a major challenge to keep the models synchronized in an automatic way.

From the proposals discussed in Section 7.1 to improve the integration of design and analysis, we can basically distil two approaches to bridge the

gap between their two models: 1) convert the design model to an analysis model on the basis of individual features [Lee, 2005; Peak et al., 1998], and 2) establish a link based on more low-level geometric elements, and support the creation of the analysis model by operations along this path [Hamri et al., 2008; Drieux et al., 2007]. The first approach can work in specialised cases, but is not general enough to handle every type of geometry abstraction. In particular combining the geometry of multiple features into a single unit of abstraction is not supported, nor are analysis models that exploit symmetry in the geometry of the model, and thus can be based on the partial geometry of some features. The second approach is more generic, but is harder to automate completely. More manual control and insight from the engineer are required, as he often has to select and fine-tune the results of multiple, geometric algorithms. Many of these issues can be mitigated, or possibly even avoided, by establishing a relation between the geometry of the design model and the geometry of the analysis model that is being prepared. With this relation in place, the process of deriving the analysis model can be supported by reasoning that builds on either representation.

None of the approaches discussed in Section 7.1 are, however, well suited for actively maintaining the analysis model in combination with the design model. For instance, when the analysis model has already been prepared, it is generally not possible to make changes to the design model, and then have the analysis model updated accordingly, without the need to derive and prepare it yet again. Nor is it possible to make changes to the geometry of the analysis view, and have these automatically propagated to the design view.

The analysis view aims to combine the good elements of the known approaches, and overcome the shortcomings. Obviously, we cannot cover the complete scope of issues that the known approaches address, so we focus in particular on the functionality that seems to be generally lacking, which is maintenance of both models by means of propagation of changes.

To achieve this, it seems a prerequisite that the analysis view is a feature model, with parametrised features, as is the case in our approach. The individual features in the design view should support the abstraction process where possible by containing options for abstraction, and possibly rules that help to automatically choose between multiple options, or handle special cases of interaction with other features; to support complex abstractions, the relation between the features of the design view and the analysis view will not be one-to-one in all cases. Applying boundary values and conditions to the model should happen by means of analysis features, which demarcate areas, line segments or points within the analysis model. In addition to the parametrised representation, a more low-level geometry, such as a faceted



representation of the model, could be associated to the analysis model and ease geometric reasoning, e.g. to determine local feature size throughout the model.

The two models consisting of features can be linked by means of operations, similar to the concept of operations for the transformation of models as used in [Hamri et al., 2008; Drieux et al., 2007], together with constraints. The use of constraints is vital for achieving the level of automation that we are aiming for. It is therefore a natural requirement that it is specified how the design and analysis views relate to each other in terms of constraints on their features. The constraints implicitly encode how the models can be varied, and thus enable propagation of changes with relative ease. Important research is how the relation of the constraints between the design and the analysis view should be managed, considering also that not all constraints that exist between features within a particular view, are valid in the other view due to the lack of a one-to-one relation between features in the two views. For instance, a constraint might refer to a feature that exists in only one of the views.

Input and maintenance of the models should, obviously, not take more effort than what is common in deriving the analysis model. Views should certainly be kept synchronised, but to prevent too frequent checking, in particular amidst modifications, synchronisation may have to be delayed to suitable moments.

## 7.4 Conclusions

Attempts to improve the integration of design and analysis models already span multiple decades. The models differ not only in representation, but also in shape. Most efforts focus on automating the process to derive the analysis model from the design model. Drawbacks of this unidirectional approach are that the steps have to be repeated each time the design model has changed, and that changes to the geometry of the analysis model cannot be propagated back to the design model. These drawbacks can be addressed by extending the multiple-view feature modelling concept to the realm of analysis.

Integrating design and analysis models through the addition of an analysis view to a multiple-view feature modelling system, implies a more equivalent treatment of the two models: both models are maintained simultaneously and kept consistent. The requirement for consistency can be implemented by linking the views in such a way that either view can be modified and that changes are propagated to the other.

Incorporation of knowledge of the analysis process and the requirements

for the analysis model is essential for improving integration of analysis with the rest of the product development cycle. Depending on, for instance, the analysis context and the level of abstraction, different requirements hold. Product development systems should 'understand' when operations should be executed automatically, or else from which limited set of operations the engineer should be offered a choice. Integrating design and analysis models by means of a multiple-view feature modelling approach, in which both models are maintained concurrently as views on a single product, provides a good way to incorporate knowledge on the transition between the two models.

In the next chapter, we elaborate the idea of incorporating analysis views in a multiple-view feature modelling approach. In particular, we look at the issue of actually linking a design model and an abstracted analysis model, as is required for the realisation of analysis views. We explore an approach that links features in the design model to corresponding abstracted features in the analysis model. By reinterpreting the constraints that define the design model, we obtain an analysis model that functions as an autonomous feature model. Although our approach is elementary, this is a first step towards bidirectional propagation of changes between the two models.

## Chapter 8

# Relating an analysis view to a design view

In the previous chapter, we have discussed analysis views. These offer a specific interface for dealing with an analysis model that is linked to a design model. This concept of linking views is not new, as it is a proven concept from multiple-view feature modelling [Bronsvoort and Noort, 2004]. For analysis views, however, there is the challenge that the geometry of the analysis model and the design model can be different.

In this chapter, we discuss the issues involved in establishing a link between a design model and an abstracted analysis view, and explore an approach to realise such a link. As we have described in Chapter 7, an analysis view can have a distinct set of features addressing analysis-specific tasks, such as imposing boundary conditions or steering mesh characteristics. However, in the context of establishing a link between the two views, our principal concern is the geometry of the features in the design view and the corresponding features in the analysis view.

The analysis model represents an abstraction of the design model. The abstraction can consist of leaving out features that have no apparent effect on the outcome of the analysis. This process is known as *defeaturing* or *model simplification*. The features that are left out in the analysis model are usually small relative to the aspects of interest for the analysis, and they will probably complicate the analysis process. Another form of abstraction is *dimensional reduction*. This would in most cases be used in conjunction with defeaturing, and results in further abstraction. Dimensional reduction commonly applies if some model dimensions, such as width or depth, are small relative to other dimension(s) in the model. These small dimensions are reduced to 0 in the abstraction, which means that their geometry is

represented by a surface or a curve, instead of a 3D volume.

There are several techniques to automatically perform defeaturing and dimensional abstraction, such as described by Lee [2005] and Sinha and Suresh [2005]. However, in spite of the presentation of many promising techniques, the impression remains that the challenge of doing this fully automatically and enabling a dynamic link between the models, is still far from being adequately solved, in particular for more complex models.

But why do these techniques have trouble to meet our needs? A likely factor is that they are mostly geometry-oriented, in particular the methods for dimensional reduction. Instead of basing the construction on the information contained in the feature model, the geometry as a whole is analysed, and then a global abstracted shape is derived, commonly based on the medial surface, medial axis, or a similar skeletal structure. In this process, however, the connection of the analysis geometry with the features, the corresponding semantics, and the constraints that define the design model, is lost.

As we know from Chapter 7, it is our premise that such a link is essential for a proper integration of design and analysis models. Without this link, it would be virtually impossible to allow making changes to the analysis model and having them propagated back to the design model, as the concept of an analysis view requires. Given the difficulties with automatic abstraction without aiming for propagation of changes between the models, we expect the establishment of such a link to be a substantial challenge.

Our approach links the two models on the basis of individual features: for each feature in the analysis model, there is a corresponding feature in the design model. The same constraints that determine how the feature is dimensioned and positioned in the design model, are used for the analysis model as well, except that some adjustments are needed to compensate for the change in dimension of some of the features. These adjustments arise from a reinterpretation of the constraints for connecting the dimensionally abstracted features. The presentation of these adjusted feature parameters in the analysis view to the user is an issue in itself, as the adjustments cause the constraints of the analysis model to be different from those that define the design model. In our discussion on linking the models, we briefly outline three ways of dealing with this.

We start out in Section 8.1 with a general description of our prototype analysis view. Next, in Section 8.2, we discuss the procedure for abstraction that has been used to derive the analysis model from the design model. It implicitly defines how the two models are related. This is followed by a presentation of the algorithm for the actual linking of the two models in Section 8.3, together with a discussion of how the user can interact with an

analysis model, whereas the design model is automatically updated. This includes a discussion of the issue of presenting the adjusted parameters to the user. After the presentation of our approach, we discuss the limitations and the issues that remain to be solved in Section 8.4. Finally, the chapter is concluded in Section 8.5.

## 8.1 A prototype for an analysis view

There are many aspects to an analysis view and its link with the design view, as we have seen in Chapter 7 and the introduction of this chapter.

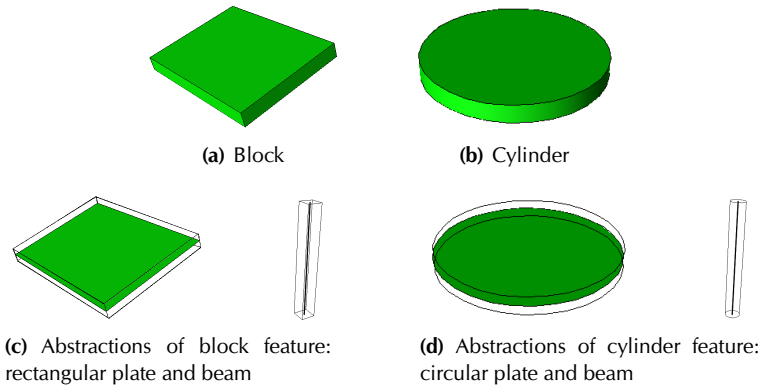
To better understand what issues we are exactly dealing with, we start by listing the principal requirements for the prototype analysis view that we take into account:

1. Support dimensional abstraction.
2. Function as a constraint-based feature model.
3. Support bidirectional propagation of model changes from and back to the design view.

It is common to perform analysis with an abstracted model, i.e. a model that has a geometry that is more suitable for a timely analysis, but gives functionally equivalent results. Features that are small and are deemed to have no discernible impact on the outcome of the analysis, should be excluded from the analysis model, as their presence can complicate the mesh generation and have a negative effect on the reliability. Also, the model can be dimensionally abstracted by substituting (part of) the 3D geometry by 2D or 1D representations. For example, a block with one dimension much smaller than the other two can be replaced for the analysis by a flat, 2D rectangular plate. Effectively, the dimensions that are too small to have an impact on the outcome of the analysis, are excluded. Hence the requirement that the analysis view should support dimensionally abstracted features.

The second requirement ensures that the model can be adapted easily, while maintaining the global relations between all features that have been specified by means of constraints. The analysis view should be parametrised in a way that is compatible with the design view: it should be impossible to make changes to the analysis view that accidentally violate the constraints and semantics of the design view. This allows for geometric optimization, where models can be analysed for a range of parameters, which is dynamically steered by the analysis process.

Propagation of changes between the views is quintessential to the concept of the analysis view, hence the third requirement.



**Figure 8.1:** Examples of the features in the analysis view

These requirements obviously have consequences for how the analysis view is composed. To enable modifications to the model in high-level terms, it is necessary that the analysis view consists of parametrised features, and constraints that describe how the features relate. The geometry of the features can be a block, a cylinder, or an abstraction thereof, i.e. a rectangular plate, a circular plate, or a beam. See Figure 8.1 for examples of the supported features. Features can use the geometry either in an additive or in a subtractive manner, i.e. features such as holes and slots are supported as well.

The features are all parametrised, i.e. each feature can represent a range of continuously varying shapes. The parameters to these features can be specified through a direct, numerical value, but also by means of a mathematical formula, or constraints, which can include references to other parameters that belong to other features or are defined by the designer. For example, a user-defined parameter *base-height* could be defined, and then a feature could have its parameter for *width* set to twice the base-height plus the width of some other feature. Such user-defined parameter are called *model parameters*.

Table 8.1 list all the base shapes that are used by the features in the analysis view, and their named entities that can be referred to by constraints. A shape can be used by multiple features that differently define whether the feature is additive or subtractive, and offer different other semantics that affects how the feature is constrained with respect to other features in the model.

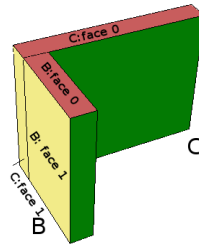
The constraints in the analysis view are attach and position constraints. A feature can be added to the model by using at least one attach constraint,

**Table 8.1:** Geometric shapes used for the analysis features, their parameters, and their geometric entities that can be referred to, e.g. in constraints

feature	parameters	faces	edges	vertices
block	width, height, length	left, right, bottom, top, back, front		
cylinder	radius, height	bottom, top	axis	
rectangular plate	width, length	bottom, top	left, right, back, front	
circular plate	radius	bottom, top		centre
block-beam	length		axis	bottom, top
cylinder-beam	length		axis	bottom, top

which requires two faces, one of the model and one of the feature, to be coplanar (parallel at distance 0), and a number of position constraints, which each requires a face of the model and a face (or axis, in case of a cylinder) of the positioned feature to be parallel at distance  $d$ . See Figure 8.2, where we see a feature  $B$  that has been attached to feature  $C$  and positioned based on two pairs of faces, namely the pair in yellow, and the pair in red. The constraint on parallelism of the attach faces can be satisfied with either identical or opposing normals of the two faces, so this has to be specified as part of the constraint. The opposing normals are most common in attachment constraints for features that both add material. Within the plane of attachment, the features are relatively positioned by specifying faces, or reference planes, one from the feature that is added and one from the rest of the model, which have to be parallel with a distance  $d$ . In the example of Figure 8.2, the distance is 0 for both position constraints.

For the abstracted features, the constraints lean on knowledge of the original, full-dimensional features. The analysis view thus cannot be considered completely separate from the design view. In the design view, both attachment and positioning are based on parallel faces, or a line/axis parallel to a face, but constraints referring to abstracted features can also be based on edges and vertices that correspond to a face of the original, full-dimensional feature. For example, when a block is abstracted to a 2D rectangular plate, it has lost four of its faces in the process. The edges on the boundary of the plate each correspond to a face of the original feature from the design



**Figure 8.2:** Attaching two features by means of constraints

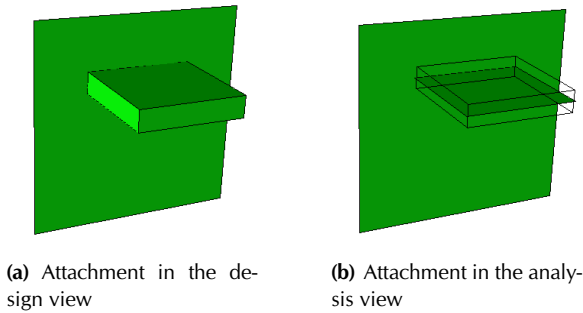
view. The meaning of a constraint that refers to an edge or a vertex that corresponds to a face that has been dimensionally reduced, depends on the orientation of the original face. Basically, the orientation of the abstracted feature must be kept consistent with the orientation of the original, full-dimensional feature. For the features that we have in our analysis view, this means that the position of the abstracted feature must coincide with that of the mid-surface, or midline, of the original feature.

Consider, for example, a block feature, which has faces that are all at a  $90^\circ$  angle with their adjacent faces, and one dimension that is relatively small to the other two. This feature can be abstracted to a rectangular plate. If there is a constraint that requires an edge of the plate feature to be coplanar with the face from another feature, then positioning the feature such that the edge lies in the face is not enough to satisfy the constraint. The plate feature is known to be an abstraction of a block, and thus it must be positioned such that it coincides with the mid-surface of the original block feature. For this example, this is equivalent to requiring that the single rectangular face that represents the plate feature is perpendicular to the face of the other feature. This is illustrated in Figure 8.3.

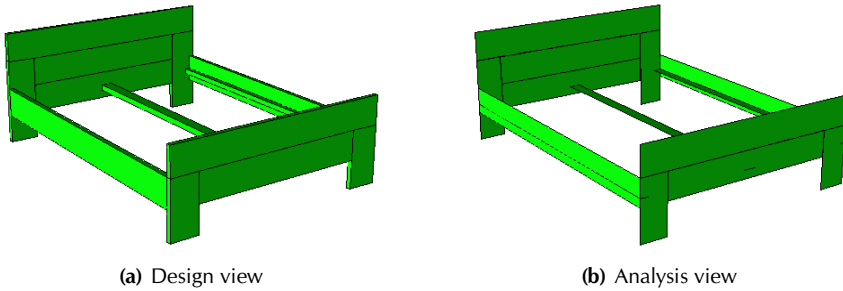
All the attach and position constraints in the analysis view have a corresponding constraint in the design view. The constraints in the analysis view function like a proxy to their original constraint, as the latter contains more information, which in some cases is required when dealing with the analysis view. For example, see Table 8.1, where the block-beam can be seen to have just a single *axis* edge to refer to, whereas the corresponding feature in the design view has four sides. The orientation of that feature is thus specified in the design view, and from here it is referenced whenever this information is relevant for an operation in the analysis view.

An example of the model geometry shown in an analysis view, in comparison to the corresponding design view, is shown in Figure 8.4. The model is parametrised with the thickness and the height of the wood boards, the





**Figure 8.3:** Meaning of the attachment constraint referring to the edge of an abstracted feature



**Figure 8.4:** An example of linked design and analysis views

height of the bed, and the length of the bed. The thickness of the wood boards has been abstracted in the analysis view, such that all the wood boards are 2D features. Manipulation of the analysis view is limited to changing the feature and model parameters. Adding features to the analysis view that have a full-dimensional equivalent in the design view, has not been looked at.

Any analysis view that fulfils the earlier listed requirements must have an automated method for generating an acceptable (abstracted) analysis view from a design view, or else we would not be able to propagate changes between the views. We discuss our method for this in the next section. A more general discussion on automatic derivation of analysis models follows in Section 8.4.

## 8.2 Procedure for automated abstraction

For our analysis view, we use an automated abstraction procedure that defines how the geometry of the abstracted analysis view and that of the design view are related. It is a simple procedure that works for the set of features and constraints in our analysis view, i.e. blocks, cylinders, plates, and beams, related by attach and position constraints.

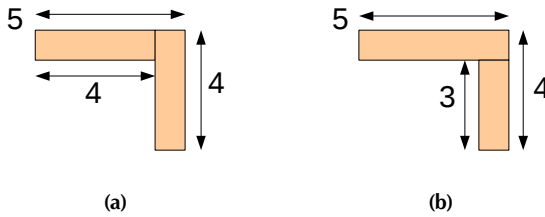
The essence of the approach is that the geometry of individual features from the design view is abstracted. We assume that there is a method for deciding which features are to be abstracted to which level, and which features can be completely ignored in the creation of the analysis model. This process is usually based on the characteristic size of the features in relation to the mesh size that is aimed for, and the value of shape parameters such as width, length and height relatively to each other. Knowledge on the specific function of the features is also important, and obviously the engineer can have a say in this as well.

Features can be abstracted to either 2D or 1D form. In the first case, the abstracted geometry is the mid-surface of the feature. However, we want the abstraction to depend on the geometry of the model as a whole, and not on the particular way the features are parametrised and constrained with respect to each other. Figures 8.5 and 8.6 illustrate this. The two models in Figure 8.5 have the same geometry, but are composed in a slightly different manner. When the features are abstracted by taking the geometry of the mid-surface of the features, while adhering to the original constraints that combine them, we end up with two different results (Figure 8.6(a) and 8.6(b), respectively). Regardless of the composition that is chosen for the model, we would like the resulting abstracted model to be the same, i.e. we prefer the solution in Figure 8.6(c). An approach to achieve this would avoid many of the inconsistencies that can be encountered when abstracting a model by effectively thinning the individual features, i.e. changing the parametrisation such that one of the shape dimensions becomes 0, while honouring the original constraints without adjustments.

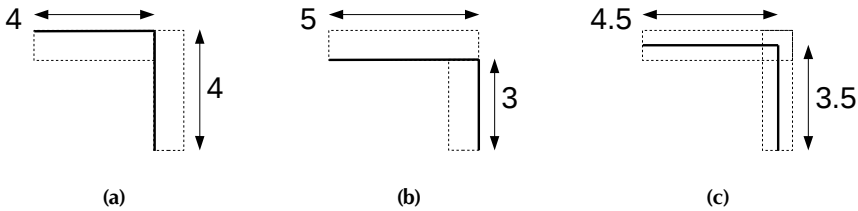
Looking at the desired solution in Figure 8.6(c), we notice two things concerning the abstracted features:

1. They are positioned such that they overlap with the position of the mid-surface from which their geometry is derived.
2. Their shape parameters are adjusted such that the constraints between them are satisfied.

The first issue is reminiscent of the relation between a geometric object and its medial axis. This is not surprising, as the medial axis, or medial



**Figure 8.5:** Two different models with an identical boundary

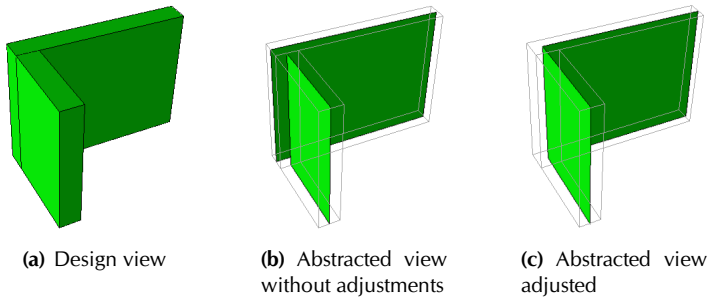


**Figure 8.6:** Approaches to abstraction of the models from Figure 8.5

surface, often plays a role in automated methods of abstraction. These methods do not look at individual features, but the geometry of the model as a whole. By pinning each abstracted feature down such that its geometry coincides with the position of the mid-surface that it was derived from, we maintain global coherence and consistency between the features.

So abstracting the individual features and pinning them to a certain position does not by itself yield an acceptable analysis model. See Figure 8.7 for a 3D example. Here, the attachment of two features is shown. In Figure 8.7(b) we see the result with the two individually abstracted features, with a clear gap between them. In Figure 8.7(c) the most obvious solution is shown. It can be achieved by taking the model in Figure 8.7(b) as a starting point, and adapting the geometry of the features based on the constraints used to attach the two features. Consequently, we can only make adjustments when constraints have actually been specified. If the blocks in Figure 8.7(a) would not be attached, but just happened to touch each other due to how their dimensions were specified, then we would not adjust the dimensions of the abstracted features such that they properly meet. We do not guess how the design was intended, but only act on constraints that are explicitly present in the design view.

The adjustments to the shape of the features can be either positive, i.e. an increase in length, or negative, i.e. a decrease in length. In Figure 8.7, for ex-



**Figure 8.7:** Attachment of abstracted features

ample, both these types of adjustment occur. The feature in front is extended to bridge the gap with the other feature, whereas the side of the other feature is shortened a bit, such that we retain a proper corner as intended by the constraints. For our analysis view, it is a rather straightforward procedure to connect the abstracted features, corresponding to the specified constraints, by adjusting the dimensions of the features. With more complex features and ways of connecting them, we may need additional strategies. More on this in Section 8.5.

We recall that the attach and position constraints in the analysis view are essentially a single type of constraint on parallelism.

An abstracted feature can be regarded as the limiting geometry resulting from changing the shape parameters of the corresponding design feature such that a certain shape dimension becomes 0. In this process, certain faces can be reduced to edges, and edges can be reduced to vertices. We thus have a correspondence of certain edges in the abstracted feature to certain faces in the original, full-dimensional feature. In a sense, the abstracted features have ‘lost’ the latter entities in the process of abstraction, which is why we refer to them as *non-persistent* entities. The geometric entities for which a dimensional correspondence remains between the abstracted and the original feature, i.e. the *persistent* entities, differ in their position. This is due to the constraint that pins the abstracted feature down relatively to the position of the corresponding, original feature. Comparing the position of the persistent entities in the abstracted and original model, it becomes clear that they differ by the distance to the mid-surface in the full-dimensional feature. In other words: replacing a full-dimensional feature by an abstraction based on its mid-surface, can result in a discrepancy of the distance from the original boundary to the mid-surface for certain attach constraints. In Figure 8.7(b), one discrepancy is the gap between the two features, another

is the excess in length of the feature in the back.

We can compensate the discrepancies by adjusting the geometry of the features that through a constraint refer to a persistent entity. We obviously cannot translate the persistent entity, as we would risk losing global consistency in the positioning of the abstracted features. Only adjustments to the geometric shape of features are allowed, countering the discrepancies locally, instead of making global changes to the position of features (see Figure 8.7(c)).

The automated procedure for deriving an abstracted model, consists merely of finding all adjustments needed to combine all individually abstracted features into a proper analysis view. Which individual features are present in the abstracted model, and their class of abstraction, is given as input to the procedure. The method is outlined in Algorithm 1.

An essential part of the algorithm is identifying whether a constraint (on parallelism) calls for adjustment of one of the features that are part of the constraint. The abstracted features are, in a sense, responsible for the discrepancy, so at least one of the two features in the constraint has to be abstracted. If one of them is, then it is determined whether the abstraction affects the validity of the constraint, i.e. whether it causes a discrepancy. If so, then it is checked whether the shape parameters of the other feature can be adjusted to bridge the gap. This is the case if this other feature has not been abstracted along the same dimension as the first feature.

The outlined procedure does not specify what the adjustment entails. For our set of features, the adjustment is simply a relatively small extension or retraction of the feature to bridge the gap between the features such that the constraint remains valid. If the constraint is between faces with an opposing normal, then a small extension is needed, else a small retraction (an extension with negative length) is warranted.

It is not enough to adjust the abstracted features based only on direct constraint relations. See, for example, Figure 8.8. Here an adjustment based on the constraint between features *B* and *C*, affected the connection of features *A* and *B*. We can avoid this by propagating the adjustments throughout the abstracted view. For each adjustment that has been recognized, it needs to be checked whether other constraints exist that depend on the entity that has been adjusted. For example, look back at Figure 8.8: we first note that an adjustment to the face of feature *B*, which is attached to feature *A*, is called for due to the abstraction of feature *C*. Thereafter, we find that the constraint between feature *A* and *B* is affected by this adjustment as well. Therefore feature *A* also needs an adjustment to keep features *A* and *B* attached as they were. This is taken care of by Algorithm 1.

The propagation process cannot result in an infinite loop. It ends when

---

**Algorithm 1: Find Feature Adjustments**


---

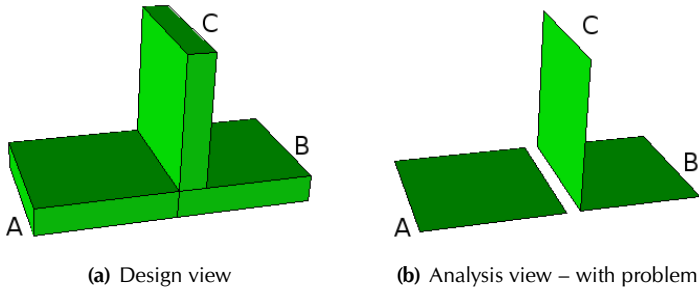
**Adjustment rule:** if a persistent entity is constrained parallel to a non-persistent entity by constraint  $C$ , then the non-persistent entity should be adjusted by  $f_{adj}$

```

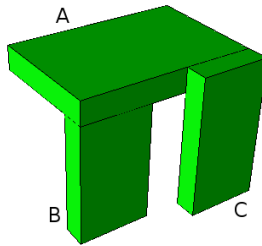
for constraint  $C \in$  design view do
  if  $C$  warrants feature adjustment  $f_{adj}$  to feature  $F$  then
    if  $F$  has no existing adjustment conflicting with  $f_{adj}$  then
       $\lfloor$  adjust  $F$  by  $f_{adj}$ 
    else
       $\lfloor$  abort: conflicting adjustments
     $\lfloor$  propagate( $f_{adj}$ )

propagate( $f_{adj}$ ):
while fresh adjustments (start with  $f_{adj}$ ) have been found do
  for each adjustment  $\bar{f}_{adj}$  do
    for each constraint  $C$  that refers to the adjusted entity  $e_{adj}$  do
      if  $C$  warrants feature adjustment  $f'_{adj}$  to feature  $F$  then
        if  $F$  has no existing adjustments then
           $\lfloor$  adjust  $F$  by  $f'_{adj}$ 
           $\lfloor$   $\rightarrow$  fresh adjustment found
        else if  $F$  has adjustments conflicting with  $f_{adj}$  then
           $\lfloor$  abort: conflicting adjustments
  
```

---



**Figure 8.8:** Need for propagation of adjustments



**Figure 8.9:** Conflicting feature adjustment

no new adjustments have been made. The number of adjustments that can possibly be made is clearly finite. The process can, however, end in a state of conflicting adjustments. See, for example, Figure 8.9, where features *B* and *C* attached to feature *A* result in conflicting adjustments.

### 8.3 Linking the design and analysis views

In the previous section, we have presented a procedure for an automated derivation of an abstracted analysis model from the design model. The resulting model consists of features that each individually correspond to a feature in the design model, together with a set of adjustments to those features. The adjustments are such that the constraints, as they are present in the design model, remain satisfied. The meaning of these constraints between abstracted features depends on the corresponding full-dimensional features, as was discussed in Section 8.1.

The rules for deriving the abstracted model from the design model are the key to linking the two models, as they define the relation between the two models. With this algorithmic understanding of the relation, we can

automatically maintain both models when changes are made to either of them. In our view, there is no distinction between rules for deriving the abstraction, and rules for maintaining the relation between the abstraction and the design model from which it was derived. If you can do one, then you can do the other as well.

Linking of the two models is achieved by relating the parameters of the set of abstracted features in the analysis view to the corresponding parameters in the design view. The abstracted features have a reference to the original feature, and their parameters are formed by coupling the parameter of the feature from the design view and the adjustments. In other words, the definitions of the parameters in the analysis models do not exist independently, but depend entirely on the corresponding definitions in the design model in combination with the adjustments.

With this setup, we can maintain the resulting analysis view under modification of the parameter values in either model. In our discussion we discern two kinds of parameters:

- feature parameters
- model parameters.

Feature parameters are specific to an individual feature, such as width, height, or length. In our analysis view, this also includes parameters that were abstracted from the design view, and correspond to a dimension that does not exist in the geometry of the analysis view. These parameters are nonetheless part of the analysis view and its features, not through geometrical presence, but as input to the analysis. For example, if it would result from analysis that a certain thickness that exists as abstracted parameter in the model, is not sufficient, then it should be possible to modify this parameter in the analysis view. Model parameters are used in the design model for relating feature parameters to each other and help in defining a global, consistent parametrisation of the model, e.g. a user-defined parameter such as a thickness that is referenced by multiple features. Model parameters can be used in the definition of feature parameters, and feature parameters can appear in the definition of model parameters, as long as there are no circular dependencies.

Propagating changes in the design view to the analysis view is fairly straightforward. All changes to feature and model parameters are automatically reflected in the analysis view, since it does not maintain these parameters independently, but in fact uses those of the design view. We only need to update the set of adjustments. The adjustments correspond to feature parameters in the original model that have been abstracted in the analysis view, so updating is only necessary when such a parameter has



been modified. After the adjustments have been updated, the analysis view can be updated by changing the dimensions of the features that have been affected by the change. We presume that in most cases the changes to the model would not lead to essential changes in the abstraction, but to be sure we would have to check this. It could happen that after the change, a feature that used to be abstracted, should no longer be abstracted, or vice versa. Another possibility is a change in abstraction, e.g. first the width and later the depth should be abstracted. Also, the outcome of the defeaturing could be different. Determining this falls outside the scope of our approach, but if the changes are known, the analysis view can be updated accordingly.

The propagation of changes in the analysis view to the design view is somewhat different. Since the parameters depend directly on the parameters in the design view coupled with the adjustments, we can calculate the change in the design view parameters from the change in analysis view parameters. This is essentially reversing the adjustment. However, the change of parameters in the analysis view is not as straightforward as one would think. On a technical level, it is just a matter of taking the adjustments correctly into account when translating the input of the user back to the design view. The issue here, however, is with the input of the user. This has no bearing on the working of our approach, but it is an interesting point nonetheless.

What values and parameters are presented to the user when he edits parameters in the analysis view? For example, the exact length of a feature does not need to be identical in both views, because of the adjustments. In the design view, the length parameter can be a numerical value, or be specified as a formula referring to user-defined parameters or the parameters of other features. In the analysis view, the definition is based on the corresponding parameter in the design view. So what should it show? There are basically three options: 1) the adjusted design view value, 2) the design view value together with the adjustments, 3) the design view value.

The first option is to show the adjusted parameter definition from the design view. If this is a numerical value then it is simple to adjust. Only the feature parameters are directly affected by the adjustments, so when their value is requested the value is adjusted. If a parameter is defined in terms of a formula, then the formula can be presented as is, since the referenced parameters will themselves be adjusted if necessary. This approach results in an analysis view that feels completely independent of the design view. A disadvantage could be confusion when a single user works with both models due to the slightly different numerical values.

The second option is to show the definition of the parameter in the design view together with the adjustment. In this case the adjustment is not done

completely transparent, but the user is made explicitly aware that the values in the two views are slightly different. This could be done by showing both the adjusted and the unadjusted definitions, or make the adjustment part of the definition such that it stands out as a separate unit. With this approach, there is no longer the risk of confusion due to differences in corresponding values between the views. However, the user is confronted with the parameter adjustments, a technical aspect underlying the link between the views, which he might not care for since his focus is on the analysis view.

The third option is to show the parameter definitions from the design view. This also works in a completely transparent manner, like the first option, and avoids confusion over the slightly different values. However, in this case there is a small inconsistency between the values that are shown to the user, and the actual dimensions of the geometry in the analysis view. Obviously, the user is aware that some parameters refer to abstracted dimensions, since he can edit the thickness parameter of a feature that is shown with zero thickness in the view. The user does not necessarily have a problem with discrepancies between the values shown to him and the dimensions of the geometry in his view for such parameters. However, he might have more difficulty with the incongruity when non-zero lengths of the geometry in the view are involved. A solution could be to make the user aware of this difference by showing the geometry with the unadjusted measurements in his view when he is editing the parameter.

For the underlying approach, it does not really matter which option is used. However, there might be a difference to the user. Will his decisions on how to change the model be different depending on how the model parameters are presented to him? Although small variations are probable, as a 20 % change to a parameter of the analysis view is different from a 20 % change to the corresponding parameter in the design view, it should not really matter to the end result. The principal idea behind abstraction is that certain details are not relevant to the outcome. Since the differences between the parameter values of the design and analysis view are due to precisely these details that were ignored, we expect the effect of using either set of parameter definitions to be insignificant.

Although the establishment of a link between the design and the analysis view is possible with our simple approach sketched above, there are undoubtedly many remaining issues. In the next section we briefly discuss some of the shortcomings of our approach to linking design and analysis models.

## 8.4 Open issues

The approach we have explored for linking an analysis model with dimensional abstractions to a design model, is admittedly limited. The set of features and constraints is small, and the success of the method depends on assumptions that may not hold for all realistic models.

In our approach, we abstract features on an individual basis, since this enables us to have a true feature model in the analysis view. Such a model cannot be the result of algorithms for dimensional reduction that are purely geometry-oriented, unless some kind of feature recognition would be used. It seems doubtful that the result of this model constructed on the basis of recognition would be better than what could be derived from the original features and constraints that define the design model. On the other hand, we question whether it is always the best choice, or even possible in every case, to create the abstracted model on the basis of individual features.

The features in our prototype offer straightforward ways for dimensional abstraction, but for a more complex feature the way to abstract its individual geometry might not be immediately clear. It could very well be that the right abstraction depends on how the feature is combined and interacts with other features. We imagine the possibility that multiple features in a design model could be best represented by a single feature in the analysis view. Our current approach cannot handle such cases, as we essentially assume a 1-to-1 relationship between individual features of the design and the analysis views.

Knowing how the geometry of an individual feature can be abstracted, is not enough. In addition to a complete set of rules for adjusting feature parameters based on the constraints, which can combine individual abstracted features into an analysis model, we also need rules to decide which features should be abstracted and, if multiple abstractions are possible, in which way. In our current approach, these decisions were part of the input. Some self-evident heuristics on the relative sizes of various shape parameters can be used here, but looking at just the individual features might give unwanted results. The choice for abstraction of a particular feature, when looking at it in combination with other features, might be different from what would be concluded from just its individual shape. This is why geometry-oriented algorithms for abstraction are popular, as they can take the whole geometry into account, and do not need to worry about the abstraction of individual features. Possibly a geometry-oriented approach could be used in conjunction with a feature-based approach in the derivation of the analysis view for supporting these choices in the abstraction.

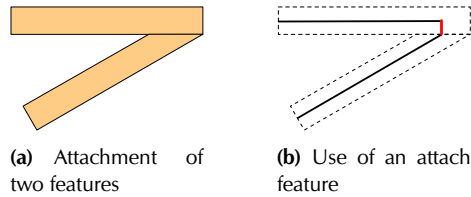
Another way to support the abstraction process is to incorporate seman-

tics from the feature model in the decision procedure. This incorporation of knowledge from other views, is one of the underlying ideas of the analysis view, or views in general. To some extent it is known what the role of the features is, whether they belong to a class of features that is usually excluded from the analysis model, and whether there is an abstraction that is preferred by default. This requires rich models that contain information that supports an algorithmic understanding of our problems. As discussed in Chapter 7, this is easier to achieve in specialised environments, where the number of different features is confined, and interest is limited to a particular class of analysis problems.

A further factor of complication is dealing with defeaturing, i.e. leaving out features from the design model in the analysis model. We have not really touched upon this with our approach. Assuming we know which features to take out, we should be able to handle this, since our feature modelling system can perform this operation and return a consistent and logical model. This should work for most features considered for defeaturing, as they usually have been added in the later stages of the design, and thus have no other features depending on them. Difficulty arises when other features depend, through their constraints, on a feature that is to be removed. Some situations could be handled by dropping the references and replacing them by constant values that correspond to the current state of the model, as this would at least yield a solution. However, we then have lost information in the process, and we do not know how to deal with this in the adjustments of the constraints for the abstracted analysis model. If corresponding constraints in the design and the analysis model were to depend on different features, then propagation becomes an issue.

Another issue with propagation of changes from an abstracted analysis model to a design model, is ambiguity. In our approach we assume that the changes to the analysis model are limited to changes to existing parameters. In that case, the design model can be updated based on the parameter correspondence, while taking the parameter adjustments into account. In general, however, propagating changes from a dimensionally abstracted model to a full-dimensional model is not possible without ambiguity. For instance, when attaching new features to the boundary of abstracted geometry, there are multiple ways to translate this attachment to the full-dimensional model. Possibly, it could be determined on the basis of heuristics what outcome is likely to be desired, or otherwise the user would be required to refer to geometric elements from the full-dimensional model when making changes to the abstracted model.

The only constraints that we have explored for our linking of models are perpendicular attach and position constraints, i.e. the mid-surface or midline

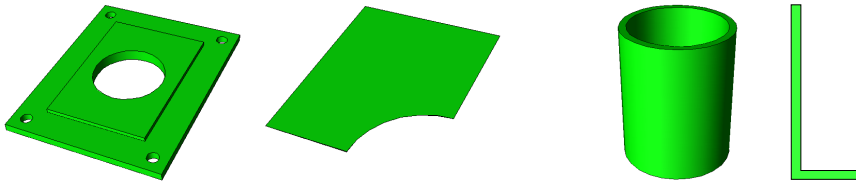


**Figure 8.10:** Connecting non-perpendicular midlines with an attach feature

of the abstracted feature has a perpendicular orientation with respect to the other feature. In that case the shape parameters of the features can be used to adjust the geometry of the model. In the case that the abstracted geometry does not make a perpendicular attachment to the other feature, it is less clear how the adjustment should be made. Adjusting the shape parameters might give unwanted results, or maybe no results at all, as the abstracted geometry does not intersect. A solution could be the introduction of a special attach feature that connects two features in the abstracted view, such as illustrated in Figure 8.10. However, though this solution may work, it might not result in the analysis model an engineer would have built. This is not necessarily a problem for the analysis, but it could be a barrier for acceptance by engineers who would have preferred a different result. Of course, there are other constraints than just attach and position constraints. It has to be looked into, if and how these can be included into the adjustment procedure as well.

In our approach, we leave the design model more or less untouched, and attempt to achieve integration by focussing on the analysis model and its path of derivation from the design model. We can, however, imagine a solution that requires essential changes to the representation of the design model and the tasks involved in specifying the model. Constraints in the design model, for example, could be extended to include directions or hints on the intention of the constraint in various cases of abstraction for an analysis model. We have not explored this particular direction, but we are wary of such approaches, as they might require the designer to make choices and specify relations that fall outside his design focus. Keeping views separated, at least from a user's perspective, is fundamental to our general approach.

It is the great variety in features, models, abstractions, analyses, etc. that makes a general solution to this problem hard. Some abstract models exploit symmetries in the model, thus greatly reducing the computational work. See Figure 8.11, which shows two examples of a design model and its abstracted analysis model. It seems likely that at least some involvement of the engineer would be required, in relating such an analysis model to the design model.



**Figure 8.11:** Two design models with a corresponding abstracted analysis model

This is essentially a failure to derive the analysis model in a completely automated manner. Given all the difficulties that we have mentioned, such a partial failure is to be expected from time to time. It is an interesting question how this should be dealt with, since we want to maintain a transparent link between the views. Is it possible to let the engineer intervene in the derivation of the abstract models, and yet have the views transparently linked afterwards?

Clearly, much research has yet to be done, before we will have an analysis view with the functionality proposed in Section 7.3.

## 8.5 Conclusions

We have created a prototype implementation for linking an analysis view and a design view. In this implementation, modelling is done by means of scripts that perform the modelling operations. With a few simple features and constraints, we have demonstrated a way to achieve propagation of changes to the geometry of both the design and the analysis model. This is a first step towards the realisation of analysis views.

The central idea of our approach is to have a correspondence between the individual features in the analysis model and those in the design model. The features in the design model are combined into a single geometry by means of constraints, and these provide an indication of how the connection of features is intended. We attempt to reinterpret these constraints for the analysis model, such that the intention behind the combination of the features is preserved. We do this by adjusting shape parameters of individual features. For our simple example models, this approach works well.

Since the adjusted parameter definitions in the analysis view are basically the same parameters as in the design view, there is a question of how to present these parameters of the analysis view to the user. The same model parameters that were used in the design view, should be present in the analysis view. These parameters, however, refer to a different geometry. The first option is that it is represented as a completely independent model,

without any mention of the adjustments that have been applied. Some of the parameter values will be slightly different from the corresponding values in the design view, but only if the user works with both views at the same time, this might be a source of confusion. The second option is that the user is made aware of the differences in geometry by showing the adjustments in some way in the interface for changing the parameter definitions. This would, however, interfere with the aim that the analysis view is really a separate view, where the only concern is the analysis context. The last option is that the same user-defined parameters are shown in the interface of the analysis model, whereas the adjustments are only used for the actual geometry that is shown in the view. From the point of ease of use, and the aims of the analysis views, either the first or the last approach is preferred. However, both have some risk of confusion, as either the parameter values between the two views are slightly different, or changes to the parameter values in the analysis view do not correspond 1-to-1 with the changes of the geometry in that view. This may not be a problem, as both effects are in the order of the adjustment values, and these are usually small in comparison to the geometry as a whole. User testing would be required to get a clearer picture of the drawbacks and advantages of the various approaches to present the parameter definitions and values of the analysis view.

As we have pointed out in the previous section, there are many issues remaining. Although our approach is a first step, much research is left to be done, before analysis views could be used in practice.

## **Acknowledgements**

We kindly thank Rick van der Meiden for the use of his constraint solver.





## Chapter 9

# Conclusions and future research

This thesis addresses improvement of the integration of design and analysis. Historically, the two disciplines evolved independently, but with the growing capabilities of both, the need for interaction developed. Nowadays, analysis is an established constituent of the product development cycle. However, the integration between the design and analysis phases can still be characterised as poor, or even lacking.

An important factor in this poor integration is the need to switch frequently between various models, such as the design model, the analysis model, and the analysis mesh. These transitions often involve some kind of conversion of models, which is rarely completely automated, and thus leads to tedious and redundant manipulation of models. In this work, we aimed to improve the integration of design and analysis by increasing the reuse of information when going back and forth between the various phases and their corresponding models.

The first aspect we have looked at, is reducing the time spent on meshing an analysis model after a change in the design. To this end, we have developed a procedure for efficient tetrahedral remeshing, which works by relating the current and previous analysis model, and by identifying the similarities and differences in their geometry.

The second aspect we have looked at, is the integration of design and analysis models. We introduce the concept of analysis views for multiple-view feature modelling, and describe how views interact in this approach.

Firstly, in Section 9.1, we summarise the conclusions drawn from these two research aspects, and from the work as a whole. Secondly, in Section 9.2, we discuss how the work in this thesis could be extended, and how it relates to possible future developments for the integration of design and analysis.

## 9.1 Conclusions

Remeshing of analysis models, based on the analysis model and the mesh from the previous iteration, can improve the efficiency of the product development cycle, in particular when the modifications to the analysis model are small, the mesh has many elements, and the meshing procedure is time-consuming. Small modifications are more likely when the development of a product has progressed further, and so is the number of mesh elements, as the accuracy of the analysis becomes more important at that stage. Meshing algorithms that are time-consuming often have a variational nature, minimising an energy functional related to some quality measure, and therefore delivering high quality meshes. Such meshes are advantageous for performing analysis, as they lower the computation time and increase the overall reliability of the analysis.

Our tetrahedral remeshing approach delivers such high quality meshes. The method can handle changes in topology between the original and modified models, such as addition or removal of features, which cannot be handled with morphing-based techniques. The efficiency gains are higher when the ratio of boundary samples to nodes is higher, but even for the lowest acceptable ratio the gains are evident.

The remeshing approach is based on variational tetrahedral meshing. We showed that with some adjustments, this algorithm can be suitably applied to meshing of mechanical models. These enhancements are in particular focussed on obtaining a correct representation of the boundary in the mesh, as this is the principle concern of applying the Delaunay-based method to mechanical models. The runtime of the algorithm is not noticeably affected by our enhancements, as the repeated updates of the Delaunay triangulation clearly dominate the computational costs. Some models, however, remain difficult to handle with this method, as effectively an approximate conforming Delaunay mesh is constructed. If small dihedral angles are present in the model, or regions where different parts of the boundary are close, then a very high number of elements will be needed for a correct representation of the boundary. In general, however, the approach behaves well for meshing mechanical models. Also, because it is a Delaunay-based method, we can copy mesh subsets with relative ease. Only nodes need to be copied, without taking explicit care for the connectivity, which makes it very suitable to base our remeshing approach on.

Very important to our remeshing approach is the new concept of feature difference. This description identifies and categorises, for each feature, the geometric entities of the model that it covers. For each entity, including cells of volume, it is recorded whether the geometry is persistent, old, or new.

Additionally, the interpretation of the entity is taken into account, and the changes of interpretation are captured by the feature difference as well. For our purpose of remeshing, the interpretation reflects for faces, edges and vertices whether the entity was part of the model boundary, and for cells the additive or subtractive nature of the cell.

The feature difference and the combined model, which is a data structure derived from the feature difference, enable us to identify similar geometry between related models, such that it can be used by our remeshing procedure. This is possible since the differences and similarities are represented in a geometrically exact manner. The description also connects closely to our intuitive idea of how two models differ, as it is based on how individual features evolve and interact with the other features in the model. This is sensible since incremental changes to feature models are made in terms of addition, removal, or reparametrisation of features. The cellular model is essential to the representation of the feature difference. The information contained in the feature difference model is virtually unobtainable from a basic BRep structure. The concept of feature difference is very useful for our remeshing procedure, and we therefore suppose that there are more applications that might benefit from this approach to model comparison.

Altogether, the new remeshing approach that has been presented can be profitable for variational tetrahedral meshing of mechanical models, but the basic idea and some concepts used in it, can also be valuable for other meshing approaches.

For the remeshing procedure it is required that the original and the modified analysis model can be related to each other, i.e. we must be able to map the features between the models, so that we know how the individual features correspond between the two models. This requires consistency between versions of the analysis models. Since an analysis model is an abstraction of the design model, this implies a relationship between these two models as well. There are several ways to obtain an analysis model from a design model, each with different underlying data structures and characteristics for manipulation. For automatically maintaining consistency between iterations of an analysis model, and, consequently, between analysis models and the design model, we need to collectively maintain the models. The concept of analysis views for multiple-view feature modelling supports this.

Analysis views offer an integration of design and analysis in which the relationships between all models are automatically maintained in a transparent manner. This avoids having to repeatedly derive an analysis model, or manually update earlier derived models, after changes to the design. They also enable bidirectional propagation of changes, i.e. changes

can be made in any of the views, not just the design view. Although multiple-view feature modelling has already demonstrated its viability in general, a particular challenge here lies in the maintenance of views with different geometry.

The analysis model is an abstraction of the design model, and as such, similar features can be found in both models. Linking these two models, even if the analysis model is a dimensional abstraction, can be done on the basis of features and the constraints that define how the features are related. In case of dimensional abstraction, the constraints need to be adjusted to compensate for the differences in feature dimensions. The user can edit either representation, because through the adjusted constraints, the constraint relations in the design model can be updated.

The work on this aspect has shown that the concept of analysis views for multiple-view feature modelling is both feasible and promising, but much work remains to be done here to fully achieve an analysis view with the capabilities that we envision.

## 9.2 Future research

In our remeshing approach, the Delaunay property defines the connectivity of the mesh. Obviously, when the connectivity had been changed to further optimise the mesh, e.g. to remove near-slivers by flipping, this will not be transferred to the new mesh by merely copying the nodes. In that case, we would need to copy the connectivity too, or repeat the final optimisation step. In general, the ability to copy mesh connectivity is a requirement when basing the remeshing procedure on alternative meshing algorithms that do not have the connectivity implicitly defined. It would be interesting to implement the remeshing strategy for such algorithms, and measure the corresponding performance improvements.

The node density is currently assumed to be uniform over the model. Obviously, when the geometry of the model changes, then the requirements for the mesh can change as well. In particular the required mesh density may change, and this needs to be taken into account when applying the remeshing procedure. Some regions may have to be adapted. Through the use of analysis features for the density requirements, the changes herein could be determined exactly. The feasibility of this, and the impact on the performance improvements, have to be studied.

The benefits of using a high quality mesh and of the remeshing procedure have not been tested within the context of actual analyses. Realistic experiments should be carried out to confirm the posited advantages, such as shortening the preparation and analysis stages, and providing a higher

degree of reliability. In particular, such experiments would include a comparison with other meshing methods.

The feature difference relates features from two versions of a model, and thus requires that the two models have their origin in the same modelling system, and that they are a variant of each other. When the models originate from different systems, or when they have not been constructed with a feature modelling system, then the creation of the feature mapping between the models would be a challenge on its own.

We have demonstrated how to link a design and an analysis model for a very limited set of features and constraints. The approach assumes that every feature in the analysis model relates to a single feature in the design model, i.e. that there is a one-to-one relation between the features of the analysis model to the features from the design model. This is not always the case in practice. Also, some features from the design model might not be included in the analysis model, as they are not significant for the outcome of the analysis, and may even complicate and interfere with the analysis process. This is an issue when there are features in the analysis model of which the constraints depend on features that do not exist in the analysis model. An extension of the approach that includes more types of features and constraints, and resolving the latter and many other remaining issues, would be very interesting.

Ideally, we would have a complete assimilation of analysis into design, which would imply that the developer would no longer distinguish between the separate tasks, but instead analysis would transparently be used to steer the design process. This, however, is nowhere near to being realised, so the coming years will focus on the integration of the two disciplines. The latter will involve more and easier interaction primarily based on the exchange of information on updates to the model, between the various tasks, which could very well be achieved through a range of views on the product in the spirit of what we have proposed.

Incorporation of knowledge of the analysis process and the requirements for the analysis model is essential for improving integration of analysis with the rest of the product development cycle. Depending, for instance, on the analysis context and the level of abstraction, different requirements hold. Product development systems should 'understand' when operations should be executed automatically, or at least know from which limited set of operations the engineer should be offered a choice. Integrating design and analysis models by means of a multiple-view feature modelling approach, in which both models are maintained concurrently as views on a single product, provides a good way to incorporate the relevant knowledge on the transition between the two models.

Although indirectly a lot has been written on the combined workflow of design and analysis, in terms of specific problems and industrial cases, little is known about how industry at large deals with the interaction of the two disciplines. It would be useful to have more statistically backed insight on this, in relation to the size and complexity of the models, the type or level of abstraction of the analysis models, and the type of simulations. Collecting such information, however, would be a huge effort, also because industry is not always keen on disclosing the issues they are experiencing. On the other hand, it might have a significant impact on the future research on integration of design and analysis.

# Bibliography

- Acikgoz, N. and Bottasso, C. L., 2007. Metric-driven mesh optimization using a local simulated annealing algorithm. *International Journal for Numerical Methods in Engineering*, **71**(2):201–223.
- Ainsworth, M. and Oden, J. T., 1997. A posteriori error estimation in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, **142**(1-2):1 – 88.
- Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M., 2005. Variational tetrahedral meshing. *ACM Transactions on Graphics*, **24**(3):617–625.
- Amenta, N. and Bern, M., 1999. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, **22**(4):481–504.
- Apel, T. and Dobrowolski, M., 1992. Anisotropic interpolation with applications to the finite element method. *Computing*, **47**:277–293.
- Arabshahi, S., Barton, D. C., and Shaw, N. K., 1993. Steps towards CAD-FEA integration. *Engineering with Computers*, **9**(1):17–26.
- Bagemihl, F., 1948. On indecomposable polyhedra. *American Mathematical Monthly*, **55**:411–413.
- Baker, T. J., 2005. Mesh generation: Art or science? *Progress in Aerospace Sciences*, **41**(1):29 – 63.
- Beall, M. W., Walsh, J., and Shephard, M. S., 2004. A comparison of techniques for geometry access related to mesh generation. *Engineering with Computers*, **20**(3):210–221.
- Benzley, S. E., Perry, E., Merkley, K., Clark, B., and Sjaardama, G., 1995. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elastoplastic analysis. In *Proceedings of the 4th International Meshing Roundtable*, pages 179–191. John Wiley & Sons.
- van den Berg, E. and Bronsvoort, W. F., 2010. Validity maintenance for freeform feature modeling. *Journal of Computing and Information Science in Engineering*, **10**(1):011006/1–14.

- van den Berg, E., Bronsvoort, W. F., and Vergeest, J. S. M., 2002. Freeform feature modelling: concepts and prospects. *Computers in Industry*, **49**(2):217–233.
- Bespalov, D., Regli, W. C., and Shokoufandeh, A., 2006. Local feature extraction and matching partial objects. *Computer-Aided Design*, **38**(9):1020–1037.
- Biasotti, S., Marini, S., Spagnuolo, M., and Falcidieno, B., 2006. Sub-part correspondence by structural descriptors of 3D shapes. *Computer-Aided Design*, **38**(9):1002–1019.
- Bidarra, R. and Bronsvoort, W. F., 2000. Semantic feature modelling. *Computer-Aided Design*, **32**(3):201–225.
- Bidarra, R., de Kraker, K. J., and Bronsvoort, W. F., 1998. Representation and management of feature information in a cellular model. *Computer-Aided Design*, **30**(4):301–313.
- Bidmon, K., Rose, D., and Ertl, T., 2004. Intuitive, interactive, and robust modification and optimization of finite element models. In *Proceedings of the 13th International Meshing Roundtable*. Sandia National Laboratories.
- Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Levy, B., 2010. *Polygon Mesh Processing*. AK Peters.
- Bozdoc, M., 2010. Marian Bozdoc's history of CAD. <http://mbinfo.mbdesign.net/CAD-History.htm> (accessed June 2010).
- Branets, L. and Carey, G. F., 2005. A local cell quality metric and variational grid smoothing algorithm. *Engineering with Computers*, **21**(1):19–28.
- Brenner, S. C. and Scott, L. R., 2007. *The Mathematical Theory of Finite Element Methods*. Springer, 3rd edition.
- Bronsvoort, W. F., Bidarra, R., and Nyirenda, P. J., 2006. Developments in feature modelling. *Computer-Aided Design and Applications*, **3**(5):655–664.
- Bronsvoort, W. F. and Noort, A., 2004. Multiple-view feature modelling for integral product development. *Computer-Aided Design*, **36**(10):929–946.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., 1996. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK.
- CGAL Editorial Board, 2006. CGAL, Computational Geometry Algorithms Library. <http://www.cgal.org/> (accessed March 2006).
- Chand, K. K., Diachin, L. F., Li, X., Ollivier-Gooch, C., Seol, E. S., Shephard, M. S., Tautges, T., and Trease, H., 2008. Toward interoperable mesh, geometry and field components for PDE simulation development. *Engineering with Computers*, **24**(2):165–182.



- Chen, L., 2004. Mesh smoothing schemes based on optimal Delaunay triangulations. In *Proceedings of the 13th International Meshing Roundtable*, pages 109–120. Sandia National Laboratories, Williamsburg, VA.
- Chen, L. and Xu, J., 2004. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, **22**(2):299–308.
- Cheng, S.-W., Dey, T. K., Edelsbrunner, H., Facello, M. A., and Teng, S.-H., 2000. Sliver exudation. *Journal of ACM*, **47**(5):883–904.
- Chew, L. P., 1989. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Comp. Science Dept., Cornell University.
- Chong, C. S., Kumar, A. S., and Lee, H. P., 2007. Automatic mesh-healing technique for model repair and finite element model generation. *Finite Elements in Analysis and Design*, **43**(15):1109–1119.
- Cifuentes, A. O. and Kalbag, A., 1992. A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis. *Finite Elements in Analysis and Design*, **12**(3–4):313–318.
- Cohen-Steiner, D., de Verdière, E. C., and Yvinec, M., 2002. Conforming Delaunay triangulations in 3D. In *SCG '02: Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 199–208. ACM Press, New York.
- Date, H. and Onosato, M., 2008. Triangular mesh deformation based on dimensions. *Computer-Aided Design and Applications*, **5**(1–4):287–295.
- Delaunay, B., 1934. Sur la sphère vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvoennyka Nauk*, **7**(6):793–800.
- Dittmer, J. P., Jensen, C., Gottschalk, M., and Almy, T., 2006. Mesh optimization using a genetic algorithm to control mesh creation parameters. *Computer-Aided Design and Applications*, **3**(6):731–740.
- Drieux, G., Léon, J.-C., Guillaume, F., Chevassus, N., Fine, L., and Poulat, A., 2007. Interfacing product views through a mixed shape representation. Part 2: Model processing description. *International Journal on Interactive Design and Manufacturing*, **1**(2):67–83.
- Du, Q., Faber, V., and Gunzburger, M., 1999. Centroidal Voronoi tessellations: applications and algorithms. *SIAM Review*, **41**(4):637–676.
- Du, Q. and Wang, D., 2003. Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations. *International Journal for Numerical Methods in Engineering*, **56**(9):1355–1373.
- Du, Q. and Wang, D., 2006. Recent progress in robust and quality Delaunay mesh generation. *Journal of Computational and Applied Mathematics*, **195**(1):8–23.

- Edelsbrunner, H. and Guoy, D., 2002. An experimental study of sliver exudation. *Engineering with Computers*, **18**(3):229–240.
- Eppstein, D., 2001. Global optimization of mesh quality. <http://www.ics.uci.edu/~eppstein/pubs/Epp-IMR-01.pdf> (accessed October 2010).
- Field, D. A., 1988. Laplacian smoothing and Delaunay triangulations. *Communications in Applied Numerical Methods*, **4**(6):709–712.
- Foucault, G., Cuilliere, J.-C., Francois, V., Léon, J.-C., and Maranzana, R., 2008. Adaptation of CAD model topology for finite element analysis. *Computer-Aided Design*, **40**(2):176–196.
- François, V. and Cuillère, J.-C., 2000. 3D automatic remeshing applied to model modification. *Computer-Aided Design*, **32**(7):433–444.
- François, V., Cuillère, J. C., and Gueury, M., 1999. Automatic meshing and remeshing in the simultaneous engineering context. *Research in Engineering Design*, **11**(1):55–66.
- Frey, P. J. and George, P.-L., 2000. *Mesh Generation: Application to Finite Elements*. HERMES Science Publishing, Oxford, Paris.
- George, P. L., Hecht, F., and Saltel, E., 1991. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, **92**(3):269 – 288.
- Hamri, O., Léon, J.-C., Giannini, F., Falcidieno, B., Poulat, A., and Fine, L., 2008. Interfacing product views through a mixed shape representation. Part 1: Data structures and operators. *International Journal on Interactive Design and Manufacturing*, **2**(2):69–85.
- Hoffmann, C. M. and Joan-Arinyo, R., 1997. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, **23**(2–3):287–299.
- Hoffmann, C. M. and Joan-Arinyo, R., 1998. CAD and the product master model. *Computer-Aided Design*, **30**(11):905–918.
- van Holland, W. and Bronsvort, W. F., 2000. Assembly features in modeling and planning. *Robotics and Computer Integrated Manufacturing*, **16**(4):277–294.
- Hughes, T. J. R., Cottrell, J. A., and Bazilevs, Y., 2005. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, **194**(39-41):4135–4195.
- Joe, B., 1995. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, **16**(6):1292–1307.
- Klingner, B. M. and Shewchuk, J. R., 2007. Aggressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23. Springer.

- de Kraker, K. J., Dohmen, M., and Bronsvooort, W. F., 1997. Maintaining multiple views in feature modeling. In C. M. Hoffmann and W. F. Bronsvooort, editors, *Proceedings of Solid Modeling '97, Fourth Symposium on Solid Modeling and Applications, 14–16 May, Atlanta, USA*, pages 123–130. ACM Press, New York.
- Lawson, C. L., 1977. Software for  $C^1$  surface interpolation. In J. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press.
- Lee, K., 1999. *Principles of CAD/CAM/CAE systems*. Addison-Wesley, USA.
- Lee, K. Y., Armstrong, C. G., Price, M. A., and Lamont, J. H., 2005. A small feature suppression/unsuppression system for preparing B-rep models for analysis. In L. Kobbelt and V. Shapiro, editors, *SPM '05: Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, pages 113–124. ACM Press, New York.
- Lee, S. H., 2005. A CAD-CAE integration approach using feature-based multi-resolution and multi-abstraction modelling techniques. *Computer-Aided Design*, 37(9):941–955.
- Liu, W. and Yang, Y., 2007. Multi-objective optimization of an auto panel drawing die face design by mesh morphing. *Computer-Aided Design*, 39(10):863–869.
- Mackerle, J., 2001. 2D and 3D finite element meshing and remeshing: a bibliography (1990-2001). *Engineering Computations*, 18(8):1108–1197.
- Martino, T. D., Falcidieno, B., and Hassinger, S., 1998. Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment. *Computer-Aided Design*, 30(6):437–452.
- Matula, D. W. and Sokal, R. R., 1980. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12:205–222.
- van der Meiden, H. A. and Bronsvooort, W. F., 2007. Solving topological constraints for declarative families of objects. *Computer-Aided Design*, 39(8):652–662.
- Nyirenda, P. J., Bidarra, R., and Bronsvooort, W. F., 2007. A semantic blend feature definition. *Computer-Aided Design and Applications*, 4(6):795–806.
- Open CASCADE S.A.S., 2010. Open CASCADE. <http://www.opencascade.org/> (accessed June 2010).
- Owen, S., 2006. Meshing research corner. <http://www.andrew.cmu.edu/user/sowen/mesh.html> (accessed October 2010).
- Owen, S. J., 1998. A survey of unstructured mesh generation technology. In *Proceedings of the 7th International Meshing Roundtable*, pages 239–267. Sandia National Laboratories, Dearborn, MI.

- Parthasarathy, V. N., Graichen, C. M., and Hathaway, A. F., 1994. A comparison of tetrahedron quality measures. *Finite Elements in Analysis and Design*, **15**(3):255–261.
- Peak, R. S., Fulton, R. E., Nishigaki, I., and Okamoto, N., 1998. Integrating engineering design and analysis using a multi-representation approach. *Engineering with Computers*, **14**(2):93–114.
- Pernot, J.-P., Falcidieno, B., Giannini, F., and Léon, J.-C., 2008. Incorporating free-form features in aesthetic and engineering product design: State-of-the-art report. *Computers in Industry*, **59**(6):626–637.
- Persson, P.-O., 2006. Mesh size functions for implicit geometries and PDE-based gradient limiting. *Engineering with Computers*, **22**(2):95–109.
- Quadros, W. R., Shimada, K., and Owen, S. J., 2004. Skeleton-based computational method for the generation of a 3D finite element mesh sizing function. *Engineering with Computers*, **20**(3):249–264.
- Rajan, V., 1994. Optimality of the Delaunay triangulation in  $R^d$ . *Discrete and Computational Geometry*, **12**:189–202.
- Reddy, J. N., 2005. *An Introduction to the Finite Element Method*. McGraw-Hill, 3rd edition.
- Regli, W. C. and Spagnuolo, M., 2006. Introduction to shape similarity detection and search for CAD/CAE applications. *Computer-Aided Design*, **38**(9):937–938.
- Requicha, A. A. G. and Voelcker, H. B., 1985. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, **73**(1):30–44.
- Rivara, M.-C., Hitschfeld, N., and Simpson, R. B., 2001. Terminal-edges delaunay (small-angle based) algorithm for the quality triangulation problem. *Computer-Aided Design*, **33**(3):263–277.
- Schneiders, R., 2006. Mesh generation & grid generation of the web. <http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html>.
- Schönhardt, E., 1928. Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Mathematische Annalen*, **98**:309–312.
- Shah, J. and Mäntylä, M., 1995. *Parametric and Feature-based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons, Inc.
- Sheffer, A. and Unger, A., 2001. Efficient adaptive meshing of parametric models. *Journal of Computing and Information Science in Engineering*, **1**(4):366–375.
- Shephard, M. S., Beall, M. W., O'Bara, R. M., and Webster, B. E., 2004. Toward simulation-based design. *Finite Elements in Analysis and Design*, **40**(12):1575–1598.

- Shewchuk, J. R., 1996. Robust adaptive floating-point geometric predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. ACM Press, New York.
- Shewchuk, J. R., 2002a. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In *Proceedings of the 11th International Meshing Roundtable*, pages 193–204. Sandia National Laboratories, Ithaca, NY.
- Shewchuk, J. R., 2002b. What is a good linear element? Interpolation, conditioning, and quality measures. In *Proceedings of the 11th International Meshing Roundtable*, pages 115–126. Sandia National Laboratories, Ithaca, NY.
- Shimada, K., 2006. Current trends and issues in automatic mesh generation. *Computer-Aided Design and Applications*, 3(6):741–750.
- Si, H., 2006. TetGen, a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. <http://tetgen.berlios.de/> (accessed november 2007).
- SIAM, 2010. The history of numerical analysis and scientific computing. <http://history.siam.org/> (accessed June 2010).
- Sinha, M. and Suresh, K., 2005. Simplified engineering analysis via medial mesh reduction. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 207–212. ACM Press, New York.
- Spatial Corporation, 2006. 3D ACIS modeler. <http://www.spatial.com/products/acis.html>.
- Stark, J., 2005. *Product Lifecycle Management: 21st Century Paradigm for Product Realisation*. Springer-Verlag, London.
- Sutherland, I. E., 1963. Sketchpad, a man-machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference, Volume 23*, pages 329–346. Cleaver-Hume Press, London.
- Sypkens Smit, M. and Bronsvort, W. F., 2007. The difference between two feature models. *Computer-Aided Design and Applications*, 4(6):843–851.
- Sypkens Smit, M. and Bronsvort, W. F., 2008. Variational tetrahedral meshing of mechanical models for finite element analysis. *Computer-Aided Design and Applications*, 5(1–4):228–240.
- Sypkens Smit, M. and Bronsvort, W. F., 2009a. Efficient tetrahedral remeshing of feature models for finite element analysis. *Engineering with Computers*, 25(4):327–344.
- Sypkens Smit, M. and Bronsvort, W. F., 2009b. Integration of design and analysis models. *Computer-Aided Design and Applications*, 6(6):795–808.

- Tangelder, J. W. and Veltkamp, R. C., 2008. A survey of content based 3D shape retrieval methods. *Multimedia Tools and Applications*, **39**:441–471.
- Tautges, T. J., 2001. The generation of hexahedral meshes for assembly geometry: survey and progress. *International Journal for Numerical Methods in Engineering*, **50**(12):2617–2642.
- Tech Soft 3D, 2006. HOOPS. <http://www.hoops3d.com/products/3daf.html> (accessed December 2007).
- Teng, S.-H. and Wong, C. W., 2000. Unstructured mesh generation: theory, practice, and perspectives. *International Journal of Computational Geometry and Applications*, **10**(3):227–266.
- Thakur, A., Banerjee, A. G., and Gupta, S. K., 2009. A survey of CAD model simplification techniques for physics-based simulation applications. *Computer-Aided Design*, **41**(2):65 – 80.
- Unruh, V. and Anderson, D. C., 1992. Feature-based modeling for automatic mesh generation. *Engineering with Computers*, **8**(1):1–12.
- Voronoi, G., 1907. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, **133**:97–178.
- Weiler, K. J., 1988. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In *Geometric Modeling for CAD Applications*, pages 3–36. North Holland, Amsterdam.

# Summary

Analysis, nowadays, plays a major role in the product development cycle. It enables to check whether products will function to specification, it helps to optimise designs for material use and physical characteristics, and it promotes innovation by facilitating the exploration of less conventional designs, without having to construct costly and time-consuming test models.

Historically, there was limited interaction between design and analysis, as they started out as more or less separate disciplines. As their respective capabilities advanced, the need for more frequent interaction increased. Over recent decades, a closer interaction between design and analysis has become very common, but their historic disparity is still evident by a clear lack of integration between the two tasks.

In this thesis, we are concerned with improving the integration of design and analysis, within the context of feature modelling. This means to decrease the need to perform tedious or redundant tasks related to the transition from design to analysis, or vice versa. It should be accomplished by remedying the lack of reuse of information captured in different phases of the design and analysis process, and will lead to cheaper, more reliable, and more efficient product development.

We focus on two aspects within the design-analysis loop, and describe how improvements can be made here. The first aspect concerns efficient meshing of analysis models, whereas the second aspect concerns the integration of design and analysis models.

The first aspect we deal with, is the efficient remeshing of analysis models. Quality meshes for analysis are popular, since they can decrease the runtime of the calculations, lower the odds of failure of the analysis, and generally increase the reliability of results. However, quality meshes take more time to generate. Within the product development cycle, analysis is performed multiple times, each time after some modification of the design model. Since the changes in the geometry, after such a modification, are often minor and local in scope, there is a great deal of similarity between the meshes of successive analysis models as well. We exploit these similarities for the

efficient construction of a new mesh, based on the previous mesh.

Our work is based on the variational tetrahedral meshing algorithm. This algorithm generates tetrahedral meshes with a very high quality, but it is not really suitable for meshing mechanical models. We discuss the limitations, and present improvements to overcome these, which we illustrate by examples.

In order to reuse a mesh from a previous iteration of an analysis model, we need a precise geometrical description of the similarities and differences between two models. Since we work with feature models, and features have a strong connection to how humans perceive a model and, by extension, the similarities and differences between models, we base our approach on the features that compose the models. For each feature, we determine a description of the difference, which we call the *feature difference*. It describes how the geometry and interpretation between two models changed, *from the point of view* of that feature. All the individual feature differences together constitute the *difference model*. We give a detailed explanation of the feature difference by means of examples, and discuss its implementation.

Building upon the feature difference, and the improvements to variational tetrahedral meshing, we present a method for efficient tetrahedral remeshing. The method basically looks for the similarities between two models, based on the features that compose them. For regions of similarity, mesh elements are copied from the previous mesh to the new mesh, and for the remaining regions new mesh elements are constructed. We take specific care that the quality of the new mesh is on par with the quality of the previous mesh, which in particular is important for those regions where mesh elements of different origins meet. The efficiency gains are illustrated by means of examples.

The second aspect that we deal with in this thesis, is the integration of design and analysis models. In practice, the representation of design models and analysis models is often different. This commonly results in a need for conversion of a design model to an appropriate analysis model. We describe the differences in representation, and various approaches to integration that have been suggested in the past. We propose a tighter integration of the two models, by means of *analysis views*. The analysis view extends the multiple-view feature modelling paradigm, by offering an interface that is specific for dealing with analysis models, but is linked to the representation of the design model. This enables bidirectional propagation of changes between the two models.

The analysis view offers new challenges over other views in multiple-view feature modelling, as the geometry of the design and analysis model can be different. The analysis model can even be dimensionally abstracted.



We discuss the difficulties that this poses, and present an algorithm for linking a design and an analysis view, which enables propagation of changes between the two views. We discuss its implementation, and the interaction of the user with both views.

Finally, we present conclusions on what has been achieved, and reflect critically upon the issues that remain. We also look ahead which further improvements we might expect in the future with respect to the integration of design and analysis.



# Samenvatting

Productanalyse neemt tegenwoordig een belangrijke plaats in binnen de ontwerpcyclus van producten. Een dergelijke analyse maakt het mogelijk om na te gaan of producten aan de vereiste specificaties voldoen, helpt om producten te optimaliseren voor materiaalgebruik en fysische eigenschappen, en stimuleert innovatie door de verkenning van minder gangbare ontwerpen mogelijk te maken, zonder dat er geld en kostbare tijd hoeft te worden besteed aan het maken van testmodellen.

In het verleden was er maar beperkte interactie tussen ontwerp en analyse, aangezien deze als afzonderlijke disciplines zijn ontstaan. Met de toename van de mogelijkheden van beide disciplines, groeide de behoefte aan meer interactie tussen ontwerp en analyse. Gedurende de laatste decennia is er wel meer interactie gekomen, maar de historische kloof is nog steeds merkbaar aanwezig in de vorm van een duidelijk gebrek aan integratie tussen de twee taken.

In dit proefschrift houden we ons bezig met verbetering van de integratie van ontwerp en analyse in de context van feature modelleren. Dit houdt in dat de noodzaak tot het uitvoeren van omslachtige en overbodige handelingen bij de overgang tussen de ontwerp- en de analysefase wordt verminderd. Het dient te worden gerealiseerd door het gebrek aan hergebruik van informatie die tijdens verschillende fasen van het ontwerp- en analyseproces wordt ingewonnen, te verhelpen. Dit zal leiden tot goedkopere, betrouwbaardere en efficiëntere productontwikkeling.

We lichten er twee aspecten uit binnen de cyclus van ontwerp en analyse, en beschrijven verbeteringen op deze deelgebieden. Het eerste aspect betreft efficiënte meshgeneratie voor analysemodellen, en het tweede de integratie van ontwerp- en analysemodellen.

Het eerste aspect dat aan bod komt is het efficiënt opnieuw genereren van een mesh, d.w.z. een rooster van tetraëders, van een analysemodel. Meshes voor analyse van hoge kwaliteit zijn populair, aangezien ze de rekentijd kunnen verminderen, de kans op falen van de analyse verlagen, en in algemene zin de betrouwbaarheid van de analyse vergroten. Het genereren van

meshes van hoge kwaliteit kost echter meer tijd. Binnen de ontwerpcyclus wordt er meerdere keren analyse uitgevoerd, telkens nadat een reeks van modelaanpassingen heeft plaatsgevonden. Aangezien de wijzigingen in de geometrie, na zo'n reeks van aanpassingen, doorgaans beperkt en lokaal van aard zijn, is er ook een grote overeenkomst tussen de meshes van opeenvolgende analysemodellen. Wij gebruiken deze overeenkomst voor het efficiënt opnieuw genereren van een mesh voor analyse op basis van het eerdere mesh.

Ons werk is gebaseerd op het variational tetrahedral meshing algoritme. Dit algoritme genereert meshes opgebouwd uit tetraëders van een zeer hoge kwaliteit, maar het is niet echt geschikt voor toepassing op mechanische modellen. We bespreken de beperkingen en presenteren verbeteringen om deze te compenseren; de bereikte resultaten worden door voorbeelden ondersteund.

Om een mesh van een eerdere iteratie van een analysemodel te kunnen hergebruiken, hebben we een exacte geometrische beschrijving nodig van de overeenkomsten en verschillen tussen de twee modellen. Aangezien we met feature modellen werken, en features sterk gerelateerd zijn aan hoe wij mensen een model intuïtief beschouwen, ook wat de overeenkomsten en verschillen tussen twee modellen betreft, baseren wij onze aanpak op de features waar de modellen mee zijn opgebouwd. Voor elk feature bepalen we een beschrijving van het verschil en dit noemen we het *feature difference*. Het beschrijft hoe de geometrie en de interpretatie daarvan verschilt tussen twee modellen, *vanuit het gezichtspunt* van dat feature. Alle individuele feature differences samen, vormen het *difference model*. We geven een gedetailleerde uitleg van het feature difference begrip aan de hand van voorbeelden en bespreken de implementatie ervan.

Voortbouwend op het feature difference en de verbeteringen van variational tetrahedral meshing, presenteren we een methode voor het efficiënt opnieuw genereren van tetraëder meshes. De methode komt erop neer dat de overeenkomsten, op basis van de features waaruit ze zijn opgebouwd, tussen de twee modellen worden gezocht. Voor overeenkomstige gebieden kunnen mesh elementen van het eerdere mesh worden gekopiëerd naar het nieuwe mesh, en in de resterende gebieden worden nieuwe mesh elementen geconstrueerd. We zorgen ervoor dat de kwaliteit van het nieuwe mesh vergelijkbaar is met die van het eerdere mesh. Dit is met name van belang voor de gebieden waar mesh elementen van een verschillende herkomst bij elkaar komen. De winst in efficiëntie wordt geïllustreerd aan de hand van voorbeelden.

Het tweede aspect dat aan bod komt in dit proefschrift is de integratie van ontwerp- en analysemodellen. In de praktijk verschilt de representatie

van deze twee modellen vaak. Dit heeft tot gevolg dat het ontwerpmodel moet worden omgezet naar een geschikt analysemodel. We beschrijven de verschillen tussen de representaties, en een aantal aanpakken voor integratie die in het verleden zijn voorgesteld. Wij streven naar een nauwere integratie van de twee modellen door middel van *analyse-views*. De analyse-view is een uitbreiding van het multiple-view feature modelling concept. Door een specifieke interface voor de omgang met het analysemodel te bieden, maar deze te verbinden met de representatie van het ontwerpmodel, wordt het automatisch uitwisselen van wijzigingen tussen de twee modellen mogelijk gemaakt.

De analyse-view biedt nieuwe uitdagingen binnen multiple-view feature modelling, aangezien de geometrie van het ontwerp- en analysemodel kunnen verschillen. Het analysemodel kan zelfs een abstractie met lager dimensionale elementen zijn. We bespreken de moeilijkheden die dit met zich meebrengt, en presenteren een algoritme voor het verbinden van een ontwerp- en een analyse-view dat het automatisch uitwisselen van wijzigingen tussen de twee views mogelijk maakt. We bespreken de implementatie hiervan en de interactie van de gebruiker met beide views.

Tenslotte presenteren we conclusies op grond van wat er bereikt is, en reflecteren we kritisch op de nog onopgeloste kwesties. We kijken ook vooruit naar welke ontwikkelingen we in de toekomst mogen verwachten op het gebied van integratie van ontwerp en analyse.



# Curriculum Vitae

Matthijs Sypkens Smit was born on April 15, 1980, in Utrecht, the Netherlands. He graduated in 1998 from the Utrechts Stedelijk Gymnasium, a grammar school with a history dating back to 1474. In 2005 he completed the academic study Computational Science, focussing on large-scale scientific computing and modelling, at the Department of Mathematics of the University of Utrecht.

In October 2005 he started working as a Ph.D. student in the Computer Graphics and CAD/CAM Group, of the Electrical Engineering, Mathematics and Computer Science faculty at Delft University of Technology. The title of the research project was *Integration of design and analysis models*, and it concerned the improvement of the product design cycle by bringing geometric feature modelling and numerical analysis of these models closer together. The research culminated in the current thesis during the course of 2010.