Beautification of Reverse Engineered Geometric Models

A thesis submitted in partial fulfilment
of the requirement for the degree of Doctor of Philosophy

Frank Curd Langbein

June 2003

Cardiff University
Department of Computer Science

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

A copy of this document in various transparent and opaque machine-readable formats and related software is available at `http://www.langbein.org/research/BoRG/`.

To Lieselotte, Kurt and Sonia
for their patience and support.

Abstract

Boundary representation models reverse engineered from three-dimensional range data suffer from various inaccuracies caused by noise in the data and the model building software. *Beautification* aims to improve such models so that they exhibit exact geometric regularities representing the original, ideal design intent. In this thesis an approach to beautification as a post-processing step solely working with the boundary representation is investigated. Geometric regularities approximately present in the model are detected and a consistent subset of these regularities is imposed on the model. Only models of engineering objects whose surfaces can be represented by planar, spherical, cylindrical, conical and toroidal faces with sharp edges or fixed-radius rolling ball blends are considered. A large number of mechanical parts can be constructed from these surfaces and there are robust reverse engineering methods for them.

A novel approach to approximate geometric regularities is introduced. They are handled as approximate symmetries of discrete properties of boundary representation elements. This leads to new efficient detection methods for different approximate regularity types classified by the underlying symmetry type. Due to the ambiguity present in approximate models many approximate regularities are detected, which are unlikely to be mutually consistent. Hence, a consistent subset of regularities likely to represent the intended design has to be selected. Expressing regularities in terms of geometric constraints and interpreting constraints in a topological context results in a new efficient solvability test for constraint systems based on degrees-of-freedom analysis. In order to select likely, consistent regularities they are added sequentially in order of a priority to a constraint system. A regularity is selected if the expanded constraint system remains solvable. The selected constraint set is solved numerically and an improved model is rebuilt from the solution. Experiments show that this approach can be used to improve reconstructed models.

Acknowledgements

First an foremost, I would like to thank my supervisor, Ralph Martin, for invaluable discussions and advice throughout the course of this research, and for many helpful comments on the draft of this thesis.

I wish to thank my colleagues, Dave Marshall, Bruce Mills, and Chunhua Gao, for many helpful discussions on beautification. In particular, thanks to Bruce with whom I have worked closely on approximate regularities. I have enjoyed our numerous discussions on the subject and many other topics. Thanks to Dave for his comments on my work and proofreading this thesis.

I would also like to thank Pál Benkő and Tamás Várady from the Hungarian Academy of Sciences for helpful discussions on reverse engineering systems and CADMUS Consulting and Development Ltd. for providing reverse engineering software.

Furthermore, I wish to thank my colleagues from the Vision and Geometry laboratories, Ana, Darren, Gavin, Gundon, Jovisa, Julia, Paul, and Peter, for many stimulating discussions of ideas of mutual interest.

This research has been supported by the UK EPSRC Grant GR/M78267. I am also grateful to the Computer Science Department of Cardiff University for supporting me to finish this thesis in time.

A special thanks to the developers of free software without which the presented algorithms could not have been implemented on a stable, reliable platform. Science seeks to explain and understand the universe; engineering seeks to apply scientific knowledge to practical problems. Neither should be done for or controlled by commercial interest which seeks to restrict knowledge and its use making it an instrument rather than a purpose.

My deep gratitude to my parents, Kurt and Lieselotte, for supporting me all this time, and to Sonia, for her support and patience, for many useful discussions of this research and other ideas, for proofreading this thesis, and for keeping me entertained — a difficult, demanding task shared mainly with the Inkies, Candia, Tony, and Adam, and their fans.

Contents

List of Figures

List of Tables

List of Algorithms

Notation

$X \cup Y$	union of sets X and Y
$X \cap Y$	intersection of sets X and Y
$X \setminus Y$	subtraction of set Y from set X, i.e. X without Y
$Y \subset X$	set Y is contained in or equal to set X
$X \times Y$	Cartesian or direct set product of sets X and Y
$\overline{X}$	closure of set X
X°	interior of set X
$x!$	factorial of positive integer x
$\binom{x}{y}$	binomial coefficient of non-negative integers x, y
$\lvert X \rvert$	cardinality of set X or absolute value of real number X
$\lVert x \rVert$	norm of vector x (Euclidean norm or two-norm unless stated otherwise)
$\lVert x \rVert_\infty$	maximum norm of vector x
X^{-1}	inverse of matrix (or real number, function, etc.) X
x^t	transposed matrix or vector x
$f: X \to Y, x \mapsto f(x)$	f is a function from set X to set Y mapping each element x of X to an element $f(x)$ in Y
$=_\epsilon$	an approximence (a reflexive, symmetric relation that is an equivalence under certain conditions; see Section 2.3.2)
∇f	Nabla operator, i.e. gradient of function f
$\nabla^2 f$	Hessian matrix of second partial derivatives of function f
$\frac{\partial f}{\partial x_l}$	partial derivative of f with respect to x_l
ΔT_A	ΔT_P for length distances (see Section 3.1.2)
ΔT_F	ΔT_P for loop feature distances (see Section 3.1.2)
ΔT_L	ΔT_P for angular distances (see Section 3.1.2)
ΔT_P	minimum distance between two features in order for them to be considered different (see Section 3.1)
δ_k	Dirac distributions on the unit circle $\mathbb{S}^1$ (see Section 3.1.2)
ε	machine precision

$\omega(r)$	priority of regularity r (see Section 4.2)
$\omega_a, \omega_b, \omega_c, \omega_e, \omega_q, \omega_{ra},$ ω_{sv}, ω_t	merit functions and constants to compute the priority $\omega(r)$ (see Section 4.2)
$\Pi_{l=1}^{n} X_l$	Cartesian or direct set product of sets $X_1, \ldots, X_n$
π	ratio of the circumference to the diameter of a circle
$\sum_{l=1}^{n} m_l$	sum of $m_1, \ldots, m_n$
$\mathrm{avg}_P(x_1, \ldots, x_n)$	average of elements $x_1, \ldots, x_n$ of the metric space P
$\mathfrak{C}$	set of all cells in $\mathbb{E}^3$ (see Section 2.2.1)
$\mathbb{C}$	set of complex numbers
$\mathbf{C}^\infty$	indicates smooth, ∞-differentiable function or manifold
$\mathrm{ceil}(x)$	smallest integer greater than or equal to the real number x
$d_P(x, y)$	metric in metric space P; P omitted if obvious
$DF(x)$	functional matrix of partial derivatives of vector-valued function f at x
$\mathbb{E}^n$	n-dimensional Euclidean space
$\mathrm{floor}(x)$	largest integer less than or equal to the real number x
$\gcd(x, y)$	greatest common divisor of integers x and y
I	identity operation
$I\langle F \rangle$	ideal of F (see Section 5.2.2)
$L_{\max}$	largest length present in a geometric model (see Section 3.1)
$\mathbb{N}$	set of natural numbers, i.e. $\{0, 1, 2, \ldots\}$
$f(n) = O(g(n))$	there exist numbers C and n_0 such that $f(n) \leq Cg(n)$ for all $n \geq n_0$
$\mathfrak{P}(X)$	power-set of set X
$\mathbb{P}^n$	n-dimensional real projective space
$\mathbb{Q}$	set of rational numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	n-dimensional vector space over $\mathbb{R}$
$\mathbb{R}_+$	set of non-negative real numbers, i.e. $\{x \in \mathbb{R} : x \geq 0\}$
$\mathrm{round}(x)$	the integer closest to the real number x
$\mathbb{S}^n$	n-dimensional sphere
$S(n, e)$	*dependency sub-graph* of a node n due to an edge e in a constraint graph (see Section 6.1)
$\mathrm{sign}(x)$	$+1$, 0, -1 indicating the sign of the real number x
T_m	approximation of largest possible tolerance level for clustering features (see Section 3.1.1)

Chapter 1

Introduction

Reverse engineering geometric models extracts sufficient information from a physical object to reconstruct a CAD model for a particular purpose like redesign, reproduction or quality control. If we say that engineering converts a concept into an artefact, then reverse engineering converts an artefact into a concept. Ideally, for applications like redesign, the reverse engineered model should exhibit exactly the same geometric properties present in the original, ideal design. Rather than trying to create a very detailed and exact description of the physical object suitable for creating an exact copy or for inspection purposes, we are interested in reverse engineering the shape of an engineering object such that the description of the model represents the original design intent. For this purpose we use a state-of-the-art reverse engineering system which can create a boundary representation model representing the object's natural surfaces from dense three-dimensional range data. However, due to inaccuracies in the measured data from the object and approximation and numerical errors during the reconstruction process, this model is approximate in the sense that it exhibits intended geometric regularities such as symmetries only approximately. We propose to improve such models in a post-processing step, which we call *beautification*.

In Section 1.1 we start with an overview of current reverse engineering techniques and applications, focused on a typical reverse engineering system which is available to us. Then we describe the beautification problem in Section 1.2. In Section 1.3 we present previous approaches and our approach to beautification. In Section 1.4 we give an overview of the remaining chapters in this thesis.

1.1 Reverse Engineering Geometric Objects

There are many applications of reverse engineering. In the context of this thesis we are interested in reverse engineering shape and its geometric properties related to design intent. But note that a broader interpretation relating to functional properties and mechanisms is

also possible. For our purposes we may loosely classify reverse engineering applications into reproduction, inspection and redesign. By measuring a physical object, all of these applications aim to create a description of the object covering some aspect of its shape which is not available from other sources. The requirements for the model description differ depending on the application type. Note that we do not claim that this classification covers all possible applications and there may be other classification schemes. Here it is simply used to discuss some basic requirements for the model descriptions.

We limit the discussion to reverse engineering systems which start with measurements producing dense data sets to describe the surface of the physical object. For an overview of optical, tactile, acoustic and magnetic measurement methods see [130]. All these methods interact with the surface using some physical phenomenon to acquire positional data. The type of data analysis to obtain the positional information as well as its accuracy depends on the particular measurement method. In our particular case we use a three-dimensional laser scanner producing dense range data by triangulating the position of a laser beam projected onto the object. This produces data at high accuracy in relatively short time. However, an object has to satisfy certain conditions such that it can be measured, e.g. the object should have no deep cavities which cannot be probed by the laser scanner (see Section 1.2).

The aim of reproduction applications is to create a description of a physical object which allows its recreation. This is often necessary when the original drawings or CAD models were lost or are not available. The process is expected to create an exact copy of the original or at least a copy of the original which differs only at a relatively close tolerance. No real analysis or modification of the data is required and thus it can be achieved with a description close to the original measurements from the object (e.g. a point set or a triangular mesh). This type of application can be seen as the origin of reverse engineering [128]. The pantograph [117] was used to copy, at any predetermined scale, arbitrary geometric shapes over a mechanical linkage. Copy lathes and mills represent more contemporary, automated versions of this machine. The basic principle is to follow the surface of the source object mechanically and use this to steer the movement of a cutter device over the relations in a parallelogram. Only low-level positional information is required for this and no CAD model is involved.

However, the aim of reverse engineering is not simply to create a nearly exact copy of the physical object via a digitised data set or as another physical object. Our goal is to create a concept, i.e. a CAD model which can be analysed and modified. For this an appropriate description of the object's boundary has to be generated. For different types of objects different reverse engineering techniques are required. In this thesis we consider engineer-

ing objects rather than natural or artistic objects. We can classify engineering objects by the types of surfaces required to describe their shape. Some objects have a shape which can be represented by simple analytic surfaces like planes and natural quadrics. Other objects require more geometrically complex free-form surfaces. We only consider objects of the first type for reasons which will be laid out below (in Section 1.2 we give a detailed description of the objects we consider).

For inspection applications, very accurate data from the physical object is required in order to analyse the properties of the object. Depending on the properties considered, the object may be measured in different ways. We may, for instance, be interested in quality control and compare the measured data with an ideal data set. The main tasks involved are analysis of the data and matching it to an existing data set in the context of the particular application. Note that inspection also requires an existing specification of what we are looking for. For instance, in order to test the planarity of a surface, simply fitting a plane to it and replacing the data points by a plane would be inappropriate. Instead we require the error between a best-fit plane and the data points. But in order to test whether two planes are orthogonal we require an explicit representation of the planes to measure the angle between the normals. Note that this may be done from a few measurements rather than dense range data under certain assumptions about the quality of the surfaces involved. In general discrete measurements are not very reliable. The above two tasks may also be combined to a single task fitting two planes to the data sets under the constraint that they be orthogonal. But for inspection the measured data would have to be compared to the two fitted planes and it would have to be verified that the fitted planes are indeed orthogonal.

Similarly, in order to generate custom fits to human surfaces, for mating parts such as prostheses or helmets, we require accurate data about these surfaces. This data then has to be analysed appropriately to adjust the mating part. The basic requirements for this type of applications are quite similar to inspection applications.

For redesign applications it is necessary in general to have a higher-level representation of the object. While accuracy is still important, small variations like a scratch in a surface do not have to be captured (unless the scratch is deliberate). A suitable representation would, for instance, be a boundary representation solid model describing the object in terms of its natural surfaces. It is important that this representation contains accurate, intended geometric regularities in order for it to be useful for redesign. Applications include tasks where clay or wood models are used to create, evaluate and improve models (e.g. in the automobile industry). For another application an existing mechanical component is scanned, and based on its design, a new model is created. It may also be required to change an existing part such that it better suits its purpose. The focus of redesign applications is to

capture the design intent of the physical object. For this, we still require high accuracy, but the emphasis is on the object's geometric regularities such as symmetries, aligned cylinder axes, orthogonal or parallel planes, etc. In typical engineering objects these regularities are not random, but are present to fulfil a functional or aesthetic purpose.

Often CAD models generated by reverse engineering software are only approximate in the sense that they exhibit intended geometric regularities only approximately. The cause for this may be an imperfect physical object altered by wear or the particular manufacturing method used to make it. It may also be caused by inaccuracies in the reconstruction process introduced by numerical and approximation errors. Such problems may be eliminated in the model by explicitly considering design intent such that the created models are more suitable for modification in CAD applications. In the context of redesign our ultimate goal is to realize an intelligent three-dimensional scanner which is able to automatically determine the design intent of a physical object with only minimal human interaction. Such a system would provide a simple, high-level user interface for both engineers, and inexperienced and non-engineering users to reconstruct suitable CAD models.

In the following we give an overview of a state-of-the-art reverse engineering system upon which this thesis builds. This system was originally developed by several partners, including Cardiff University, in the EU funded RECCAD project. It has since been further developed by the Hungarian Academy of Sciences and CADMUS Consulting and Development Ltd. We are grateful to them for providing it for use in this project. Due to noise in the measured point set and noise introduced by the reconstruction algorithms, the resulting CAD models are unlikely to show all regularities expected by an engineer. This problem is specified in more detail in Section 1.2, and in Section 1.3 we present previous approaches and our approach to address it.

1.1.1 Typical Reverse Engineering Systems

In the following we give an overview of the main phases of a typical reverse engineering system. The system available to us follows these steps. See Figure 1.1 for a flowchart of the process. For a general overview of reverse engineering see Várady et al. [129, 130].

The process starts with a data acquisition phase where raw measurement data from the physical object is collected. Some pre-processing on this data is required in order to combine multiple measurements from different viewpoints. The most crucial phase of the process is the segmentation and surface fitting phase. Here the individual (natural) surfaces of the object have to be determined and surfaces of suitable geometric types have to be fit. The method first creates a triangular mesh for the object surface. Based on this

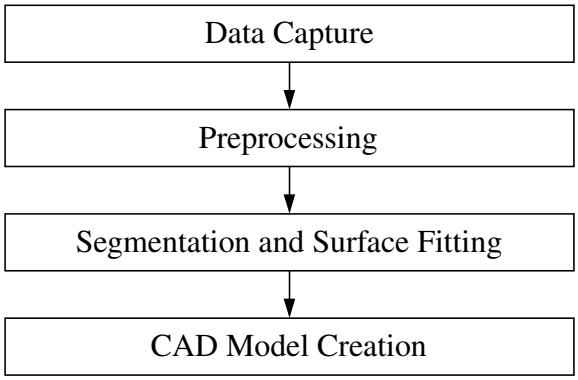


Figure 1.1: Main Reverse Engineering Phases.

mesh the data is segmented into subsets representing natural surfaces of the object. For each of the subsets, one or multiple analytic surfaces are fitted individually. However, the segmentation and surface fitting cannot be separated completely from each other, and the methods have to be carefully chosen to work together. Already during segmentation we have to consider the surface types to be fitted later to the subsets. Finally, after appropriate surfaces have been found, a complete boundary representation model is created by stitching the surfaces using adjacency relationships. In the following we give a more detailed description of the processes involved in each of these steps in the reverse engineering system we use.

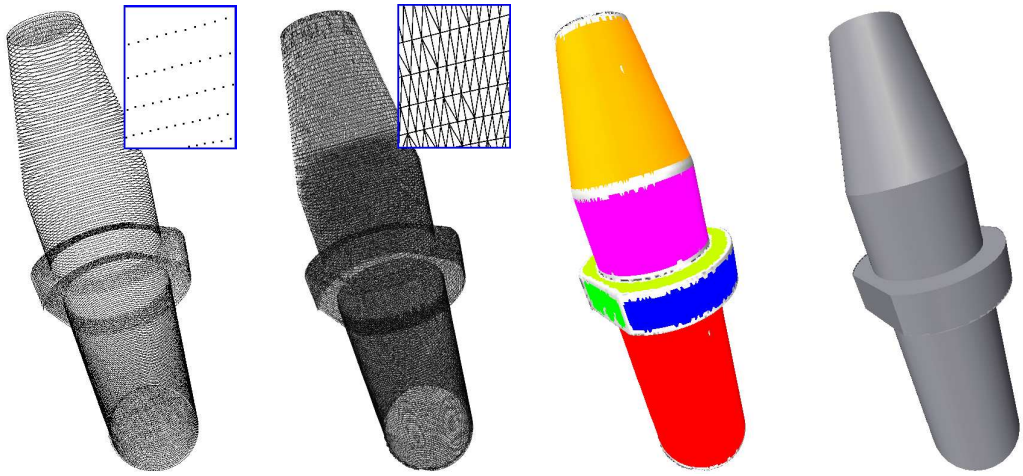
The particular approach taken starts with a data acquisition phase where multiple views of a physical object are obtained using a commercial three-dimensional laser scanner. Note that in order to capture data about all the surfaces of an object, we always require multiple views. The measured data is imperfect, and in particular it is noisy and incomplete, due to problems of precision, accessibility and occlusion which occur when measuring real objects.

In a pre-processing phase the multiple views are merged into a single point cloud. For this the user has to provide some approximate information about how the different views are related. This can be done either by identifying specially attached markers like calibration balls in the various views, or by providing approximate transformations representing the relations between the different views. In both cases we have a set of transformations telling us approximately how to combine the views into a single point set. This approximation can be improved by using methods based on the original iterative closest point (ICP) algorithm [12]. The iterative closest reciprocal point (ICRP) algorithm creates especially good results [70]. Both methods pair the closest points in the two views and compute a motion that minimises the mean square error between the paired points.

For ICRP a point pair is only considered if it represents the closest point from a point in the first view to a point in the second view and vice versa. There are various additional variants of the ICP method [115]. Registering multiple range views simultaneously as opposed to pairwise registration of views also improves the data correspondence as the sets are merged as a whole [34].

In the next step the point set is triangulated using an algorithm which merges locally defined triangulations into a consistent global triangulation [66]. Optionally, we can add a decimation step after the triangulation, for efficiency reasons. The aim of decimation is to reduce the total number of triangles in the mesh without changing the topology and only small changes to the geometry. Depending on whether a vertex lies on a boundary, on an internal edge, a corner, etc. in the mesh a criterion based on the distance of the vertex to a plane or an edge is used to decide whether the vertex should be removed [118]. The triangulated data set is the basis for segmentation, which splits the point set into disjoint subsets which mark connected regions of the points sharing some common property. The subsets are considered to represent the natural faces of the model. There are various approaches to segmentation, often based on region growing techniques. In this system, direct segmentation is used, i.e. a local property of the surface like surface normals or curvatures is approximated based on the triangulation, and the subsets comprise point sets that are connected in the triangulation and for which the local property is consistent. Points for which a clear decision cannot be made are not added to any subset: it is assumed that they are close to an edge, or lie on a blend.

To each of the subsets a surface (or surfaces) of appropriate type is fitted separately [10, 89, 134, 135]. Multiple surfaces may have to be fitted to a point subset in case of smooth edges between the surfaces, e.g. created by a tangential intersection between a cylinder and a plane. For single surfaces a faithful least-squares fitting algorithm, which is robust in the sense that the results become closer to simple surfaces, i.e. planes, cylinders, cones or spheres, as the principal curvatures of the surfaces fitted decrease or become equal [89]. First simple analytic surfaces are considered for fitting the point subset. If this is not possible, the subsets are considered to be *multiple smooth regions*, which have to be subdivided further [10]. If translational or rotational symmetries are found for these cases, a smooth sweeping profile is determined where curve segments correspond to surfaces. To fit the sweep profile a constrained curve fitting algorithm is used. For cases where this also fails more sophisticated segmentation has to be used and several surfaces have to be fitted simultaneously under tangency constraints. Note that segmentation cannot be fully separated from surface fitting: they highly interact with each other. For a successful segmentation we would first like to know the type of surface each point subset represents, but this is determined by the surface fitting methods, for which known point subsets are



(a) (b) (c) (d)

Figure 1.2: Reverse Engineering a Solid Model.

required.

After an appropriate surface (or surfaces) has been fitted to each of the subsets, the points left-over from segmentation between the subsets have to be considered to determine the adjacency information for the fitted surfaces. The left-over points can be employed to detect whether adjacent surfaces meet at sharp edges or are connected by blends [67]. For sharp edges the point subsets are expanded in a *fattening* step which assigns the left-over points to the most likely surface. This also yields a sharp edge between the surfaces, which can be recomputed by intersecting the fitted surfaces with each other. The resulting faces are stitched to create an *initial* boundary representation model.

We ignore blends in this thesis. However, Kós et al. [67] present a method to determine the radius of fixed-radius rolling ball blends from the left-over points. From this information we can instruct the geometric modelling kernel to construct appropriate blends. Hence, we can label edges between surfaces connected by blends with a blend radius and ignore blends for beautification. Afterwards the blends can be added by the kernel.

Figure 1.2 shows the stages of reverse engineering a solid model. Point set (a) in Figure 1.2 was obtained by scanning a simple physical object and registering multiple views of it. Figure 1.2(b) shows the triangulation created from the point set, Figure 1.2(c) shows the segmented model, and Figure 1.2(d) shows the reconstructed model.

The system described has certain limitations with respect to object types, especially the surfaces involved. In this thesis we only consider objects which can be handled robustly by the reverse engineering system. This means we consider engineering parts composed of only planar, spherical, cylindrical, conical and toroidal surfaces that either intersect at

sharp edges or are connected by fixed radius rolling ball blends. There are reliable surface fitting methods available for these surfaces [10, 89, 134, 135] and many realistic engineering objects can be generated using only these surface types [95, 116]. As described above blends can be identified and represented as edge and vertex attributes and thus we ignore them in the rest of this thesis.

Experiments with this system show that simple engineering objects of the above mentioned type can be successfully reconstructed. The main problems present in this system are related to registering multiple views and finding appropriate tolerance values for the segmentation.

As multiple views of a three-dimensional object are necessary in order to capture data from all its surfaces (assuming that all surfaces can in principle be measured by a three-dimensional laser scanner), we have to combine these views into a single data set. This requires information from the user about the approximate relations between the data sets. The registration algorithms are not perfect, and often there is still a relatively large error left. This is especially a problem if the data set does not have sufficient distinct features, such as sharp edges, for matching, e.g. for objects with a high degree of rotational symmetry as in Figure 1.2.

It is also often hard to find appropriate parameters for segmenting the model. The parameters indicate when to consider neighbouring geometric properties (computed from the triangulation) as different: each control parameter applies globally to the whole data set. Often parameter values which are appropriate for one part of the model do not work for another part. In some cases involving smooth edges the surface fitting algorithm can fit multiple surfaces, but this strongly depends on the tolerances used. For instance, a smooth edge between a relatively large cylindrical surface and a small plane may be too difficult to find and the whole surface may be fitted by a single cylinder. Problems like this show how closely the segmentation and surface fitting steps are related. For complex models more robust methods still have to be developed.

1.2 The Beautification Problem

State-of-the-art reverse engineering systems of the type described in Section 1.1.1 can create valid boundary representation models approximating physical objects. More robust methods for all the steps are still required, but there is already a relatively large class of objects which can be handled. However, the reconstructed models suffer from various inaccuracies resulting from sensing errors arising in the data acquisition phase as well as

approximation and numerical errors in the reconstruction process. This means that certain intended regularities, like parallel planes or aligned cylinder axes, may only be approximately present in the reverse engineered model. However, in order for such models to have the greatest usefulness to redesign applications, these intended regularities should be present as they are an important part of engineering designs.

We refer to this problem as the *beautification problem*. In this section we introduce the beautification problem and give a detailed description of the object types we consider and our assumptions for beautifying reverse engineered geometric models. In the following we refer to the models created by the above type of reverse engineering system as *initial models* as they still require improvement in order to represent the design intent.

Errors arising during the data acquisition phase may be reduced by improving the precision of the sensing techniques. Yet measurement will always be subject to a certain tolerance as a perfect sensor cannot exist. Further noise is introduced to the process by numerical methods used during reconstruction. This can be reduced by using numerically more stable methods, but it cannot be avoided as long as we are computing with floating point numbers. Furthermore, depending on the methods used for the reconstruction, certain assumptions about the data set are made and this influences the way the data is approximated. For instance, the error metric used to fit a surface to a point set determines how well the surface is approximated. Different error metrics favour different surface types and surface parameters to approximate the point sets. We refer to these types of errors as approximation errors. In general, sensing, numerical and approximation errors can be reduced by more stable methods. While this increases the precision, it cannot be increased arbitrarily. Thus, the resulting models will always be subject to a tolerance.

In our reverse engineering system the main sources of noise are the registration, segmentation and surface fitting steps as mentioned in Section 1.1.1. In particular, as surfaces are fitted independently to each point subset after segmentation, any special relations between the surfaces and other geometric entities are not preserved. While originally an orthogonality relationship between two planes may be well preserved in the point set, fitting planes independently to appropriate subsets may introduce a larger error. A local error in a planar point subset, which is relatively large compared to the error elsewhere in the subset, may cause a relatively large global error for a plane being fitted to the subset in order to minimise the overall error (we get many points with a small error rather than a localised larger error).

In order to represent the design intent we also have to consider other sources of error such as possible wear of the object and the particular manufacturing method used to make it. For instance, when using a mould to create a part, the side faces are usually slightly

rotated in order to ensure that the part can easily be removed from the mould.

Hence, to ensure that intended geometric regularities are present, they have to be deliberately enforced at some stage of the reverse engineering process. We aim to expand an existing reverse engineering system such that it can capture the design intent properly and improve the initial model with respect to design intent. We restrict ourselves to certain engineering objects which can be handled by our system as described in the following.

Due to the limitations of the surface fitting methods, we only consider objects whose surface can be represented by planes, cylinders, cones, spheres and tori. The regularities considered here are also less likely to occur in objects which require the use of free-form surfaces and are less useful for this type of object. Moreover, higher order analytical surfaces are rarely used in engineering objects. The objects may also contain fixed-radius rolling ball blends. We assume that these blends have been identified in the model and are represented by edge attributes. They can easily be added to the model using standard blend algorithms and we will ignore them for the rest of this thesis. Many typical mechanical parts fulfil these restrictions. However, our approach to beautification should be designed in a way that it can easily be expanded with respect to the surface types and the considered regularities.

We consider objects of limited complexity, i.e. with no more than about 200 faces. This is a reasonable limit achievable with the above reverse engineering system. The objects have to be physically small enough to be put on a typical three-dimensional scanner, e.g. they should fit inside a 50cm cube, and they should be light enough to be lifted onto and positioned on the scanner bed by a human. Any features have to be large enough (bigger than about 5mm) to provide sufficient data to be able to properly fit surfaces, and there should be no deep cavities that cannot be probed by a three-dimensional laser scanner. These assumptions also make it unlikely that we have to handle non-manifold models or other models which create complicated problems for the usual boundary representation. In general we refer to these objects as *pragmatic solids*.

As we consider design intent in terms of geometric regularities, the geometric properties of the objects should be significant for their application. This means that the particular shape of an object is important for its purpose. Often a shape that suits a certain purpose is also regarded as beautiful due to its symmetrical properties and other similar regularities. It also has to be possible to derive the regularities from the object alone. For instance, the shape should not have been designed to mate with another complicated shape or to fit in a particular space such that without access to additional information the object's shape would appear irregular or random. Parts designed on the same regularity principles are also likely to work well together as long as the regularities are suitable for the functions.

Hence, the type of regularities we consider (which were derived from a part survey; see Section 2.4) have to be suitable for the part.

Furthermore, we assume that the design intent of the object is of limited complexity. This is hard to specify exactly and we will only give the basic idea here. It mainly relates to principal limitations of what can be done if we have only approximate models. Given a certain tolerance we can only distinguish properties (e.g. cylinder radii, plane normals, etc.) which differ by more than this tolerance. Given a large number of properties from a complex part, we may get a sequence of properties which cannot be distinguished within the tolerance, but form a relatively large set. For instance, consider the cylinder radii $0.1, 0.2, 0.3, 0.4, \ldots, 10.0$ at a tolerance 0.5. They are obviously not all equal, but we do not have a clear criterion to decide which of them should be equal. Sets like this may be interpreted in many ambiguous ways. For details of the kind of problems complex design intent may cause, see Chapters 7 and 8.

A typical part will often have a major design intent like a global symmetry which is only broken by a few features. Many typical engineering objects also only have a small number of different "important directions" forming directional properties such as plane normals, cylinder axis directions, etc. In particular we may have an object generated from a block such that there are a small number of (one, two or three...) major orthogonal systems of directions, with a few additional directions present. If we have a complex object which generates a large number of directions with only small angles between them we cannot make decisions based upon them. Given, for instance, a regular prism with approximately 5° angles between the plane normals, we can detect it provided the initial model is accurate enough. However, if the directions do not relate to a prism, but to independent sub-parts of an object, it may be hard to decide which combinations of directions form regularities without additional information about the structure of the object. Some may relate to a prism, but others may be completely unrelated. The beautification system presented in this thesis is intended for objects of medium complexity, with only a relatively small number of independent sub-parts, and a design intent of limited complexity.

1.3 Approaches to Beautification

In this section we discuss various approaches to the beautification problem. We first consider previous approaches to the problem and then present an overview of our particular approach. In previous work machining features and geometric constraints were considered for including design intent in the reverse engineering process. Both approaches can

in principle be used for beautification, but little has been previously done on the beautification problem per se.

Thompson et al. [128] consider feature-based reverse engineering of mechanical parts. In their system a human identifies features like slots and pockets in the three-dimensional point set interactively. The system requires user identification of the type and the approximate location of each feature. This information can then be used to reconstruct the model by fitting parametric feature models instead of simple surfaces to the three-dimensional point set. The approach is in particular justified for engineering objects which are generated from a small number of machining operation types. Fitting feature models improves the accuracy of the generated models and they are in particular useful for feature-based CAD applications.

The human interaction is used because low-level information about an object's surface in form of point sets is often not enough to make decisions about higher-level design intent. We try to go beyond the requirement of human interaction by first extracting a higher-level model as a boundary representation with natural surfaces expressed as analytic surfaces. From this representation more information about the actual design intent of the object may be derived automatically. The feature-based reverse engineering system also only considers a given set of machining feature types. While this is often sufficient, more general classes of geometric regularities will provide a system which can handle more general objects. In principle a feature model can also be regarded as a collection of faces and other geometric entities specified further by certain constraints.

Geometric constraints are a useful tool for representing design intent in a CAD model. The topology of a boundary representation model gives the basic structure of the model. The geometry implicitly represents the detailed design intent by various parameter values. Geometric constraints can be used to explicitly specify the desired relations between the geometries in terms of the parameters. Thus, they are also an obvious choice for including design intent in reverse engineering. There are two different ways to use geometric constraints when building a model. They can be used during the surface fitting phase to ensure that surfaces are fitted to three-dimensional point subsets which satisfy necessary conditions describing the geometric relations between them. Alternatively, constraints can be used to improve the initial model without further reference to the three-dimensional point data. Here the constraints specify the desired relations between the geometries in the boundary representation model. A constraint system describing the design intent of the model can be solved to adjust the parameters of the initial model directly.

Benkő et al. [9] and Werghi et al. [133, 136] consider fitting multiple surfaces to three-dimensional point sets under geometric constraints. So, rather than fitting surfaces indi-

vidually, they are fitted simultaneously using the constraints as a set of conditions which the surface parameters have to fulfil in addition to providing a good fit to the three-dimensional point data. Thus, for instance, two planes might be fitted simultaneously under the constraint that they be orthogonal. An exact formulation of the problem, usually as a numerical minimisation problem where the constraints are either part of the objective function or constrain the parameter space for the optimisation, together with efficient and numerically stable algorithms for finding a solution are given in [9, 133, 136].

In order to use constraints for reverse engineering, we require some way to obtain them in the first place. The above approaches to constraint fitting assume that the regularities are available from some other source. One possible source is for a user to manually specify the constraints. This is tedious and error-prone, and can often lead to inconsistent constraint systems. A system which can automatically detect constraints is more desirable. For constrained fitting we could first create an initial model without constraints, then use this model to determine appropriate approximate regularities, which could then be used to determine the constraints for a constrained fitting phase. However, this approach requires constraints which are consistent with the fitting conditions and the constrained optimisation methods employed are quite expensive due to the large number of data points. In general it is hard to ensure automatically that the constraints are consistent with surface fitting as the constraints and the fitting conditions are not on the same type of geometric entities. The fitting conditions relate the point subsets to the surface representations, the geometric constraints are between the surface parameters and parameters for other elements in the boundary representation. There are efficient graph-based methods for constraints which may yield a fast consistency test without requiring to solve any equation system.

For automated constraint detection we require a regularity detection process to find appropriate approximate regularities. As the regularities are only approximately present in the initial model, the set of all detected approximate regularities is likely to contain inconsistencies, i.e. unintentional or accidental regularities which conflict with the intended ones. This means we have to select a consistent subset of the regularities found. Consistency here has to consider two aspects. On the one hand the constraint system derived from the regularities has to be solvable as otherwise a suitable model cannot be realised at all. On the other hand the selected regularities should also describe a consistent, likely design intent.

In principle constrained fitting can lead to an accurate and correct model, but it is computationally very intensive. Constrained fitting is an optimisation process trying to satisfy fitting conditions and constraints simultaneously. For each point subset describing a natural surface of the object, a distance between the fitted surface and each point in the subset

has to be computed in the objective function. In addition the constraints limit the domain over which the optimisation runs by some condition. Surface fitting and constraint satisfaction both require numerical computation with large data sets.

In this thesis we propose instead to only refer to the initial model and improve it without further reference to the point set. This means we first detect geometric regularities in the initial model and then impose a consistent set of them on the model to obtain an improved model. This would only require a constraint solver and no constrained fitting, which is much more efficient as it operates on a smaller data set. It only involves the geometric elements of the boundary representation of the initial model, rather than the considerably larger point set. Solving the constraint system can be done in the context of these elements which allows us to argue about the consistency of the constraints using a graph without solving any equation systems (see Chapter 6). If we detect sufficient regularities in the initial model, it may be possible to select a consistent set of regularities which constrains all parameters of the model such that only one uniquely determined model can be derived from it. This means that using the point data is not required as no further adjustments can be made to the model without violating at least one of the constraints. In the case that the constraints allow for multiple models, we can still use the model which is in some sense closest to the initial model.

Independent of how the improved model is built (by solving a constraint system on the model or by constrained surface fitting), methods to handle geometric regularities in terms of geometric constraints are required for an automated system. For this we require methods to determine approximate geometric regularities in the initial model and a process to select regularities consistently. As constraint solving is less computationally intensive than constrained fitting, we propose to reconstruct the model from the solution of the constraint system without further reference to the point data. We also expect to detect a large number of regularities which are likely to over-determine the model. Thus, we have to select an appropriate consistent subset of regularities, which is likely to completely determine the model. Hence, using the point data in addition to the constraints is unlikely to change the solution of the constraint system. Instead it will make it harder to find a solution as it will introduce inconsistencies between the regularity constraints and the fitting conditions. These inconsistencies are hard to determine and resolve in terms of which set of conditions and constraints should be chosen, and algorithms for this appear to be computationally very expensive. While the fitting conditions may be given a higher priority it is not simple to determine which of the constraints are consistent with them as the geometric entities they refer to are not the same.

We now give an overview of *beautification* as a post-processing step for the initial model,

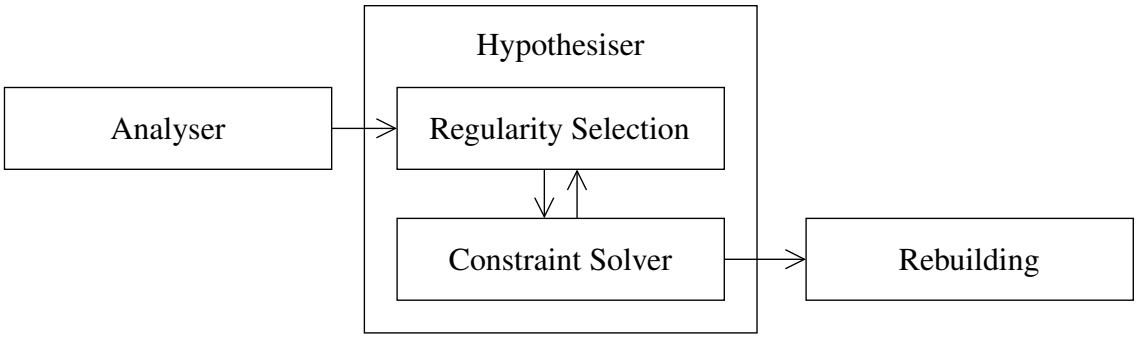


Figure 1.3: Beautification Strategy.

as proposed in this thesis. It can be seen as a part of the model building phase. To beautify a model means to detect and enforce certain regularities which are almost or exactly present in the initial model. We also have to take care to stay as close as possible to the initial model and keep the modifications as small as possible such that the regularities are fulfilled without introducing unnecessarily large changes in the model. In order to improve an initial model we may have to change the geometry of the elements of the boundary representation and also the topology of the boundary representation. In the context of this thesis we only consider changes to the geometry and assume that the topology is correct. Some topological changes, like removing small faces which are caused by segmentation problems, may be done before the beautification here. Others, like fixing holes or adjusting blends to ensure that a valid, complete model is created, may be done afterwards. Such topological changes are considered separately [40]. Methods developed for healing, i.e. repairing models which are considered to be broken when imported into another CAD application, e.g. due to different model representation and tolerance schemes, may also be useful in this context. See, for instance, [8, 104].

Our proposed beautification process consists of three main steps as shown in Figure 1.3. First we *analyse* the initial model to detect approximate geometric regularities expressed by geometric constraints, e.g. we look for approximately aligned cylinder axes, approximately orthogonal planes, etc. As these regularities are only approximately present, the detected regularity set is likely to contain inconsistencies. Hence, in a *hypothesiser* step we select an appropriate consistent subset of constraints which represents the likely original design intent. In the hypothesiser we must check whether the selected regularities represent a solvable constraint system, which is done by the *constraint solver*. The *regularity selection* method takes care of selecting regularities which are likely to represent the original design intent. Note that these two components of the hypothesiser cannot be com-

pletely separated from each other. Finally, the constraint solver has to compute a solution to the selected constraint system from which an improved model can be *reconstructed*.

1.4 Overview

We now give an overview of the remaining chapters in this thesis. The general structure follows quite closely the component structure of our proposed beautification system. We first discuss regularities and their detection (Chapters 2 and 3). Then we introduce a regularity selection strategy focused on the likelihood of a regularity being part of the original design intent, under the assumption that we have a method to decide the solvability of a constraint system (Chapter 4). In the following chapters we discuss geometric constraint systems and methods to determine the consistency of geometric constraint systems (Chapters 5 and 6). Finally we present the results of some experiments (Chapter 7) and end with some conclusions (Chapter 8). Note that beautification has not yet been considered as such in previous work. While related work exists, the presented concepts and algorithms represent a novel approach to beautification. We will give references to related work in the individual chapters as appropriate. The novel work presented in this thesis has been published in [72, 74, 75, 76, 77, 94] or is accepted for publication [73].

In Chapter 2 we start by introducing a novel general concept for approximate geometric regularities in the initial model. Our notion of approximate regularities has to be suitable for improving the model in subsequent steps, and we require efficient methods to detect them. By considering combinatorial properties together with geometric properties of the elements involved in a boundary representation, we derive a precise concept of approximate regularities suitable for identifying regularities in engineering objects. We introduce special properties as features (which are not machining features like slots and pockets, but points in a property space). These features change in the property space in a similar way to the way in which a model changes in the three-dimensional Euclidean space under isometries. Approximate symmetries of these features form the basis for our approximate regularities. This new concept covers a wide variety of approximate regularities. Theoretically it is justified to call them regularities as they are based on symmetries and we also have empirical evidence of their presence in typical engineering parts.

Our approach for detecting approximate regularities as a combinatorial problem of limited complexity leads to efficient algorithms presented in Chapter 3. By combining the combinatorial and geometric properties of the features, we derive detection algorithms which provide exact, non-arbitrary results indicating the presence of approximate regularities. We derive various general methods for detection of certain regularity types which can be

applied to detect particular regularities. New algorithms for the detection of approximate global symmetries, special types of approximate partial symmetries, and approximate congruences of discrete sets of points in a metric space are introduced. Furthermore, a novel algorithm for detecting special values for scalar values based on continued fractions is presented.

Regularity detection considers various relations between the elements used to represent a CAD model to find potential regularities. These do not have to be mutually consistent. In particular, as we aim to describe the whole model by constraints, we have to find many regularities. Hence, in Chapter 4 we present a general selection strategy to select a subset of regularities likely to be wanted for the ideal design of the model. The selected regularities must be mutually consistent and should be likely to represent the original, ideal design intent. The likelihood of a regularity being part of the original, ideal design is computed as a priority. In order of priority, regularities are added to a constraint system which is tested for solvability. Only regularities which lead to solvable constraint systems are selected for the final set of constraints. In particular our selection strategy separates the detection from the actual decision which regularities should be enforced. This decision consists of two aspects, which cannot be completely separated: selection of consistent regularities and selection of intended regularities.

One of the crucial subtasks of the regularity selection strategy discussed is the test whether a consistent set of regularities remains consistent if an additional regularity is added. For this we require an efficient solvability test for constraint systems. In Chapter 5 we discuss constraint systems, and different approaches to determine their solution and their solvability. We also present a numerical approach based on an optimisation method to find the solution of a constraint system and consequently determine whether it is solvable.

A more sophisticated approach to the solvability problem based on degrees-of-freedom analysis, interpreted in a topological context, is presented in Chapter 6. By analysing the topological dimensions of the parameter spaces used to describe the geometry of the model, and how constraints limit these dimensions, we can determine the solvability of a constraint system in a generic sense without actually solving the system. By considering the topological properties of the spaces involved we derived a novel method to determine the solvability of a constraint system, which is closely related to degrees-of-freedom analysis. Our new topological interpretation also improves the understanding of the structures underlying degrees-of-freedom analysis.

In Chapter 7 we present the results of experiments which beautify various models using our beautification system. We have tested the system with simulated and real three-dimensional point sets captured from test objects, from which initial models have been

created. Our beautification system was then used to improve these models. Certain principal restrictions for beautifying an approximate model exist due to the ambiguity introduced by accepting certain tolerance levels. But the experiments showed that we can improve reconstructed models with respect to design intent with our approach.

In summary in this thesis we investigate a novel approach to automatically improve reverse engineered geometric models with respect to design intent, in a post-processing step working solely with the boundary representation model. In particular we make the following main contributions to the solution of the problem:

- A theoretical framework for a certain type of approximate geometric regularities in boundary representation models based on approximate symmetries.
- Efficient algorithms for the detection of approximate regularities relating to congruences, repetitions, global symmetries and special values in boundary representation models.
- A strategy to select mutually consistent regularities described by geometric constraints which are likely to represent the original, ideal design intent of the reverse engineered model.
- A topological interpretation of geometric constraints leading to an efficient test to determine the solvability of geometric constraint systems.

Chapter 2

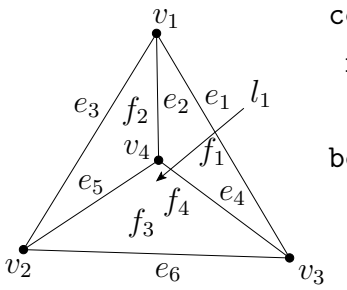
Representation and Regularity of Shape

For beautification we need a concept for approximate geometric regularities in the initial model which may be part of the original, intended design. These regularities have to be suitable to improve the model in the subsequent steps and we require efficient methods to detect them. In this chapter we introduce a boundary representation of geometric models. Based on this representation we discuss in detail exact and approximate geometric regularities. This leads to a precise notion of approximate regularities suitable for identifying regularities in engineering objects. It is based on an exact definition of approximate symmetries as regular arrangements of subsets of discrete sets describing certain properties of the elements (features) used to represent the model in special property spaces. We derive a simple general concept for detecting approximate regularities. We also list the particular regularities chosen for improvement of the models. The particular detection methods are described in Chapter 3.

In [74, 75, 76, 77] the author developed novel concepts and methods for regularities relating to subsets of all Objects used to represent the geometric model. In collaboration with B. I. Mills an algorithm and the theory to detect global symmetries of point sets and boundary representation models was developed in [93, 94]. Both approaches are unified here into a single, general framework for geometric regularities of boundary representation models.

2.1 Boundary Representation

Our beautification system has to handle pragmatic bulky solids as discussed in Section 1.2. In this section we will give a brief introduction to the representation of such solids. A solid is a bounded subset of three-dimensional Euclidean space $\mathbb{E}^3$, which has a non-empty interior and is closed. Hence, we define the term *solid* to refer to bounded point sets equal to the closure of their interior, and whose topological boundary is the union of a finite number of manifolds. Usually the manifolds should be orientable and smooth [100]. Note



$$
\begin{aligned}
\texttt{cells} &= \{v_1, v_2, v_3, v_4, e_1, e_2, e_3, e_4, e_5, e_6, f_1, f_2, f_3, f_4, l_1\} \\
\texttt{rank} &= \{(0, \{v_1, v_2, v_3, v_4\}), (1, \{e_1, e_2, e_3, e_4, e_5, e_6\}), \\
&\quad (2, \{f_1, f_2, f_3, f_4\}), (3, \{l_1\})\} \\
\texttt{bound} &= \{(e_1, \{v_1, v_3\}), (e_2, \{v_1, v_4\}), (e_3, \{v_1, v_2\}), \\
&\quad (e_4, \{v_3, v_4\}), (e_5, \{v_2, v_4\}), (e_6, \{v_2, v_3\}), \\
&\quad (f_1, \{e_1, e_2, e_4\}), (f_2, \{e_2, e_3, e_5\}), \\
&\quad (f_3, \{e_1, e_3, e_6\}), (f_4, \{e_4, e_5, e_6\}), \\
&\quad (l_1, \{f_1, f_2, f_3, f_4\})\}
\end{aligned}
$$

$r : \texttt{cells} \to \mathfrak{P}(\mathbb{E}^3)$

Figure 2.1: Boundary Representation of a Tetrahedron.

that many of the concepts, like the term solid, introduced in this chapter apply to more general spaces, but we restrict the discussion to $\mathbb{E}^3$.

We represent the structural information about a solid by a *boundary representation*, also known as a *geometric realisation of an algebraic complex*. It has been proven to be useful for both practical and theoretical uses [6, 29, 30, 37, 50, 98]. Figure 2.1 illustrates the boundary representation of a tetrahedron. It is represented as a set of vertices, edges, faces and lumps, which we call *cells*, with the structural information stored in a `rank` and a `bound` relation. Each cell has a name such as e_1 or f_1 and the `rank` relation states the dimension of the cells. The `bound` relation lists which cells are contained in the boundary of a cell. This forms the algebraic complex, which is usually referred to as the topology of the model. The geometric realisation, $r : \texttt{cells} \to \mathfrak{P}(\mathbb{E}^3)$, assigns each cell a point set in $\mathbb{E}^3$, i.e. each vertex is mapped to a location, each edge is mapped to a curve, each face is mapped to a surface, and each lump is mapped to a three-dimensional subset of $\mathbb{E}^3$. The realisation is usually referred to as the geometry of the model. The algebraic complex and the geometric realisation form a *geometric complex* or a *boundary representation*.

We assume that the realisation $r(c)$ of a cell c is path-wise connected and closed in the relative topology of an appropriate subspace, i.e. a cell is a solid in an appropriate subspace. The `rank` relation classifies each cell according to some condition which essentially relates to the cell's dimension. The intersection of the realisation of two cells should always be the realisation of one or more cells of lower rank. For an element $(c, \{b_1, \ldots, b_n\})$ of `bound` the rank of c is larger than those of the b_l. The geometric realisation of each boundary cell b_l is a subset of the boundary of the realisation of c, i.e. $r(b_l) \subset r(c) \setminus r(c)^\circ$ for all l, and the boundary of c is created by the b_l, i.e. $r(c) \setminus r(c)^\circ = \bigcup_{l=1}^{n} r(b_l)$ in the relative topology of an appropriate subspace. We get the usual notions of a boundary loop of a face and a boundary shell of a lump by partitioning the boundary sets in `bound` into

connected subsets. In the following we will often identify the cell c with its realisation $r(c)$. Note that a realisation $r(c)$ of a cell c can also be interpreted as an embedding of an abstract manifold c into $\mathbb{E}^3$.

A boundary point p of a polyhedron P may be classified by the `rank` relation as a vertex, edge or face according to whether the minimal dimension of the intersection of P with a plane through p is 0, 1, or 2 respectively, locally at p. The path-wise connected components of the sets of each type of point form the natural cells of the polyhedron. By construction it is apparent that the realisation for a face is a planar region, for an edge a linear segment and for a vertex a point. The realisation in combination with the dimension and the boundary relations form a boundary representation of the polyhedron.

More generally the realisations of the cells may be any continuous map from planar regions, line segments, and points. In case these realisations still form well-defined faces with proper boundary relations we will still get a proper boundary representation. However, defining the rank of a point in the solid is more complicated and not uniquely determined. For instance, consider a cone with its apex. If we define the dimension in topological terms, then the apex is of rank 2 as there is a neighbourhood of the apex which is homeomorphic to an open disc. If we require in addition that the surface is differentiable, then the apex is of rank 0. Thus we would have to create an additional vertex and extend the `bound` and `rank` relations accordingly.

A *stratification* [120] of a set $X \subset \mathbb{R}^n$ is a partition of X into $\mathbf{C}^\infty$ sub-manifolds $\{X_l\}$ such that $\{X_l\}$ is locally finite at each point of X. A stratification $\{X_l\}$ satisfies the *frontier condition* or *weak frontier condition* if $(\overline{X_l} \setminus X_l) \cap X_k \neq \emptyset$ implies $\overline{X_l} \supset X_k$ or if the set of the connected components of all X_l satisfies the frontier condition respectively. Our boundary representation structure can also be interpreted as a stratification of a solid satisfying one of the frontier conditions. Our cells become the manifolds (strata) X_l. Using a local regularity criterion such as that the points of the strata are of the same topological type (there exist two homeomorphic neighbourhoods for each point pair in a strata) or the Whitney condition requiring the strata to be smooth can be used to generate the structure. In particular this can generate the `rank` relation. Gomes et. al. [45] describe this in more detail for geometric models. For the mathematical background see Shiota [120]. This concept has also been used for the Djinn interface [6]. A detailed discussion of the definition for the dimensions of points for the `rank` relation and stratifications is beyond the scope of this brief introduction. However, since we begin with a boundary representation, it is not typically an issue for beautification.

Finally note that the geometry of the faces in our reconstructed models are limited to planar, spherical, cylindrical, conical and toroidal surfaces as stated in Section 1.2. Con-

0-dimensional cells:	**2-dimensional cells:**
Vertex	Planar face
1-dimensional cells:	Spherical face
Straight edge	Cylindrical face
Circular edge	Conical face
Elliptical edge	Toroidal face
Intersection edge	**3-dimensional cells:**
	Lump
1-dimensional connected boundaries:	**2-dimensional connected boundaries:**
Polygonal loop	Shell
General loop	

Table 2.1: Cell Types and Connected Boundaries for Our Solids.

sequently the edge geometries may be any intersection curve which can be generated by these surface types. For our solids we only consider cells of the types listed in Table 2.1. The main cells are the faces and the vertices, but we also consider the edges in certain cases. As we always assume that we have a complete solid, each edge is an intersection of faces, but identifying straight, circular and elliptical edges may still be useful when detecting certain regularities. Lumps in this context are general path-wise connected three-dimensional manifolds as subsets of $\mathbb{E}^3$. In addition to this we also handle connected boundary cells of other cells as loops and shells. We specifically consider loops which form a (flat) polygon in order to easily handle regularities relating to planar faces.

2.2 Geometric Features

We describe and detect regularities in terms of properties of the cells in a model (see Section 2.3). A property is in general an attribute in some property space attached to a single cell or a group of cells. It has a *type* and a cell or a group of cells can have multiple properties of the same type. For instance, a torus has two radii as length properties, a centre as a position and a direction for the axis. To give a property a geometric meaning we require that it changes in a similar way to the cell under isometric transformations. For instance, the radii of a torus are not changed by an isometry, but translations and reflections change the centre, and rotations and reflections change the direction of the axis.

For the purpose of regularity detection and later for detecting the solvability of constraint

systems to improve a model (see Chapters 5 and 6) we only consider special properties which we call *features*. Note that here features do not relate to machining features like slots or pockets. They could, however, under certain conditions be used to describe machining features as well. A property is called a feature if the property space, now also called *feature space*, fulfils certain conditions such that it is suitable for regularity detection as well as constraint solving. We require that features not only change in a similar way to the cells under isometric transformations, but that the relations between features of different cells transformed by the same isometry are also preserved. An exact definition with examples is given below.

2.2.1 Properties and Features

A property of a cell is an element of some property space P, e.g. normals of planar cells in $\mathbb{S}^2$. P represents the type of the property. We can associate a cell with its properties by a function, $f:\ \mathfrak{P}(\mathfrak{C}) \to \mathfrak{P}(P)$, where $\mathfrak{C}$ is the set of all cells in $\mathbb{E}^3$. Some elements of $\mathfrak{P}(\mathfrak{C})$ may be mapped to the empty set, i.e. they do not have a property of the type P. f maps a set of cells to a set of all properties of the type P associated with the set of cells. f can also be seen as a relation on $\mathfrak{C} \times P$ and often f in this form is actually a function on a subset of $\mathfrak{C}$ if each cell has exactly one property of the given type, e.g. the radii of all spherical cells. Defining f as a function on the power-sets simplifies the notation. We will always ignore the sets in $\mathfrak{P}(\mathfrak{C})$ which are mapped to the empty set in $\mathfrak{P}(P)$.

In order for f to be geometrically meaningful we require that a property of a cell changes in the same way as the cell under isometric transformations. This means for each isometry $T:\ \mathbb{E}^3 \to \mathbb{E}^3$ there is a mapping $T^*:\ P \to P$ such that

$$f(T(C)) = T^*(f(C)) \text{ for all } C \in \mathfrak{P}(\mathfrak{C}) \tag{2.1}$$

under the extended action of f, T and T^* on sets and T operating on the geometric realisation of the elements of C such that it maps a cell (or a group of cells) in $\mathfrak{C}$ to another cell (or group of cells) in $\mathfrak{C}$. Note that more generally we could choose another transformation group from which we choose the T.

For instance, for a cylinder the radius and the direction of the axis are two properties. We can map the cylinder to the value of its radius in $\mathbb{R}_+$ and the direction of its axis in a direction space. A direction can be represented as a point on the unit sphere $\mathbb{S}^2$. However, in the context of regularity detection we do not consider the orientation or sidedness of cells and thus we cannot distinguish between opposite directions for an axis. Hence, we identify opposite directions in $\mathbb{S}^2$ which creates the two-dimensional real projective

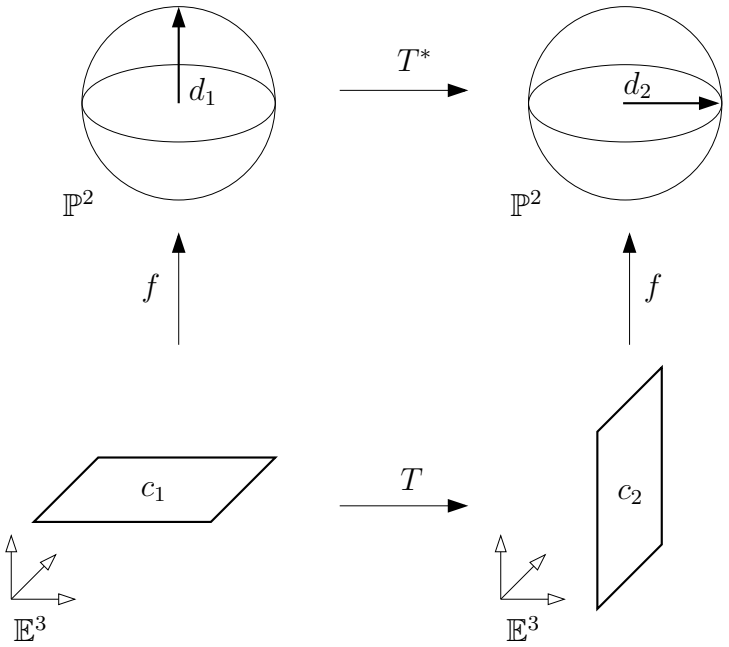


Figure 2.2: Diagram of Mappings for Property Definition.

plane $\mathbb{P}^2$ as directional property space. $\mathbb{P}^2$ is also generated by taking all lines through the origin. After transforming the cylinder by an isometry, it has the same radius as the original cylinder. Thus T^* for the length space is the identity as isometries do not change lengths present in the set of cells they transform. The direction may be changed by rotating and reflecting the cell. Using only the rotation and the reflection parts of an isometry T we get T^* on $\mathbb{P}^2$. Similarly the normal of a plane is a directional property. The diagram in Figure 2.2 visualises (2.1) for plane normals, where T maps a cell c_1 to c_2 and f maps c_1 and c_2 to the properties d_1, d_2 respectively which are related by T^*.

As an example for something that is not a property in terms of the above definition, consider the minimal distance of any cell in $\mathfrak{C}$ from the origin. f maps each cell on its minimal distance from the origin. Let T be a rotation around some axis which does not contain the origin. Let S be a sphere with a centre on this axis such that p is the closest point on S to the origin, i.e. $f(S) = d(p, 0)$, where d is the Euclidean metric in $\mathbb{E}^3$. Rotating S by T will not change its distance from the origin, i.e. $T^*(d(p, 0)) = d(p, 0)$. Now let R be a planar face through p such that $f(R) = d(p, 0)$. In general rotating R with T will change its distance from the origin, i.e. $T^*(d(p, 0)) \neq d(p, 0)$. Hence, this f is not a property.

We call a property a *feature* if the property space is a path-wise connected, smooth, abstract manifold with a metric and the T^* in (2.1) are isometries. The manifold conditions mainly relate to the constraint solving algorithms and will be discussed in Chapters 5 and 6. The metric with the condition that the T^* are isometries is used for regularity de-

tection. It essentially means that the relative relations between features of different cells are preserved under isometric transformations. For the directions in $\mathbb{P}^2$ we can define a metric by taking the smaller angle between the two lines for the directions and get a directional feature space.

Note that not every property is a feature. For vertex cells we can define a property by mapping each vertex position v to a non-uniformly scaled position Av, e.g. by doubling the x-coordinate, where A is represented by a matrix with different non-zero entries on the diagonal and zeros elsewhere. This is a property and for a given T we have $T^* = ATA^{-1}$. However, it is not a feature as T^* is not isometric.

2.2.2 Selecting Suitable Features

There are many options how to choose features for the cells. Many of these options may select equivalent features with different representations, i.e. the feature spaces are isomorphic such that the action of each T^* is preserved by the isometry. We are only interested in really different features. The main purpose of the introduction of features is to describe geometric aspects of cells by discrete point sets suitable for efficient regularity detection and constraint solving. The selection of features may also depend on the particular domain of a class of objects and the related desirable regularities. We will first discuss some general principles of how features could be derived from cells and then list our particular choice of features for the solids we consider for beautification. The particular selection is intended to be suitable for improving general engineering objects. For applications with a more narrow range of objects, or objects with other surface types, or other applications, a different selection may be appropriate. Our methods can in principle deal with any such features.

Following Klein's Erlanger Program [64, 141] a geometric property of a cell (or a group of cells) in $\mathbb{E}^3$ is any property which remains invariant under isometric transformations of $\mathbb{E}^3$. For a straight edge, the length, or for a sphere, the radius, are such features. For them T^* is the identity independent of the isometry T. As such features relate to internal properties of the cell or the group of cells from which they are derived, we call them *intrinsic features*.

Given, for instance, two orthogonal planes, we could refer to the $\pi/2$ angle between the plane normals as an intrinsic feature of the cell pair. However, as we intend to detect such arrangements using each angle between the normals of each plane pair and, more generally, the set of angles between the normals of a set of planes, this method is not suitable. We would have to generate features for each set of planes or even each set

of faces in the boundary representation. Instead, we choose to define the normal for planar faces as a single directional feature. This feature type is no longer invariant under isometric transformation, but changes in the same way that the plane does when rotated and reflected. As the T^* for the directional features is an isometry in $\mathbb{P}^2$, the relations between the normal features of a set of planes transformed by an isometry T remain invariant when transformed by T^*. This is the main reason why we require T^* to be an isometry for features. We call such features *extrinsic features* as they are used for relations between cells.

For the constraints, we have to consider another aspect of the features. In a torus, for instance, we can define the minor and the major radius, and the sum and the difference of these two radii, as intrinsic features. However, the minor and major radii depend on the radii sum and difference. To handle these dependencies we should choose one set of features as being dependent on the other set of features. We call the dependent features *extended features* and the primary ones *base features*. These dependencies create additional constraints later. Note that extended features depend on base features, but are not necessarily completely determined by base features. To avoid problems in the dependencies there should be a clear distinction between extended and basic features without any loops in the dependencies.

Another principle for defining features is to concentrate on basic properties of the cells. For instance, for finite cylindrical cells we could define the sum of the radius and the length in the direction of the axis as a length feature. However, in general it is unlikely that this feature would have any meaning for shape regularities. For similar reasons we do not consider features like the area of a face. The area may depend on functional purposes, e.g. minimal surfaces, and simple area properties are already represented by features like edge lengths and radii.

Sometimes it may be possible to create features of features. For instance, for cylinder axes as features we could define the intersection point of cylinder axes as a feature of the cylinder axis features. However, we will always define these in terms of the underlying cells in the solid. We must take care to record the dependencies between such features properly.

Our features have types, indicated by the feature space in which they exist. We primarily consider positional, directional, length and angle features. Positions are points in $\mathbb{R}^3$ with the usual Euclidean metric. Directions are points in $\mathbb{P}^2$ with the smaller angle between the two lines representing these points as metric. For lengths we have $\mathbb{R}_+$ (including 0) with the absolute value of the difference between two points as metric. For angles we use $\mathbb{S}^1$ with the shorter distance on the circle between two points as metric.

In this context the length of a straight edge and a radius of a cylinder are both length features. It is, however, useful to consider sub-types of features like lengths or radii for certain regularities to avoid generating too many not very likely relations as regularities. Furthermore, we also label the features with their specific function in the cell, so that we can distinguish between things like the major and the minor radius of a torus. Here it is important to note that each feature is not just a point in a feature space, but also has a label identifying the cell it relates to and its particular relation to the cell. While the points in the feature space for two distinct features may be equal, their labels are not and thus the features remain distinguishable.

In Table 2.2 we list the basic features we use for beautification for the cell types listed in Table 2.1 (note that we do not define features for lumps, shells, general loops and intersection edges as these do not appear to play a major role for general geometric regularities). Besides listing features and their type for each type of cell, we also mark the features as extrinsic (E) or intrinsic (I), and possibly extended (X). For a degenerate torus, we only use the radius sum if the outer hull, sometimes referred to as an *apple* torus, represents the surface, and the radius difference is used if the inner hull, sometimes referred to as an *lemon* torus, defines the surface. For non-degenerate tori we use the radius sum and difference.

In the table we introduce two new feature types: *axis* and *polygon*. An axis is a line in $\mathbb{E}^3$ and for our cells it represents a central rotational symmetry axis. It is represented by a direction coupled with a position, and usually depends on a directional and sometimes a positional feature of the cell. We use it to detect special arrangements of axes where we employ information gathered about the directional and positional features of the cells. More details are discussed with the related regularity detection algorithms in Chapter 3.

We also consider polygonal loops of planar faces for regularities and define a special feature type for these. The details will be explained in Section 3.1.2. Basically we represent each polygonal loop of a planar face by an n-dimensional vector over $\mathbb{R}$ and use the Euclidean metric to find similar polygonal loops. We define a feature for each polygonal loop rather than for the face to simplify handling planar faces with multiple loops. For each of these loops we also have a *root point* which is the centroid of the vertices of the loop. Combining the root point with the plane normal generates an axis.

The edge features can be regarded as optional, because in simple models regularities relating to them only mirror the regularities between the faces. Table 2.2 also only lists the basic features used for regularity detection. In addition we use an intersection point feature to represent intersections of axes of cell pairs having axes. These are handled as positional features to detect axis intersection regularities. Furthermore, we introduce one-

Cell(s)	Features	Type	Class
Vertex	Location	Position	E
Straight Edge	Edge direction	Direction	E
	Axis	Axis	EX
	Distance between end-points	Length	I
Circular Edge	Centre	Position	E
	Normal of circle plane	Direction	E
	Axis	Axis	EX
	Radius	Length	I
	Angle of circle segment	Angle	I
Elliptical Edge	Centre	Position	E
	Normal of ellipse plane	Direction	E
	Axis	Axis	EX
	Major direction of ellipse	Direction	EX
	Minor direction of ellipse	Direction	EX
	Major radius	Length	I
	Minor radius	Length	I
Polygonal Loop (group of edges)	Root point	Position	E
	Axis	Axis	EX
	Polygonal loop	Polygon	I
Planar Face	Normal	Direction	E
Spherical Face	Centre	Position	E
	Radius	Length	I
Cylindrical Face	Axis direction	Direction	E
	Axis	Axis	EX
	Radius	Length	I
Conical Face	Apex	Position	E
	Axis direction	Direction	E
	Axis	Axis	EX
	Semi-angle	Angle	I
Toroidal Face	Centre	Position	E
	Direction	Direction	E
	Axis	Axis	EX
	Major radius	Length	I
	Minor radius	Length	I
	Sum of radii (unless lemon)	Length	IX
	Difference of radii (unless apple)	Length	IX

Table 2.2: Basic Features of Cells Marked Extrinsic (E), Intrinsic (I), Extended (X).

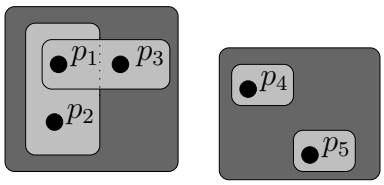
and two-dimensional partial position features as positions in $\mathbb{E}^1$ and $\mathbb{E}^2$. They are used to represent points which are equal when projected onto special lines or planes. We also get an angular feature from the angle between two directions, which is only used if the angle is not part of a more complex regularity (see Section 3.2.1). Additional auxiliary features are introduced to detect regularities, but are not used directly to represent them. These will be introduced as needed in Chapter 3.

2.3 Geometric Regularities

In this section we introduce our concept for geometric regularities of boundary representation models. We only consider regularities related to the geometric nature of the model. We do not consider regularities related to the function of the model or other properties not directly related to shape. We first define exact regularities based on regular arrangements (symmetries) of the features discussed in the previous section. We then expand this to approximate regularities where the congruence between sets in the exact case becomes an approximate congruence and additional conditions are employed to get well-defined answers for the presence of an approximate regularity.

A simple regular arrangement, which we will use quite often, is the congruence of cells with respect to features, e.g. equal radii or lengths or parallel directions. We can express this by saying that a certain set of features remains invariant under the identity transformation of the feature space. More generally consider the isometries in the feature space. A set of features which remains invariant under a sub-group of the isometries represents a regularity. It is possible to expand this to other groups of transformations. Usually it is not the whole set of features of a particular type derived from a model which remains invariant under the transformation group. We have to find appropriate subsets, which are maximal, i.e. there is no set which contains the subset and remains invariant under the same transformation group. For instance, we may have eight positions which exhibit cubic symmetry and we cannot expand the set of eight positions by other positions from the model such that the expanded set still has cubic symmetry.

We can expand this further and include incomplete symmetries, i.e. there is a subset which would be symmetric if certain additional features, which are not present, are added. However, note that any set can usually be expanded to be invariant under any transformation group, so we require some additional conditions. The basic idea is that there is sufficient information present in the feature set such that sufficient transformations can be derived from it which generate the group. E.g. given a set of equi-spaced points on a line we



Sets at tol. level ϵ_1:

$C_1 = \{p_1, p_2\}$
$d(p_1, p_2) < \epsilon_1$
$C_2 = \{p_1, p_3\}$
$d(p_1, p_3) < \epsilon_1$
$C_3 = \{p_4\}$
$C_4 = \{p_5\}$

Sets at tol. level ϵ_2:

$C_5 = \{p_1, p_2, p_3\}$
$d(p_1, p_2) < \epsilon_1$
$d(p_1, p_3) < \epsilon_1$
$\epsilon_1 < d(p_2, p_3) < \epsilon_2$
$C_6 = \{p_4, p_5\}$
$\epsilon_1 < d(p_4, p_5) < \epsilon_2$

Figure 2.3: Approximence Relations between Points at Two Tolerance Levels.

have an incomplete translation symmetry, but we require at least two points in order to determine the distance and a third point to avoid trivial cases.

In the case of an exact regularity the feature set and its image under one of the transformations in the group are exactly the same. This congruence generates an equivalence relation between the elements of the feature set. The equivalence relation leads to clean concepts and efficient detection algorithms (see Section 2.3.1). In the approximate case the congruence is only approximately present. This generates a relation which is reflexive and symmetric, but not necessarily transitive. In the following we call such a relation an *approximence*. The lack of transitivity generates ambiguous situations where global information is required for algorithms to work correctly.

In certain situations an approximence can be transitive when restricted to the relevant features. In these cases we can use concepts similar to the exact case for efficient detection algorithms. Essentially we detect tolerance levels at which the approximence behaves transitively on the relevant features. For instance, consider the points in Figure 2.3 where we seek approximately equal positions, i.e. we try to match the set with itself. At tolerance level ϵ_1 we have four clusters of points and as two of the clusters have one point in common the related approximence is not transitive. If we increase the tolerance to ϵ_2 we get two clusters and the approximence becomes an equivalence. Note that when considering the right clusters alone the approximence is locally transitive with respect to the two points at both tolerance levels. This is important for approximate models subject to different local tolerances.

2.3.1 Exact Geometric Regularities

We first define an exact regularity of a solid model in terms of its features. Let F be the set of features of a particular type in a feature space P derived from a solid. F is a discrete

set of points in P. Each element of F is labelled to uniquely identify it and relate it to the cells of the model. The label also allows us to distinguish two features with the same value in P. In order to describe exact regularities we only consider two elements of F to be the same if their distance in P is 0. For computational purposes, we test for equality by checking if the distance is smaller than the machine precision ε. Thus the assumption is valid if the distinct features in F are more than ε apart from each other.

Let G be a transformation group on P preserving the structure of P, i.e. a group of isometries in P. Let R be a subset of F such that for each $g \in G$ we have $g(R) = R$, where the action of g is expanded to sets. Furthermore, R is maximal, i.e. there is no superset $R^* \subset F$ of R for which this is true for G. We call (G, R) an exact regularity of F and thus an exact regularity of the solid. The type of the regularity is indicated by the Group G and the feature space P. For instance, if G is the symmetry group of a cube D_4 and P is a positional space we have a cubic symmetry regularity. Using the sub-types of the features in P this can be refined to sub-type regularities which only limit the elements of F considered, but not the feature space.

For a regularity (G, R), the elements of G are associated with permutations of the features in R. Let the feature set R be a set of mutually distinct features $r_1, \ldots, r_n$ and let $g \in G$. Then g induces a permutation σ of the labels $1, \ldots, n$ of the features in R. For $g(r_k) = r_l$ we get $\sigma(k) = l$. In order to avoid ambiguities we have to assume that the values of the features in R are mutually different. Otherwise g induces more than one permutation. In this context identity regularities become a special case only associated with the identity permutation. After identities have been detected and identical features are replaced by single features representing the identical feature sets, we can use the result to find non-trivial symmetries as permutations.

Let R be a set of mutually different features $r_1, \ldots, r_n$. We say that a permutation σ of the labels of features in R is *distance preserving* if for each pair $r_k, r_l \in R$ we have $d(r_k, r_l) = d(r_{\sigma(k)}, r_{\sigma(l)})$. Any isometry g for which $g(r_k) = r_{\sigma(k)}$ for $k = 1, \ldots, n$ induces the permutation σ. In general g is not unique for a given σ as R may lie in a sub-space of P. Let $G(\sigma)$ be the set of isometries on P inducing a distance preserving permutation σ on F. The $G(\sigma)$ for the set $\Sigma(F)$ of all distance preserving permutations σ on F describe an equivalence relation $\sim$ on the set G of all isometries under which R is invariant in P. For exact regularities G is always a group and factoring it by the equivalence relation created by the $G(\sigma)$ generates a unique matching between permutations and the elements of the factored group $G_{/\sim}$. Hence, by detecting distance preserving permutations $\Sigma(F)$ we find the factored group $G_{/\sim}$ which contains sufficient information to describe the relations between the features. Note, however, that we may also have to consider the sub-space of

P that contains R in order to enforce those relations using constraints.

For instance, for n points $p_1, \ldots, p_n$ arranged in sequence symmetrically around a circle in $\mathbb{E}^3$, the permutation $\sigma : l \mapsto 1 + (l \bmod n)$ is induced by a $2\pi/n$ rotation around an appropriate axis or by a reflection of the points at the plane of the circle in combination with this rotation. By detecting all distance preserving permutations of the points we find the rotations and reflections in the plane of the circle that keep the points invariant. This represents the factored group $G_{/\sim}$. Expanding these isometries to the whole space $\mathbb{E}^3$ gives G.

If R is the set of *all* features F of a particular type (or sub-type) derived from a solid, we say that the regularity (G, R) is a *global* regularity. Global symmetries of positions are an example for global regularities. For other regularities we may first have to find an appropriate subset of F in order to find a particular symmetry, e.g. a subset of axes symmetrically arranged on a cylinder. As this means that in addition to finding the symmetries we also have to find an appropriate subset, detecting such regularities is computationally more expensive than detecting global regularities. We call these regularities *partial* regularities. For partial regularities we only present detection methods for special cases where we can use conditions that avoid the consideration of all subsets of F to avoid computational intractability (see Section 3.2). For instance, we only consider axes with parallel directions and not the set of all axes.

We have an *incomplete* regularity if there is a superset R^* of some $R \subset F$ such that R^* is invariant under some transformation group and R contains all elements that are in R^* and F. An incomplete regularity can be global or partial, i.e. it may or may not relate to all features in F. As it is always possible to expand a given set R to a symmetric one there has to be sufficient evidence in R to justify the regularity. We base this justification on the presence of transformations that can be derived from R directly and generate the group G. A group G is generated by elements $g_1, \ldots, g_n$ if any element g of G can be represented as a product of the g_l where the same element g_l and its inverse can occur multiple times in the product. For instance, a $\pi/2$ rotation around an axis generates the group of $k\pi/2$ rotations around this axis. If we partition R into subsets R_l we can detect the generators in R by detecting transformations which map the R_l onto each other. The more often we detect the same generator or transformations g that can be generated by a single generator, i.e. $g = g_l^r$ for some $r \in \mathbb{N}$, the more likely the incomplete regularity related to the generator is present.

Consider, for instance, n points $p_1, \ldots, p_n$ arranged in sequence symmetrically on a circle in $\mathbb{E}^3$. We choose to ignore the reflections and so the regularity is created by $k2\pi/n$, $k \in \mathbb{Z}$, rotations. By removing a single point a $2\pi/n$ rotation still maps some of the

remaining points onto each other and we still detect some $k2\pi/n$ rotations between the points. The more points we remove the less often we will find these rotations. Other rotations, especially rotations by larger angles, may still exist, though (even as complete regularities). Also note that if there are only two points left we could still argue that there is a reflection present and even some evidence for a rotation. However, as this is true for *any* point pair there is not sufficient evidence for an incomplete regularity. The precise conditions for incomplete regularities are discussed with the particular regularity types in Section 3.2.

In particular, repetitions are suitable for incomplete regularities. If G is generated by a single transformation g, i.e. $G = \{g^r : r \in \mathbb{Z}\}$, we call G a *repetition*. If g is a translation in some direction then the regularity is necessarily incomplete as we only have finitely many features. However, if we have sufficiently many points on a line a fixed distance apart, we can still refer to it as an incomplete regularity.

In order to detect regularities in solids, we have to specify which feature types or subtypes we intend to consider, and which type of symmetries we are looking for, e.g. only the identity, any isometry, only rotations, only reflections, etc. We can detect regularities by essentially considering distance preserving permutations of the features. For this we first have to detect identity regularities of the features. For *partial regularities*, we use in addition certain conditions limiting the feature subsets considered. For *incomplete regularities*, we need conditions describing the allowed symmetries with respect to the feature set.

There is another type of regularity which we can detect if we have a structure in P which explicitly selects special values. For lengths we have a special point 0 and from there we can detect special distances from this point as special length values. Similarly for angles and the special angle 0. But for directions and positions, for instance, we only consider relative relations. There is no other structure unless we add some reference frame which would also become a set of features in these spaces, again yielding relative relations. It appears that the usefulness of special values is limited to intrinsic features.

A variety of approaches to the detection of exact symmetries in point sets exists. These are essentially the methods that could be used to detect exact regularities. Exact symmetry detection of planar sets of points and lines is possible in $O(n \log n)$ time [137]. Detecting symmetries of point and line configurations and three-dimensional polyhedra in $\mathbb{E}^3$ has the same time order [126]. The basic approach for exact symmetry is to sort the points according to the distance from their centroid and then check how many there are at each distance, which essentially reduces the problem to the complexity of sorting algorithms. A general method for detecting partial symmetries, which is of low polynomial time order

(a) (b)

Figure 2.4: Points with Approximate Cubic and Pyramidal Symmetry.

depending on the space and types of symmetries, is discussed in [16]. For partial symmetries, multiple centroids have to be considered. One approach is to sort isosceles triangles (a, b, c) with respect to the rotation they describe by mapping $a \mapsto b$ and $b \mapsto c$. Note that symmetries by reflection in the exact case can easily be determined by appropriately combining the $\binom{n}{2}$ possible point pairs that can be exchanged by reflection. All of these methods are limited to the exact case as they rely on matching the points exactly. Further algorithms for detection of exact symmetry are given in [58, 87, 88, 127].

2.3.2 Approximate Geometric Regularities

To beautify reverse engineered geometric models, we require a concept of approximate geometric regularities. This new concept must generalise exact regularities, and include it as a distinguished special case. In the exact case, regularities are symmetries of feature sets. In the approximate case, we define them in terms of approximate symmetries. Instead of transformations that leave the feature set invariant, we seek transformations that map the feature set onto another set which is only approximately congruent to the original feature set. There are various definitions for this (see Section 2.3.3). Our main goal is to find a notion of approximate symmetry that is suitable for efficient detection methods in the context of geometric modelling.

One computational aspect of exact symmetry is its implication of local congruence, i.e. the fact that something matches locally, automatically means that it matches globally. This admits greedy detection algorithms which try to gradually extend partial symmetries to complete ones. In the approximate case the greedy approach does not work in general. Locally there may be matches between approximately equal features of the model which are globally invalid such that backtracking is required. For instance, given the two sets $A = \{1, 2, 3, 1.2\}$ and $B = \{1.2, 2.2, 3.2, 0.8\}$ we can try to match the elements at a tolerance 0.3. If we start with the elements in A in sequence as listed we match 1 to 1.2,

2 to 2.2 and 3 to 3.2 and then we don't find a match for 1.2. Thus we have to backtrack trying in principle all other matchings in order to find a match within the given tolerance, which in this case would be matching 1 to 0.8, 2 to 2.2, 3 to 3.2 and 1.2 to 1.2. But note that while this matching works, we cannot really tell whether 1 should be matched to 0.8 or to 1.2. We present a way to restore the validity of the greedy approach by constructing a definition such that non-transitive cases cannot occur.

Recall that we can associate symmetries in the exact case with distance preserving permutations of the feature set. In the approximate case we can expand this to approximately distance preserving permutations which map distances to each other within some given tolerance ϵ. Because in the approximate case the transformations are hard to determine and not uniquely defined, we will use approximately distance preserving permutations to define approximate symmetries.

Let F be a set of features of a solid in some feature space P and let $R \subset F$ be a set of n features $r_1, \ldots, r_n$. Let $s =_\epsilon t$ if and only if $|s - t| < \epsilon$ for $s, t \in \mathbb{R}$ and let $D(R) = \{d(r_l, r_k) : l, k \in L\}$ for $L = \{1, \ldots, n\}$. A pair (ϵ, σ) of a positive real number ϵ and a permutation σ is an approximate symmetry of R, if $=_\epsilon$ is an equivalence relation on $D(R)$, and $|d(f_l, f_k) - d(f_{\sigma(l)}, f_{\sigma(k)})| < \epsilon$ for all $l, k \in L$.

For instance, the set of eight three-dimensional points in Figure 2.4(a) has all the approximate symmetries needed for cubic symmetry. By moving the points on the top face closer together, as in Figure 2.4(b), some of the approximate symmetries detected in case (a) do not yield an unambiguous matching for the points. Hence, (b) has only the approximate symmetries related to a square pyramid.

Combining the approximate symmetries of R at a tolerance ϵ generates a set of approximately distance preserving permutations G. Furthermore, we require that R is maximal with respect to the approximate symmetries in G. In a similar way to the exact case, we call (ϵ, G, R) an approximate regularity of F and thus of the solid from which F was derived. An approximate regularity with $\epsilon = 0$ is equivalent to an exact regularity. The notions of global, partial and incomplete regularities also apply to approximate regularities.

Note that G does not have to be a group, as it is not necessarily closed. For example, the point set $R = \{(10, 1), (0, 10), (-10, 1), (0, -10)\}$ admits the permutation corresponding to $T(x, y) = (-y, x)$ as an approximate symmetry for $\epsilon = 1.5$, but not $T^2(x, y) = (-x, -y)$: see Figure 2.5 which shows R marked by circles and $T(R)$ and $T^2(R)$ marked by squares. In practice, the permutations detected often form a group, but this cannot be guaranteed. Working with groups, however, is computationally intensive. We use methods

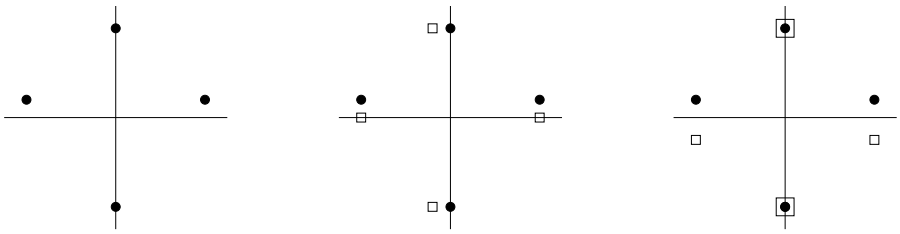


Figure 2.5: Example for Approximate Symmetries which are not Closed.

which quickly determine the set of permutations that preserve the distances, obtaining more information about the object. If a group is required then a maximal sub-group, or the minimal super-group of the set may be used.

By requiring that the approximately distance preserving permutations in G are exactly distance preserving in an appropriate sub-space of P which contains R, we get the exact relations for the features to fulfil the exact regularity. Based on this we can then associate the permutations with isometries in P. In cases where G is not a group we may have to make it into a group in order to ensure that the specified relations are unambiguous.

For an *identity* regularity we only have to detect an $=_\epsilon$ relation which is an equivalence on $D(F)$. This creates a transitive clustering of F into equivalence classes. The smallest ϵ values for which we get a new equivalence relation $=_\epsilon$ represent the consistent tolerance levels present in the feature set. In Section 3.3.1 we will present a clustering algorithm specifically designed to detect those tolerance levels for global symmetries. For other regularities we will describe a clustering method in Section 3.1 which uses two additional parameters to reduce the number of different tolerance levels to only the "interesting" levels, which depend on the error in the model, and which cut off too large tolerance levels.

After detecting the identity regularities we can replace the equivalence classes at a tolerance level by single features (of the cells that the original features relate to) and detect approximate symmetries of these features at this tolerance level. Depending on the regularity, we may choose only features of a special type or sub-type, or those which fulfil certain conditions, e.g. only select axes approximately parallel to some direction. Detection methods for these symmetries are described in Section 3.2 and 3.3.

For the definition of approximate symmetry we require $=_\epsilon$ to be an equivalence relation on $D(R)$ such that R is clustered into equivalence classes. However, each element of R is associated with one or more cells in the boundary representation. Unless each R is related to a unique cell the equivalence classes of R will not create equivalence classes of these

cells. In certain situations, as will be described in Section 3.1, this is, however, desirable. For instance, consider axis intersections. We can detect them by creating (approximate) intersection positions of $\binom{n}{2}$ axis pairs of n axes and cluster these positions. Then $=_\epsilon$ is an equivalence relation on $D(R)$ if for a cluster with m elements all $\binom{m}{2}$ distances between the elements are smaller than ϵ. However, the elements of the cluster may not represent the intersections of all axes related to the intersection. If l axes are involved in the cluster of axis pairs, then the cluster has to have $\binom{l}{2}$ intersection points of axis pairs in order for all intersection points of the axes to be considered. It could be said that we transfer the transitivity condition from the axis pair intersection point features to the axis features used to generate them.

As for exact regularities, we also consider approximate *special value* regularities. This simply means that a feature value is close to the special structure identifying special values in the feature space (see Section 3.4).

2.3.3 Alternative Approaches to Approximate Regularities

Our definition of approximate regularity is based on a definition of approximate symmetry. In the following we discuss the relations between our definition of approximate symmetry and alternative approaches.

One approach to detecting approximate symmetry is to compute a measure of asymmetry [91, 143]. This could be a real number, determined from information about an object, which is zero if and only if the object is symmetric. Typically precisely defined, and often easy to compute, these measures can be highly useful, for example in the determination of the effect of asymmetry of molecules on the melting point of a material [143]. But such measures suffer from calibration problems when applied in geometric modelling. It is difficult to design a measure that will agree with the human visual system over a wide range of models. Furthermore, the precise threshold at which an object is declared to be symmetric or not is arbitrary, based on supposition and subjective evaluation.

Another approach is to find a symmetric object with a small distance (e.g. Hausdorff [48]) from the object in question. Although there is a certain arbitrariness in the choice of metric, this often does not matter for determining the nearest symmetric object. There are two discrete approaches for this. Alt et al. [5] determine a group of transformations such that the images of the set under the transformations remain approximately equal. The time order for this approach is high-polynomial depending on the precise context. Iwanowski [55] tries to find a set with exact symmetry which is approximately equal to the original set. This approach is NP-complete. The reason for the high time order can be

seen intuitively in the case of multiple sequences of points with little distance between one and the next, but a large distance between the first and the last, e.g. two clusters of points arranged on two straight lines. It is not clear which point should be mapped to which, and a combinatorial search may be required to determine this (also see introductory example in Section 2.3.2). We assert that for our application the situation that leads to this problem is not interesting. If the points are too close with respect to the error bound it is hard to tell what sort of object is involved. For these cases it may be more appropriate to replace the points by a curve or a surface or by a single point with larger tolerance bounds.

Using Alt's approach [5] we acquire information about transformations which keep the set approximately the same. From this we do not immediately know how to rectify the object. We could try to compute an induced permutation for each transformation. If found, we could use this information to rectify the object. But we might not find such a permutation, because this approach does not always respect the structure of the object. Consider using the Hausdorff distance and the set $R = \{1, 2, 28, 29, 30\}$. The isometry $T(x) = 31 - x$ produces $T(R) = \{1, 2, 3, 29, 30\}$. The Hausdorff distance between R and $T(R)$ is 1. The structure preserving maps of sets are the bijections. But any bijection between R and $T(R)$ produces a minimum distance of 25, which is not at all close.

Iwanowski [55] detects a symmetric set approximately equal to the original set, which is potentially more useful for rectification. However, as a consequence of producing more information, the computation is NP-complete. The difficulty in this approach is that the overlap of error bounds may be non-transitive leading to a non-local matching problem. We avoid this by suggesting that as a property of a solid, the non-transitive case does not justify the assertion of approximate symmetry in the original object. The approximate reflective symmetry of $R = \{1, 2, 28, 29, 30\}$ about 15 is better expressed as a symmetry of the object obtained by replacing each cluster of points by a single point.

The relation between our approach and the two discrete approaches is complicated. Both of the above approaches begin with a specification of the group of symmetries which the method should detect. In our approach we try to determine which symmetries exist. While we may determine a description of the result as an abstract group afterwards, it plays no explicit role in the actual computation. This is an advantage since the pragmatic task is to *find the symmetries of this set* rather than *detecting if the set has such and such a symmetry*. Similarly our method automatically determines suitable tolerance levels.

Moreover, both methods have to handle spatial isometries, in the knowledge that any exact isometry chosen might not be the correct one. If we deal with only one isometry per potential symmetry, we have the problem of determining the candidate. Alternatively, if we deal with the set of all suitable isometries then we require a method for describing them

symbolically. A method of handling generic sets of isometries would be complicated, and not justified. We solve this problem by converting it into a combinatorial problem, in which the potential symmetries are expressed as a permutation selected from a finite set. In this way, each permutation we examine corresponds to a set of isometries but no explicit computation with these sets is required.

2.4 Common Regularities for Engineering Objects

We have surveyed about 600 mechanical components to determine common geometric regularities which are likely to occur in objects we wish to beautify [95]. These objects included small engine parts, fittings and brackets for optical systems, plastic fittings, caps and connectors, sliding fittings for cupboards, a general selection of CAD models from online repositories and company catalogues, and parts from other surveys.

A generic algorithm to detect approximate global and partial symmetries suitable for all of our feature spaces could detect most of our general regularity types. However, a method which could particularly handle *general* partial symmetries appears to be computationally intensive, as a large number of subsets would have to be examined for approximate symmetry. An efficient algorithm does not yet exist. Moreover, for incomplete regularities we require conditions for extending the sets to complete symmetric ones. Therefore, the survey was conducted to identify *specific* common cases so that we could concentrate on developing efficient methods for specific regularities.

About 97% of the parts exhibited important geometric regularities which could be classified using our regularity concept. This justifies our approach of trying to exploit such regularities to improve the quality of reverse engineered models. Common regularities are given in Table 2.3 for each feature type. We consider direction, axis, (axis) intersection, position, partial position (generated by projection of points onto special lines and plane), length and angle, and polygon features. Note that after detecting axis intersections we treat the intersection positions in the same way as positions and do not list the other regularities separately for them. We list a general description of the regularity together with the involved symmetry types, and whether it is a global (G), partial (P) or incomplete (I) regularity. We also have special values (S) of scalar parameters and ratios between scalar parameters of the same types. For each of the regularities we list the general type of isometries used for the symmetry group in Table 2.3. The number in the last column indicates how common the particular geometric regularity is with 5 being nearly always present to 1 being rare as determined manually. Later chapters will show how we detect and enforce such regularities.

Feature Type	Regularity	Symmetries	Class	Freq.
Direction	Parallel directions	Identity	G	5
	Symmetries of directions	Isometries	G	1
	Rotational symmetries of directions	Rotations	PI	4
Axis	Aligned axes	Identity	G	3
	Parallel axes arranged equi-spaced along lines and grids	Translations	PI	3
	Parallel axes arranged symmetrically on cylinders	Rotations	PI	2
Intersections	Axes intersecting in a point	Identity	G	3
Position	Equal positions	Identity	G	2
	Point set symmetries	Isometries	G	3
	Equi-spaced positions arranged on a line or a grid	Translations	PI	3
	Positions arranged symmetrically on a circle	Translations	PI	1
Partial Position	Equal positions when projected on a special line or plane	Identity	G	3
Length/Angle (Scalar)	Equal scalar parameters	Identity	G	5
	Special scalar parameter values	—	S	3
	Simple integer relations between scalar parameters	—	S	4
Polygon	Similar polygons	Identity	G	4

Table 2.3: Geometric Regularities Marked as Global (G), Partial (P), Incomplete (I), Special Value (S) with their Estimated Relative Frequencies on a Scale from 1 to 5.

We have identity regularities for all feature types. Note that this includes a method to find similar polygonal loops independent of a scaling factor. We also seek global symmetries of the feature sets. For scalar and polygon features these are not relevant. For directions we could look for global symmetries, but most of these are either covered by the rotational symmetries (see below) or they are not very common. Global directional symmetries require extremely regular objects with regular structures similar to the Platonic or semi-regular polyhedra. So while the regularity is listed in Table 2.3, we do not explicitly detect it. Similarly, global symmetries for axes are not common and the usual cases are covered by special arrangements of parallel axes (see below). For intersection and position features they are more common, but still require highly regular objects.

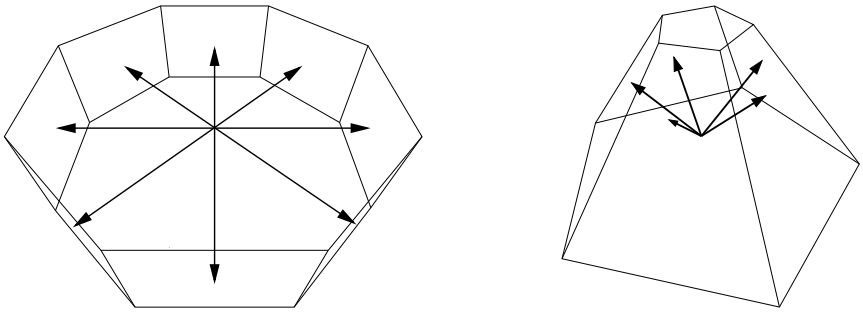


Figure 2.6: Rotational Symmetries of Directions.

A further type of regularities is based on general partial symmetries. However, as they appear to be quite expensive to detect we did not consider the general case but have concentrated on special types relating to repetitions. These special types allow us in addition to consider incomplete partial regularities. We look for directions arranged symmetrically around a circle in $\mathbb{P}^2$. These arrangements relate in general to prismatic or pyramidal structures of (not necessarily adjacent) faces as indicated in Figure 2.6. For prismatic arrangements the directions lie on a great circle of the unit sphere, and for pyramidal structures the directions lie on a small circle of the unit sphere.

Furthermore, parallel axes can be arranged along lines and grids with equal spacing between them, and they can be arranged symmetrically around a cylinder. Both types generate partial regularities of axes under the condition that the axes are parallel. We have similar arrangements for positional features, but note that most of these arrangements are often associated with similar axis arrangements. For grids we only consider orthogonal grids as a common special case. Other grids are only indirectly represented by regular arrangements on lines. Considering approximate arrangements in grids in general is likely to generate a large number of regularities and ambiguities.

In addition to positions, we have partial position features and seek positions which are equal when projected onto special planes or lines. The special lines and planes are derived from the main directions in the model as identified by the parallel direction regularities. For instance, in the prism in Figure 2.6 there are pairs of vertices, one in the top and one in the bottom plane, which are equal when projected onto the top or bottom plane. Similarly all vertices of the top plane (and the bottom plane) are equal when projected onto the central axis.

Scalar parameters from faces and edges are either lengths or angles. For each type separately, we look for special values including integers and simple fractions. We also try

to find simple integer relations between pairs of scalar parameters of the same type, i.e. relations of the form $n_2 s_1 = n_1 s_2$ with the scalar parameters s_l and some integers n_l. We handle these integer relation regularities as special value of ratio features. The methods to detect these special value regularities are presented in Section 3.4.

2.5 Summary

We have presented a well-defined concept for approximate geometric regularities. It provides exact answers to the question of the approximate presence of regularities in a solid. The concept has been designed in order to derive efficient detection methods which will be presented in Chapter 3. Its basis is a clean notion of a boundary representation for solid objects from which discrete, special properties are derived as features. Symmetric arrangements of these features in a feature space lead to regularities. By combining the geometric and combinatorial nature of the object, a non-arbitrary, efficient concept for approximate symmetries has been derived. The idea is to link combinatorial symmetries (distance preserving permutations) with geometric symmetries that induce them to create a new mathematical entity. Essentially, we seek levels of approximation for which it can be asserted that the features behave symmetrically.

Depending on whether the regularity is global or partial and complete or incomplete different detection methods are required. Only certain partial regularities are considered due to the intensive computational requirements for general partial symmetries. These regularities relate to repetitions which in addition allow us to consider incomplete regularities. Common regularities have been identified which can be described in terms of our regularities and show that we require methods to detect identity regularities, global symmetries and partially, incomplete regularities for repetitions, and special values.

Chapter 3

Detection of Approximate Geometric Regularities

In Chapter 2 we introduced our notion of approximate geometric regularities based on features derived from a boundary representation model and approximate symmetries of these features. The introduction of distinct features makes regularity detection a combinatorial problem suitable to be solved by a computer. In this chapter we present algorithms to detect our approximate regularities. By combining the combinatorial and geometric properties of the features we derive detection algorithms which provide exact, non-arbitrary results indicating the presence of approximate regularities. We present four general methods for the detection of the following approximate regularities: congruent features; simple partial, incomplete symmetries (repetitions) of features; global symmetries of features; and special values for scalar features. Using these algorithms we show how common regularities which were described in Section 2.4 are detected.

To find approximately equal features we detect distinct tolerance levels at which the features fulfil certain regularity conditions. For this the author modified a hierarchical clustering algorithm based on a closest pair strategy developed by Eppstein [35]. The resulting clusters are further used to detect repetitions and, where applicable, special values to represent the clusters are determined. The methods for this were developed by the author and they were first introduced in [74, 75, 76, 77]. Furthermore, in collaboration with B. I. Mills an algorithm to detect global symmetries was developed [93, 94].

3.1 Approximate Congruences

Approximate congruences of features, expressed by saying that a set of features remains invariant under the identity, like approximately parallel directions, are an important regularity type. As noted in Section 2.3 they are the basis of more complex regularities. Regularities which can be described in this way are, for instance, parallel directions, aligned

axes, etc. (see Table 2.3). Using these regularities we find symmetries like special rotational symmetries of directions or special symmetries in the arrangements of axes later (see Section 3.2).

Recall that we defined approximate regularities in terms of approximately distance preserving permutations of feature clusters generated by an equivalence relation $=_\epsilon$ on the features (see Section 2.3.2). As for approximate congruence regularities we solely consider the identity permutation, we only have to find relations $=_\epsilon$ on the feature set which are an equivalence. In the following we discuss a general clustering algorithm used to find such relations. Then we show how this algorithm is used to find our particular approximate congruence regularities.

3.1.1 Consistent Hierarchical Clustering

We wish to detect approximately congruent sets of features $F = \{f_l\}$ in a feature space P with metric d. In general this can be done by employing clustering algorithms. In order to detect appropriate relations $=_\epsilon$ for multiple tolerance levels ϵ we require a hierarchical clustering algorithm. In addition the clusters have to fulfil a regularity condition such that we get an equivalence relation. We call clusters which fulfil this regularity condition *consistent*.

By seeking consistent clusters at multiple tolerance levels we avoid the requirement of setting a maximum tolerance which is hard to find. Under the condition that all desired regularities are detected, the number of unwanted regularities can in general only be minimised, but not completely avoided, if we use a maximum tolerance. In order to remove all unwanted regularities we would have to discard desired regularities as well. A hierarchical structure of consistent clusters more accurately represents the structures present in the initial model. Combining all consistent clusters at a tolerance level into a set R gives an approximate congruence regularity for the subset R of F.

To represent feature clusters by a single feature, we assume that we have an averaging method avg_P to combine two features f_1, f_2 in P. It generates a new feature $f = \mathrm{avg}_P(\omega_1, f_1, \omega_2, f_2)$ where ω_1, ω_2 are two positive weights such that $d(f, f_l) \leq d(f_1, f_2)$, $l = 1, 2$.

A variety of approaches exist to the clustering problem [1, 47, 145]. The most common approach is the agglomerative (bottom-up) technique. Initially each cluster C_l consists of a single element c_l, which also represents the cluster at tolerance 0. We start with the smallest distance $d(c_l, c_k)$ representing the closest cluster pair in the current set and

combine the two clusters C_l, C_k to give a new cluster C^* represented by the average feature $c^* = \text{avg}_P(|C_l|, c_l, |C_k|, c_k)$. C^* replaces the clusters C_l and C_k. This is repeated until only one cluster remains at the root of a cluster tree. A brute force solution searching for the closest elements each time requires $O(n^3)$ time for n elements. Note that there are alternative numerical [2] and top-down [142] techniques. If we allow an approximate solution to the clustering problem there are sub-quadratic algorithms [15]. Also note that we do not limit the number of clusters, but generate as many clusters as required. Thus, algorithms for k nearest neighbour search [119] are not suitable.

Eppstein [35] uses a closest pair strategy to improve the brute force approach for the agglomerative technique. We can use a distance matrix containing the distances between the features and maintain a quad-tree like data structure to keep track of the closest pair. We store the distances $d(c_l, c_k)$ between the elements c_l and c_k representing clusters in a matrix D_n for $l < k$ for n elements. The elements are grouped arbitrarily into pairs (c_l, c_{l+1}) and the distance between two pairs is defined to be the minimum distance between the four elements of the two pairs. These pairs define a new closest pair problem in a matrix $D_{n/2}$ of half the size. Continuing this recursively we get the closest pair in D_1 as the element left in the last matrix at the root of this structure. From there we can extract it and update the matrices. After initialising we have to update at most two rows and two columns of each of the matrices for each update operation. Note that if at any stage we have a matrix D_l with an odd number l of elements, we can introduce one additional dummy element to make l even.

This algorithm is listed in Algorithm 3.1. We store the distances in a single matrix D and maintain index matrices I_j identifying the solutions to the closest pair problems. Instead of copying the solutions of the original matrix D into a new matrix $D_{n/2}$ and keeping track of which features are used for $D_{n/2}$, we represent the solution of the closest pair problems by indices pointing to the position of the closest distance in the original D. If D_{lk} is the position of the smallest distance representing the solution of a closest pair problem for I_j then the index matrix I_j represents this by storing the index $ln + k$ (interpreting D as a vector). From the index in I_j we can easily generate the indices l, k for D. Note that in order to save space we actually store D as a triangular matrix where the index computation is only slightly more complicated (the indices are of the form $\sum_{j=0}^{l} j + k$, $k < l$).

The initialisation of the matrices solves the closest pair problems recursively. In sequence we remove the closest pairs left in the data structure to create the hierarchical cluster structure. After a closest pair has been removed from the data structure updating is done by marking one of the clusters as removed. The other cluster is replaced with a new cluster by updating the distances in D and then adjusting the corresponding rows and columns in

Algorithm $\texttt{hcluster}(\texttt{F}, \texttt{d}, \Delta\texttt{T})$
Create hierarchical cluster structure of the n-vector F of features $\texttt{f}_\texttt{l}$, $l = 1, \ldots, n$, using the metric d and the minimum tolerance constant $\Delta\texttt{T}$. The resulting hierarchical cluster structure is reported as tree.

I. Initialise cluster structure C and distance matrix D. For each feature $\texttt{f}_\texttt{l}$:
   1. Create the cluster $\texttt{C}_\texttt{l}$ in C containing the feature $\texttt{f}_\texttt{l}$ with average feature $\texttt{c}_\texttt{l} = \texttt{f}_\texttt{l}$ and tolerance $\texttt{t}_\texttt{l} = 0$.
   2. For $\texttt{k} = 1, 2, \ldots, \texttt{l} - 1$ let $\texttt{D}_{\texttt{lk}} = \texttt{d}(\texttt{c}_\texttt{l}, \texttt{c}_\texttt{k})$.

II. Recursively generate the index matrices for D by starting with an index matrix $\texttt{I}_1$ representing the closest pairs in D itself. In each recursion step do the following:
   1. Generate an index matrix $\texttt{N} = \texttt{I}_\texttt{j}$ half the size of the previous matrix $\texttt{M} = \texttt{I}_{\texttt{j}-1}$ containing only elements below the diagonal.
   2. Set each element $\texttt{N}_{\texttt{lk}}$ to represent the smallest distance of the four distance values $\texttt{M}_{2\texttt{l},2\texttt{k}}$, $\texttt{M}_{2\texttt{l},2\texttt{k}+1}$, $\texttt{M}_{2\texttt{l}-1,2\texttt{k}}$ and $\texttt{M}_{2\texttt{l}-1,2\texttt{k}+1}$ (if $\texttt{l} \neq \texttt{k} + 1$).

III. While there is a closest pair $\texttt{C}_\texttt{l}$, $\texttt{C}_\texttt{k}$:
   1. Merge clusters $\texttt{C}_\texttt{l}$ and $\texttt{C}_\texttt{k}$ to form a new cluster $\texttt{C}^*$.
   2. Mark cluster $\texttt{C}_\texttt{k}$ as removed.
   3. Replace $\texttt{C}_\texttt{l}$ with $\texttt{C}^*$ and update the l-th row and column of D and all corresponding entries in the I_j.

Algorithm 3.1: Hierarchical Clustering Algorithm.

the index matrices I_j by solving the new closest pair problems.

Initially generating D and the index matrices requires $O(n^2)$ time. Let $T_1(n)$ be the time for generating the index matrices for n clusters. We get the recurrence $T_1(n) = O(n^2) + T_1(n/2) = O(n^2)$. So the initialisation is $O(n^2)$ in time and $O(n^2)$ in space. Inserting / replacing and deleting require the same amount of operations. An insertion requires $O(n)$ changes to the distance matrix D_n. Let $T_2(n)$ be the time required for updating the index structure for n clusters. Then for updating the index matrices after an insertion we get the recurrence $T_2(n) = O(n) + T_2(n/2)$, thus $T_2(n) = O(n)$. So for n clusters an insertion or deletion requires $O(n)$ time. Hence, the complete clustering algorithm requires $O(n^2)$ time at a cost of $O(n^2)$ memory, assuming that the computation of d and avg_P requires constant time and space [35]. As the number of features is of the order of the number of faces in a model (up to about 200 faces, see Section 1.2) memory is readily available for our application.

Thus overall we have a fast hierarchical clustering algorithm at acceptable extra costs for memory. Note that alternative hierarchical clustering algorithms could be used as well. In

case memory has to be conserved other closest pair approaches discussed by Eppstein [35] are suitable. A better time performance may be achieved with the sub-quadratic approximate algorithms [15]. But the methods have to be suitable for modification such that the resulting cluster hierarchy consists of consistent clusters. We describe the modification required for Algorithm 3.1 in the following.

Using Eppstein's original algorithm on our features creates a complete cluster hierarchy as a tree where the leaves contain the single features at tolerance level 0 and the other nodes represent clusters containing sub-clusters at a certain tolerance level. However, we are only interested in consistent clusters. In order to only keep the consistent clusters in the tree we modify the merging of clusters. When merging two clusters to form a new cluster we only keep them as separate sub-clusters of the new cluster if they are consistent.

Following our definition of approximate symmetry, a cluster is consistent if the related $=_\epsilon$ relation is transitive. This means that the distances between all features in the cluster are smaller than the tolerance level ϵ and the distance to features in other relevant clusters is larger than ϵ. In addition to this condition we introduce a minimum distance constant ΔT_P, which has to be provided by the user. ΔT_P indicates the minimum distance two features have to be apart in order for them to be considered different. Experiments with the clustering algorithm supported the introduction of ΔT_P as it avoids the potential problem that relatively small changes in the distances between features can create additional consistent clusters at intermediate tolerance levels. It indicates the minimum error we expect in the initial model from the reconstruction process and also simplifies the regularity selection process by reducing the number of available options.

Each cluster C_k is represented by an average feature c_k and a tolerance t_k, which is the maximum distance of the elements in the cluster from c_k. We combine two clusters C_l and C_k to give a new cluster C^*, which is represented by the average feature $c^* = \text{avg}_P(|C_l|, c_l, |C_k|, c_k)$ with tolerance t^* computed as the maximum distance of the elements of C^* from c^*. All features in C^* are at most $2t^*$ apart from each other. C_l becomes a sub-cluster of C^* if it is sufficiently apart from C_k, i.e. $d(c_l, c_k) \geq 3t_l + t_k + \Delta T_P$ (see Figure 3.1: the closest possible position to C_k of an element of C_l is t_l apart from c_l and the closest possible position to C_l of an element of C_k is t_k apart from c_k; the distance between these two position has to be larger than $2t_l$ in order for C_l to be consistent and we expand this distance slightly by ΔT_P). Otherwise the elements and sub-clusters of C_l are directly added to C^*. Similarly C_k only becomes a sub-cluster if $d(c_l, c_k) \geq 3t_k + t_l + \Delta T_P$. These conditions ensure that the distances between the clusters are sufficiently larger than the distances between the elements of the clusters.

We handle the clusters as balls in P and check if they are sufficiently far apart from each

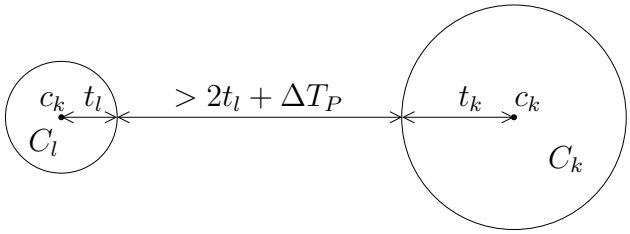


Figure 3.1: Distances Between Clusters for Consistency Condition.

other in order to be considered distinct. This ensures that the clusters are equivalence classes in a local sense as the clusters are sufficiently apart locally, but the tolerances may not work globally (see example from Figure 2.3). However, we do not necessarily find all possible partitions into equivalence classes. The hierarchical structure determined by the algorithm is coarser than this, but it does not require globally consistent tolerance levels. For global symmetry detection we present an alternative clustering method in Section 3.3.1. This method detects all tolerance levels at which we get equivalence relations on the whole feature set. This is slightly more expensive than Algorithm 3.1, requiring $O(n^2 \log n)$ time and $O(n^2)$ memory.

As noted in Section 2.3.2, depending on the feature type, further conditions may apply to decide if a cluster is consistent. For instance, an intersection point of n axes should be a cluster of $n(n-1)/2$ intersections of axis pairs, i.e. the transitivity condition on the cluster is transfered to the original elements in the boundary representation. These form additional conditions for consistency, and are described below in combination with the particular feature types. They can easily be added to the consistency test.

After we created the complete hierarchy of consistent clusters we recompute an average feature for each cluster to better fit the elements of the cluster (e.g. in a least squares sense) and compute precise tolerances. Doing this during clustering would make the merging of clusters more expensive (e.g. by solving a least squares problem). The influence of this on the cluster structure is negligible due to the coarser hierarchical structure.

As all features are combined into a single top-level cluster, the hierarchy contains clusters at high tolerance levels which are not likely to represent a desired regularity. Thus, we discard clusters where a large jump between the tolerance levels occurs. To detect this we need an approximation T_m for the largest possible tolerance level, which is automatically derived from the initial model (see Section 3.1.2). T_m is required since all features of a specific type from a model may be close to each other, in which case the largest tolerance jump is above the largest tolerance of the clusters. Note that ΔT_P has to be chosen small enough for the model such that $T_m \gg \Delta T_P$.

We take the largest cluster tolerance t which is smaller than T_m. If there is a cluster tolerance larger than t, but not larger than $t + \Delta T_P$ we have found an appropriate tolerance level. Otherwise, we set t to the next larger tolerance and repeat the test until an appropriate t has been found. This is done to avoid cutting off clusters at tolerance levels which are close to each other. All clusters with tolerances larger than t are removed from the tree and we are left with a forest of approximate congruences represented by consistent clusters. We refer to clusters in this forest which directly (not over sub-clusters) contain elements from the original feature set F as *base* clusters. These will be important to limit the cases considered when using the cluster results for the detection of other regularities.

3.1.2 Approximate Congruence Regularities

Our common regularities (see Section 2.4) which can be detected as approximate congruences are parallel directions, equal positions, equal positions when projected onto special lines or planes, aligned axes, axes intersecting in a point, equal lengths and angles, and similar polygons. In this section we show how to detect these using the clustering algorithm in the feature spaces introduced in Section 2.2.

In general we take the features in an appropriate feature space with a metric and an averaging method and cluster them using a user defined ΔT_P and a consistency condition. After clustering, the average features representing the consistent clusters are recomputed and the cluster tolerances are updated accordingly. Then using an appropriate T_m we cut off the cluster hierarchy at a large tolerance jump. ΔT_P is either a length tolerance ΔT_L or an angular tolerance ΔT_A for most of our regularities. By increasing ΔT_P, features are more likely to be judged the same. T_m is automatically derived from the model. Let $L_{\max}$ be the largest length present in the model, which is approximated by taking the largest distance between the position features of the model. Then an appropriate T_m for lengths is $L_{\max}/2$. Since we identify opposite directions, our angles are in the interval $[0, \pi/2]$ and thus $T_m = \pi/4$ is appropriate for angles.

Approximately parallel directions are detected by clustering directions in the feature space $\mathbb{P}^2$ using a metric that compares the (smaller) angle between two directions d_1 and d_2, $d_{\mathbb{P}^2}(d_1, d_2) = \arccos(|{d_1}^t d_2|)$. Furthermore, we define the weighted average of two directions d_1 and d_2. If ${d_1}^t d_2$ is non-negative, i.e. d_1 and d_2 point roughly in the same direction, the weighted average is $(\omega_1 d_1 + \omega_2 d_2)/(\omega_1 + \omega_2)$ with positive weights ω_1, ω_2. If ${d_1}^t d_2$ is negative, the weighted average is $(\omega_1 d_1 - \omega_2 d_2)/(\omega_1 + \omega_2)$.

We improve the average directions representing the clusters in a least squares sense. Let d_l, $l = 1, \ldots, n$, be the direction features in a cluster. We find a new average direction

x by minimising the error of the linear system ${d_l}^t x = 1$, $l = 1, \ldots, n$, in a least squares sense using singular value decomposition of the row vector matrix $[d_1, \ldots, d_n]$. Note that all the d_l must point in the same half space as the original average direction x_0, which can be achieved by replacing d_l by $-d_l$ if ${d_l}^t x_0 < 0$. Also note that the solution to the optimisation problem might not be a unit vector, so x has to be normalised.

We find approximately equal positions by clustering them in $\mathbb{E}^3$ using the Euclidean metric and the usual weighted average between positions.

The parallel direction clusters are used to find partially equal positions. Positions are considered to be partially equal in 2D with respect to a direction if they are equal when projected onto a plane orthogonal to that direction. Furthermore, they are considered to be partially equal in 1D with respect to a direction if they are equal when projected onto a line in the given direction. Both regularities can be detected by clustering the projected positions as positions in $\mathbb{R}^2$ or $\mathbb{R}^1$. We remove clusters from the hierarchy which contain projected positions which are as close together as the original positions, as these represent approximately equal positions detected earlier. Optionally, we may choose to consider only major parallel direction clusters in models with many different directions to reduce the number of partially equal positions. In case an orthogonal system has been detected (see Section 3.2.2), we consider the involved parallel direction clusters to be major directions. Furthermore, we consider a parallel direction cluster which contains the direction of an axis with at least $1/3$ of the faces in the model having an axis parallel to it as major direction (we have three "main directions" in $\mathbb{E}^3$ and if these are evenly distributed between the faces we expect that we have $1/3$ of the faces pointing in each direction, e.g. consider a model with all directions belonging to an orthogonal system).

As all directions in the parallel direction cluster hierarchy are considered, there may be positions considered to be partially equal with respect to different directions. To eliminate such ambiguous cases we compare clusters from the same partially equal cluster hierarchy to find clusters with the same positions and remove the cluster with the larger tolerance. Note that there is also an ambiguity between 1D and 2D partial equality. Positions that are equal when projected onto a plane are also equal when projected onto a line in that plane. In case the selected directions contain the normal of the plane and the direction of the line we get both regularities. As cases like this do not create a contradiction, we do not explicitly check for it. Furthermore, the regularity indicating that the positions are equal when projected on the line may involve additional positions. This yields additional information about the relations between the positions (see Figure 3.2).

Length and angle parameters are clustered in $\mathbb{R}_+$ using the absolute value of the difference between the scalar values as metric and the weighted average of positive reals.

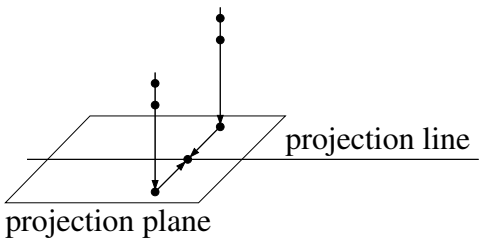


Figure 3.2: Partial Equality of Positions when Projected on Lines and Planes.

Using the parallel direction clusters we find sets of parallel axes by extracting the direction features which are associated with an axis feature. Using the average direction of the direction features of the axes we can then project the positions of the axes onto a plane. To find an appropriate position of the plane such that the error for the projection is reduced we first project the axis positions on a line through the origin parallel to the average direction and take the average of the positions on this line to set the distance of the projection plane. The projected axis positions can be clustered as positions in the Euclidean plane to detect approximately aligned axes.

To find axis intersection points we create an approximate intersection point feature as the centre of the shortest line segment between the two axes. Note that this point is only an approximate intersection point if the axes nearly meet. However, we consider all such points. This means an intersection point is a position in $\mathbb{E}^3$ with an associated length value specifying the distance between the two axes. Also note that we use actual axes for this and not any averages representing parallel direction clusters. Furthermore, we only intersect axes that belong to different direction base clusters, to avoid trying to intersect approximately parallel axes.

For intersection positions we expand the consistency condition. An axis intersection cluster which contains n different axes is consistent if it has $m = n(n-1)/2$ intersection points of axis pairs. However, as some of the axes might be parallel, some intersection points might not be present, so the cluster is still consistent if it has at least $m - p$ intersection points where p is the number of parallel axis pairs in the cluster as indicated by a parallel direction cluster at a tolerance smaller than the tolerance of the axis intersection cluster t. To compare the angular tolerance of the parallel direction cluster with the positional tolerance of the axis intersection cluster we convert the angular tolerance t to a positional tolerance $L_m \sin(t) + \Delta T_L$. We also check if the distances between the axes are consistent with the cluster tolerances. If a cluster contains an intersection point feature with an axis distance larger than the sum of the cluster tolerance and ΔT_L, the cluster is not consistent. This eliminates cases where the axes do not intersect within the toler-

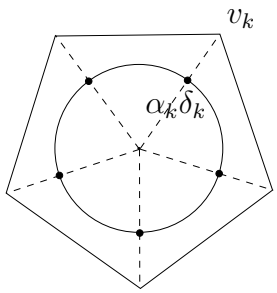


Figure 3.3: Representing Polygons as Distributions on the Unit Circle.

ance, but the centroids of the shortest line segment between the axes are close together. In general these cases cannot be considered regular.

As another regularity, we seek approximately similar polygons. The basic idea of our method is to represent each polygon as a function and then cluster the Fourier coefficients of these functions [11]. There are various ways to represent a polygon as a function. One way is to define a curvature of a polygon. Let α_k be the angle at the k-th vertex of the polygon with m vertices v_k such that $v_0 = v_m$ and let l_k be the accumulated length of the line segments from v_0 to v_k. In our particular approach we use a *distribution* (see [26] for details) f of the form $f = \sum_{k=0}^{m-1} \alpha_k \delta_k$. δ_k is the Dirac distribution defined on the unit circle $\mathbb{S}^1$ with the singularity at $2\pi l_k/l_m$ such that $\langle \delta_k, \phi \rangle = \phi(2\pi l_k/l_m)$ for $\phi \in \mathbf{C}^\infty(\mathbb{S}^1)$ where $\mathbf{C}^\infty(\mathbb{S}^1)$ is the space of infinitely differentiable functions on $\mathbb{S}^1$ in the complex plane identified with the infinitely differentiable, 2π periodic, complex functions on $\mathbb{R}$. This means that we project the polygon on the unit circle and represent the projection as a sum of distributions $\alpha_k\delta_k$, which represent the α_k at the projected position of v_k on the unit circle indicated by δ_k (see Figure 3.3). The complex Fourier coefficient of order $j \in \mathbb{Z}$ of the distribution f is

$$u_j = \frac{1}{2\pi}\langle f, \exp(-ij\cdot)\rangle = \frac{1}{2\pi}\sum_{k=0}^{m-1} \alpha_k \exp\left(-ij2\pi\frac{l_k}{l_m}\right). \tag{3.1}$$

For each polygon we compute the first n Fourier coefficients of f, which gives us a complex vector u of Fourier coefficients with n components u_j. When comparing two of these vectors u, v we use the similarity measure $d_{\mathbb{C}^n}(u,v) = \sum_{j=1}^{n} |\,|u_j| - |v_j|\,|$, comparing only the amplitude and not the phase. Note that if we compare the phase we would not match congruent polygons which can be matched by rotations. The averaging method for clustering is the usual weighted average of complex vectors. The vectors are clustered with a tolerance ΔT_F. By using the Fourier transform of f, we compare loops independently of scale. Note that f could be chosen differently and that we could also split the polygons into small sections and compare those [11]. While there are other methods to

compare polygons, our method is efficient and simple, and provides a natural metric for the clustering algorithm.

Let p be the minimum of $(l_{k+1} - l_k)/l_m$, i.e. the smallest ratio between an edge length and the circumference of the polygon. Let P be the minimum p from all polygons being compared. Then at least $\mathrm{ceil}(1/P)$ Fourier coefficients have to be computed to ensure that all relevant frequencies are considered. As the Fourier coefficients of f can be computed exactly, only a small number of coefficients is sufficient to characterise the shape of the polygon. Typically we use the first 10 coefficients.

For an m-sided, regular polygon, the only non-zero coefficients are those of order km, $k \in \mathbb{Z}$. In the approximate case the values of these coefficients are sufficiently larger than the others, so that they can be used to identify regular polygons. It appears that this can also be used to find polygons which are based on an m-sided, regular polygon, with minor modifications like an additional or missing vertex.

In summary, we have presented an efficient hierarchical clustering algorithm which generates consistent clusters at different tolerance levels. It can be used to detect various approximate congruence regularities. Some of these regularities form the basis for the approximate repetition regularities discussed in Section 3.2 and special value regularities discussed in Section 3.4.

3.2 Approximate Repetitions

Based on the approximate congruences detected with the methods discussed in Section 3.1 we can now look for special partial and incomplete regularities (see Section 2.3). A partial regularity is a symmetry of a subset of the feature set. To avoid having to check all or at least a large amount of subsets, we only present detection methods for special cases where certain conditions are used to find interesting subsets. Incomplete regularities are symmetries of a feature set appropriately expanded by additional features. As we can make any given feature set symmetric by adding more features, we require rules indicating the cases in which we should expand a given set of features.

Hence, we are only interested in repetitions which are generated by a single isometry g. Given a subset R of a feature set F in some feature space P, we have a repetition if the group G under which R remains invariant is generated by g. For instance, given n points equally spaced around a circle, a $2\pi/n$ rotation generates a repetition for these points. In particular for translational repetitions, but also for rotational repetitions, the regularity

may be incomplete. We also have to identify an appropriate subset of F before we look for the repetitions.

In order to find appropriate subsets of F we first identify a suitable subset fulfilling some simple condition. For instance, for directional features we detect directions which lie on a circle in $\mathbb{P}^2$. If we interpret $\mathbb{P}^2$ as lines through the origin this means we seek lines which lie either on a cone with the origin as apex or on a plane through the origin. Then we analyse each of these feature sets for subsets which are invariant under a repetition. We seek a minimum number of repetitions and it has to be possible to determine the generating transformation from mapping features onto other features in the subset. In the case of directions, for each angle between a direction pair on a circle in $\mathbb{P}^2$, we get a rotation, and can then group these into transformations which are generated by the rotation with the smallest angle. Each such group describes a repetition which may be incomplete.

We first discuss the algorithm for detecting approximate repetitions for directions on circles in $\mathbb{P}^2$ and then show how this algorithm can be modified to find similar repetitions relating to axes and positions.

3.2.1 Approximate Repetitions of Directions — Planar Case

Given a set of direction features from a boundary representation model, we wish to find directions lying on circles which form partial, incomplete repetitions. Instead of using the directions directly from the model, we use the features f_l representing parallel direction clusters detected by the clustering algorithm from Section 3.1. Each f_l has a tolerance t_l from the cluster and can be interpreted as a feature of the cells C_l from which the features in the clusters were obtained. Here we only consider the base clusters in the hierarchy as the directions of the other clusters are already indirectly represented by the base clusters. The hierarchy will be used later do determine possible higher tolerance levels at which the base cluster directions can be considered parallel.

A set of directions $\{d_l\}$ on a circle satisfies the equation system $|{d_l}^t x| = a$, where x is the centre of the circle in $\mathbb{P}^2$. If $a = 0$, we have a direction plane with normal x and if $a \in (0, 1)$, we have a direction cone with axis x. Thus, we have to distinguish between a planar case where we have to determine the centre x and a conical case where we have to determine the centre x and the cone angle represented by a. Note that for the planar case, taking the absolute value of ${d_l}^t x$ is not required, and we can drop it for a conical case if we ensure that all directions lie in the same half-space.

First consider the planar case (the conical case is discussed in Section 3.2.2). Each linearly independent pair of directions defines a plane through the origin, which we call a *direction*

plane feature. By clustering the normals of these planes in the same way as clustering direction features for finding parallel direction clusters we detect sets of directions which lie on a great circle in $\mathbb{P}^2$. Note that we can consider base cluster features to be linearly independent at the lowest tolerance level, as for each base cluster direction pair f_l, f_k we have $d_{\mathbb{P}^2}(f_l, f_k) > t_l + t_k + 2\max(t_l, t_k) + \Delta T_P > \Delta T_P$ due to the clustering consistency condition.

As direction plane features are generated by direction pairs we have to expand the consistency condition for clustering in a similar way to that used for clustering axis intersections. A direction plane cluster is consistent only if it contains $n(n-1)/2$ plane normals generated from n parallel direction base clusters. Here we also have to consider the parallel direction cluster hierarchy. Consider a direction plane cluster consisting of n plane normals p_{lk} generated by parallel direction base cluster pairs d_l and d_k. This cluster is consistent if there are $n(n-1)/2$ different p_{lk}. If this is not the case, we count the number m of direction pairs (d_l, d_k) for which there is no p_{lk} in the cluster, but which are considered to be parallel in the parallel direction cluster hierarchy at a higher tolerance level, still smaller than the tolerance of the direction plane cluster. We still consider the cluster to be consistent if there are $n_1(n_1-1)/2$ different p_{lk} for $n_1 = n - m$, as there are m parallel direction pairs.

After the cluster hierarchy has been generated and cut off at a large tolerance jump, we recompute the average normal x of the planes using a least squares error method. For directions d_l, $l = 1, \ldots, n$, on a great circle with normal x, we get the linear system ${d_l}^t x = 0$, $l = 1, \ldots, n$, under the constraint $\|x\| = 1$. Thus we minimise $\|Dx\|$ for $x \in \mathbb{P}^2$ with the row vector matrix $D = [d_1, \ldots, d_n]$. Let USV^t be the singular value decomposition of D. As U and V are unitary they do not change the norm of a vector. Thus $\|Dx\|$ is optimal if $V^t x = e_l$ where e_l is the l-th standard basis vector corresponding to the smallest singular value of D, i.e. the solution to the least squares problem is $x = Ve_l$.

The direction plane cluster build the subsets in which we look for repetitions. Each cluster represents a set of directions approximately orthogonal to the direction plane feature of the cluster. In these sets we look for subsets such that the angles between the directions in this subset are integer multiples of a base angle β, i.e. the directions as points on the unit sphere are arranged equi-spaced around a great circle. The angle β represents the underlying generating isometry. The subsets left invariant by the group generated by the rotation can be incomplete in the sense that not all multiples of β need to be present. We present an algorithm to detect these symmetries for direction planes and later discuss modifications for direction cones and other similar repetitions.

Algorithm planar_direction_repetition($\{\mathtt{d_l}\}_{l=1,\ldots,\mathtt{m}}, \mathtt{t}, \mathtt{N_{max}}$)
Detect repetitions in $\mathtt{m}$ directions $\mathtt{d_l}$ on a great circle of $\mathbb{P}^2$ with tolerance $\mathtt{t}$ where $\mathtt{N_{max}}$ indicates the smallest allowed rotation angle. The regularities are reported as arrays of clusters indicating which cluster of directions lies approximately on which position of the circle with respect to a generating rotation angle and a direction indicating the centre of the circle in $\mathbb{P}^2$.

I. Compute the angles $\alpha_{\mathtt{lk}}$ between directions $\mathtt{d_l}$ and $\mathtt{d_k}$ for $\mathtt{l} < \mathtt{k} < \mathtt{m}$.

II. For all reference directions $\mathtt{d_k}$ with $\mathtt{k} < \mathtt{m}$ and for all angles $\alpha_{\mathtt{lk}}$ with $\mathtt{k} < \mathtt{l} \leq \mathtt{m}$:
   1. Find the candidates $\beta_{\mathtt{kn}} = \pi/\mathtt{n}$ such that $\mathtt{n} < \mathtt{N_{max}}$ and $|\beta_{\mathtt{kn}} - \alpha_{\mathtt{lk}}| < \mathtt{t}$.
   2. Add $\beta_{\mathtt{kn}}$ to the list of candidates for $\mathtt{d_k}$ unless it is an integer multiple of one of the $\beta_{\mathtt{kn_0}}$ already in the list.
   3. If $\beta_{\mathtt{kn}}$ has been added to the list, remove any $\beta_{\mathtt{Kn_0}}$ from the list which is an integer multiple of $\beta_{\mathtt{kn}}$.

III. For each reference direction d_k with $\mathtt{k} < \mathtt{m} - 1$ consider the subset $\{\mathtt{d_k}, \ldots, \mathtt{d_m}\}$ and for each candidate $\beta_{\mathtt{kn}}$ with $\mathtt{n} = 1, \ldots, \mathtt{N_k}$, where $\mathtt{N_k}$ is the number of base angle candidates for $\mathtt{d_k}$:
   1. For each direction $\mathtt{d_j}$ with $\mathtt{k} < \mathtt{j} \leq \mathtt{m}$:
      A. If for $\mathtt{f} = \alpha_{\mathtt{jk}}/\beta_{\mathtt{kn}}$, we have $|\mathrm{round}(\mathtt{f})\beta_{\mathtt{kn}} - \alpha_{\mathtt{jk}}| < \mathtt{t}$, then record $\mathtt{d_j}$ to be a $\mathrm{round}(\mathtt{f})$ multiple of the base angle. If there is already a direction $\mathtt{d}$ recorded as the multiple $\mathrm{round}(\mathtt{f})$, then only replace $\mathtt{d}$ by $\mathtt{d_j}$ if the tolerance of $\mathtt{d_j}$ is smaller than the tolerance of $\mathtt{d}$.
   2. If the planar angle-regular subset found has at least two elements:
      A. If we already found a subset with a base angle which is a divisor or a multiple of the base angle of this subset, the two sets have common directions and the error between the two initial directions is smaller than t, then merge the two subsets and adjust the base angle.
      B. Otherwise, create a new planar angle-regular set with base angle $\beta_{\mathtt{kn}}$.

Algorithm 3.2: Finding Planar Angle-Regular Direction Sets.

Let $\{d_l\}$ be a set of directions in a direction plane and let α_{lk} be the angle between d_l and d_k. We call the directions d_l planar *angle-regular* if there is a $\beta \in \{\alpha_{lk}\}$ such that $\beta = \pi/n$ for $n \in \mathbb{N}$ and for each α_{lk} there is an integer p such that $\alpha_{lk} = p\beta$. As we identify opposite directions we only need to consider angles in the interval $(0, \pi]$. We do not require that all multiples of β are present, but based on which multiples are present we decide whether an angle-regular set is reported as a regularity. We call the directions approximately angle-regular if the α_{lk} are approximately integer multiples of β.

As a direct consequence of the requirement that the rotation has to be derivable from the

features in the set we only look for base angles which are (approximately) present as an angle between directions. For instance, if we have two approximate angles $2\pi/6$ and $5\pi/6$ relative to some direction, the underlying base angle $\pi/6$ is not detected. However, the angles between the involved directions of such an arrangement *are* found as special angle values.

We check the directions d_l from the direction plane clusters for angle-regular arrangements. As the direction planes were created using parallel direction clusters, the d_l are the average directions of these clusters. To avoid ambiguous angle-regular arrangements due to the tolerances involved, we derive a maximum N_{max} for the allowed n values of the base angles. Let t be the sum of ΔT_A and the maximum of the tolerances of the parallel direction clusters and the direction plane clusters. If n is larger than or equal to $N_{\text{max\_tol}} = \text{floor}(\pi/(4t))$, angle regularities with base angles π/n and $\pi/(n+1)$ cannot be distinguished at tolerance level t. Thus we set $N_{\text{max}} = \min\{N_{\text{max\_tol}}, N_{\text{max\_user}}/2\}$, where $N_{\text{max\_user}}$ is provided by the user to compensate for small tolerance values: even if the tolerances are small an n larger than about 36 (a base angle of $10°$) may not be of interest. We divide $N_{\text{max\_user}}$ by two so that we can also use it for the conical case where opposite directions in the plane cannot be considered equal and thus we have twice as many directions to consider (see Section 3.2.2).

Given a set of m directions $\{d_l\}$, we look for a minimum number of subsets which are approximately angle-regular with respect to t. The algorithm for this consists of three main steps. First we compute all angles α_{lk}, $k < l \leq m$, between the directions. From these angles we derive a set of possible base angles β_{kn} for each d_k, which we call the reference direction for the angles β_{kn}. Note that more than one base angle candidate can be close to a single α_{lk} depending on the tolerances. In the third step we try to find angle-regular subsets by checking the angles α_{jk} for each reference direction d_k and all base angle candidates β_{kn} (see Algorithm 3.2).

To compute the angles between the directions in step I, the reference direction d_k is used to choose the angle that lies consistently to the right of the reference direction d_k with respect to the direction plane normal q (see Figure 3.4). With $v = q \times d_k$ we have

$$\alpha_{lk} = \begin{cases} \arccos({d_k}^t d_l) & \text{if } v^t d_l \geq 0, \\ \pi - \arccos({d_k}^t d_l) & \text{if } v^t d_l < 0. \end{cases} \tag{3.2}$$

This allows us to identify which of the $k\pi/n$, $k = 0, \ldots, n-1$, directions is occupied by a particular d_j for some $n \in \mathbb{N}$.

In step II we look for base angle candidates β_{kn} for each set $\{d_k, \ldots, d_m\}$ referenced by d_k. We have to check the relations between the base angles added for a single reference

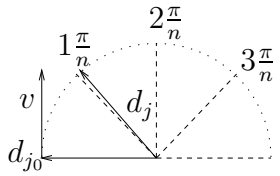


Figure 3.4: Planar Angle-Regular Arrangement of Directions.

direction to ensure that we use the smallest base angles and eliminate multiples.

In step III we check each set $\{d_k, \ldots, d_m\}$ for $k < m - 1$ for angle-regular subsets with respect to the candidates β_{kn} found in the previous step. The reference direction is always an element of an angle-regular subset of the set. Thus, for each β_{kn} we seek approximately angle-regular subsets of $\{d_{k+1}, \ldots, d_m\}$. A d_l for $l \in \{k+1, \ldots, m\}$ belongs to the angle-regular subset, if, for $f_l = \alpha_{lk}/\beta_{kn}$, we have $|\mathrm{round}(f_l)\beta_{kn} - \alpha_{lk}| < t$. However, we only allow one direction for each multiple of β_{kn}, as the directions have already been clustered into parallel directions. Hence, if we have two directions d_{l_1} and d_{l_2} for which $p = \mathrm{round}(f_{l_1}) = \mathrm{round}(f_{l_2})$, we use the one that is closer to p, i.e. the one for which $|p - f_{l_j}|$ is smaller.

Step I of the algorithm requires $O(n^2)$ time to compute the angles for n directions in the direction plane cluster. In step II we create $O(n^2)$ times a constant number m of base angles β_{kn} and check if they are already represented in the list of base angle candidates found before. The length of this list is also bound by the constant m. Thus step II requires $O(n^2 m)$ time with $m \ll n$. In step three we have two nested loops over each reference direction and each associated base angles, i.e. step III.1 and III.2 are executed $O(nm)$ times. Step III.1 is a loop over all directions, i.e. we have $O(n)$ time. Step III.2 records detected regularities and also compares the directions in the regularity with previously detected regularities. This takes at most $O(n^2)$ time. However, usually the number of directions involved in a regularity is bound by a constant (it does not grow with the number of faces in the model), so III.2 usually takes $O(1)$ time. In case as many directions are involved in a regularity as we have directions, we expect to find only one regularity which again means that III.2 requires only $O(1)$ time. So we expect that all three steps require $O(n^2) + O(n^2 m) + O(nm) \times (O(n) + O(1)) = O(n^2 m) = O(n^2)$ time for a direction plane cluster. We create $O(N^2)$ direction plane features where N is the number of faces in the model. However, the consistency condition for clustering ensures that the number of base direction plane clusters is $O(N)$. The depth of the cluster hierarchy is bound by a constant for usual engineering objects as we cut off the hierarchy at a large tolerance jump and only accept consistent clusters. So finding planar angle-regular direction sets for the whole model is expected to take $O(N^2)$ time.

Once we have found an angle-regular subset for a particular base angle and reference direction, we first check if that subset can be merged with a set found before. This is the case if one of the base angle candidates of the two subsets is a multiple of the other, the subsets have common elements and the error between the two reference directions is smaller than t. If there is no such set, we create a new regularity. While the algorithm has been designed to avoid cases where sets have to be merged, they cannot be avoided completely due to the approximate nature of the problem. Comparing the sets rather than the relations between the base angle candidates is more robust and therefore preferred.

After the angle-regular subsets have been detected, we further check the distributions of the directions in each angle-regular subset. For a base angle π/n we have possible directions at positions 0 to $n-1$ and we check if every m-th position is occupied starting with some direction in the set. m should be a divisor of n and we seek a minimum number of different m. We mark the occupied positions in a Boolean array and check for all occupied directions and all divisors m of n if every m-th position is occupied in the array. This results in a matrix $\{r_{ml}\}$ where r_{ml}, $l < m$, indicates if every m-th position is present starting with the direction at position l. We check all directions with the smallest m first, such that we detect redundant arrangements in the matrix and only report the smallest non-redundant m.

If any of the computed α_{lk} are not involved in an angle-regular subset, we check whether we can find a special value for this angle, using the algorithm to detect special angle parameters described in Section 3.4.

3.2.2 Approximate Repetitions of Directions — Conical Case

The symmetrical arrangements of directions lying on a cone are detected in a similar way to the planar case. To find *direction cone* features we have to consider an angle and a direction. For each triple d_1, d_2, d_3 of directions representing parallel direction base clusters, we generate a direction cone with direction c and semi-angle α by solving the linear system ${d_l}^t x = 1$, $l = 1, 2, 3$. From this we get the cone parameters as $\alpha = \arccos(|x^t d_1|/\|x\|)$ and $c = x/\|x\|$. As the d_l are in general only approximations of the directions, we avoid finding nearly flat direction cones that actually represent direction planes or direction cones that represent approximately parallel directions by rejecting cones for which $\alpha < \Delta T_A$ or $|\pi/2 - \alpha| < \Delta T_A$.

One way to find sets of directions on a cone would be to cluster the generated direction cones as pairs (c_l, α_l) [77]. This, however, means that we compare the direction cones with respect to the cone semi-angle and the cone axis direction at the same time mixing

both rather separate aspects. Better results can be achieved if we consider the angles separately creating a cone angle hierarchy and use this hierarchy to guide the clustering of cone directions. To cluster the angles, we generate a cone angle feature for each direction cone and cluster them using ΔT_A and the metric $d(\alpha_1, \alpha_2) = |\alpha_1 - \alpha_2|$. The cluster hierarchy is cut off at a large tolerance jump using $T_m = \pi/4$.

We do a depth first traversal of the angle cluster hierarchy, clustering the cone directions at the lowest level of the angle hierarchy first and reporting the results to higher hierarchy levels. At the lowest level in the hierarchy we simply collect all cone directions belonging to the angles combined in that cluster and cluster them in the same way as the direction planes. A direction cone cluster is only consistent if n directions cones are generated by $n(n-1)(n-2)/6$ different parallel direction base clusters. Analogously to the direction plane consistency check, we also check if directions on the cone are approximately parallel at higher tolerance levels in the parallel direction hierarchy. Note that in this case we have to keep non-consistent clusters at the top level in the current hierarchy for the next angle cluster. At a higher tolerance level from the angle cluster they may be merged to form consistent clusters.

When moving upwards in the angle cluster hierarchy the clustering results of lower levels are combined into a single set of clusters and the clustering of the cone directions is continued using them. At each level the clusters are checked for consistency and only the consistent ones are preserved, while the others are simply reported to the next higher angle cluster level until the top-level angle clusters are reached. When combining clusters at higher levels, those which are marked consistent are always added as sub-clusters to new clusters.

This results in a direction cone cluster hierarchy which is created by only comparing the directions of the cones. To avoid mixing cones with different angles, the angle clusters are used to ensure that only cone directions with similar angles are combined at different tolerance levels. The process is illustrated in Figure 3.5. We have two cone angle clusters at tolerance levels t_0, t_1 combined to a single cluster at a higher tolerance level t_2. First the direction cones are clustered with respect to the direction in the sub-clusters. The two separate cluster structures are combined at tolerance level t_2 and we continue the clustering from there such that the two corresponding cluster pairs are combined for a cone angle at tolerance t_2.

We finally recompute the average direction cone representing a cluster built from the direction base clusters d_l by solving the linear system ${d_l}^t x = 1$, $l = 1, \ldots, n$, in a least squares sense using singular value decomposition to give the average semi-angle and the average cone direction.

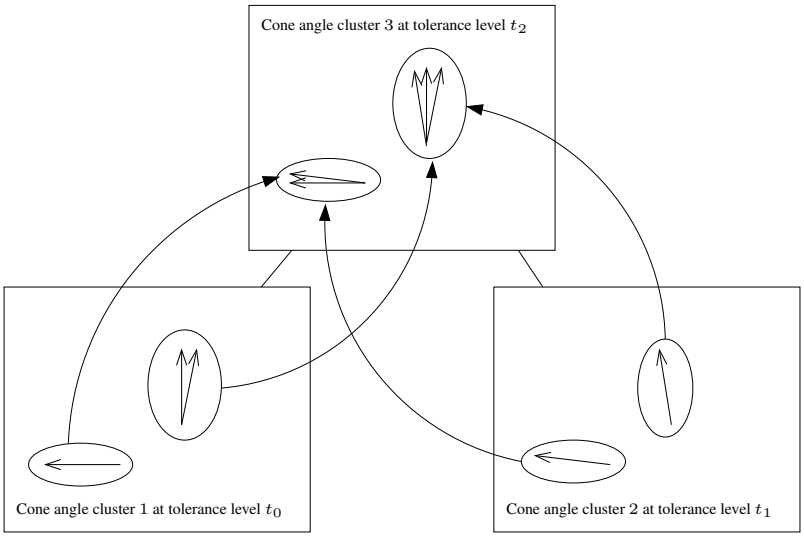


Figure 3.5: Cone Angle Cluster Hierarchy to Guide Direction Cone Clustering.

In order to apply the algorithm to detect repetitions of directions in a plane, we project the directions $\{d_l\}$ in the cone onto the plane through the origin orthogonal to the axis of the cone. Due to the projection, opposite directions on the plane actually represent different axis directions on the cone. Therefore, we have to use base angles of the form $2\pi/n$ in the angle-regular definition. However, each axis direction on the cone can still point in one of two directions, and so we always project the direction pointing to the same side of the plane orthogonal to the cone normal c, i.e. ${d_l}^t c > 0$.

As opposite directions are no longer identified in this case, the maximum n for the base angles is $N_{\text{max}} = \min\{N_{\text{max\_tol}}, N_{\text{max\_user}}\}$ with $N_{\text{max\_tol}} = \text{floor}(\pi/(2t))$. The tolerance t is derived in the same way as for the planar case. Note that three axis directions forming an orthogonal system generate a special conical angle regularity with semi-angle $\arccos(1/\sqrt{3})$ and angle $\pi/2$ between the axis directions.

Note that in general we can also handle directions in a plane without the repetitions as regularities, especially if the normal of the direction plane is approximately parallel to a direction in the model. Direction cones alone do not represent an important regularity, as a model can easily create many different direction cones.

3.2.3 Approximate Repetitions of Axes and Positions

We are also interested in repetitions of parallel axes and positions in planes or lines. This relates to translational repetitions, which are always incomplete. However, we can use an approach similar to direction repetition detection. Instead of distances on a circle

measured by angles we have distances on a line. As the distances on a line do not have to be fractions of a maximum length there are no restrictions on the distances between the features.

First consider parallel axes. We already detected approximately aligned axes with respect to a parallel direction cluster. We can use the results generated by clustering axis positions in an appropriate plane to find axes on a line, a gird, or a circle which are regularly spaced. This yields repetitions of the axes. Note that we only consider orthogonal grids as the most common case with relatively small ambiguity.

For each parallel direction cluster for which we have a cluster hierarchy for aligned axes we use the projected axis positions to detect parallel axes along lines and grids. For each pair of projected points, we create a line. These lines are clustered into approximately parallel lines and in each cluster of parallel lines we group the points generating the lines into point sets lying approximately on the same line. Then we check whether the points on the average lines are regularly spaced. By further examining pairs of approximately orthogonal groups of regularly spaced, parallel lines, we find the grids.

In the first step we generate a line for each pair of points p_1, p_2 in the plane. To cluster the lines into parallel lines, we represent the lines by vectors $d = p_2 - p_1$. For two such vectors d_1, d_2 with $\|d_2\| \geq \|d_1\|$, we use the similarity measure $\delta(d_1, d_2) = \|d_1 - ({d_1}^t u)u\|$ with $u = d_2/\|d_2\|$, and ΔT_L as the cluster tolerance. Using the distance between d_1 and its projection on d_2 instead of the angle between the two vectors allows us to take the distance between the points into account. If we used the angle between the vectors alone, two points which are on different parallel lines in a grid and sufficiently far apart might generate a line which is approximately parallel to the grid lines. For the same reason, we also define the weighted average for clustering in terms of the lengths,

$$\mathtt{avg}_{\mathtt{pl}}(d_1, \omega_1, d_2, \omega_2) = w \left(\frac{d_1}{\|d_1\|}\omega_1 + \operatorname{sign}({d_1}^t d_2)\frac{d_2}{\|d_2\|}\omega_2 \right) \tag{3.3}$$

with $w = (\|d_1\|\omega_1 + \|d_2\|\omega_2)/(\omega_1 + \omega_2)^2$.

For each parallel line cluster, we solve the linear system ${d_l}^t x = 1$, $l = 1, \ldots, n$, in a least squares sense where the d_l are the normalised direction vectors of the parallel lines in the cluster. The solution x is used as the average direction for the parallel line cluster and the cluster tolerance is updated. After this the cluster hierarchy is cut off using $T_m = L_{\max}/2$.

The clusters represent sets of parallel lines, but we still have to group the points which generate these lines to find distinct lines on which the points lie approximately. If we have two approximately parallel lines each generated by two points, we can consider these points to lie approximately on the same line if (at least) one of the points is used to

generate both lines. In general we can use this to find the distinct lines by detecting sets of points connected by lines in a parallel lines cluster. We start with pairs of points each representing a single parallel line in the cluster. As long as we can find two sets of points which have at least one point in common we merge the sets.

Each of the point sets representing points that are approximately on a line is further examined for regular arrangements of the points on the line, i.e. we look for base distances such that the distances between a subset of the points on the line can be represented as integer multiples of a base distance, analogously to the method used for angle-regular arrangements (Algorithm 3.2). The main difference is that we do not have a special value such as π/n for the base distance, but any distance could be a base distance.

After these steps we have sets of parallel lines with points on them where some subsets of them might be marked as distance-regular. To generate grids, we search for orthogonal pairs of distance-regular parallel line sets. Lines which are not distance-regular or for which we cannot find an orthogonal partner are noted as simple regularities. For the orthogonal pairs of line sets, we try to generate regular grids.

Each orthogonal pair is handled separately. First the two sets of parallel lines in the pair are grouped into lines with compatible distance-regular arrangements. In the following approximate always refers to an equality within the corresponding parallel lines cluster tolerance plus ΔT_L. Two parallel lines belong to the same distance-regular group if one of their base distances is approximately a multiple of the base distance of the other and the distance between the two reference points on the line in the parallel direction is approximately an integer multiple of the base distance. This produces two lists of groups which contain compatible distance-regular lines. Corresponding elements of each group form an orthogonal pair of compatible distance-regular lines. These pairs generate grids in such a way that the distances between the lines in one group fit on the distance-regular arrangement of the other group and vice versa.

The generated grids are not unique in the sense that various diagonals of a grid can form a distance-regular line, and combining orthogonal pairs of these diagonals can form additional grids. Thus, we have to find and remove such diagonal lines and grids, and add additional points on them to the fundamental grid. A line is a diagonal of a grid if the reference point of the line lies on the grid and the base distance d_l of the line can be generated from the two base distances d_1 and d_2 of a grid. Let D_j be the unit vector representing the directions for the distance d_j in the grid and D_l be the direction of the line. The line is a diagonal of the grid if

$$d_l D_l \approx \text{round}\left(\frac{d_l {D_l}^t D_1}{d_1}\right) d_1 D_1 + \text{round}\left(\frac{d_l {D_l}^t D_2}{d_2}\right) d_2 D_2. \tag{3.4}$$

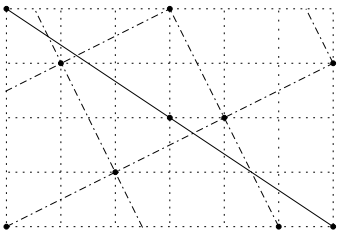


Figure 3.6: Combining a Fundamental Grid with Diagonal Lines and Grids.

Another grid with base distances d_3, d_4 and the corresponding directions D_3 and D_4 is a diagonal of the fundamental grid if its reference point is on the grid and the distances are compatible. Without loss of generality, we assume that $d_1^2 + d_2^2 < d_3^2 + d_4^2$, i.e. the diagonal of the second grid is longer than the diagonal of the first one. Then the distances of the two grids are compatible, if Equation (3.4) holds for $l = 3$ and $l = 4$. The grid with the shorter diagonal is the fundamental grid and we eliminate the one with the longer diagonal. Figure 3.6 illustrates a diagonal grid marked with dot-dashed lines and a distance-regular diagonal line drawn solid on a fundamental grid marked with dashed lines. Only the positions of the fundamental grid which are also on the illustrated diagonals are marked.

Any distance-regular line that is not combined to give a grid or removed as a diagonal of a grid is noted as a regularity. In addition we also try to find special values for base distances of distance-regular lines and grids (see Section 3.4).

Approximately parallel axes can be arranged equally spaced around a cylinder. For cylinder, cone and torus axes we check if they approximately lie on a cylinder and are symmetrically arranged. For this we can use the projections of the root points onto a plane and decide if the points lie on circles. We use the base clusters of aligned axes clusters and remove any clusters which do not contain at least one cylinder, cone or torus axis.

In some cases, a set of points may lie both on a grid and on a circle, so we only consider points derived from axes which do not lie on a large grid, having more than 5 points on it, and with enough points on each line of the grid in each direction. For one grid direction let n_0 be the maximum number of points on a single line and let n_1 be the average number of points on a line. There are enough points on each line of the grid in this direction, if $n_0 > 2$, $n_1 \geq 3/4 n_0$ and there are at least two occupied lines in this direction, or $n_0 = 2$, $n_1 = 2$ and there are more than two lines in this direction. This condition could be adjusted to suit other special cases, but it helps to avoid finding many circles which are actually produced by a single large grid.

Each triple of remaining clustered points generate a circle. We only consider circles with

radii larger than ΔT_L and smaller than $L_{\max}$, and the smaller arc of the circle described by the three points should be larger than ΔT_A. For n points there are $n(n-1)(n-2)/6$ possible circles, which can easily become too many for our hierarchical clustering algorithm. Thus, we first group the circles by clustering their positions and stop as soon as the distance between the pairs of average positions is larger than some $d_{\max}$. Note that we do not create a cluster hierarchy as this is only intended to reduce the amount of circles considered to be approximately congruent. The hierarchical clustering structure is created using these sets later (see below). $d_{\max}$ determines how many different average positions we consider. The larger it is, the more possible relations between circles we take into account. We set it to $\Delta T_L \min\{L_{\max}/\Delta T_L, D_{\max}\}$ with some user-defined limit $D_{\max}$, which should be as large as possible with respect to the memory available or time we are prepared to take.

Given three points p_1, p_2, p_3 in the plane such that $p_l = p_1 + p_{l,x}b_1 + p_{l,y}b_2$ with $p_{3,y} \neq 0$, where b_1, b_2 are two orthogonal vectors representing the plane such that $b_1 = (p_2 - p_1)/\|p_2 - p_1\|$, the centre of the circle defined by the points is

$$c = p_1 + \frac{p_{2,x}}{2}b_1 + \frac{{p_{3,x}}^2 + {p_{3,y}}^2 - p_{2,x}p_{3,x}}{2p_{3,y}}b_2. \tag{3.5}$$

We get the radius from the distance between c and any p_l.

We find similar circles by considering each group of circles separately to reduce the amount of memory required. The circles in a group should be clustered with respect to their position and radius. Our strategy to cluster them is similar to that used for finding direction cones. First we create a cluster hierarchy for the radii and cut it off using $T_m = L_{\max}/2$. Then we use the radius cluster hierarchy to guide the clustering of the positions. A circle cluster is consistent if for n different points in the cluster generating the circles, we have $n(n-1)(n-2)/6$ circles in the cluster. The resulting position cluster hierarchy is finally cut off using $T_m = L_{\max}/2$.

Once we have the circle cluster, we recompute a circle that fits the points in a least squares sense. For this we minimise the function $F : (c, r) \mapsto \sum_l (\|p_l - c\| - r)^2$ for the circle centre c and the radius r with the points p_l. As $r = 1/n \sum_l \|x_l - c\|$ for a minimum of F, we actually minimise $G : c \mapsto \sum_l (\|p_l - c\| - 1/n \sum \|p_l - c\|)^2$ using the Nelder-Mead downhill simplex method [99, 107]. The downhill simplex method performs a multidimensional minimisation based on appropriately modifying an initial simplex in order to find a small simplex enclosing the minimum. It does not require the evaluation (nor existence) of derivatives, but requires many function evaluations. It was chosen as it is simple to implement and works well to find the unique minimum of G. But alternatives, like quasi-Newton methods, could be used as well.

Finding repetitions of positional features on lines, grids and circles is done in the same way as finding repetitions of axes. The only difference is that we use positional features rather than projected axis positions (but we still have to project the positions onto an appropriate plane or line as they only lie approximately on a plane or a line). To find positional features which are approximately on a line or a plane, we can use the results from the partially equal positions. Positions which are equal when projected onto a line lie in a plane, and positions which are equal when projected onto a plane lie on a line.

In summary, by creating feature clusters we detect subsets of all features from a model which represent interesting cases to look for repetitions. By considering all distances between the features in these subsets we can combine the distances to compatible distances to detect repetitions. The generating transformation of a repetition is represented by the base distance. We presented a general algorithm to detect such repetitions which can easily be adjusted to various different feature types.

3.3 Approximate Global Symmetries

We wish to determine the approximate symmetries of a set of positions F. According to our definition of approximate symmetries as regularities (see Section 2.3.2). This means we have to find an equivalence relation $=_\epsilon$ on F. We then require approximately distance preserving permutations σ on F which essentially map the equivalence classes of $=_\epsilon$ onto each other and describe a geometric symmetry of F. Alternative approaches to the detection of approximate symmetries and the relations to our approach were discussed in Section 2.3.3.

Following the definition our approach to finding the approximate symmetries consists of two parts. We first detect tolerance levels at which the relation $=_\epsilon$ is an equivalence and then look for approximate symmetries at these levels.

3.3.1 Distinct Tolerance Level Detection

We have to detect appropriate tolerances ϵ to determine the permutations σ. For ϵ less than the smallest non-zero distance between the points in P, the approximence $=_\epsilon$ is an equivalence on F. This is also true for any ϵ greater than the maximum distance between the points. The problem is to determine which intermediate values of ϵ produce an equivalence relation.

Algorithm `distinct_tolerance_levels(Points)`
Detect distinct tolerance levels for point set `Points`. The output consists of a list of consistent clusterings with the corresponding tolerance levels where each clustering is described by a set of point clusters with their centroid.

I. For each `P` in `Points`:
   1. Create a graph component `C` containing `P`.
   2. Initialise node count `C.nodes` $= 1$ and edge count `C.edges` $= 0$.

II. Initialise counter of incomplete components `Incomplete` $= 0$.

III. Create a list `Edges` of each distinct point pair in `Points`.

IV. Order `Edges` by the distance between the points.

V. For each `E` in `Edges`:
   1. If `Incomplete` $= 0$, report the set of current components as a consistent clustering at a tolerance level of half the minimum distance between the centroids of the clusters.
   2. If `E` connects two nodes in a single `C`, then `C.edges` $=$ `C.edges` $+ 1$ and if `C` is complete, decrement `Incomplete` by 1.
   3. Else `E` must connect two distinct components $\mathtt{C}_1$ and $\mathtt{C}_2$:
      A. If $\mathtt{C}_1$ is complete, increment `Incomplete` by 1.
      B. If $\mathtt{C}_2$ is complete, increment `Incomplete` by 1.
      C. Merge $\mathtt{C}_1$ and $\mathtt{C}_2$ to a single component `C`, set `C.nodes` $=$ $\mathtt{C}_1$`.nodes` $+$ $\mathtt{C}_2$`.nodes`, and `C.edges` $=$ $\mathtt{C}_1$`.edges` $+$ $\mathtt{C}_2$`.edges` $+ 1$.
      D. If `C` is complete, decrement `Incomplete` by 1.

Algorithm 3.3: Detect Distinct Tolerance Levels in Point Sets.

Note that the clustering algorithm from Section 3.1 may be used for detection of the distinct tolerance level at which $=_\epsilon$ is an equivalence. However, we require tolerance levels at which we get an equivalence relation on the whole set of points. This cannot immediately be derived from the cluster hierarchy which only contains locally consistent clusters. Thus, we use a graph based algorithm instead, particularly designed to find tolerance levels for global symmetry detection.

Since $=_\epsilon \subset F \times F$ is symmetric, $(=_\epsilon, F)$ is an unordered graph. For it to be an equivalence relation each connected component of this graph must be a complete sub-graph (the components represent equivalence classes). A connected component with m nodes is complete if and only if it has $m(m-1)/2$ edges. By keeping track of the number of edges in a component when we create the relation it is possible to detect complete components efficiently.

Assuming all the points in F are distinct, $=_0$ is the discrete partition. Thus, the initial graph has each point in a component of its own, and the graph is an equivalence relation. We can consecutively add edges to the graph from a list of point pairs ordered by distance and keep track of the number of nodes and edges in each component. The algorithm is listed in Algorithm 3.3.

The most critical part of the algorithm is merging of components as it will be done many times at the core of the algorithm. On the one hand we require fast merging and on the other hand we want to be able to determine quickly when an edge connects two nodes of the same component. Careful bookkeeping techniques can keep this down to logarithmic complexity. We can store the components as small trees. When merging the components we make the root of the shallower tree point to the root of the deeper tree. To check if two nodes are in the same component we can move upward in the tree from both nodes until we reach the same node in which case the nodes are in the same component. If we reach the root of a component tree first, the nodes are in different components.

Each relation generated as the edges are added is a superset of the one before. Hence, each equivalence relation is a coarsening of the previous one, and is obtained by merging some of the equivalence classes previously found. The set of centroids of the classes is considered to be a new point set, whose symmetries are to be determined as described in the next sub-section. It is apparent that the distance between any two of these new points is at least twice the tolerance. Thus, the association of a transformed point to a nearby (within tolerance) point is unique if it exists.

3.3.2 Symmetry Detection

We have to find approximate symmetries of the clustered points at a tolerance level t as permutations. This is done for each consistent clustering reported by Algorithm 3.3. We do this by detecting distance preserving permutations of the centroids of the clusters. An exhaustive search over all $n!$ permutations would work, but is computationally too expensive. However, a more efficient search can be devised since metric preservation is a monotonic property. The permutations are described by point pairs indicating which point is mapped to which other point. We can build up the permutations consecutively one pair at a time where the pair lists are partial injections of a set into itself. This generates a tree with an empty partial injection at the root and a child is generated from its parent by adding an additional point pair. The leaves of this tree are permutations. For any distance preserving permutation the partial injections in the tree leading to it also have to be distance preserving. Thus, if we build the permutations one pair at a time, then once

any partial injection has been shown to be non-metric-preserving no extension of it needs to be considered.

Once we know where an isometry takes a non-degenerate simplex, we also know exactly where it takes the whole of space. Similarly if we know approximately where some non-degenerate simplex is mapped to, by a permutation, then we can tell roughly where the other elements of the set are mapped. Thus it is sufficient to build up the partial injections for the points in a non-degenerate simplex, which is chosen as described below. This yields distance preserving mappings between a simplex explicitly chosen from F and arbitrary simplices build from points in F. In the following we call the explicitly chosen simplex the *registration simplex*.

We still have to check if the mappings between the simplices also map all other points approximately onto each other. Given a non-degenerate simplex the distances between the points in the simplex and another point p uniquely determine the position of p. Thus, in three-dimensional Euclidean space four distances from the four vertices in a given non-degenerate tetrahedron determine a point uniquely. Hence, given a distance preserving partial injection between two simplices of points in F, we can check whether it induces a distance preserving permutation on all points by matching the four-dimensional vectors of distances from the two simplices. Due to the detection of distinct tolerance levels we have a unique matching of two four-dimensional vectors if they are no further than t apart.

The centroid of the points in F must be mapped to itself by a symmetry of F. So by expanding F by its centroid we can eliminate one linear scan from the search of distance preserving partial injections by always adding the centroid to the registration simplex. Moreover, maximal volume would be a desirable characteristic of a registration simplex, but its computation is intensive. Instead we build a *greedy* simplex by finding the pair of points with the largest distance between each other, then add the point that makes the largest triangle, and finally add the point that makes the largest tetrahedron. In practice it was found that choosing the first point as furthest from the centroid worked well.

We want the chosen partial injection to fix the rest of the permutation to the greatest degree possible. It can be shown that the edges of the greedy simplex are on average at least one half the length of the maximal simplex. The idea of picking this sort of simplex is that we want a reasonably wide spread collection of points so that bringing these points into approximate correspondence will be as restrictive as possible on the rest of the points.

Algorithm 3.4 lists the algorithm to find approximate symmetries of the centroids of the point clusters at a tolerance level. The algorithm works for three-dimensional points, but can easily be modified for other dimensions. First a large tetrahedron is detected. Then we

Algorithm `detect_symmetry`$(\texttt{Points}, \texttt{t})$
Detect approximate symmetries of point set `Points` at tolerance `t`. The approximate symmetries are reported as permutations of the points.

I. $\texttt{P}_0 =$ centroid of `Points`
II. $\texttt{P}_1 = \texttt{P}$ in `Points` with maximal $length(\texttt{P}, \texttt{P}_0)$
III. $\texttt{P}_2 = \texttt{P}$ in `Points` with maximal $area(\texttt{P}, \texttt{P}_1, \texttt{P}_0)$
IV. $\texttt{P}_3 = \texttt{P}$ in `Points` with maximal $volume(\texttt{P}, \texttt{P}_2, \texttt{P}_1, \texttt{P}_0)$
V. For each $\texttt{Q}_1$ in `Points`:
    1. If $|d(\texttt{P}_0, \texttt{P}_1) - d(\texttt{Q}_0, \texttt{Q}_1)| < \texttt{t}$, for each $\texttt{Q}_2$ in `Points` without $\texttt{Q}_1$:
        A. If $|d(\texttt{P}_0, \texttt{P}_2) - d(\texttt{Q}_0, \texttt{Q}_2)| < \texttt{t}$ and $|d(\texttt{P}_1, \texttt{P}_2) - d(\texttt{Q}_1, \texttt{Q}_2)| < \texttt{t}$, for each $\texttt{Q}_3$ in `Points` without $\texttt{Q}_1$, $\texttt{Q}_2$:
            i. If $|d(\texttt{P}_0, \texttt{P}_3) - d(\texttt{Q}_0, \texttt{Q}_3)| < \texttt{t}$ and $|d(\texttt{P}_1, \texttt{P}_3) - d(\texttt{Q}_1, \texttt{Q}_3)| < \texttt{t}$ and $|d(\texttt{P}_2, \texttt{P}_3) - d(\texttt{Q}_2, \texttt{Q}_3)| < \texttt{t}$ and $\texttt{expand}(\texttt{P}_0 \rightarrow \texttt{P}_0, \texttt{P}_1 \rightarrow \texttt{Q}_1, \texttt{P}_2 \rightarrow \texttt{Q}_2, \texttt{P}_3 \rightarrow \texttt{Q}_3, \texttt{Points}, \texttt{t})$, report symmetry as permutation of points.

Algorithm 3.4: Detect Approximate Symmetries of Point Sets.

try to map the points $(\texttt{P}_0, \texttt{P}_1, \texttt{P}_2, \texttt{P}_3)$ of the tetrahedron to any tuple $(\texttt{Q}_0, \texttt{Q}_1, \texttt{Q}_2, \texttt{Q}_3)$ of the cluster centroids stopping each attempt as soon as the mapping is not distance preserving within the tolerance level. Once we found a mapping for the tetrahedron which preserves the distances approximately we still have to check whether it can be expanded to the remaining points in the method `expand`. This is done by using the distances of the points from the registration simplex and finding a point q for each point p such that the distances of p from the tetrahedron $(\texttt{P}_0, \texttt{P}_1, \texttt{P}_2, \texttt{P}_3)$ are approximately the distances of q from the tetrahedron $(\texttt{Q}_0, \texttt{Q}_1, \texttt{Q}_2, \texttt{Q}_3)$. If this succeeds for all points we have found an approximate symmetry as a permutation of points.

The level of approximation t used is half the distance between the two closest points. Thus, points that are far away constrain the distance preserving permutations proportionately more than points that are close to the centre of the model. This means that points which are close may prevent the detection of a symmetry of some subset that is considerably larger than the distance between the points. However, the large discrepancy in the distances between the points that are close and the size of the set means that the points close together will be combined to a cluster at some later stage in the process and the symmetry of the larger shape will be noticed.

For points on long thin rods the algorithm ignores any cross sectional symmetry. Consider a long thin prism. The points at the ends are likely to be combined to a single point during the consistent clustering detection and the symmetry detection algorithm will con-

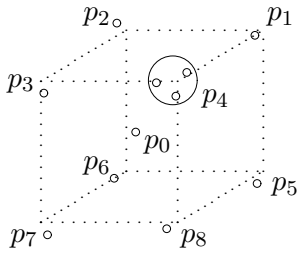


Figure 3.7: Approximate Symmetry Detection Example.

sequently only detect the mirror symmetry. In order for the rotational symmetry to be detected a much higher accuracy is required which may not be possible to realize under the condition of getting a consistent clustering. In retrospect, as a first analysis this is justified. However, a second pass could be used in which the model is first expanded orthogonally to its longitudinal axis. In a similar manner a point set whose primary structure is planar might be expanded orthogonally to the plane to examine its secondary symmetries. Both these ideas are simple to implement in the context of the original algorithm.

We used a similar approach to detect approximately congruent sub-parts in a geometric model [39]. But this is not further discussed here.

3.3.3 Symmetry Detection Example

To illustrate the operation of the algorithm consider the points shown in Figure 3.7. They are arranged approximately at the corners of a cube except that at one corner there are three points, and the centroid p_0 is also included. We detect three consistent clusterings. Firstly we have each point in a separate component, secondly the three points at one corner of the cube are combined into a single component labelled p_4, and finally all points are in a single component.

At a tolerance of 0.0 the model is clearly asymmetric and has only the identity transformation as an approximate symmetry. The first tolerance level produced by the algorithm, however, is half the distance between the closely grouped points in the corner. If the points on the other corners are not out of place by more than this, then the model, at this tolerance level, is approximately a cube with one truncated corner. This has a three-fold rotational symmetry about each diagonal of the cube, as well as three related mirror symmetries.

The next tolerance selected is about half the edge length of the cube. At this tolerance we might select the points p_0, p_1, p_2 and p_3 as the large tetrahedron. The centroid p_0 has to be mapped to itself. We can try to map p_1 to p_2 which succeeds as $d(p_0, p_1) \approx d(p_0, p_2)$.

Next we can try to map p_2 to p_4. While $d(p_0, p_2) \approx d(p_0, p_4)$ the distances $d(p_1, p_2)$ and $d(p_2, p_4)$ are not approximately equal. So we have to try another option for p_2, for instance, we could try p_3. In this case we have $d(p_0, p_2) \approx d(p_0, p_3)$ and $d(p_1, p_2) \approx d(p_2, p_3)$. Finally we can try to map p_3 to p_4 and check that $d(p_0, p_3) \approx d(p_0, p_4)$, $d(p_1, p_3) \approx d(p_2, p_4)$ and $d(p_2, p_3) \approx d(p_3, p_4)$. This means we found a valid mapping for the tetrahedron and have to complete the mapping by considering the registration coordinates of the points $p_4, \ldots, p_8$. This yields a mapping of p_4 to p_1, p_5 to p_6, p_6 to p_7, p_7 to p_8 and p_8 to p_5. Continuing with the other options of mapping the tetrahedron we determine the remaining cubic symmetries.

At tolerances greater than half the length of the edge there are no more consistent clusterings until, at about $\sqrt{3}/2 \approx 0.866$ of the edge of the cube, all the points are close enough to the centroid to be grouped together as one point. Of the four tolerances, which might be 0.0, 0.1, 0.5 and 0.9 of the edge of the cube, it is the middle two that characterise the approximate symmetry of the model. The symmetries at zero and infinite tolerance are generic and are not checked explicitly by our algorithm. All finite models have spherical symmetry at infinite tolerance, and only exactly symmetric models have non-trivial approximate symmetry at zero tolerance.

3.3.4 Complexity Analysis

Our algorithm consists of two methods where the symmetry detection method (Algorithm 3.4) is called for each consistent clustering of points at distinct tolerance levels determined by the clustering method (Algorithm 3.3). We will show that the time complexity of this method is $O(n^{3.5} \log^4 n)$ and in practice we expect $O(n^2 \log^4 n)$ time for engineering objects.

The clustering method requires $O(n)$ time to initialise the components for n points and $O(n^2 \log n)$ time to create a list of point pairs ordered by distance. Detecting distinct tolerance levels is done in a loop adding the point pairs in order as edges to the graph. Adding a point pair to a graph may merge two connected components of the graph. Storing the components as trees requires $O(\log m)$ time to check if a newly added edge lies in a single component where m is the average size of the component, and merging two components takes constant time. Thus, the time complexity of the clustering method is $O(n^2 \log n + n^2 \log m)$. As $m \leq n$, we have $O(n^2 \log n)$.

The symmetry detection method to detect the symmetry requires $O(n)$ time to find the tetrahedron. As the centroid is part of the tetrahedron and has to be mapped to itself we have to find matches for three more points in three nested loops. The time required to

match the first two points is bound by $O(n^2)$. A bound on the number of matches for these two points can be found from the result that the number of points a given distance apart is no more than $O(n^{1.5})$ [46]. After the centroid and two more points have been matched successfully, the next point can only match a constant number of points, but it takes $O(n)$ time to find out which points. So the total time taken in matching is bound by $O(n^2+n^{2.5}) = O(n^{2.5})$ time, producing $O(n^{1.5})$ possible matchings for the four points. In order to match the rest of the list, for simplicity, our implementation uses the direct $O(n^2)$ approach. So we have $O(n^{2.5})$ for the loop and $O(n^{1.5}) \times O(n^2)$ for the expansion of all possible matchings. This yields and overall time complexity $O(n^{3.5})$ for the symmetry detection method in our implementation.

However, a better approach treats the matching of the rest of the points as a problem of finding a data point in a rectangular piece of four dimensional space (defined by the distance from each of the four points). This can be done with a once off $O(n \log^3 n)$ setup time, for each set of four points, and then $O(\log^4 n)$ query time for each point [27]. This means that $O(n^{1.5} n \log^3 n) = O(n^{2.5} \log^3 n)$ time will be taken over the whole execution, doing the setup operations, and $O(n \log^4 n)$ time per set of four will be spent doing the matching. In each case this is $O(n^{2.5} \log^4 n)$, which is thus the improved time order of the symmetry detection method.

The clustering method requires $O(n^2 \log n)$ time. Each time symmetry detection is called the clustering method has made the partition of points coarser. Thus, the immediate limit to the number of calls is n. This limit can be reached by a collection of points built up one at a time, adding each point a bit further away each time. The symmetry detection method requires $O(n^{2.5} \log^4 n)$ time. So the order of the algorithm in the worst case is $O(n) \times O(n^{2.5} \log^4 n) + O(n^2 \log n)$ which is $O(n^{3.5} \log^4 n)$ using the improved point matching algorithm. Analogously our implementation with the simple point matching requires $O(n^{4.5})$ time.

The order of the symmetry detection method is actually an overestimate because the figure $O(n^{1.5})$ (for the number pairs at a given distance) is an overestimate. The actual value is between $O(n)$ and $O(n^{1.5})$, most likely closer to the lower than to the upper limit. Furthermore, with normal engineering objects, even reaching the bound of $O(n)$ is unlikely. For an object to have as many distinct partitions as points would require the object to have interesting features on as many scales as there are points. In practice an engineering object, even a complex one, will have only a few levels of size of feature. Thus the actual number of times that the symmetry detection method would be called is bound by a constant, and is $O(1)$. These two issues taken together lower the order by $n^{1.5}$ bringing the expected performance in practice on engineering objects to $O(n^2 \log^4 n)$ (and $O(n^3)$

using simple point matching).

In summary, our definition for approximate symmetries from Section 2.3.2 leads directly to an efficient and practical algorithm to detect global symmetries. Theoretically and practically our algorithm is faster than the best algorithms known for other definitions of approximate symmetry. However, it cannot be easily generalised to detect partial regularities and repetitions, which, at the moment, are better handled by methods like the one introduced in Section 3.2.

3.4 Special Values

We are interested in finding special values for length and angle features. While the other regularities discussed so far in this chapter relate to congruences and in general special arrangements of features, we also require parameter values for the exact realization of the geometries and the special arrangements. For instance, we require values for cylinder radii, edge lengths, etc., but also exact values for the cone angle of conical angle-regular directions, distances of axes arranged regularly on a grid, etc. We assert that these values are likely to be simple fractions on some scale as described below. Even if the original design did not specify such a special value, in engineering objects the exact values are usually subject to a tolerance and it is likely that we find a special value within this tolerance.

For each average parameter value p for a length or angle cluster, we seek some special values $f_b(n_k/m_k)$ close to p. Here n_k and m_k are integers and $f_b : \mathbb{R} \to \mathbb{R}$ is a member of a family of functions representing the scales on which we look for simple fractions, which depends on the parameter type. A special value is close to p if it is within the tolerance of the cluster or the appropriate ΔT_P, whichever tolerance is larger.

The functions f_b are usually of the form $f_b : v \mapsto vK_b$, where K_b are base units for length or angular measurements. For length units K_b depends on the measurements in the model and could for instance be 1.0, 0.1 or 2.54 (cm to inch conversion). For angles we use base units π and $\pi/180$ (degrees) and in addition the special function $f_{\mathrm{t}} : v \mapsto \arctan(v)$.

To find special values for some value v, we seek integer pairs n_k, m_k for each relevant f_b such that $|f_b^{-1}(v) - n_k/m_k| < t_0$. The tolerance t_0 depends on the cluster tolerance t_c and the appropriate ΔT. It also depends on the scale represented by f_b. For the linear f_b we set it to $t_0 = \max(t_c, \Delta T)/K_b$ to have consistent tolerance values independent of the constant K_b. As $\arctan(x) = x + O(x^3)$, we use $t_0 = \max(t_c, \Delta)$ for f_t, which is a good approximation as long as t_c is small. Note that for f_{t} we have the condition

$||v| - \pi/2| > \Delta T_A$ assuming that $v \in [-\pi, \pi]$, which can easily be achieved as v is an angle. The algorithm to find the integer pairs approximating some $w = f_b^{-1}(v)$ is described in the next sub-section. Note that we may have to eliminate duplicates, if we consider multiple functions f_b, e.g. $\arctan(1) = \pi/4$

Furthermore, we look for simple ratios between the average cluster parameters. Instead of selecting special values independently the ratios relate them to each other which may yield a more consistent choice of special values. For each pair of average parameters v_l and v_k, we seek two integers n_j, m_j such that $|v_l/v_k - n_j/m_j| < t_\mathrm{r}$, where t_r is a separate user-defined tolerance, and the clusters have been ordered such that $v_l/v_k \geq 1.0$. $1/1$ ratios are avoided as they represent approximate congruences in the cluster hierarchy. We also require that the difference between the two parameters is larger than the maximum of the two cluster tolerances. To find the integers we use the algorithm mentioned above with $w = v_l/v_k$.

Special values determined in this way have to be handled with care by the subsequent beautification steps. We may obtain several special values for each parameter and while a preferred value can be chosen using some merit function, there is no clear indication which one is the desired special value. Some special values are particularly simple, while others may be closer to the input data.

Special values might also depend on manufacturing and functional purposes. We do not consider the former for the ideal model. For values which are not simple rational numbers and depend on functional purposes, other specialised methods will need to be developed. Values depending on functional purposes are usually also subject to a tolerance. If we choose values within these tolerances, the beautified model should be usable.

3.4.1 Finding Simple Fractions

We have reduced the problems of finding special parameter values and ratios to finding a list of simple fractions approximating a real number w within a tolerance t_0. For this we assume that integer values of w are always special, and without loss of generality we also assume that w is non-negative. As integers are always special, we first find the closest integer a_0 to w and note it as a special value if it is within the tolerance t_0. The remaining problem is to find simple fractions for the signed remainder w_0.

Finding an integer relation between real numbers, using Euclid's algorithm for two numbers, or the PSLQ algorithm for more than two numbers [36], and recognising numerical constants [7] are related problems, but they assume that a close approximation is required.

Algorithm simple_frac$(\mathtt{w}, \mathtt{n}, \mathtt{m}, \mathtt{s}, \mathtt{M}, \mathtt{M}_0, \mathtt{neg}, \mathtt{t}_0, \mathtt{t}_1)$
Recursive algorithm to find simple fractions for $\mathtt{sn}/\mathtt{m} \pm \mathtt{w}$ which approximates w within the tolerance $\mathtt{t}_0$. w is the value in $[0, 1]$ which still has to be approximated by a fraction, n, m are the two integers representing the fraction $\mathtt{n}/\mathtt{m}$ found so far with the sign indicated by s, M is the maximum denominator, $\mathtt{M}_0$ is the value for M with which the recursion has been started, and neg indicates whether w has to be added or subtracted from $\mathtt{n}/\mathtt{m}$. Once the error is smaller than $\mathtt{t}_1$ no further simple values are computed. The algorithm reports the special fractions in a list.

I. Let $\mathtt{a} = 1$.
II. While the denominator $\mathtt{b} = \mathrm{round}(\mathtt{a}/\mathtt{w})$ is not larger than M:
    1. If $\mathtt{b} > \mathtt{a}$:
        A. Let $\mathtt{r} = \mathtt{w} - \mathtt{a}/\mathtt{b}$.
        B. If r is negative, set neg_r to true and $\mathtt{r} = |\mathtt{r}|$. Otherwise neg_r is false.
        C. If neg is false, the new numerator is $\mathtt{p} = \mathtt{nb} + \mathtt{ma}$. Otherwise:
            i. If $\mathtt{nb} < \mathtt{ma}$, the new numerator is $\mathtt{p} = \mathtt{ma} - \mathtt{nb}$ and $\mathtt{s} = -\mathtt{s}$.
            ii. Otherwise the new numerator is $\mathtt{p} = \mathtt{nb} - \mathtt{ma}$ and $\mathtt{neg\_r} = \mathtt{not}(\mathtt{neg\_r})$.
        D. The new denominator is $\mathtt{q} = \mathtt{ma}$.
        E. Reduce the fraction $\mathtt{p}/\mathtt{q}$ to simplest terms.
        F. If $\mathtt{r} < \mathtt{t}_0$, add $\mathtt{sp}/\mathtt{q}$ to the special values list, if it is not already in it.
        G. If $\mathtt{r} > \mathtt{t}_1$ and $\mathtt{sp}/\mathtt{q}$ was not already in the special values list, call simple_frac$(\mathtt{r}, \mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{M}_0\mathtt{M}, \mathtt{M}_0, \mathtt{neg_r}, \mathtt{t}_0, \mathtt{t}_1)$
        H. Let $\mathtt{a} = \mathtt{a} + 1$.

Algorithm 3.5: Recursive Algorithm for Finding Simple Fractions.

Instead, we try to find some special values not too distant from w, which are in some sense *simple* rather than *as close as possible* to w.

A straight-forward method for finding fractions within a tolerance is to determine the numerators n for a given denominator m by multiplying w_0 by all integers $m = 1, \ldots, M_0$, where M_0 is some maximum allowed denominator. If $|w_0 - n/m| < t_0$ with $n = \mathrm{round}$ $(w_0 m)$, i.e. if n/m is within the tolerance limit, then n/m is a suitable special fraction. However, as we also wish to find fractions close to w_0, we must use large values for M_0, which makes this method expensive.

If $1/(2t_0) < m$, then more than one n/m for a fixed m could be in the interval $(w_0 - t_0, w + t_0)$. To avoid this ambiguity we set the maximum allowed value for m to $M_0 =$

$$
\begin{array}{llll}
0.59 & = 1/2 & + & \underbrace{0.09}\\
 & & & = 1/11 \quad [\rightarrow \mathbf{13/22}] \quad - \quad 0.000909\\
 & & & = 2/22 \quad - \quad 0.000909\\
 & = 2/3 & - & \underbrace{0.076667}\\
 & & & = 1/13 \quad [\rightarrow \mathbf{23/39}] \quad - \quad 0.000256\\
 & = 3/5 \quad [\rightarrow \mathbf{3/5}] & - & 0.01
\end{array}
$$

Table 3.1: Finding Special Values for 0.59 with $\mathbf{t_0 = 0.05}$, $\mathbf{t_1 = 0.01}$ and $\mathbf{M_0 = 5}$.

$\mathrm{floor}(1/(2t_0))$. Additionally we limit M_0 by two configuration parameters M_{min} and M_{max}, say 3 and 10, to avoid too small or too large an M_0 if we have a large or small tolerance t_0.

An alternative to the simple method is to approximate real numbers using continued fractions [63]. We approximate w_0 by $a_1 = \mathrm{floor}(1/w_0)$ such that $w_0 = 1/(a_1 + w_1)$ and continue approximating w_1 and so on until w_l is smaller than some tolerance t_1. While this quickly computes a good approximation, it does not generate all special values close to w, especially as the iterative process based on the function $c : x \mapsto 1/x - \mathrm{floor}(1/x)$ behaves chaotically [23].

Our method thus combines the simple method with the continued fraction method to find a number of simple special values as well as some other special values closer to w_0 without checking a large number of possible denominators. This leads to a recursive algorithm checking some fractions with the simple method at each recursion level. On recursion level l, we have a fraction n_l/m_l approximating w_0 with a remainder w_l. In a loop for $a = 1, 2, \ldots$, we approximate w_l by fractions a/b with $b = \mathrm{round}(a/w_l)$ as long as $b < M_l$, where M_l is a limit for the denominators at this recursion level. We add each b/a to n_l/m_l. If this new fraction is not already in the list of special values, we add it to the list. If the new error w_{l+1} is larger than t_1, we call the algorithm recursively on w_{l+1} with an increased denominator limit $M_{l+1} = M_0 M_l$. This increase ensures that we still find valid approximations on higher recursion levels.

The complete recursive algorithm considering the signs of all involved values is listed in Algorithm 3.5. Due to the behaviour of the function c from above, it is more likely to find fractions with small denominators if $|w_0| > 0.5$. As w_0 is the remainder to the closest integer, we have $|w_0| \leq 0.5$. Thus, we start with $w = 1.0 - |w_0|$, $n = 1$, $m = 1$ and the flag `neg` $=$ `true` (otherwise we would start with $w = |w_0|$, $n = 0$, $m = 1$, `neg` $=$ `false`). M is initially M_0 and s is -1 if w_0 is negative and 1 otherwise.

Table 3.1 contains an example for 0.59. The special values generated in the process are

marked with $\rightarrow$. For the intermediate results $1/2$ and $2/3$, which are too far away from 0.59 to be accepted as a special values, we do one further recursion step to find more special values for the corresponding errors.

At recursion level l we have $w = y_{l-1} + x_l$ where w is the original value in $[0, 1]$ for which we seek special fractions. y_{l-1} represents a special fraction approximating w which has been found so far with the error x_l. x_l is approximated by fractions k/b_k with $b_k = \mathrm{round}(k/x_l)$ for $k = 1, \ldots, n$. We get the error

$$x_{k,l+1} = \left| \frac{k}{\mathrm{round}\left(\frac{k}{x_l}\right)} - x_l \right|. \tag{3.6}$$

We have $x_{k,l+1} \leq 1/2$ and thus also $x_l \leq 1/2$. Let $\varepsilon_k = \mathrm{round}(k/x_l) - k/x_l$ with $|\varepsilon_k| < 1/2$. Hence, we get

$$x_{k,l+1} = \left| \frac{k}{\frac{k}{x_l} + \varepsilon_k} - x_l \right| = \left| \frac{\varepsilon_k x_l}{\varepsilon_k x_l + k} \right| |x_l| \leq \frac{1}{3} |x_l| \tag{3.7}$$

as $|\varepsilon_k x_l| \leq 1/4$. Thus the error at recursion level l is smaller than $1/2(1/3)^l$ considering that at the first recursion level $l = 0$ we have an error of at most $1/2$. Hence, for tolerance $t_1 < 1$ the maximum recursion depth is limited by $L = \mathrm{ceil}(|\log(2t_1)|/\log(3))$.

As in each loop we have the condition $b_k < M_0^{l+1}$, we check at most $N_l = \mathrm{ceil}((1/3)^l (M_0^{l+1}/2 + 1/4))$ fractions k/b_k in each loop at recursion level l. However, we eliminate duplicates. We also frequently find fractions at small recursion levels l which are very close to x_l, i.e. some $x_{k,l+1}$ are very small for relatively small l. These two factors mean that many of the branches in the search for special values are eliminated at small recursion levels and the maximum recursion level L is only reached by a few branches. Hence, we actually check a considerably smaller number of fractions. It appears that we check at most a constant number of fractions on each recursion level (across all recursive calls) which is determined by M_0. Hence a rough bound on the overall number of fractions considered is $O(M_0|\log(t_1)|)$. This is considerably better than a worst case analysis considering the maximal values L and N_l which represent a limit on the recursion branches and the values checked in the loops rather than the usual amount of fractions tested.

Note that the algorithm can only miss special values whose denominator is larger than M_0. The precision increases with the depth of recursion. With appropriate t_1 and M_{max} we get the simple method as a special case. Also note that we can easily expand the method to detect special fractions of the form $(a_k/b_k)^r$ for $r = 1, 2, \ldots$ or $r = 1/2, 1/3, \ldots$ by calling Algorithm 3.5 with $w^{1/r}$ for all r in question.

The presented method to find special values as simple fractions on various scales is a combination of the simple search method with a continued fraction approach. Combining

both methods allows us to search for interesting special values in an interval around the original value. In addition we are also able to find close approximations of the original value without searching an extremely wide range of denominators in a loop.

3.5 Summary

We have presented four basic algorithms to detect approximate congruences, repetitions, symmetries and special values of scalar parameters. We have also shown how to use these algorithms to detect the common geometric regularities identified in Section 2.4. The algorithm designs are based on our definition of approximate regularities from Section 2.3.2, and they were refined based on experiments with various models (see Chapter 7). They were mainly adjusted to simplify the task of selection described in Chapter 4. The resulting algorithms provide reliable results indicating the presence of approximate regularities in the initial model at various tolerance levels. The regularities form the basis for the following step where an appropriate subset of the regularities is selected as basis for improving the initial model.

The basic assumption behind the algorithms is that we can describe certain aspects of boundary representation models by a set of distinct features. The features have to be sufficiently apart from each other in order to be distinguishable within the accuracy of the initial model. We take all features of a certain type or sub-type and then examine the set of features without further reference to the model. Thus, the features have to be distinguishable independent of their relation to the model. Note that for complex models, usually consisting of several sub-parts, this assumption may no longer be true. The features may not be sufficiently distinguishable without considering the structure of the model. The analysis methods given here were designed for models of simple to medium complexity with only a relatively small number of completely independent sub-parts (see Section 1.2).

Chapter 4

Regularity Selection and Model Rebuilding

The regularity detection methods described in Chapter 3 detect a large set of approximate regularities present in the initial model at various tolerance levels. The regularity detection considers various relations between the cells in the model to find potential regularities which do not have to be mutually consistent. As we do not use a strict tolerance limit we get some regularities at rather large tolerances. Moreover, the detection methods seek multiple relations between the same set of features, e.g. multiple special ratios between the same scalar parameters. Hence, the set of detected regularities is likely to contain many inconsistencies and an appropriate subset has to be selected. The selected regularities must be mutually consistent and should be likely to represent the original, ideal design intent. In this chapter we present a general selection strategy, which includes an algorithm to detect simple inconsistencies, and priorities used to select likely regularities. We also describe how to construct an improved model from the set of selected regularities. The consistency test will be discussed in Chapters 5 and 6 in form of a solvability test of geometric constraint systems.

The author devised a general regularity selection strategy (see Section 4.1) based on sequential selection of regularities in order to reduce the number of regularity sets checked for consistency and desirability. An exhaustive search over all regularity subsets or any large amount of subsets would be too expensive considering the high computational costs of testing the solvability of constraint systems. The general selection strategy includes merit functions for priorities to determine the sequence in which the regularities are considered (see Section 4.2) and a rule-based method for detecting simple inconsistencies (see Section 4.3). In [72] the selection strategy was first introduced in combination with a numerical solvability test (see Section 5.6). In [73] it was used with a more efficient topological solvability test (see Chapter 6). We also discuss a simple reconstruction method to rebuild a new model from the selected regularities in this chapter (see Section 4.4). It was developed by the author employing standard surface-surface intersection algorithms.

Note that problems related to fixing topological problems during reconstruction are not considered in this thesis (see Section 1.3).

4.1 Selection Strategy

Given a set of potential regularities, our task is to select a consistent subset which can be used to improve the initial boundary representation model so that it more closely represents the original, ideal design intent. The regularities detected in the initial model can be expressed by sets of geometric constraints (see Section 5.5). We assume that we have a solvability test algorithm available to determine if a given set of regularities described by constraints is solvable. This method has to be initialised with $\mathtt{G} = \mathtt{init\_solvability}(\mathtt{R})$ for the set of all regularities `R` in order to express the regularities with appropriate constraints and create a data structure `G` for the solvability test. `G` depends on the solvability test used and is not required if we directly compute a solution of each constraint system to test its solvability. As this approach is computationally too expensive (see Section 5.6) we store some information about the solvability properties of the constraint system describing the previously selected regularities such that we only have to consider the changes introduced by the constraints for a new regularity.

The method $\mathtt{solvable}(\mathtt{G}, \mathtt{S}, \mathtt{r})$ returns a Boolean indicating if the regularity `r` is consistent with the set of regularities `S` under the condition that `r` and `S` do not themselves contain internal inconsistencies. `G` is the data structure reported by `init_solvability` and contains information about the solvability properties of `S`. If `solvable` returns `true`, then it also updated `G` to represent the constraint system `S` expanded by `r`. Note that for the implementation `G` can be hidden in an object describing the solvability properties of a constraint system `S`.

The above `S` and `r` are constraint sets grouped by regularities. The solvability methods are described in detail in Chapters 5 and 6 using a purely numerical approach and alternatively a more efficient topological approach related to degrees-of-freedom analysis. For the topological approach `G` is a (hyper-)graph where the nodes describe the geometric objects (the cells from the initial boundary representation model). The non-directed (hyper-)edges describe a geometric constraint by connecting the involved nodes. For instance, a geometric constraint system describing distances between vertices forms a graph with the vertices represented by the nodes and the constrained distances between the vertices are described by the edges. As in general a constraint may refer to more than two cells we have a hyper-graph with edges between more than two nodes.

We also assume that we have a method priorities(R) which computes a numerical value $\omega(\mathtt{r})$ for each regularity r in the set of all regularities R, indicating the likelihood of r being part of the original, ideal design (see Section 4.2). Also recall that we have dependencies between regularities, e.g. we first require a parallel direction regularity before we can add an aligned axes regularity (see Section 2.2.2 for an overview and Section 5.5 for particular dependencies). Let dependencies(r) be the set of regularities the regularity r depends on. For the selection strategy we only need to know if all regularities in dependencies(r) are already selected.

Even if we do not consider topological changes of the model we still require a description of the topology in order to enforce the dependencies between the features and ensure that a valid model is generated from the regularities. The features not only have to fulfil the regularity constraints, but also have to describe the cells such that the geometric intersections of the cells are consistent with the algebraic complex (the geometric realisation of the cells has to be consistent with the algebraic complex), e.g. the position of a vertex must lie on all surfaces the vertex is part of. See Section 5.5 for a description of the geometric constraints T used to describe the topology for our models. We can handle such constraints as a special regularity describing the geometric relations required by the topology. This regularity always has to be included in the set of selected regularities. Note that as the regularities already describe exact relations between intersecting cells (either directly or indirectly), and as the regularity detection process considers all such relations, we only require a basic set of topological constraints. For instance, general constraints that simply require two faces to intersect (without specifying details about the intersection) are not necessary. We only have one topology regularity as we do not consider changes in the topology. But note that it is not sufficient to introduce multiple topology regularities describing topological alternatives. In order to consider topological changes we also have to change the cells in the boundary representation.

Furthermore, there are dependencies between features such as the requirement that the apex feature of a cone has to lie on the axis feature (see Section 2.2.2). These can also be described by constraints as discussed in Section 5.5. We assume that these relations are also part of the topological constraints T.

We have two basic choices for selecting subsets from the set of all regularities R. We could consider all subsets of R and try to find those which are consistent and desirable. Alternatively we could sequentially build a subset by adding elements of R in order of a priority indicating the desirability of each regularity. Note that in both cases we must always include the topology regularity and feature dependency constraints as well as check whether the dependencies between the regularities are met.

The naive approach for considering all subsets would create all of them and then check if they fulfil the dependency and solvability conditions. This is clearly too expensive, but the costs can be reduced by growing regularity subsets. When creating the subsets we can start checking for solvability of the subsets from the beginning and eliminate a subset and all its expansions as soon as a set was determined to be unsolvable. However, we would still have to check a large number of subsets for solvability which is an expensive test and we would also have to store a lot of information about the solvability properties of the different subsets. As even for models of medium complexity we can get $n \approx 500$ regularities or more this approach is not feasible as there are 2^n subsets.

It may be possible to refine the approach of growing regularity subsets using a backtracking strategy. We could start with some regularity set, expand it and backtrack whenever some condition on the selected set of regularities becomes true. This means that we would always only have one set of selected regularities. But this condition is not trivial. Clearly we can backtrack if we have found a set which cannot be expanded any further, but at the same time this set may be used to beautify the model. We could develop an absolute measure for the desirability of these sets and seek a set with a maximal desirability measure. But this would still mean that we have to check a large number of regularity sets for solvability and would have to explore all combinations of regularities which yield a solvable constraint system.

As an alternative for finding a set with a maximal desirability measure we could use an evolutionary [44] or randomised approach. Bit strings indicating which regularities are part of the selected constraint system can be used to represent the population. By evaluating the selected systems with respect to solvability and desirability a maximum could be found using a genetic algorithm. It is not clear how long such an approach would take, but testing for solvability of multiple constraint systems is expensive and so a straight-forward evolutionary approach may still take a long time. More sophisticated approaches considering solvability properties may improve this. But note that in any case a measure indicating the desirability of a regularity set is required which allows comparing different selections consistently. Geometric reasoning in combination with artificial intelligence techniques may provide an appropriate method for this evaluation. This is, however, beyond the scope of this thesis and left as future work for more complex models.

For the models considered in this thesis a simpler approach proved to be usable. Using a sequential selection method which tries to add regularities in order of a priority is suitable to improve simple to medium complexity models (see Chapter 7 for the experimental results). Instead of a method to determine the overall desirability of a regularity selection

we use priorities only to decide which regularity to choose in case of an inconsistency. This way we get a selection of regularities with high priorities by at most considering each regularity once for the solvability test.

We require a subset with likely regularities as indicated by priorities $\omega(r)$. While in principle we could select the set with the highest sum of $\omega(r)$ in the selected subsets, this actually requires that the sum has a meaning. Instead of being a priority, the $\omega(r)$ would have to be an absolute merit. While the sums do not have to be a probability distribution, there has to be an underlying probability distribution such that consistent decisions can be made based on these sums. In general it is easier to construct meaningful $\omega(r)$ for our regularities if they are only used to compare the likelihood of regularities being part of the ideal design.

Hence, we have chosen a sequential approach of constructing a single subset which is grown in order of the regularity priorities. We reject those regularities which create inconsistencies in the selected subset. A summary of the general selection method is given as Algorithm 4.1. Using the set of detected regularities `R` and the constraints `T` describing the model topology it selects a subset of `R` (including `T`) with high priorities which represents a solvable constraint system. It uses the solvability test methods from above, which are discussed in detail in Chapters 5 and 6, and a pre-selection mechanism based on simple selection rules as described in Section 4.3. The regularities and constraints are marked as *inactive* if they cannot be selected due to a selection rule, as *active* if they can be selected and as *selected* if they have been added to the set of selected regularities.

Initially we have a list of potential regularities. Calling `init_solvability` in step I adds sets of constraints between the features to each regularity to derive a low-level description of the high-level regularities suitable for the solvability test (see Section 5.5). Depending on whether we choose the numerical or topological solvability test, the constraints are created slightly differently. However, in both cases they form a (hyper-)graph `G` where the constraints represent the edges between the nodes which represent the cells with their features (see Chapter 5). `G` is used by the actual solvability test to record information about the solvability of the selected constraints. Moreover, `priorities` computes numerical priorities in $[0, 1]$ for each regularity.

The constraint sets describing different regularities may contain several constraints between the same geometric objects, e.g. multiple special values are considered for a single scalar parameter such as a cylinder radius. It can also happen that different regularities contain exactly the same constraints, e.g. an orthogonal system regularity and a planar angle-regular direction regularity with the base angle $\pi/4$ in a eight-sided regular prism, which suggest the same $\pi/2$ angle between two directions. We can detect these using the

Algorithm `select(R, T)`
Select and report a consistent subset of the regularity set `R` with high priorities. `T` is the set of constraints describing the model topology.

I. Call `G = init_solvability(R)` and `priorities(R)` to create appropriate constraint sets and priorities for the regularities in `R`.
II. Detect multiple constraints between the same geometric objects and run the initial regularity selection using the function `init_selection(R, G)` (see Section 4.3). This marks selectable constraints and regularities as active, and all others as inactive.
III. Call `solvable(G, S, t)` for all constraints `t` in `T` (in order to properly build up `G`), add them to the selected constraint set `S`, and mark them as selected.
IV. Repeat until `R` does not contain any elements marked active:
    1. Remove the active regularity `r` from `R` with the highest priority for which all regularities `dependencies(r)` it depends on are marked selected.
    2. Let $S_0 = \emptyset$ and for all `u` in the set of non-selected constraints `U` of `r` do
        a. If `solvable(G, S, u)` is `true` add `u` to S_0 and `S`, and mark `u` as selected.
        b. Otherwise, reject `r`, i.e. mark all constraints in S_0 as active, remove them from `S`, remove all regularities depending on `r` from `R`, mark `r` as inactive, call `check_selection(r, R)` to adjust the selection with respect to the selection rules from step II, and exit the loop of step IV.2.
    3. If all elements of `U` were successfully added, mark `r` as selected.
V. Report the selected regularities / constraints as the set of selected regularities.

Algorithm 4.1: Select Consistent, High Priority Regularities.

constraint graph `G`. If these constraints have the same constants, the related regularities are consistent, so we replace them by a single constraint with a reference to it from each involved regularity. This avoids adding the same constraint multiple times to the constraint system. If the constraints involve different constants, then the related regularities are mutually inconsistent and only one regularity can be added to the constraint system. This generates a very simple selection rule. There are more complex situations which generate similar rules. For instance, we detect cases where distance constraints between points are limited by incidence constraints between points, e.g. the distances between a point p_0, and two other points p_1 and p_2 which are constrained to be equal, have to be the same. These rules are detected by `init_selection` and an appropriate initial selection based on these rules is made as well, such that the regularities and constraints are marked as active and inactive. In general we try to mark as many high-priority regularities as active as allowed by the rules. At this stage we cannot simply remove the inactive regularities

with low priority as we may later still reject an active regularity, which in turn may allow the reactivation of a low priority regularity (see step IV.2.b).

In step III we select the constraints describing the topology of the model and add them to a constraint system `S`. While we know that they are consistent and can be enforced on the model as we have an initial model and do not consider topological changes, we still have to call `solvable` to properly build up the solvability information about the constraint system (at least in the general case, for specific implementation of `solvable`, like the numerical solvability test in Section 5.6, where `G` is not used, the call may be omitted).

In step IV we select the regularities. We take the active regularity with the highest priority from the list for which all regularities it depends on are already selected (step IV.1). The constraints of this regularity which are not yet added to the constraint system due to another regularity are checked in turn by `solvable` (step IV.2) and are added to the constraint system (step IV.2.a). If `solvable` reports that a constraint created an inconsistency in the constraint system, we reject the regularity and remove all constraints added for this regularity from the system (IV.2.b), and all regularities which depend on it are also removed from the regularity list. The new constraints for the regularity are all marked active again as they may still be added as part of another regularity. The regularity itself is marked inactive and we call `check_selection` to verify if the selection rules from step II allow the activation of another regularity due to the inactivation. The `check_selection` method may mark regularities active again which were previously marked inactive as a previously marked active regularity is now inactive (see Section 4.3 for details).

We only select each regularity as a whole and not parts of it. For instance, consider a cluster of parallel directions. If only one constraint in the regularity which makes a certain direction parallel to all the other directions creates an inconsistency, the complete regularity is rejected. For those cases Algorithm 4.1 could easily be modified to accept sub-regularities. As described in Section 5.5, a regularity is usually represented by a basic structure and then the relation between the features from the cells in the model are added to this structure. For instance, for parallel directions the directions are made parallel to an auxiliary direction cell. For more complex regularities like repetitions we have to create more than one auxiliary cell and specify relations between these cells. The constraints required for the basic structure of a regularity always have to be added. Constraints which specify the relation between the features of the cells and the basic structure may be omitted. However, if too many of these regularities are omitted, e.g. only one direction is in a parallel direction regularity, the regularity may become meaningless. Hence, more detailed considerations to derive additional rules for splitting regularities are required. Note that under the assumption that the initial model consists of a relatively small number of

independent sub-parts and has a consistent design intent (see Section 1.2), splitting the regularities is not a major issue and was thus omitted.

In the following we first describe the computation of the priorities in Section 4.2, then we present the ideas behind the selection rules in Section 4.3. Finally we show how to reconstruct an improved model from the results of the selection algorithm.

4.2 Prioritising Geometric Regularities

To resolve inconsistencies between regularities, a mechanism is required to select regularities which are likely to be present in the model's ideal design. As discussed in Section 4.1 we base the selection on a priority computed using merit functions. The decision about which regularities to choose is non-trivial and often there is more than one sensible choice.

The priorities we use are real numbers in the interval $[0, 1]$ where 1 indicates a very likely regularity. The selection process in Algorithm 4.1 ensures that regularities with higher priorities are selected if they are inconsistent with lower priority regularities. It does not, however, maximise the priority sum of all selected regularities. The priorities are only used to compare regularities in case of an inconsistency. They are not absolute measures for quality or desirability and thus they do not necessarily have to be based on a probability distribution or a similar mechanism.

There are many choices for the types of functions to compute the priorities. Functions modelled to compare regularities in a consistent way (e.g. functions which produce high priorities for regularities with small errors and low priorities for regularities with large errors) usually create similar selection results independent of the particular functions used. However, as the priorities indicate which regularities are preferred they also involve many user-defined constants. This plays a major role in the choice of functions as it should be possible for the user to predict the effect of changing the constants. Therefore we have chosen simple functions, often of the same type, such that constants with similar meaning have similar effects.

Functions completely different from those considered below could be used to model different selection criteria. Our particular functions mainly consider the accuracy to which the regularity is present in the initial model and the desirability of the regularity solely from a geometric point of view. So we consider measures for geometric aspects of the regularities relating to the symmetry types, geometry types, etc. and the particular arrangement and number of cells of the model involved in the regularity.

To decide the priority $\omega(r)$ of a regularity r we consider the accuracy to which the regularity's constraints are satisfied in the initial model, and the quality or desirability of the regularity. We compute $\omega(r)$ as a weighted average of a measure $\omega_e(r)$ of the numerical accuracy of the regularity in the initial model, a figure of merit $\omega_q(r)$ for the quality or desirability of the regularity, and a constant $\omega_b(r)$ describing a minimum desirability for each regularity type. We split the desirability in two values: $\omega_q(r)$ indicates the desirability with respect to the specific arrangements, constants, etc. involved in the regularity and $\omega_b(r)$ is a constant indicating a minimum for the desirability which only depends on the general regularity type. Furthermore, the weighted average of these values is weighted by a constant $\omega_c(r)$ indicating how common the regularity is (determined by surveying a range of engineering components, see Section 2.4 and [95]). Thus,

$$\omega(r) = \omega_c(r)\left(c_e\omega_e(r) + c_q\omega_q(r) + c_b\omega_b(r)\right) \tag{4.1}$$

where all constants and functions are in $[0, 1]$ and $c_q + c_e + c_b = 1$ (to get a weighted average in $[0, 1]$), e.g. $c_e = 3/6$, $c_q = 2/6$, $c_b = 1/6$. The maximum of $\omega(r)$ is $\omega_c(r)$ and the minimum for an undesirable regularity with high error is $\omega_c(r)\omega_b(r)c_b$. Thus with $\omega_c(r)$ we set a maximum priority depending solely on the regularity type which allows us to favour common regularities. This is modified by a value for the error $c_e\omega_e(r)$ and a value for the desirability $c_q\omega_q(r) + c_b\omega_b(r)$. Here $c_q\omega_q(r)$ indicates the desirability of the specific instance of the regularity and $c_b\omega_b(r)$ avoids to some extent that desirable regularities with relatively large errors get too low priorities.

$\omega_e(r)$ and $\omega_q(r)$ depend on specific parameters for the regularity and are not just constants as are $\omega_c(r)$ and $\omega_b(r)$. We use two basic function types for this. A function of the form

$$\mathbb{R} \to [0, 1], \quad x \mapsto \frac{1}{1 + cx} \text{ for } c \in \mathbb{R}_+ \tag{4.2}$$

is used if small values of x are desirable and large values are not. c indicates how fast the function drops towards zero. For cases where a specific value $p \in \mathbb{R}$ of x is favoured we use functions of the form

$$\mathbb{R} \to [0, 1], \quad x \mapsto \exp\left(-|c(x - p)|^d\right) \text{ for } c \in \mathbb{R}_+, d \in \mathbb{R}_+. \tag{4.3}$$

Here c indicates how fast the function drops towards zero, and d indicates at what distance of x from p the function starts dropping quickly towards zero (d also has an influence on how fast the function drops towards zero).

For our regularities we have angular errors in radians and length errors in length units from the constraints. Thus, for $\omega_e(r)$ we often have to combine an average angular error e_r and an average length error e_l. ω_e should be close to 1 for small errors and drop quickly

towards 0 when the errors become too large. By converting the angular error to length units using the maximum length L_m in the model we may define

$$\omega_e(r) = \frac{1}{1 + c_l(L_m \sin(e_r) + e_l)} \tag{4.4}$$

where c_l is a user-defined constant indicating the base length unit for the model, e.g. $c_l = 1$ or 2.54 (for inch in centimetre units). L_m could be chosen to be the diagonal of the bounding box of the model, but note that sometimes smaller values like the maximum edge length or a median / average value may be more appropriate to avoid emphasising angular errors between small faces or edges too much.

$\omega_q(r)$ describes the desirability or quality of the regularity if it could be enforced exactly on the model, computed by considering the regularity type, the arrangement of the involved cells, and features and special values involved. For instance, consider a planar rotational symmetry of directions. One quality aspect is the special value for the base angle of the rotation. We also take the number of pairwise adjacent faces into account as this relates the regularity to planar angle-regular directions in a prism (in a prism with n faces we have n pairwise adjacent faces). Furthermore, we consider the ratio of the number of possible different directions which can be part of a planar angle-regular arrangement to the number of directional features actually present in the regularity (for a base angle of π/n we have n different directions which may not be all occupied as the regularity can be incomplete). Another aspect which can be used is the number of directional features associated with cells of the same geometric type. We first define some quality aspects used to compute $\omega_q(r)$. The choice of the quality aspect is based on the information present in the regularities and the methods used to detect them. They indicate certain aspects of the regularity like special values or the type and number of faces involved, etc.

All special values involved in regularities have the form $v = f_b(\pm n/m)$ with integers n, m, and a function $f_b : \mathbb{R} \to \mathbb{R}$ representing the scale used to find the special values (see Section 3.4). f_b is usually of the form $x \mapsto xb$ where b is some base value like π or 1. We evaluate the quality of special values using the function

$$\omega_{sv}(m, f_b) = \frac{2q(f_b)}{2 + c_0 l + c_1 \left(\frac{m}{M^K} - 1\right)} \tag{4.5}$$

where $q(f_b)$ is a constant in $[0, 1]$ evaluating the desirability of the function f_b which indicates the scale (e.g. $q(x \mapsto x\pi) = 1$, $q(x \mapsto x\pi/180) = 0.8$, $q(x \mapsto \arctan(x) = 0.4$, $q(x \mapsto x) = 1, \ldots$), M is the base used to represent m (usually 10), l is one less than the number of digits required to represent m in the base M, and K is the number of consecutive zeros in the representation of m in base M starting with the lowest valued digit. c_0 is a constant indicating the importance of the length of the representation of

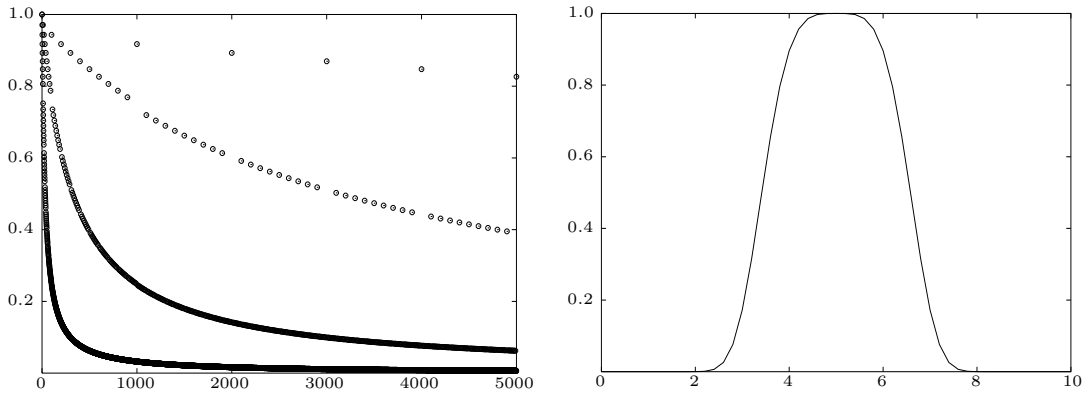


Figure 4.1: Graph of Quality Merit Functions for (a) Special Values of Denominators $\boldsymbol{\omega_{sv}(\mathbf{m}, \mathbf{x} \mapsto \mathbf{x})}$ over $\mathbf{m}$ and (b) Common Boundary Elements $\boldsymbol{\omega_a(\mathbf{X}, 0.5)}$ over $\mathbf{n(X)}$.

m, c_1 is a constant indicating the importance of the non-zero part of m, e.g. $c_0 = 0.01$, $c_1 = 0.005$. The graph of $\omega_{sv}(m, x \mapsto x)$ is shown in Figure 4.1(a) for special values of the form n/m. One can identify points on separate curves of the type of Equation (4.2) for values with m of the type $p * 1000$, $p * 100$, $p * 10$ with $p \in \mathbb{N}$ representing the non-zero part of m. The formula for ω_{sv} has been chosen to favour special values with a small non-zero part m/M^K and short representations in the base M.

Another quality aspect involves the number $n(X)$ in a set X of $|X|$ cells involved in the regularity which have a common boundary element. For instance, symmetrically arranged directions relating to pyramidal or prismatic arrangements of pairwise adjacent faces are favoured. It is computed as

$$\omega_a(X, p) = \exp\Big(- \big(c_w\big(p|X| - n(X)\big)\big)^{c_p}\Big) \tag{4.6}$$

with user-defined constants c_w and c_p (e.g. $c_w = 0.11$, $c_p = 4$) where the parameter p indicates the most desirable number of adjacent objects. We get high priorities for adjacent arrangements close to the desirable arrangement indicated by p. We can set X to the set of faces F which should share common edges, the set of vertices V which should be connected by edges or all cells O which should have a common boundary element. p is 1 if we desire arrangements of the elements in loops and 0.5 if we desire adjacent pairs. Figure 4.1(b) shows the graph of $\omega_a(X, 0.5)$ for an X with 10 elements over the number of elements $n(X)$ with common boundary elements in X.

For regular arrangements of directions or axes we have a base distance which is a special value $(n/m)b$. For symmetrically arranged directions and axes on a cylinder with base angle π/m we have $2m$ different positions, and for axes on a line we count the number

of positions between the first and the last occupied position. We prefer arrangements for which most of the possible positions are occupied and use a quality aspect $\omega_{ra}(r)$ when appropriate for a regularity r. If all consecutive positions are occupied $\omega_{ra}(r)$ for this arrangement is $\omega_{sv}(m, f_b)$. Otherwise, we have a list of smallest integers k for which all positions (starting with an arbitrary position) with the distance $k(n/m)b$ between them are occupied. For each k we add $m/(kn)\omega_{sv}(m/\gcd(m, kn), f_b)$ to ω_{ra}. For axes arranged regularly on a grid we have two orthogonal directions. For each of the directions we can project the occupied positions in the grid on a line and handle the line like equi-spaced axes along a line. The average of the quality for both directions gives the quality of the grid arrangement.

We also count the number $c(t)$ of geometric objects of the same type t involved in a regularity for the geometric objects O and compute the quality

$$\omega_t(O) = \frac{1}{|O|} \sum_{t \in \text{ObjectTypes}} c(t) \exp\left(-\Big(t_w\big(|O| - c(t)\big)\Big)^{t_p}\right) \tag{4.7}$$

with constants t_w and t_p, e.g. $t_w = 0.05$, $t_p = 2$. The graph of $\omega_t(O)$ is similar to $\omega_a(X, p)$ shown in Figure 4.1(b). It favours regularities with consistent object types, e.g. two parallel planes are preferred to a regularity making a cylinder axis parallel to a plane normal. Note that O for the computation of ω_t of polygonal loops are the faces connected by the edges in the loop.

Depending on the regularity type, we select appropriate quality aspects and compute their weighted average to give $\omega_q(r)$ as listed in Table 4.1. For example, for parallel directions we mainly consider the quality $\omega_{sv}(0, x \mapsto x\pi)$ of the parallel angle, but also prefer regularities with objects of the same geometric type. For symmetrically arranged directions in a plane, we put the main emphasis on the number of occupied positions and faces arranged in a loop as computed by $\omega_{ra}(r)$ and $\omega_a(F, 1)$, but also consider the special angle value for the planar arrangement and the geometric types involved.

Table 4.1 also lists the constants we use for ω_c and ω_b. ω_q, ω_c and ω_b were derived from a part survey estimating the frequency of regularities in simple mechanical components [95]. The constants were in addition refined to adjust the priority order of regularities in various example models. Users may adjust these values depending on the types of models arising in a particular application domain.

The values for all the constants considered in this section were chosen to give good results for all our example models considered in Chapter 7. In all cases optimising the constants for the individual model improves the selection of regularities. This mainly relates to particular instances of the models and other details rather than to major regularities of

Regularity r	$\omega_{\mathbf{c}}(\mathbf{r})$	$\omega_{\mathbf{b}}(\mathbf{r})$	$\omega_{\mathbf{q}}(\mathbf{r})$
Parallel directions	1.00	1.00	$0.8\omega_{sv}(0, x \mapsto x\pi) + 0.2\omega_t(O)$
Rotational symmetries of directions (planar)	1.00	1.00	$0.2\omega_{sv}(2, x \mapsto x\pi) + 0.3\omega_a(F, 1) + 0.1\omega_t(O) + 0.4\omega_{ra}(r)$
Rotational symmetries of directions (conical)	0.90	0.70	$0.3\omega_{sv}(m, f_b) + 0.3\omega_a(F, 1) + 0.1\omega_t(O) + 0.3\omega_{ra}(r)$
Orthogonal system	1.00	1.00	$0.6 + 0.3\omega_a(F, 1) + 0.1\omega_t(O)$
Special angle between directions	0.90	0.60	$0.8\omega_{sv}(m, x \mapsto x) + 0.2\omega_a(0, 0.5)$
Aligned axes	0.97	0.85	$\omega_t(O)$
Parallel axes arranged equi-spaced on grids	0.90	0.85	$0.3\omega_t(O) + 0.7\omega_{ra}(r)$
Parallel axes arranged equi-spaced on lines	0.88	0.75	$0.2\omega_t(O) + 0.8\omega_{ra}(r)$
Parallel axes arranged symmetrically on cylinder	0.95	0.90	$0.3\omega_t(O) + 0.7\omega_{ra}(r)$
Axes intersecting in a point	0.90	0.80	$0.1\omega_a(F, 1) + 0.9\omega_t(O)$
Equal positions	0.80	0.55	$\omega_t(O)$
Point set symmetries	0.95	0.90	1
Positions arranged equi-spaced on grids	0.90	0.85	$0.3\omega_t(O) + 0.7\omega_{ra}(r)$
Positions arranged equi-spaced on lines	0.88	0.75	$0.2\omega_t(O) + 0.8\omega_{ra}(r)$
Positions arranged equi-spaced on circles	0.88	0.75	$0.2\omega_t(O) + 0.8\omega_{ra}(r)$
2D partially equal positions	0.85	0.65	$0.5\omega_a(V, 0.5) + 0.5\omega_t(O)$
1D partially equal positions	0.83	0.60	$0.5\omega_a(V, 0.5) + 0.5\omega_t(O)$
Equal lengths / angles	0.90	0.75	$\omega_t(O)$
Special values for lengths / angles	0.85	0.70	$\omega_{sv}(m, f_b)$
Special ratios between lengths / angles	0.80	0.55	$\omega_{sv}(m, x \mapsto x)$
Similar polygons	0.85	0.55	$\omega_t(O)$

Table 4.1: Constants and Merit Functions for Regularity Priorities.

the model. We often have a choice for special values or ambiguous regularities which are about equally likely with respect to their accuracy in the initial model and their desirability

(e.g. we can either allow for a high inaccuracy and make two planes parallel or only accept small errors and thus favour an angle of say 3° between the planes). When adjusting the constants to choose special values and ambiguous regularities we usually use additional knowledge about the original design intent. Further details are given with the experiments in Chapter 7.

While the order for selecting the regularities can be adjusted by changing the constants, the rather large number of constants makes it hard to predict the effect of such changes for a user who is not aware of the internal relations. A method to choose the constants depending on a few multiple-choice questions presented to a user might improve this. In the current system the priorities are sufficient to improve the model, but a more intelligent decision process considering more complex relations between regularities and the model on a global level could improve the regularity selection with respect to design intent. Neural networks could be used to compute the priorities or directly make the decision which regularity to choose next. A belief network could also make decisions about which of the inconsistent regularities to include. This should be studied in combination with more sophisticated general selection methods for more complex geometric models where complexity here mainly relates to ambiguous and multiple, independent design intents present in the model.

The various priority and error values could also be used to filter out regularities with small priority values or large errors. This provides a simple mean for users to eliminate unlikely regularities and it can speed up the selection process as fewer regularities have to be considered. By providing the user with the priorities computed for a particular regularity set the obviously unlikely cases can often easily be identified manually. In particular a filter removing regularities with high errors (or low ω_e) is useful to reduce the number of undesirable regularities.

4.3 Selection Rules

To speed up the selection process we use a simple mechanism to avoid simple inconsistencies. There are obvious inconsistencies between sets of regularities of the same type involving the same geometric objects but with different special values. For instance, we have multiple special values for the distance between two points or multiple choices of the cone semi-angle for conical angle-regular direction features. These and similar inconsistencies can easily be detected and eliminated before we determine the solvability of the constraint system. We detect constraints between the same features or between features made equal by incidence constraints. This results in selection rules which are used to

make a pre-selection in step II of Algorithm 4.1 and adjust the selection options in case a regularity is rejected.

As the candidate regularities often specify multiple potential relations between the same features, eliminating simple inconsistencies reduces the number of regularity selections we have to check for solvability and thus speeds up the algorithm. However, not all inconsistencies can be removed in this way as there are more complicated dependencies induced by multiple regularities between different geometric objects.

By traversing the list of constraints from all regularities, we can detect sets of constraints between the same features. We first note constraints between the same features with the same constants involved to avoid adding the same constraints more than once to the system. These constraint sets are replaced by a single constraint and the regularities contain references to this single constraint. Furthermore, we detect regularities associated with constraints between the same features with different constants as these cannot be added at the same time. For this we create a rule indicating that only one of the contradictory regularities can be added simultaneously.

Further inconsistencies arise from incidence constraints (e.g. there can only be one special ratio between two scalar parameter pairs (a, b), (a, c) if we also have a constraint $b = c$). We expand the inconsistency check by considering these incidences when checking if the constraints involve the same geometric objects. This creates rules with a condition that the rule only applies if the incidence constraints are present. We get conditional rules of the form that if certain regularities are part of the constraint system then other combinations of regularities cannot be part of it. Note that in the current implementation we never consider more than one incidence constraint. In principle, we could expand this to multiple incidence constraints at the expense of increasing the costs of enforcing the rules (more rules have to be considered in a network of rules). Detecting these inconsistencies can easily be implemented using the constraint graph. Care should be taken to avoid checking too many incidence constraint combinations, and increasing their depth as this may take longer than checking these conditions with the solvability test.

In step II of Algorithm 4.1 the method `init_selection` is called. It first constructs selection rules using the constraint graph as described above, and then marks all regularities as active. For each of the rules it then calls Algorithm 4.2 to enforce a selection rule on the regularities by marking some of them as inactive. For regularities which remain active, all their constraints are marked active as well. As the constraints can be part of multiple regularities they have to have a separate marker.

To handle the selection rules algorithmically we require a simple way to represent them. The rules from the inconsistency detection stage can in general be expressed as selection

Algorithm $\texttt{enforce}(\texttt{r}, \texttt{D})$
Enforce rule $\texttt{r}$ with the regularity sets $\texttt{R}_1$, $\texttt{R}_2$ and integers $\texttt{n}_1$, $\texttt{n}_2$ on the constraint system without activating any of the regularities in the set $\texttt{D}$. This algorithm changes the activation markers of the regularities.

I. Deactivate regularities in $\texttt{R}_\texttt{k}$ to satisfy $\texttt{r}$, if there are more than $\texttt{n}_1$ active regularities in $\texttt{R}_1$ for all $\texttt{k} = 1, 2$ for which $\texttt{R}_1 \neq \emptyset$ and note the deactivated regularities in $\texttt{d}$:
   1. For $\texttt{k} = 1, 2$ find all active sets $\texttt{A}_\texttt{k}$ in $\texttt{R}_\texttt{k}$ and remove the $\texttt{n}_\texttt{k}$ elements with the highest priority.
   2. Find the set $\texttt{A}_{\texttt{k}_0}$ with the smallest largest priority.
   3. Call $\texttt{deactivate}(\texttt{c}, \texttt{d})$ for all $\texttt{c} \in \texttt{A}_{\texttt{k}_0}$.

II. Find a set $\texttt{X}$ of regularities which may be activated due to the deactivations by checking for each $\texttt{c} \in \texttt{d}$ and for each previously enforced rule $\texttt{e}$:
   1. If before step I for $\texttt{e}$ there were at least $\texttt{n}_1 + 1$ active elements in $\texttt{R}_1$ and now there are less, or $\texttt{R}_1$ is empty and there were $\texttt{n}_2$ active elements in $\texttt{R}_2$ and now there are less, add all inactive regularities from $\texttt{R}_2$ to $\texttt{X}$.
   2. If before step I for $\texttt{e}$ there were at least $\texttt{n}_2 + 1$ active elements in $\texttt{R}_2$ and now there are less, or $\texttt{R}_2$ is empty and there were $\texttt{n}_1$ active elements in $\texttt{R}_1$ and now there are less, add all inactive regularities from $\texttt{R}_1$ to $\texttt{X}$.

III. $\texttt{D} = \texttt{D} \cup \texttt{d}$.

IV. Try to activate all regularities in $\texttt{X}$:
   1. Recursively add all inactive children and dependent regularities of the elements in $\texttt{X}$ to $\texttt{X}$ and remove those which are in $\texttt{D}$ (including those which depend on them).
   2. In order of priority call $\texttt{active}(\texttt{c}, \texttt{X})$ for each element $\texttt{c}$ of $\texttt{X}$.

V. Call $\texttt{enforce}(\texttt{r}, \texttt{D})$ for all already enforced rules $\texttt{r}$ which may be affected by the activations in IV.

Algorithm 4.2: Enforce a Selection Rule.

rules which involve two sets of regularities R_1, R_2 and two non-negative integers n_1, n_2. A rule is violated if more than n_k elements of R_k are active for all $l = 1, 2$ for which $R_k \neq \emptyset$ (in general this can be done for any number of n_k, R_k pairs). One interpretation of this is, that if at least $n_1 + 1$ elements of R_1 are active, then at most n_2 elements of R_2 are allowed to be active. In this form R_1 and n_1 represent the condition, which is derived from the incidence constraints, and R_2 and n_2 represent the regularities creating the inconsistency under this condition. Note that if $R_1 = \emptyset$ and $n_1 = 0$ we have an unconditional rule.

Without trying to define this exactly, in a general sense the rules can be seen as hyper-

edges between regularity nodes. The nodes connected by each edge are divided into two sets. To enforce rules we can add the rules in sequence to this graph and adjust the labels (active/inactive) of the regularities to satisfy the new rule. As changing the labels also influences previously enforced rules we have to check them again. This can be done by following all edges in the graph which involve regularities for which the labels changed. In the worst case this means that we have to follow all paths in this graph (again without trying to define this precisely). While this makes our approach rather expensive, the number of paths is limited by the number of selection rules we detect. As the current implementation only considers a few simple, but rather common rules the number of paths we have to consider is relatively small. The practical costs of our approach depend on the number and type of rules detected and is hard to determine.

Due to the high costs of enforcing a rule, expanding the rules to more general cases is not feasible. Also for each inconsistency type which can be expressed with our rules a specialised method has to be implemented. While the rules on single distances in the feature space are easy to detect, there are far more complicated cases. For instance, consider the distances between $n \gg 2$ points in $\mathbb{E}^3$. There are many rules for which of these distances can be constrained at the same time. If we ignore special cases relating to special distance values any set of $3n-6$ distances is consistent. Specifying this in terms of our rules of what cannot be selected at the same time creates $\binom{n(n-1)/2}{3n-6}$ rules specifying that if $3n-6$ distances in the first set are selected than none of the remaining $\binom{n}{2} - 3n + 6$ distances can be selected (without considering any special cases). The selection rules were not created for these cases and there is a far more efficient way for the generic case (see Chapter 6).

Hence, with the selection rules in the current implementation we can only resolve simple inconsistencies between constraints on the same geometric features with different constants where at most one of the involved features is constrained to be incident with another one. As this is often the case with our regularities it can still reduce the number of regularities considered for the solvability test. The algorithm is not intended for other cases. In particular it is used to reduce the time required for the numerical solvability test. As the topological solvability test is considerably less expensive than the numerical test there is no real requirement to use this method. But note that we still have to check for equal constraints to avoid adding the same constraints more than once (which causes problems for the topological test as well as for solving the constraint system numerically). Thus, the detection of simple inconsistencies between constraints on the same features with different constants can easily be done and we still use it for the topological solvability test.

For each selection rule `r` Algorithm 4.2 is called with `D` $= \emptyset$. `D` is used to keep track of deactivated regularities which basically avoids following cycles in the graph. It adjusts

the selection status of the regularities such that in addition to the already enforced rules the new rule is also satisfied and the regularities with the highest priorities are active. For this we have to consider deactivating regularities in $\mathtt{R}_1$ or $\mathtt{R}_2$ to enforce the new rule `r` (step I). In case of a violation of a rule we have to deactivate elements in one of the $\mathtt{R}_\mathtt{k}$ such that at most $\mathtt{n}_\mathtt{k}$ elements are still active. Let $\mathtt{A}_\mathtt{k}$ be the set of elements we would have to deactivate in $\mathtt{R}_\mathtt{k}$ to satisfy the rule and $\omega_\mathtt{k}$ be the largest priority in $\mathtt{A}_\mathtt{k}$. Then we choose to deactivate the set with the smallest $\omega_\mathtt{k}$. This way we keep regularities with larger priorities active rather than trying to maximise the priority sum of all active regularities. Note that the priorities have been designed to compare regularities in case of an inconsistency, but not for a global desirability.

The deactivations may allow the activation of other regularities as it may reduce the number of active elements in one of the $\mathtt{R}_\mathtt{k}$ sets for another, already enforced rule (step II and IV). If we activated any other regularity we have to check already enforced rules involving these regularities (step IV). To avoid activating and deactivating regularities over and over again in the recursive calls of `enforce` we keep a set of deactivated regularities and do not allow any of them to become active again during the recursion. Note that this set is reset to $\emptyset$ for each initial call of `enforce` for each rule.

The function `deactivate(c,d)` used in `enforce` deactivates a regularity `c` and adds it to the set `d`. It also deactivates any regularities depending on it recursively (its parents in the hierarchy and those explicitly marked as dependent) and adds them to `d`.

The function `activate(c,X)` tries to activate a regularity `c` with the option that the regularities in the set `X` may also be activated. It first checks if the regularities on which `c` depends are active. If one is inactive and in `X`, it removes it from `X` and tries to activate it calling `activate` recursively. If not all dependencies are active or can be activated, `c` cannot be activated. Furthermore, for all already enforced rules involving `c` we check if `c` can be activated. Assume `c` is an element of $\mathtt{R}_2$ and more than $\mathtt{n}_1$ elements of $\mathtt{R}_1$ are active or $\mathtt{R}_1$ is empty. In order of highest to lowest priority, the inactive regularities in $\mathtt{R}_2$ which are in `X` are removed from `X` and we try to activate them recursively as long as less than $\mathtt{n}_2$ elements of $\mathtt{R}_2$ are active or `c` is next in the order (which means the rule allows the activation of `c`). We handle the case where `c` is an element of $\mathtt{R}_1$ similarly. If all the rules involved allow the activation of `c`, `c` is activated. Any previously enforced rule for which the activation of `c` caused exactly $\mathtt{n}_\mathtt{k}+1$ elements of $\mathtt{R}_\mathtt{k}$ to be active for either $\mathtt{k}=1$ or $\mathtt{k}=2$ is added to the set of rules which have to be checked in step V of Algorithm 4.2.

For instance, assume that we have four regularities A, B, C, D, which are all marked active and for which $\omega(A)<\omega(B)<\omega(C)<\omega(D)$, and three rules we wish to enforce (see Table 4.2). First we add the rule with $R_1=\emptyset$, $n_1=0$, $R_2=\{A,B\}$, $n_2=1$, i.e.

A	B	C	D	3 rules enforced in sequence:
•	•	•	•	Initially all regularities are active
				Rule 1: $R_1 = \emptyset, n_1 = 0, R_2 = \{A, B\}, n_2 = 1$
□	•	•	•	At most either A or B, not both, can be active → Deactivate A due to lower priority
				Rule 2: $R_1 = \{B\}, n_1 = 0, R_2 = \{C, D\}, n_2 = 1$
□	□	•	•	If B is active, then at most either C or D, not both, can be active → Deactivate B due to lower priority
				Recheck rule1, as B was modified
•	□	•	•	Rule 1 allows reactivation of A → Activate A
				Rule 3: $R_1 = \{B, D\}, n_1 = 0, R_2 = \{A, C\}, n_2 = 1$
□	□	•	•	If B or D are active, then at most either A or C, not both, can be active → Deactivate A due to lower priority
				Recheck rule 1, as A was modified
□	□	•	•	Cannot reactivate B due to rule 2 → Final regularity activation

Figure 4.2: Selection Rule Enforcement Example for Regularities $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ with $\boldsymbol{\omega}(\mathbf{A}) < \boldsymbol{\omega}(\mathbf{B}) < \boldsymbol{\omega}(\mathbf{C}) < \boldsymbol{\omega}(\mathbf{D})$ (• indicates active, □ inactive regularities).

either A or B, not both, can be active. Calling `enforce` deactivates A due to the lower priority. We add a second rule with $R_1 = \{B\}$, $n_1 = 0$, $R_2 = \{C, D\}$, $n_2 = 1$, i.e. if B is active, then either C or D, not both, can be active. We deactivate B in this case. This influences the selection for the first rule and we can activate A again. Finally, we enforce a third rule with $R_1 = \{B, D\}$, $n_1 = 0$, $R_2 = \{A, C\}$, $n_2 = 1$, i.e. if either B or D is active, then either A or C, not both, can be active. We deactivate A to satisfy the rule, leaving only C and D active. Considering the first rule we have to check if B can be activated due to the deactivation of A. Calling `activate` initially succeeds in this, but the recursive check deactivates B again due to the second rule.

In step IV.2.b of Algorithm 4.1 we call `check_selection(r, R)` to check if the deactivation of the regularity `r` allows us to activate some other regularities due to the selection rules. The method for this is very similar to Algorithm 4.2. We basically call `activate`

for each of the deactivated regularities which are part of selection rules that also refer to `r` and call Algorithm 4.2 for rules which may be violated due to the newly activated regularities.

4.4 Constructing an Improved Model

The regularity selection process (Algorithm 4.1) results in a list of consistent regularities as determined by the solvability test which have in general high priorities. They are represented by constraints which describe the improved model. In this section we briefly describe how to construct an improved model from this constraint system and the additional information from the initial model.

If we use a numerical solvability test we already have the numerical solution of the constraint system from the last successful call of the solvability test. The topological solvability test does not compute a solution and thus we still have to solve the system. Our version of the topological solvability test does not produce a decomposition of the constraint system which would allow a symbolic solution, so we simply call the numerical solver to find a solution of the constraint system. This is followed by the actual reconstruction process.

A numerical approach is justified as we already have a valid boundary representation model which should be close to the improved model. It can be used to provide a good initial value for the numerical solver. As the initial value is close to the solution we only require a few iterations. If a solution exists the optimisation method will converge to one that is close to the initial value [42]. In the case of a discrete set of solutions this is sufficient to improve the model. In the case of non-discrete solutions close to the initial value, the detected regularities are insufficient to specify a unique model. As we aim to find a large number of regularities this is unlikely, and did not occur in any of our tests. However, we can still choose the solution closest to the initial model. Symbolic methods based on the results of the solvability test using the graph representation of the constraint system are an alternative to our numerical solver. It may be possible to modify the solvability test to create a decomposition plan for a symbolic constraint solver [51, 52]. For the description of the numerical solvers we used see Section 5.6. The topological approach is described in Chapter 6.

From the numerical solution of the constraint system an improved model is rebuilt using the topological information from the initial model with the feature values obtained from the solution. We create new faces using the solution of the constraint system and reintersect them to obtain the complete model. From the solution of the constraint system

we have the vertex positions and the faces. We get sharp edges by using a standard surface-surface intersection algorithm for the adjacent surfaces (as provided by the ACIS solid modelling kernel). For a closed curve the intersection gives the edge immediately. Otherwise we have to limit it with vertices to get the edge. If multiple options arise for the intersection, the information in the initial model is used to determine which part of the intersection curve is required, e.g. to distinguish between a convex and concave cylindrical surface.

In the case of a smooth edge, the intersection is tangential and cannot be computed by a standard intersection algorithm. Such intersections relate to special cases which can be computed separately for all combinations of the considered surface types. For instance, as we do not consider topological changes we may try to intersect two equal planes. By checking the parameters of the planes this case can easily be detected and the intersection is simply computed as the straight line between the two vertices in the intersection. Other tangential intersections with planes, e.g. a plane and a cylinder, can be handled similarly. More generally, this applies to any tangential intersection which is a straight line. Another case is a tangential intersection between a torus and a cylinder in a circle. This case can be detected by checking whether the cylinder axis is tangential to the central circle of the torus and the cylinder radius is equal to the minor radius of the torus. The point where the cylinder axis is tangential to the circle gives the centre of the intersection circle whose radius is determined by the common radius. Note that in general it may be useful to explicitly compute special intersections which are not necessarily tangential (e.g. the intersection of a cone on top of a cylinder) in order to get analytic representations of the intersections rather than free-form curves.

Additional adjustments include moving the object to a special position and orientation with respect to certain determined vertex positions, orthogonal systems and main axes. Topological changes and problems relating to holes in the model caused by adjacent surfaces which do not properly intersect due to the adjustments are not considered here. The changes have to be small enough such that the topology need not be changed. The topological issues are discussed separately [40] and are not part of this work.

4.5 Summary

This chapter describes the core of the regularity selection and the general strategy used for it. We select regularities sequentially in order of a priority considering dependencies between them. Geometric constraints are used to determine if the selected regularities are consistent. From the solution of a selected, consistent constraint system created from

regularities with high priorities, an improved model is reconstructed. A numerical and a topological approach to the solvability test are described in the following chapters. To reduce the number of calls to the solvability test we also use simple selection rules. These are of particular importance for the numerical solvability test. The topological test is fast enough such that an even simpler version of the selection rules can be used.

Our rather simple selection strategy is sufficient for the types of models we consider in this thesis (see Section 1.2). The results of the experiments reported in Chapter 7 show that the method selects a suitable set of regularities. For more complex models with many independent sub-parts, etc. a more sophisticated approach may be required. However, this is beyond the scope of this thesis and is left as future work. Here we are more interested in regularity detection and determination of the solvability properties of geometric constraint systems.

Chapter 5

Geometric Constraints

One of the crucial subtasks of the regularity selection strategy discussed in Chapter 4 is the test whether a consistent set of regularities remains consistent if an additional regularity is added. As we express the regularities in terms of geometric constraints, this test can be implemented by checking whether a solvable constraint system remains solvable when additional constraints are added. While we ultimately require a solution of the geometric constraint system which represents an improved geometric model, the main aspect for the *selection* process is the solvability of the constraint system. In this chapter we give a general overview of geometric constraint systems, and the different approaches to testing for solvability and computing a solution. We also present a simple numerical solvability test and solver employing optimisation methods, and discuss ways to represent the regularities using geometric constraints. A more sophisticated approach to the solvability problem based on the topological dimensions of the involved sets will be presented in Chapter 6.

The main purpose of this chapter is to give a general overview of previous work in the area of geometric constraints. It also very briefly introduces the author's topological interpretation of geometric constraint systems which was first presented in [73] (see Section 5.4) and the representation of regularities using constraints [72, 73] (see Section 5.5). Furthermore, a numerical solver employing quasi-Newton (or variable metric) optimisation methods developed by the author [72] is described in Section 5.6.

5.1 Geometric Constraint Systems for Beautification

Geometric constraints have a wide range of applications in geometric modelling. For their general role in geometric modelling see [53]. For an overview of the representation of geometric constraints and geometric constraint solvers see, for instance, [17, 33, 51]. In the following we will introduce geometric constraint systems in general, and discuss their use for the beautification problem, in particular.

A *geometric constraint system* consists of a set of *geometric elements* like points, lines, planes, etc. and a set of *geometric constraints* upon them specifying desired relations between the elements. The constraints may, for instance, specify the distance between two points, the angle between two planes, or require that a point lies on a line or a plane, etc. We call finding one (or all) solutions of a geometric constraint system a *geometric constraint solving problem*. For a *geometric constraint solvability problem* we try to determine if a geometric constraint system has at least one solution (this includes a finite as well as an infinite number of solutions). Note that the solvability problem does not necessarily require the solution of a constraint system, and thus there may be a more efficient way to decide if the system is solvable since we require in some sense less information.

For the beautification problem, the geometric elements of a constraint system are the cells of the boundary representation of the initial model (see Section 2.1). As we do not consider topological changes to the model and construct the improved model by intersecting the geometry of the faces, the constraint system only determines the geometry of the cells. Basic topological relations are expressed by constraints on the geometry ensuring that it is compatible with the topology. The regularities relate to the features of the cells (see Section 2.2) and are described by constraints upon the cells specifying relations between their features. Constraints on base features can be expressed directly by constraints between the cells. For constraints on extended features we have to introduce auxiliary cells representing the extended features (e.g. an auxiliary line to represent a cylinder axis feature) and represent the dependencies between the base and extended features by additional constraints. Details of the construction of the constraint system to improve an initial model are given in Section 5.5.

The geometric constraints used to represent regularities can be classified into three types. We have *incidence* constraints which require that two cells or two features of the same type are equal (either with respect to the geometric realisation of the cells or the features in the feature space), *distance* constraints which require that two features of the same type are a positive distance apart in the feature space, and *subset* constraints which require that a cell is a subset of another cell (with respect to the geometric realisation). Note that only subset constraints can relate different types of feature spaces with each other.

For the regularity selection strategy described in Chapter 4 we have a geometric constraint solvability problem besides the regularity selection problem. We only require a solution of the complete constraint system selected to improve the model. In particular we are interested in determining whether a given, solvable geometric constraint system (possibly the empty system) remains solvable when we add an additional set of consistent constraints. We already have an initial model which can be seen as a solution to a similar constraint

system close to the solution of the new constraint system we try to construct. Hence, the question of the cardinality of the set of solutions (especially whether there is a finite or an infinite number of solutions) is relatively unimportant. Theoretically we can define the desired solution as the one closest to the initial model. Practically, we choose the solution determined by a numerical solver, which is very likely to be close to the initial model (see Section 5.6). However, it is of course desirable to have a constraint system with a discrete set of solutions which are sufficiently apart from each other such that we have a distinct solution close to the initial model. While we do not explicitly check for this, during the selection all regularities are considered in sequence. Thus, if we detect sufficient approximate regularities in the initial model, it is likely that we will select sufficient constraints to get a finite set of solutions. We can encounter certain cases where we have an infinite set of solutions, but without further regularities, we cannot add more constraints without additional regularity detection methods. The large number of regularities detected by our methods makes this situation unlikely. The main problem during selection is to ensure that we do not add too many constraints to the system, so that it has no solution.

For an inconsistent system we can only find solutions which approximately (within a possibly large tolerance minimising, for instance, the least squares error) satisfy all constraints. In this case, none of the regularities nor the required topological relations are likely to be satisfied by the resulting model. Hence, the resulting model will be broken and is very unlikely to represent any kind of improvement over the initial model.

In this chapter we present a simple *numerical* test (see Section 5.6) which determines the solvability of a constraint system by minimising a least squares error function. Using this means we have to numerically solve many constraint systems during the selection process, so this test dramatically slows down the beautification algorithm. But while our main problem is solvability, we also require a solution for the final set of selected constraints. For simplicity we always use the numerical method to solve the final constraint system, but alternative decomposition-recombination methods [51, 52] together with some other numerical or symbolic solver could certainly be employed as well. We will present a more efficient solvability test in Chapter 6.

In the following sections we first review numerical and symbolic algebraic as well as rule-based geometric approaches to geometric constraints (for a general overview also see [17, 33, 51]). Furthermore, we discuss a graph-based or topological interpretation of constraint systems which is the basis for degrees-of-freedom analysis and our solvability method discussed in detail in Chapter 6. Then we introduce our particular constraint types and how they are used to represent our regularities. Finally, we present our numerical solver in detail.

5.2 Algebraic Interpretation of Geometric Constraints

We can interpret a geometric constraint system as a non-linear equation system. Each geometric constraint is expressed as one or more equations on parameters describing the geometric elements. This gives a system of m equations for n parameters $x = [x_1, \ldots, x_n]^t$,

$$F(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} = 0 \quad \text{with } F: \mathbb{R}^n \to \mathbb{R}^m \text{ and } f_l: \mathbb{R}^n \to \mathbb{R}, l = 1, \ldots, m. \quad (5.1)$$

The f_l are usually polynomial functions, but in some cases they may involve analytical functions (e.g. trigonometric functions if we have variable angles). Methods which handle the constraint system as a non-linear equation system are based on the algebraic properties of the constraints. Such methods can be further distinguished as numerical and symbolic. In the following we give a brief overview of these methods. While they are all aimed at computing a solution of the constraint system, they can also be used as a solvability test.

5.2.1 Numerical Methods for Geometric Constraints

Numerical methods solve a particular instance of a geometric constraint system by producing a numerical solution using an iterative algorithm. Unlike symbolic methods (see Section 5.2.2), they do not solve *classes* of geometric constraint systems, creating a solution depending on certain constant parameters.

A classical, widely used method to find the solution of a non-linear equation system is the Newton-Raphson method [101]. Let x^* be a solution of a non-linear equation system $F(x) = 0$ and let $DF(x) = \left[\frac{\partial f_l(x)}{\partial x_k}\right]_{\substack{l=1,\ldots,m \\ k=1,\ldots,n}}$ be the functional matrix of partial derivatives of F. For an approximation x_0 of x^* we have $F(x^*) = 0 \approx F(x_0) + DF(x_0)(x^* - x_0)$ using the first order approximation of F at x_0. For non-singular $DF(x_0)$ we can solve $F(x_0)+DF(x_0)(x_1-x_0) = 0$ for x_1, which yields the Newton-Raphson iteration formula. This and similar methods like the secant method directly seek the solution of an equation system starting with some initial value which approximates the solution.

Alternatively, we can use an optimisation method trying to minimise a function indicating the error in an approximate solution to $F(x) = 0$ [28, 62, 122]. Given an error function $t_F: \mathbb{R}^n \to \mathbb{R}$, we try to find a solution to the equation system $\nabla t_F(x) = 0$, which is a necessary condition for an extremum of t_F. A general iteration formula to find a solution to this equation system is $x_{l+1} = x_l - \sigma_l d_l$ where x_l is an approximation to a solution, σ_l is a step size in $\mathbb{R}_+$ and $d_l \in \mathbb{R}^n$ is a direction for $l = 1, 2, \ldots$. The difference between

such optimisation methods is in the different ways of finding σ_l and d_l. The values are chosen such that there is a sufficient decrease of function values in the sequence $t_F(x_l)$, $l = 1, 2, \ldots$, in order to guarantee the convergence of the method to a (usually local) minimum. For a least squares error function $t_F(x) = F(x)^t F(x)$, methods developed to minimise quadratic functions are suitable. This means that quasi-Newton (or variable metric) methods as discussed in Section 5.6 are of use when solving geometric constraint systems by optimisation.

Iterative equation solvers as well as optimisation methods depend on the initial value for the iteration and may converge to different values for different initial values. For the Newton-Raphson method, the sets of initial values for which the method converges to a particular solution are fractal sets [105]. This makes it hard to predict which solution we will get from a given initial value, and how to change the initial value in order to find a different solution. Hence, using the initial value from our initial model may not result in a solution which produces an improved model close to the initial model.

Using the optimisation approach it is more likely that we will find a solution close to the initial value [42]. This makes this approach more suitable for our problem, as we have a good initial value from the initial model. However, the optimisation method may converge to a local rather than a global minimum, and a local minimum is not necessarily a solution to the constraint system. Using certain safeguards for the optimisation (see Section 5.6), and the fact that the initial value is close to the desired solution, makes it very likely that we will find a global minimum of the error function and thus a solution to the constraint system. This has been confirmed by our experiments (see Chapter 7).

Hence, while equation solvers and optimisation methods have major drawbacks for solving constraint systems in general, the optimisation method is suitable for solving the constraint systems arising in beautification. However, as it requires a relatively large amount of time to solve the constraint system compared to our other algorithms for beautification and the topological solvability test, it is not suitable as general solvability test.

For completeness we also shortly mention homotopy methods which avoid the problem of finding a good initial value. The basic idea is to convert the solution of a known, "simple" equation system into the solution of the given system. The homotopy equation is

$$H(x, \lambda) = (1 - \lambda)G(x) + \lambda F(x) \quad \text{for } \lambda \in [0, 1], \tag{5.2}$$

where $G(x)$ is the simple system and $F(x)$ is our given system. Using Equation (5.2) we can transform solutions of $G(x) = H(x, 0) = 0$ into solutions of $F(x) = H(x, 1) = 0$. The parameter λ determines the homotopy path from a solution of $G(x)$ to a solution of $F(x)$. For instance, a predictor-corrector approach can be used to find this path [3]. Let x

be the value on the homotopy path for a given $\lambda = \lambda_0$ (starting at a solution of $G(x) = 0$). A predictor computes x_1 as an approximate solution of $H(x, \lambda_0 + \delta) = 0$ for a small $\delta \in \mathbb{R}$. Then a corrector uses x_1 to compute a point on the homotopy path for $\lambda_0 + \delta$. Obviously the choice of the system $G(x)$ is crucial for an efficient homotopy method. It determines the number of paths from $G(x)$ which are followed to $F(x)$. Ideally each solution of $G(x)$ should lead to a unique solution of $F(x)$, but this is not true in general and often solutions of $G(x) = 0$ lead to solutions of $F(x) = 0$ at infinity. Such solutions are difficult to detect and expensive to compute [96].

For solving general geometric constraint systems homotopy methods are numerically stable and versatile methods to find all solutions of a constraint system [4]. However, they also require expensive computations and thus are limited to small systems. For further details see [3, 31, 71, 84, 113].

5.2.2 Symbolic Methods for Geometric Constraints

Symbolic methods for solving constraint systems are based on algebraic elimination in combination with root finding methods. One advantage of these methods is that they can solve classes of constraint systems. The solution can be computed such that it depends explicitly on parameters describing a particular instance of a constraint system class.

One class of symbolic methods is based on *polynomial ideals*. Let F be a set of polynomials $f_1, \ldots, f_n$. The ideal of F is $I\langle F\rangle = \{h_1 f_1 + \cdots + h_n f_n : h_l \in K[x_1, \ldots, x_n]\}$ where $K[x_1, \ldots, x_n]$ is the ring of n-variable polynomials over the coefficient field K. This means we can interpret F as the basis for the ideal $I\langle F\rangle$ which is generated by multiplying the elements of the polynomial ring with the elements of F. If we have another set G of polynomials $g_1, \ldots, g_n$ which forms a basis of $I\langle F\rangle$, then any root of F is also a root of G. A Gröbner basis is a special basis of $I\langle F\rangle$ which allows us to answer questions about consistency, ambiguity, solvability, etc. Thus the core of this approach is to transform the original system into an equivalent system represented in the Gröbner basis. The transformed system is also called a triangular system and it can be solved easily by finding roots of univariate polynomials and back-substitution. For a detailed discussion of the theory see [19, 20, 24]. It has been applied to geometric constraint systems in [65, 114].

A similar method is based on Ritt's characteristic (or triangular) sets [111, 112]. It has been used in [22, 138, 139] for theorem proving in mechanical geometry. This method decomposes the solution set of an algebraic system into set-expressions involving the solutions of simpler systems. Further methods relating to the decomposition of solution sets of polynomial systems into triangular sets and solving over $\mathbb{R}$ are presented in [61,

80, 108, 109, 110, 132]. A detailed discussion about different concepts of triangular sets can be found in [81].

Another approach can be interpreted as a generalisation of determinants for linear systems. This approach is based on the resultant of a polynomial constraint system [21]. The main idea is to expand the original system into a larger system which contains the terms of the original system as distinct variables, and then decompose and solve this system as a sparse polynomial system [43, 123, 124, 125].

A major drawback of all such symbolic methods is that they require exponential running time. Thus they cannot handle problems of the size encountered in beautification. One of the main reasons for this is that the algorithms see the problem as solving a general polynomial equation system, which covers a class of problems in some sense larger than geometric constraint solving problems. Methods which consider the specific structure of geometric constraint problems are expected to be more efficient.

5.3 Rule-Based Geometric Interpretation of Geometric Constraints

One way to consider the geometric structure of constraint systems is to represent the constraints as a set of rules and predicates. By employing rewrite rules, which represent the geometric knowledge of the solver, we attempt to find a construction sequence that satisfies all constraints. Basically, the predicates which describe the desired relations between the geometric elements are transformed into predicates which describe the position, etc. of the geometric elements.

Borning [14] presents such an approach by extending Smalltalk. A constraint is represented as rule and a set of methods that can be invoked to satisfy the constraint. The rule constructs an error expression indicating whether and how well a constraint is satisfied. The methods describe alternate ways of satisfying the constraint. By calling one of them the constraint will be satisfied. The rules and the methods are provided by the user and the system's task is to find ways to locally satisfy the constraints such that eventually a global solution is found. Another approach which computes all solutions symbolically is presented by Brüderlin [18]. Here the predicates are represented in Prolog and the construction steps are evaluated by calls to Modula-2 methods.

An approach of Verroust et al. [131] can handle dimensional, tangency and radius constraints. The constraint system is expressed in terms of mutually constrained distances

and angles which are evaluated simultaneously. Joan-Arinyo et al. [59] extend this basic system and prove its correctness. In [60] the method has been combined with a symbolic equation solver.

Gao et al. [41] describe a constraint solving system that generates a construction sequence from a declarative description of geometric diagrams. A global propagation method is used to solve problems which involve loops. Similar to a local propagation method [82], it tries to determine the position of a new geometric element from positions of already determined geometric elements. But in addition to constraints involving the new geometric element, it also considers implicit information derived from other constraints. This implicit information is computed prior to the actual constraint solving and is deduced using a fixed set of geometric axioms.

Rule-based methods explicitly represent geometric knowledge, and the representation of this knowledge is separated from the processing. This means it is relatively simple to add additional rules and adjust it to different problems. However, finding a solution is likely to be slow as the inference mechanism involves an exhaustive search.

5.4 Topological Interpretation of Geometric Constraints

The methods discussed so far are all computationally expensive. As we require a solvability test for a large number of constraint systems as differing constraints are added and removed, such methods are not suitable for our selection strategy. A more efficient method is required. In this section we briefly discuss our new interpretation of geometric constraints based on the topological type of the sets involved. This is closely related to degrees-of-freedom analysis where the topological dimensions of the involved spaces represent the degrees of freedom of the geometric elements. In this section we only introduce the general idea. A detailed discussion of this new approach and examples will be presented in Chapter 6 in combination with our topological solvability test.

From a set-theoretical point of view we can start with an unconstrained tuple of elements involved in a constraint system. The set of allowed values for the tuple is a product space of sets indicating the maximal range of values for the elements. A constraint requires that the tuple lies in a subset of the product space. For instance, consider a fixed distance constraint between two points. The unconstrained point pair can have any value in $\mathbb{R}^3 \times \mathbb{R}^3$. The constraint limits this to the subset of point pairs which are the given distance apart, i.e. one of the two points has to lie on a sphere around the other point. Hence, the set of solutions of a constraint system is the intersection of the subsets selected by the

constraints. In the case of only distance constraints between points, a point may have to lie in the intersection of multiple spheres.

For a product space generated by finite sets, this creates links between constraint satisfaction and universal algebra [56, 57]. A constraint satisfaction problem becomes a pair of relational structures, and the solutions to the problem are the structure preserving mappings between these two structures. However, for geometric constraints the elements of the product are infinite sets, which makes the problem more complicated.

However, all the sets involved in geometric constraint systems have a topological structure (one of the reasons feature spaces are required to be path-wise connected, smooth manifolds; see Section 2.2.1). The set of geometric elements of a given type are manifolds, and our regularities are described in terms of regular arrangements of features which are also elements of manifolds. The subsets selected by geometric constraints can be interpreted as (lower-dimensional) sub-manifolds of the feature space. Satisfying multiple constraints means that we intersect these sub-manifolds. The details will be discussed in Chapter 6. At the moment it is only important to notice that a constraint selects sub-manifolds of some of the manifolds for the involved geometric elements. These sub-manifolds reduce the dimension of the sets of allowed values for the elements. When intersecting these sets we can argue generically about the result of the intersection and this again gives a dimension reduction of the sets. By generic we mean a likely configuration which does not require any special conditions. For instance, in the case of distance constraints between points, we have to consider the generic intersection of two spheres. As will become clearer in Chapter 6, this intersection will be a circle, i.e. a one-dimensional sub-manifold of the three-dimensional manifold of all points in $\mathbb{E}^3$. Thus, under the assumption that the intersections are generic, we can argue solely about the dimension of the involved spaces in order to consider the solvability of a constraint system.

Finally, when all the sets are of dimension 0, we have a set of discrete solutions of the geometric constraint system. However, as our constraints only specify *relative* relations between the geometric elements it is not possible to fix the location or the orientation of the resulting geometric structure. In $\mathbb{E}^3$ this means that there should be in general six dimensions (three for location and three for orientation) left in the product space.

This type of argument does not directly produce a solution to the constraint solving problem. Instead, it investigates the structure of the constraint system, which includes its solvability properties. Due to the generic reasoning it may fail for special constraint systems. In degrees-of-freedom analysis we only consider the dimensions of the topological spaces involved in a constraint system. These dimensions are usually referred to as degrees of freedom of the geometric elements. One of the main aims of degrees-of-freedom

analysis is to create a *decomposition-recombination plan* for a constraint system [51]. An algorithm for this creates a plan to decompose the constraint system into small, solvable sub-systems. These sub-systems are usually small enough to use one of the expensive symbolic solvers. Afterwards, their solutions are then recombined by solving other small systems. Restricting the solvers to small sub-systems improves the overall efficiency of finding a solution. The main aim is to efficiently decompose the constraint systems into close-to-optimal small sub-systems. The whole system cannot be decomposed arbitrarily into small sub-systems as some constraints have to be solved simultaneously (for instance, see [32]). An overview of current decomposition-recombination planners with a clean problem formulation and measures to evaluate their efficiency is given by Hoffmann et al. [51, 52]. Our main aim for beautification is to use the approach to argue about the solvability of a constraint system and not to find an efficient decomposition, though.

There is a huge variety of methods related to the degrees-of-freedom (or graph-based) approach (e.g. [25, 38, 49, 54, 68, 69, 78, 79, 86, 102, 103, 121]). Rather than discussing all of these methods in detail we only present the basic concept. For an overview see [17]. For all of these methods, the basic idea is to argue about the degrees of freedom a geometric element has and how a geometric constraint reduces the degrees of freedom of these elements. As there are usually different ways in which a geometric constraint can be interpreted to reduce the degrees of freedom, the algorithms try to find ways to distribute these reductions between the geometric elements. A constraint system in this context can be interpreted as a (hyper-)graph with the nodes representing the geometric elements and a geometric constraint creates a (hyper-)edge between the geometric elements it involves. Degrees-of-freedom analysis methods use this graph to find a way to distribute the degrees of freedom and identify sub-graphs that describe small, solvable sub-systems. Such a sub-graph can be replaced by a simpler graph. Usually the sub-graph is replaced by a single node. However, better results can be achieved by replacing only the internal nodes of the sub-graph with a single node and keeping the frontier nodes (nodes connecting the internal nodes to the rest of the graph) [52]. This process eventually generates a hierarchical structure of sub-graphs which yields the decomposition-recombination plan. The methods differ in the ways they replace the sub-graphs, the geometric elements and constraints considered, and the ways the degrees of freedom are distributed in the graph (see [51] for a general framework for this). Many approaches rely on specific properties of two-dimensional constraints and cannot be easily generalised to three dimensions.

In Chapter 6 we present our topological solvability test. Its structure is similar to the `dense` algorithm developed by Hoffmann et al. [121], which is used to detect close-to-optimal small sub-systems in a given constraint system. However, we do not aim to decompose a constraint system, but only determine its generic solvability.

5.5 Constructing Constraints for Beautification

In this section we describe how we construct the constraints representing the approximate regularities detected in the initial model to determine the model geometry. We also use constraints to describe the relations between the geometries as required by the topology of the improved model based on the initial model. We first describe the geometric element and constraint types available to model the constraint systems. Then we present the constraints used to describe the topological structure of the model and other required relations. Finally, we show how to express our regularities in terms of geometric constraints.

5.5.1 Elements of Geometric Constraint Systems

In the following we introduce the basic elements of our constraint systems. For beautification we use constraints between the cells (see Section 2.1) of the initial model as geometric elements in the constraint system. The constraints describe the relations between the parameters required to describe the shape of the cells.

The parameters are three-dimensional vectors and scalars used to describe the base features (see Section 2.2) of the cells, and some additional parameters to completely determine the geometry of the cells such that an improved model can be constructed (see Section 4.4). The additional parameters are required, as, for instance, for a plane we also require a position which is not represented by the plane's features. The parameters describe elements of feature spaces, which also represent the type of the parameter. But not all parameters are features used for regularity detection. Table 5.1 lists the cells with the parameters used to describe them for the constraint system. Note that some positional parameters do not describe cells uniquely (positions of planes, lines, etc.). Other parameters cannot be chosen freely as there are additional dependencies (e.g. the major direction of an elliptical edge has to be orthogonal to the normal of the ellipse's plane). These additional conditions are modelled by required constraints (see Section 5.5.2).

In addition to the cells of the initial model we also use auxiliary cells as listed in Table 5.2. They are used to represent the relations of some features to the cell, e.g. the length of an edge cell becomes an auxiliary length cell which is used in a constraint specifying the distance between the two vertices of the edge. This allows us to specify relations between the edge length and other parameters without having to include a constant value. There can be a constraint setting the length value explicitly or it can be determined by constraints relating it to other length values. Auxiliary cells are also used to construct auxiliary geometry to simplify the representation of more complex regularities. For instance, for a set

Cells	Parameters	Type
Vertex	Location	Position
Straight Edge	Edge direction	Direction
	Edge position	Position
Circular Edge	Centre	Position
	Normal of circle plane	Direction
	Radius	Length
Elliptical Edge	Centre	Position
	Normal of ellipse plane	Direction
	Major direction of ellipse	Direction
	Major radius	Length
	Minor radius	Length
Polygonal Loop	Root point	Position
	Axis direction	Direction
Planar Face	Normal	Direction
	Location	Position
Spherical Face	Centre	Position
	Radius	Length
Cylindrical Face	Axis direction	Direction
	Radius	Length
	Location	Position
Conical Face	Apex	Position
	Axis direction	Direction
	Semi-angle	Angle
Toroidal Face	Centre	Position
	Direction	Direction
	Major radius	Length
	Minor radius	Length

Table 5.1: Cells Used as Geometric Elements in Constraint Systems.

of parallel directions we add constraints making each of the directions parallel to an auxiliary direction rather than adding constraints to make them pairwise parallel. In general this avoids the introduction of complex constraint types which are hard to handle by the topological solvability test. Note that while auxiliary cells are described by elements of feature spaces, they are not necessarily a cell in the sense of our boundary representation. For instance, an auxiliary cell of an edge length does not have a geometric realisation, but

Auxiliary Cells	Parameters	Type
Auxiliary line	Location	Position
	Direction	Direction
Auxiliary plane	Position	Position
	Direction	Direction
Auxiliary cylinder	Position	Position
	Direction	Direction
	Radius	Length
Auxiliary position	Position	Position
Auxiliary direction	Direction	Direction
Auxiliary angle	Angle	Angle
Auxiliary length	Length	Length

Table 5.2: Auxiliary Cells and Their Parameters.

is used to describe the geometric realisation of the boundary representation cells.

We have incidence, distance and subset constraints as listed in Table 5.3. We list the geometric description of the constraint and the equation between the parameters involved, which are either three-dimensional vectors or scalars. We assume that there is another equation for each direction vector ensuring that it is a unit vector. We have subset constraints requiring positions to lie on a surface or curve cell. These are expressed by the appropriate algebraic equations for the cell geometry. Subset constraints involving a line being a subset of some other cell are specifically required in order to express certain regularities. More general subset constraints are not needed in our system. Note that some distance constraints involve variable rather than fixed distances which are represented by an auxiliary length cell. This enables us to specify the distance value separately from the distance constraint. In the numerical approach to solvability testing, the constraints are handled as numerical equations (see Section 5.6). In the topological approach, they are primarily handled in terms of the feature spaces involved and the restrictions introduced by the constraints, as discussed in Chapter 6.

5.5.2 Required Constraints

In this section we describe the constraints which are *required* to be part of the constraint system. They either describe the relations between the cells required by the topology of the model, or necessary relations between parameters of a cell or the relation of auxiliary

Geometric Constraint	**Equation**
Parallel directions d_1, d_2	${d_1}^t d_2 = 1$
Constant angle α between two directions d_1, d_2	${d_1}^t d_2 = \cos(\alpha)$
Variable angle a between two directions d_1, d_2	${d_1}^t d_2 = \cos(a)$
Equal positions p_1, p_2	$\|p_1 - p_2\| = 0$
Constant distance l between two positions p_1, p_2	$\|p_1 - p_2\| = l$
Distance between two position p_1, p_2 is a constant multiple ν of a variable length l	$\|p_1 - p_2\| = \nu l$
Position p_0 is the average of n positions p_k	$np_0 = \sum_{k=1}^{n} p_k$
Constant value α for angle/length parameter s	$s = \alpha$
Equal angle/length parameters s_1, s_2	$s_1 = s_2$
Linear relation between lengths/angles s_k with constants α_k	$\sum_k \alpha_k s_k = 0$
Position p on cell O	$p \in O$
Line with position p_1 and direction d_1 lies on plane with position p_2 and normal d_2	${d_1}^t d_2 = 0$ ${p_1}^t d_2 = {p_2}^t d_2$
Line with position p_1 and direction d_1 lies on cylinder with position p_2, direction d_2 and radius r	${d_1}^t d_2 = 1$ $r = \|(p_1 - p_2) -$ $(p_1 - p_2)^t d_2 \; d_2\|$
Line with position p_1 and direction d_1 is axis of cell with position p_2 and direction d_2	${d_1}^t d_2 = 1$ $(p_2 - p_1)^t \times d_1 = 0$

Table 5.3: Geometric Constraints.

cells to the cells. Note that relations between auxiliary cells and the cells only have to be added to the constraint system describing the improved model in cases where the auxiliary cell is actually used by the regularities. This can easily be done at the time of regularity selection. Whenever a constraint which involves auxiliary cells is added to the constraint system, required constraints for the auxiliary cell also have to be added unless they are already part of the constraint system. This simple modification has been omitted in the description of the constraint selection algorithm (Algorithm 4.1) for clarity.

To impose the correct topology on the model, we add constraints requiring each vertex to lie on appropriate edges and faces. This does not fully specify the geometric relation between adjacent faces, but only ensures the proper intersection of faces at vertices of the model. In the case of the intersection of two adjacent planes, two distinct vertices from a common edge are sufficient to ensure that a proper straight line intersection exists. In other cases, e.g. the intersection of two cylinders, it only ensures that the intersection is not empty (the vertices have to be in it), but it does not specify the type of the intersection.

Instead of adding inequality constraints or using alternative techniques to ensure that the surfaces intersect properly for the model reconstruction, we use the regularities. As the regularity detection phase considers all possible relations between face features, and suggests multiple options for special relations, there is at least one regularity which specifies the exact relation between two adjacent faces. Thus, the regularities determine the exact relation between adjacent faces, e.g. constraining a cylinder axis to be parallel to a plane normal. The relation between adjacent surfaces is either part of a higher-level regularity such as a partial rotational symmetry of directions, or a pairwise regularity such as a special angle value between the normals of two adjacent planes. Pairwise relations which are part of a higher-level regularity may be rejected due to inconsistencies not caused by the particular pairwise relation. This means that in principle the relation between two adjacent faces is not determined exactly. But keeping all pairwise relations creates a lot of additional regularities which would make the selection process considerably more complex. Such cases may in principle also be handled by adding partial regularities. For each regularity we have certain constraints which determine the basic structure of the regularity and have to be added every time. Other constraints relate to individual faces, etc. and could easily be dropped if they create an inconsistency (e.g. a complete rotational symmetry of directions created by a prism could be reduced to a partial directional symmetry). This has not been considered further. It would also make the selection process more complicated as the reduced regularity is unlikely to have the same priority. We start with a valid initial model and try to find an improved model close to it. Hence, it is unlikely that a missing pairwise relation would break the model. Note that at least the involved vertices lie on the intersection.

If every regularity specifying a precise relation between a pair of adjacent faces is rejected due to inconsistencies, the relation is determined indirectly from other regularities. In such cases we cannot add any constraints specifying the relation between the face pair without making the constraint system unsolvable or at least over-constrained. The relation between the face pair is effectively determined by other parts of the model. But recall that appropriate vertices are always constrained to lie in the intersection of the two faces, which limits the relations between the pair. Note that it is possible to add further positions to describe the topology, especially for cases where there are no natural vertices, e.g. the intersection of a sphere with the top of a cylinder. In practice (see Chapter 7) we never observed any topological problems created by the solution of the constraint systems. Potential topological problems are not discussed in this thesis. They are handled separately by pre- and post-processing steps for the system presented here [40].

In the constraint system we treat edges as cells independent from the two surfaces they lie on. They are only used to express regularities relating to their features and not to

ensure that the surfaces intersect. When we construct the improved model all the surface intersections are recomputed, so there is no need to require that the edges are on their surfaces. Similarly we handle polygonal loops as separate cells with special relations to the model, mainly to handle the axis and the position of the loop.

To enforce *necessary* dependencies between the features, we must create various required constraints as discussed below. These dependencies are directly introduced by the definition of the features. For each elliptical edge we have to add a constraint which requires the major axis direction to be orthogonal to the normal of the ellipse's plane. For all axis features we have to create an auxiliary line and create a required constraint making it the axis of the corresponding cell. This means we can handle any regularities involving axes in terms of lines and thus can still use simple constraints, which can easily be handled in terms of the topological interpretation of constraint systems. Similarly, we require auxiliary scalars for the radii sum and difference of toroidal faces, the distance between the end-points of a straight edge, and the angle of the circle segment of a circular edge. Constraints are used to describe the relations of these auxiliary scalars to the cells. In order to describe the angle segment of a circle we also require two auxiliary lines through the circle centre and an end-point on the edge. Furthermore, the root point of a polygonal loop is constrained to be the centroid of the vertices in the loop. These required constraints are associated with their auxiliary features and are added whenever a constraint referring to one of the auxiliary features is introduced by a regularity. This is done to keep the constraint system simple. For the numerical constraint system, this especially avoids the introduction of unnecessary relations which are not relevant for beautification as all regularities related to the auxiliary cells were rejected.

5.5.3 Regularity Constraints

We describe our regularities listed in Table 2.3 by geometric constraint sets (see also Chapters 2 and 3). Any regularities which can be expressed using the constraints from Table 5.3 between cells from Tables 5.1 and 5.2 can be handled by our system.

Identity regularities indicating the approximate congruence of features are arranged in a cluster hierarchy tree, where a regularity can only be added to the constraint system if its children are also present. Furthermore, we add dependencies requiring certain regularities to be present before we can add another one, e.g. requiring a parallel direction regularity to be present before a corresponding aligned axes regularity is added. We use separate regularity hierarchies for parallel directions, equal positions, equal length, and equal angle parameters. For each cluster of approximately congruent features, we create a

corresponding auxiliary cell, and constrain the features in the set to be equal to this object. To handle hierarchies, we constrain auxiliary cells for children to be equal to the auxiliary cell for their parent.

Aligned axes and axis intersections can also be found by clustering features, where a parallel direction cluster is used to determine if axes should be aligned or intersect. Each axis is represented by an auxiliary line which is made the axis of the corresponding cell by certain constraints (see Section 5.5.2). Aligned axis regularities are marked dependent on the related parallel direction regularity. Thus, we only have to add constraints which make parallel lines equal. This is done by adding an auxiliary vertex for each cluster of aligned axes and require that this point lies on each of the axes in the cluster. For the cluster hierarchy, we create additional auxiliary lines for non-base clusters and require that these axes are equal to the axes of base clusters. Axis intersections can easily be represented by creating an auxiliary vertex for each axis intersection cluster and requiring that this vertex lies on all axes. The hierarchy is expressed by requiring that auxiliary vertices are equal.

For parallel aligned axes, we further look for regular arrangements on grids, lines and cylinders. This is expressed with the help of additional auxiliary cells. For a symmetrical arrangement of axes around a cylinder, e.g. bolt holes arranged in a circle, we create an auxiliary cylinder and require that the axes lie on that cylinder. For each of the symmetrically arranged axis locations on the cylinder we create an auxiliary plane through the cylinder axis with a normal constrained to be orthogonal to the cylinder axis. The angles between these planes are set to an appropriate integer multiple of $2\pi/n$. To enforce the symmetrical arrangement, the positions of the axes are constrained to lie on one of these planes. For parallel axes arranged equi-spaced along grids we generate planes intersecting at appropriate distances and require the axes to lie in the two planes at each intersection of the grid. The planes are constrained to be parallel and orthogonal to give the grid structure. The distances between the planes are specified by distances between vertices which are constrained to lie on an auxiliary plane and on one of two auxiliary lines. Axes arranged equi-spaced along a line are handled in a similar way.

For equi-spaced positions arranged on a line, or a grid of positions arranged symmetrically on a circle, we create an auxiliary cell structure similar to those used for axis arrangements. We have to add an additional plane to account for the fact that we have positions rather than lines arranged in this structure.

We also consider global symmetries of points. These can in principle be represented by requiring that distances between the points that are mapped onto each other by the permutations are the same. As the symmetry detection algorithm was implemented separately, this is not further considered here.

We distinguish planar and conical cases of symmetrically arranged directions. In the first case we have a set of directions orthogonal to a direction d_0, e.g. plane normals for the sides of a prism. The angles between the directions are integer multiples of π/n for $n \in \mathbb{N}$. In the second case the angle to the direction d_0 has some other fixed value and the angles between the directions projected on the plane defined by d_0 are integer multiples of $2\pi/n$, e.g. plane normals in a pyramid. To create the constraints for the planar case we create two orthogonal auxiliary directions d_0, d_1. For each direction in the set we add a constraint requiring it to be orthogonal to d_0 and with angle to d_1 being a suitable integer multiple of π/n. In the conical case we have a list of possible special values for the angle between the directions and d_0. For each of the special values we create a regularity specifying the angles between the directions and d_0 and d_1. Note that an orthogonal system is a special case of the conical case. Also note that we do not have regularities for global symmetries of directions (see Section 2.4).

To express regularities for similar polygons, we create conditions that require the angle between the directions of the straight lines to be equal, and the edge lengths (as distances between the vertices) to be related by a constant ratio. We do not use auxiliary polygon cells (or similar) to express the hierarchy, but use a specific polygon from each set of the children clusters and require these to be similar.

Also, we have cluster hierarchies of equal positions when projected onto special planes (2D partially equal) and lines (1D partially equal). The special plane and lines are derived from major directions in the model such as main axes and orthogonal systems. Positions which are equal when projected on a plane lie on the same line.

We also have lists of special values for angles between individual directions. These relate to direction pairs which are not part of a higher-level regularity like a partial rotational symmetry of directions. Note that distances between vertices are features of the edges connecting them and other distances are not considered as there would be too many. Finally, there are regularities specifying special ratios between pairs of angle or length features and lists of special values for these features. For all these regularities appropriate constraint sets with dependencies can be created using the constraints on scalars.

5.6 Numerical Solvability Test and Solver

In this section we discuss our numerical approach to the solvability problem. In Section 5.2.1 we discussed in general how to use non-linear, numerical optimisation to find a solution to a geometric constraint problem. In the following, we first assume that we

have a numerical optimisation method and discuss how to use it in the selection algorithm (Algorithm 4.1). Then we briefly discuss quasi-Newton based methods for optimisation and how to handle numerical redundancies.

As noted earlier, the basic idea is to minimise an error function $t_F : \mathbb{R}^n \to \mathbb{R}$ for a non-linear equation system $F(x) = 0$, $F : \mathbb{R}^n \to \mathbb{R}^m$. We use quasi-Newton methods, which are particularly suitable for a least-squares error function $t_F(x) = F(x)^t F(x)$ as they are based on a quadratic model. Furthermore, in general the quasi-Newton method and in particular the BFGS method are stable and efficient optimisation methods [122, 144]. They have previously been used successfully for geometric constraint problems [42]. For other numerical approaches to constraints see, for instance, Mullineux [97].

For the selection strategy in Algorithm 4.1 from Section 4.1 we require two methods for the solvability test: `init_solvability`, which initialises the solvability test appropriately, and `solvable`, which indicates whether the constraint system expanded by an additional constraint remains solvable. For our numerical solvability test, `init_solvability` creates appropriate non-linear equations (see Table 5.3) for each regularity. The parameters for all cells from the model and all auxiliary cells are represented by a large vector p over $\mathbb{R}$. We maintain appropriate references from the elements of p to the cells they describe. The vector p is initialised by the values taken from the initial model. For the optimisation method we require a vector p' which contains the values for the parameters involved in the current constraint system. p' only contains values for cells which are involved by at least one selected constraint. When adding new constraints which involve new, not yet considered cells, p' is expanded appropriately using p. Suitable references between the elements of p and p' are maintained. To avoid numerical problems created by unused values in p', it only contains the values used in the constraint system. The complete data structure `G` containing the equations and the vectors p and p' with the references is returned by `init_solvability`.

Also note that we have to add an additional numerical constraint to the system for each direction vector to normalise it. This can easily be achieved in the data structure `G` at the time we create the vector p'. It avoids having to create more complicated numerical equations which handle unnormalised direction vectors. Such more complicated equations are more likely to cause numerical instabilities and slow down the evaluation of the target function t_F.

The method `solvable` uses a quasi-Newton based optimisation method to find a solution to a selected constraint system `S`, expanded by the constraints `r` using the data structure `G`. If the optimisation converges (determined by the optimisation method as described below) to a vector x^* such that $\|F(x^*)\|_\infty$ (the maximum error of the constraints) is smaller than

some tolerance t, `solvable` returns `true` indicating that the expanded system is solvable. In this case the vector p' in `G` is also permanently expanded to include any parameters for cells which were not previously considered in the constraint system. Otherwise, the temporary expansion of this vector required to run the optimisation method is removed. With this method we select a numerically consistent constraint system. Only inconsistencies which cause an error larger than the tolerance t are detected. But this is only important if t is large compared to the tolerance of the solid modelling kernel. Otherwise, the error caused by the inconsistency is too small to cause problems. For the optimisation method various parameters which determine tolerances, step sizes, condition safeguards, etc. are required. Given a set of these parameters that ensures convergence of the optimisation method for a given beautification problem, no consistent regularities are removed.

We also use the same optimisation method to find the solution of the final system. This is the solution found for the last successful call to `solvable` if we numerically test for solvability. For the topological solvability test (see Chapter 6), we have to run the numerical optimisation separately. In the following we discuss further the optimisation methods, and a method to handle numerical redundancies to avoid numerical instabilities.

5.6.1 Quasi-Newton Optimisation Methods

We only give a brief overview of the quasi-Newton methods we use. For a detailed discussion of the methods see [13, 90, 122]. Standard approaches to quasi-Newton methods have been used, but they were chosen in particular to handle the numerical problems of our constraint systems. As noted earlier, our general iteration formula is $x_{l+1} = x_l - \sigma_l d_l$ with $x_l \in \mathbb{R}^n$, $\sigma_l \in \mathbb{R}_+$ and $d_l \in \mathbb{R}^n$ for $l = 1, 2, \ldots$. For each iteration step we have to find a direction d_l and then find a step size σ_l in this direction which sufficiently reduces t_F such that the iteration converges to a minimum. For quasi-Newton methods d_l should point approximately in the direction of the gradient of t_F at x_l. We have a choice for the linear search method to determine σ_l and for the approximation method for the Hessian matrix of the second partial derivatives used to find d_l.

For the line-search method we considered using the Goldstein-Armijo and PWS (Powell-Wolfe-Stepsize) methods [122]. While both perform well, PWS is more stable and more suitable for the BFGS (Broyden-Fletcher-Goldfarb-Shanno) quasi-Newton update. In particular PWS with BFGS guarantees a sufficient rate of descent to ensure convergence for non-convex target functions [122].

To find the direction d_l we have to solve the equation system

$$H_l d_l = \nabla t_F(x_l) \tag{5.3}$$

during each iteration, where H_l is an approximation of the Hessian matrix $\nabla^2 t_F$ of t_F. For the Hessian the BFGS update is a widely used and suitable method. Instead of the simple BFGS iteration formula we use a formula based on the Cholesky decomposition of the Hessian matrix related to the equation system (5.3) and a condition guard for H_l initiating restarts of the iteration [122]. The BFGS method with the Cholesky decomposition of H_l allows for a simple approximation of the condition of H_l (as H_l is positive definite and symmetrical, the condition is the quotient of the largest λ_{max} and the smallest eigenvalue λ_{min}). If the condition of H_l is too large (e.g. larger than $1/\sqrt{\varepsilon}$ where ε is the machine precision), we cannot determine a suitable direction of descent from Equation (5.3) as the numerical error of the solution of the linear system too large (caused by a scale problem between the small and the large eigenvalue). For similar alternative stabilisations see [92, 140]. We also tried using the SR1 iteration formula [122] with a condition guard, but the convergence rates and stability were in general not as good as for BFGS.

Further improvements to numerical stability, especially for cases involving inconsistent constraints, were achieved by using a damped version of the BFGS method [83]. Using a hybrid method switching between BFGS (or the damped BFGS) and a Gauss-Newton step improved the convergence rates and still performed reasonably well with respect to numerical stability [90].

When using the optimisation method for the *solvability* test, a numerically stable method like the damped BFGS method is preferable, as the equation systems are more likely to cause numerical problems. For the optimisation method it is not a problem that a solution does not exist as it simply finds a minimum larger than 0. But the minimum may still be close to 0, especially if we have a lot of equations, and this makes it hard to decide if the system is actually solvable or not. Furthermore, if the constraint system is solvable we have a global minimum and with sufficiently small tolerances it is possible to detect when the optimisation converged. If the system is not solvable, the convergence test has to be based on a tolerance on the step-size rather than the function value. However, as a result the method may get trapped in a local minimum without knowing if there is a global minimum of 0 or not. Note that for practical purposes we base the convergence test on a limit for the step-size, and the function value or the error of the involved equations.

Using a hybrid method with damped BFGS still performs well. When trying to *solve* a consistent constraint system the BFGS method using Cholesky decomposition in combination with a condition guard is faster and sufficiently stable. The performance can be improved further by using it with the hybrid method. Note that the optimisation method is the most crucial time factor for beautification (see Chapter 7).

During the solvability test, we also check for redundant constraints which could make the

system numerically unstable. A redundant constraint is one which can be added to the constraint system without changing the set of solutions. The problems are similar to those caused by inconsistencies in the equation system. This time, however, they are caused by the dependencies between the equations which primarily created ill-conditioned matrices. To identify numerical redundancies we use the method described in [85]. Any selected but redundant constraint equation is marked as such and not used for the target function t_F. When a new constraint is already satisfied by the current solution the constraint may be redundant. In this case we disturb the constant values involved in the constraint and try to solve the system with the modified values. If the system remains solvable, the constraint is not redundant. Otherwise, the constraint is redundant and is not added to the constraint system, but remains active. We first check for redundancy and then try to solve the system with the original constants adding all new, non-redundant constraints. But note that this redundancy test is only used for the numerical solvability test, and not for solving the complete selected constraint system when we employ the topological solvability test. Redundancies are determined (generically) by the topological approach without solving any equation system.

5.7 Summary

A central element of the selection strategy is the solvability test for constraint systems. We have discussed various direct approaches to computing solutions of constraint systems. A common problem of such approaches is that computing a solution is an expensive operation. In order to be able to solve large constraint systems we have to decompose them into smaller systems which can be solved by the direct solvers. A degrees-of-freedom based approach arguing about the topological structure of the involved space can be employed. In particular this approach can be used to determine generic solvability of constraint systems, as will be discussed in more detail in Chapter 6.

We have further presented the basic elements of the constraint systems used for beautification, and how to represent the regularities and the improved model by geometric constraints. Moreover, a numerical method for the solvability and solver problem has been given. Using numerical optimisation methods, a constraint system can be solved and its solvability can be deduced from the existence of a solution. While the numerical solvability test is slow, it also computes a solution to the constraint system. In our final approach to beautification, it is used only to solve the selected constraint system, which cannot be done directly by the topological approach.

Chapter 6

Topological Solvability Test for Geometric Constraint Systems

In Chapter 5 we have given an overview of techniques available to determine the solvability and the solution of geometric constraint systems. Given the large number of constraints detected by our methods, we require an efficient solvability test. Our topological interpretation of geometric constraint systems introduced in Section 5.4 provides such a test as it can determine (generic) solvability without actually computing a solution. In this chapter we present details of a method to decide if a given constraint system is solvable (i.e. has at least one solution) based on this approach. While it is desirable to ensure that we have a unique solution or at most a discrete set of solutions, we only check whether at least one solution exists. In particular we accept cases where there are infinitely many solutions. As we have an initial model, we can seek a solution close to it. Furthermore, the large number of regularities we detect makes under-constrained systems very unlikely.

By analysing the constraint (hyper-)graph (see Section 5.4) we can determine certain generic properties of the constraint system. We introduce our idea by using distances between three-dimensional points as an example in Section 6.1. To verify the solvability of a constraint system we consecutively add constraints to the graph (see Section 6.2) and check whether the system remains solvable (see Section 6.3). This new approach was developed by the author in the context of degrees-of-freedom analysis, and was first presented in [73]. In our approach degrees of freedom are interpreted in terms of dimensions of manifolds representing the domains of the geometric elements, and sub-manifolds representing the restrictions to the allowed values of these elements induced by the constraints. This leads directly to an efficient algorithm for adding geometric constraints as directed edges to a constraint graph indicating the dependencies and restrictions created by the constraints. By analysing the dependencies in the constraint graph we quickly determine whether a constraint system expanded by an additional constraint remains solvable in a generic sense. The mathematical background, and the scope and limitations of this approach are discussed in Section 6.4.

6.1 Distance Constraints Between Points

To illustrate the basic concepts of our methods for determining the solvability of a constraint system, we consider constraint systems consisting only of constant distance constraints between points in three-dimensional Euclidean space $\mathbb{E}^3$.

An unconstrained point in $\mathbb{E}^3$ can be at any location in space, i.e. its parameter domain is $\mathbb{R}^3$. A distance constraint between two points v_1, v_2 limits the allowed values the two points can have at the same time. One way of enforcing this is by allowing v_1 to be parameterised by an arbitrary value in $\mathbb{R}^3$ and requiring that v_2 is on a sphere of fixed radius with centre v_1. This means that v_2 can be described by a parameter on the unit sphere $\mathbb{S}^2$ in combination with the position of v_1. Thus, for v_2 we select a (lower-dimensional) sub-manifold of the parameter manifold $\mathbb{R}^3$ and this sub-manifold is homeomorphic to $\mathbb{S}^2$. We say this sub-manifold has the topological type $\mathbb{S}^2$ and often we will identify this type space with the actual sub-manifold in the parameter manifold for simplicity (we can always specify a homeomorphism between the type space and the actual sub-manifold). Obviously the role of v_1 and v_2 can be exchanged. Hence, we can interpret a distance constraint as a reduction of the parameter space $\mathbb{R}^3$ to $\mathbb{S}^2$ for one of the two points involved.

For instance, consider three vertices v_1, v_2, v_3 in $\mathbb{E}^3$ and three distance constraints between them. Each vertex can be described by a parameter in $\mathbb{R}^3$. Thus, we can assign any element of $\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$ to the unconstrained tuple (v_1, v_2, v_3). The three unconstrained vertices are shown in Figure 6.1(a), where each vertex is labelled with its name and its parameter domain.

A distance constraint $dist_1$ between the two vertices v_1, v_2 can be interpreted by either putting v_2 on a sphere around v_1 or v_1 on a sphere around v_2. Taking the first case, we select a sub-manifold of the topological type $\mathbb{S}^2$ in $\mathbb{R}^3$ for v_2. Thus, the allowed values for the tuple (v_1, v_2, v_3) can be parameterised using the product space $\mathbb{R}^3 \times \mathbb{S}^2 \times \mathbb{R}^3$. To indicate this we add a directed edge from v_1 to v_2 in the constraint graph in Figure 6.1(b) and change the domain label of v_2 to $\mathbb{S}^2$. We get an analogous result in the second case.

Similarly, a distance constraint $dist_2$ between v_1 and v_3 can be interpreted as parameterising the allowed values of (v_1, v_2, v_3) over the space $\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{S}^2$ (or, analogously, over $\mathbb{S}^2 \times \mathbb{R}^3 \times \mathbb{R}^3$ in the other case). If we impose both constraints at the same time, we get the parameter space $\mathbb{R}^3 \times \mathbb{S}^2 \times \mathbb{S}^2$ as the selection of subsets does not interfere with each other in the product space. This is illustrated in the graph in Figure 6.1(c) by adding another directed edge from v_1 to v_3 and changing the domain of v_3 to $\mathbb{S}^2$.

Finally, we add a constraint $dist_3$ between v_2 and v_3. Again we have a choice to limit v_2 or

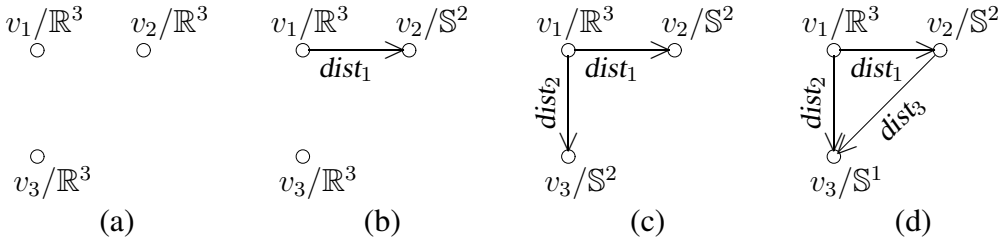


Figure 6.1: Distance Constraint Graph Between Three Points.

v_3 to $\mathbb{S}^2$. However, both points are already reduced to $\mathbb{S}^2$. If we choose to put v_3 on another sphere, v_3 has to be on the intersection of two spheres. In the generic case, two spheres intersect in a circle, and hence v_3 is now described by a parameter on the unit circle $\mathbb{S}^1$ (see Figure 6.1(d)). We cannot constrain the points any further with distance constraints and in this case we actually have a unique solution modulo rotations and translations in $\mathbb{E}^3$.

More generally, the two spheres may also intersect in a point, not intersect at all, or be equal (i.e. the intersection is a sphere). In the first case the distances between the points must be specifically chosen such that the points are collinear. In the second case the distances must satisfy $d(v_1, v_2) > d(v_1, v_3) + d(v_2, v_3)$. In the third case we have a coincidence constraint instead of a distance constraint. In all of these cases additional conditions are present which cannot be determined directly from the constraint graph and certain equation systems would have to be solved in order to detect them. Due to the special conditions they are unlikely to occur in a constraint system and we expect that in most cases the spheres intersect in a circle. Hence, we call this case generic.

If one of the special conditions apply, the generic analysis may fail to determine the solvability properly. In case the intersection is a point, the analysis would assume that the sub-manifold of allowed parameter values for a point has more dimensions than available and we may get an over-constrained system. In case the intersection is a sphere, the analysis would assume that the sub-manifold of allowed parameter values for a point has less dimensions than available and we may get an under-constrained system. In case the two spheres do not intersect at all, the analysis would actually select proper constraints, but the constraint system is not solvable due to the involved constants. For beautification, only the over-constrained case may cause serious problems, as an over-constrained system is likely not to be solvable unless the constraints are consistent by chance. This may also cause numerical instabilities for the optimisation solver due to a more complicated objective function. In the under-constrained case, we can still look for a solution close to the initial model. The case of inconsistent constants is unlikely to occur at all as the

constraints are based on a valid initial model, and usually only large changes to this model would yield inconsistent constants.

Choosing any other generic options of enforcing the distance constraints of the constraint system in Figure 6.1 will create product spaces involving $\mathbb{S}^1$, $\mathbb{S}^2$, $\mathbb{R}^3$ with the total sum of dimensions being six in the generic case. For instance, if we put v_3 instead of v_1 on a sphere (choosing the parameter space $\mathbb{S}^2 \times \mathbb{S}^2 \times \mathbb{R}^3$ in Figure 6.1(c)), the final intersection would be $\mathbb{S}^2 \times \mathbb{S}^2 \times \mathbb{S}^2$ representing a circular dependency between the points.

We add distance constraints to the graph by choosing one of the two points constrained to $\mathbb{S}^2$ and update the parameter domain by intersecting it with $\mathbb{S}^2$ under the assumption that we have the generic case. We intersect $\mathbb{S}^2$ with either $\mathbb{R}^3$, $\mathbb{S}^2$ or $\mathbb{S}^1$. In the generic case we assume that the intersection with $\mathbb{R}^3$ gives $\mathbb{S}^2$, with $\mathbb{S}^2$ gives $\mathbb{S}^1$ and with $\mathbb{S}^1$ gives $\mathbb{R}^0$. The intersection with $\mathbb{S}^1$ can actually either lead to a circle, an empty set or two points. In the generic case we get two points, and we choose one of them, i.e. we get $\mathbb{R}^0$. Clearly, the two points indicate that we have two discrete solutions. To distinguish between them we need another constraint, e.g. an inequality constraint. As our improved model should be close to the initial model, the assumption that we get a single point can be justified if we take the solution which is closer to the initial model.

In the graph context we can refer to a constraint as an undirected edge between the nodes in the graph representing the geometric elements. A constraint can usually limit the elements it constrains to sub-manifolds in various ways. Which of these options is selected can be indicated by a directed edge pointing towards the nodes which are restricted (in more complex cases involving hyper-edges labels for each end-point of the edge may have to be used instead of a simple direction). We refer to adding a directed edge to a graph for the solvability test in the above way as *distributing* the constraint in the graph. Thus, the constraint can be identified with an undirected edge in the graph, which is usually not drawn. Distributing the constraint adds a directed edge to the graph, which represents a particular way of enforcing the restrictions of the constraint on the geometric elements. When we choose another option for an already distributed constraint (see below), we refer to this as *redistributing* the constraint, i.e. changing the direction of the edge.

The dimensions of the parameter domains of the points represent the degrees of freedom of these objects (three for three-dimensional points). The reduction of these domains to (lower-dimensional) sub-manifolds indicates the number of degrees of freedom removed by a constraint (one for distance constraints). The way we distribute the constraints and the different options for their distribution is similar to degrees-of-freedom analysis. We interpret it in terms of the topology of the involved parameter spaces and their dimensions.

Obviously it is not always possible to distribute a new constraint in a given constraint

system. If both points involved in a distance constraint are already $\mathbb{R}^0$, we cannot generically intersect either of them with $\mathbb{S}^2$. However, as for each distance constraint there are two ways in which it can be added to the constraint graph, it may be possible to choose a different distribution of constraints along some of the edges in the graph such that we can add the constraint. Thus, we have to do a graph search starting at the edge of the new constraint. We have the option of doing a depth-first or a breadth-first search. In the depth-first approach we follow one particular path in the graph backwards along the directed edges to its end before we consider any other paths. Once we find an edge which can be changed (the constraint can be redistributed) we backtrack from that edge along the path and redistribute all other constraints referred to by the edges in the path accordingly. This redistribution may now allow us to distribute the new constraint. In this approach the paths may become quite long before we can redistribute a constraint and the search grows exponentially as we check all possible paths.

The alternative, a breadth-first search backwards along the directed edges, is more efficient as it finds the first edge which can be changed closest to the new edge. In addition, we do not have to consider all paths starting with the new constraint, but only find the shortest paths to the edges referring to constraints which can be redistributed. Whenever a constraint can be redistributed, we redistribute the constraints along the whole path to the edge of the original constraint. We then have to repeat the breadth-first search until we can distribute the new constraint, or all redistribution options have been exhausted.

Finding a redistribution path is similar to the `distribute` method used in the `dense` algorithm [121]. In that method the constraint graph is converted to a bipartite graph between nodes representing the constraints and other nodes representing the geometric objects. The edges in this graph connect the constraint nodes with the object nodes that they constrain. The graph is interpreted as a flow network from a source to a target. The source is connected to all constraint nodes and the target is connected to all objects. The capacity of an edge from the source to the constraint node indicates the degrees of freedom removed by the constraint node. The capacity of an edge from the object node to the target indicates the degrees of freedom of the object. The capacities of edges between constraints and objects are infinite. The `distribute` method used by `dense` tries to distribute a new constraint in such a flow network by finding a flow augmentation path to distribute the newly added flow from a constraint. This is done in a similar way to searching for the redistribution paths above. The different options we have for each edge describe the different ways the flow can be distributed through the network. Instead of changing the distribution of the flow, we change the direction of edges in our approach, but the principle behind it is the same. However, by considering the topological types of the spaces and their topological dimensions, we link the graph more closely to the constraint

system without having to consider a flow network. This also allows us to give more details about the scope and limitations of the technique and may eventually lead to a method to handle more general constraint systems reliably (see Section 6.4). Also note that we use our approach to determine the solvability of the constraint system without explicitly identifying solvable sub-systems of the constraint system. The `dense` algorithm is used to find dense sub-graphs which represent small solvable sub-systems for decomposition-recombination planners (see Section 5.4).

However, a method to distribute a constraint does not reveal if the resulting constraint system is solvable. We say that a system is solvable if there is at least one solution under the assumption that the intersections are generic. If the constraint cannot be distributed, the system is clearly unsolvable. The opposite is not true.

Consider the following simple example. If we want to determine point v_3 in the constraint system in Figure 6.1, we can do this by setting an arbitrary location for v_1, choose v_2 on a sphere around v_1 and then determine v_3 by choosing a parameter in $\mathbb{S}^1$, using the locations of v_1 and v_2. This specifies all three points up to location and orientation in $\mathbb{E}^3$. As distance constraints cannot determine the absolute location or orientation of the point set, we cannot determine the points any further. Hence, in the general case there must always be at least six degrees of freedom left. But note that for a zero-dimensional point set, i.e. one point, we require only at least three degrees of freedom, and for a one-dimensional point set, i.e. (two distinct) points on a line, we require only at least five degrees of freedom.

The directions of the edges in a constraint graph define the dependencies between the nodes. Given an arbitrary node n in the constraint graph and an edge e directed towards this node, we can follow edges backwards to determine the sub-graph $S(n, e)$ of all nodes on which n depends due to e. To detect $S(n, e)$ we mark n as visited and follow the edge e backwards marking its starting point as visited. From there we continue following all (directed) edges backwards which lead to unvisited nodes employing a greedy algorithm. We stop when no further unvisited nodes can be reached by following the edges backwards. n and all edges between n and the detected sub-graph are added to $S(n, e)$, which we call the *dependency sub-graph* of n due to e. In this sub-graph we change the parameter space of n so that only the edges in $S(n, e)$ are considered. The resulting $S(n, e)$ represents a solvable sub-graph if the sum of the remaining degrees of freedom of the nodes in $S(n, e)$ is at least six, five or three depending on the dimensionality of the points involved.

We have to change the parameter space of n in $S(n, e)$ to account for the dependencies of n over other edges not in $S(n, e)$. Each dependency sub-graph represents a restriction of n. We assume that the intersection of these restrictions is generic and can be done.

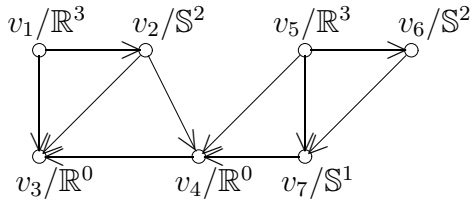


Figure 6.2: Example Constraint Graph for Dependency Sub-Graphs for Point $\mathbf{v_4}$.

This is checked whenever we distribute a constraint and compute the dimension of the resulting parameter space (a constraint removes a generic amount of degrees of freedom from a node and we must not remove more degrees of freedom from a node than it had originally). At a node we bring the geometric structures of the different dependency sub-graphs together to form a single structure. This can be done if the structures described by the sub-graphs have sufficient degrees of freedom left.

For example, consider the constraint graph in Figure 6.2. Node v_4 has three dependency sub-graphs. The first graph consists of the nodes v_1, v_2, v_4 with v_4 relabelled to $\mathbb{S}^2$ (only one constraint is on v_4 in the sub-graph). This sub-graph has 7 degrees of freedom left indicating that it is solvable. Without relabelling v_4 it would only have 5 degrees of freedom which are not sufficient. The second sub-graph consists of v_4 and v_5 with v_4 relabelled to $\mathbb{S}^2$. Without relabelling v_4 to $\mathbb{S}^2$ in the second sub-graph the graph would not be solvable (it would have 3 degrees of freedom, whereas we have two points on a line which requires 5 degrees of freedom). The third sub-graph consists of v_4, v_7, v_5, v_6 with v_4 relabelled to $\mathbb{S}^1$ and has 7 degrees of freedom.

When we have successfully distributed a new constraint as a directed edge e in the graph, we must test if the new graph is solvable. If distribution failed, we already know that the constraint system is not solvable. To test for solvability we only have to consider the changes made during distribution. Let n be the node the constraint has been distributed to, i.e. the directed edge e points towards n. We claim that the graph remains (generically) solvable if the dependency sub-graph $S(n, e)$ is solvable, i.e. it has sufficient degrees of freedom left. Originally the number of degrees of freedom of a node is the dimension of its parameter space. By distributing constraints we select sub-manifolds of the parameter space and intersect those sub-manifolds. This reduces the dimension of the sub-manifold of allowed parameter values for a node. The dimension of this sub-manifold is the amount of degrees of freedom left after constraint distribution. In general in the sub-graph $S(n, e)$ the sum of degrees of freedom has to be at least six (three rotational and three positional degrees of freedom in $\mathbb{E}^3$) in order for it to represent a solvable system. However, note that there are special cases of lower dimensional subsets embedded in $\mathbb{E}^3$ which have to

be handled separately, e.g. two points on a line with only five degrees of freedom.

First consider the case where no redistributions are required to distribute the constraint. In this case only node n was changed. The dependency sub-graphs which do not contain e did not change. So we only have to check $S(n, e)$. Under the assumption that all intersections are generic we only have to check if there are sufficient degrees of freedom in $S(n, e)$. In the case of redistributions we can consider each redistribution separately. Assume we have an edge between two points n_1 and n_2 which initially constrains n_2. When we redistribute the constraint this edge relates to, then n_1 is constrained by n_2. The degrees of freedom are moved from n_1 to n_2 and this is indicated in a change of the dependency sub-graphs of the two nodes. Initially n_2 had a dependency sub-graph over n_1 with sufficient degrees of freedom. This sub-graph is replaced by a new one for n_1 which includes n_2. Due to moving the degrees of freedom, this new sub-graph also has sufficient degrees of freedom.

6.2 Distributing Constraints

In this section we present the distribution algorithm for all our constraint types in detail. We describe the geometric objects (the cells from the model and the auxiliary cells) in terms of positions, directions, lengths, and angles as parameters. Constraints limit the allowed combinations of the values for these features. The topological type of the domain for positions is $\mathbb{R}^3$, $\mathbb{R}^2$ or $\mathbb{R}^1$ depending whether the position describes the location of a point, a line or a plane. For directions the domain is $\mathbb{S}^2$, for lengths $\mathbb{R}_+$, and for angles $\mathbb{S}^1$. Any of the constraints we consider can be interpreted as selecting a lower-dimensional subset of these domains. There is usually more than one way in which to select the subsets. When a geometric object is constrained by more than one subset, its allowed values lie in the intersection of the two subsets. We assume that all the intersections are generic as is usually done in degrees-of-freedom analysis.

In Table 6.1 we list the different options for distribution of constraints. Note that some of the distributions do not have an explicit geometric meaning, but as the distribution relates to local rather than global properties of the manifolds involved (see Section 6.4 for details), this can be ignored. For instance, for two parallel directions the cases where one of the two directions is equal to the other and thus fully determined directly relates to putting one of the two directions in $\mathbb{S}^0$. The case where d_1 and d_2 both have one degree of freedom left is indicated as d_1 and d_2 lying in $\mathbb{S}^1$. This is sufficient locally as d_1 and d_2 can be moved a short distance along a circle. However, with this labelling the global structure appears to be the torus $\mathbb{S}^1 \times \mathbb{S}^1$, which is in fact not the case. For this type of analysis only

Geometric Constraint	**Distribution**
Parallel directions d_1, d_2	d_1 or d_2 in $\mathbb{S}^0$ or d_1 and d_2 in $\mathbb{S}^1$
Constant angle α between two directions d_1, d_2	d_1 or d_2 in $\mathbb{S}^1$
Variable angle a between two directions d_1, d_2	a in $\mathbb{R}^0$ or d_1 or d_2 in $\mathbb{S}^1$
Equal positions p_1, p_2	p_1 or p_2 in $\mathbb{R}^0$ or p_1 in $\mathbb{R}^1$ and p_2 in $\mathbb{R}^2$ or p_1 in $\mathbb{R}^2$ and p_2 in $\mathbb{R}^1$
Constant distance l between two positions p_1, p_2	p_1 or p_2 in $\mathbb{S}^2$
Distance between two position p_1, p_2 is a constant multiple ν of a variable length l	l in $\mathbb{R}^0$ or p_1 or p_2 in $\mathbb{S}^2$
Position p_0 is the average of n positions p_k	p_0 or one p_k in $\mathbb{R}^0$ or p_0 in $\mathbb{R}^1$ and one p_k in $\mathbb{R}^2$ or p_0 in $\mathbb{R}^2$ and one p_k in $\mathbb{R}^1$ or p_{k_1} in $\mathbb{R}^1$ and p_{k_2} in $\mathbb{R}^2$ or p_{k_1} in $\mathbb{R}^2$ and one p_{k_2} in $\mathbb{R}^1$
Constant value α for angle/length parameter s	s in $\mathbb{R}^0$
Equal angle/length parameters s_1, s_2	s_1 or s_2 in $\mathbb{R}^0$
Linear relation between lengths/angles s_k with constants α_k	One s_k in $\mathbb{R}^0$

Table 6.1: Distribution of Geometric Constraints.

concerned with the involved dimensions it can be ignored, though. We also distinguish between constraints with constant and variable parameters, as only variable elements can be used for the distribution. For instance, a *variable* distance constraint between two positions with no degrees of freedom left can also be distributed to the distance parameter, i.e. the fixed positions determine the value of the distance.

Constraints requiring positions to lie on a surface or curve are omitted from Table 6.1. Surfaces and curves are usually described by a combination of positional, directional, angular and length features. In the constraint graph we represent them by a single node. Constraints that only relate to one of the features describing the surface can only be distributed using this particular feature, e.g. making two planes parallel only constrains their normals. But putting a vertex on a surface or curve means that any of the involved features can be used, e.g. putting a point on a plane can restrict its position as well as its normal.

Also note that positional features of surfaces are not always three-dimensional positions.

For planes we only have a one-dimensional position space (its distance from the origin), for cylinders and straight lines we have a two-dimensional position space (the position of the axis). All other objects we consider have three-dimensional position spaces.

The issues relating to faces in the constraint graph are complicated and some more details are discussed in Section 6.4. Here we only present as much as is required to understand how these objects are handled by the distribution algorithm. Whenever we put a position on a surface or a curve we reduce one of the parameter spaces of the surface or curve or the parameter space of the position by one or two dimensions respectively. We can choose any of the parameter spaces involved for the distribution. The types of the parameter spaces may vary and the intersections between them can be complicated and involve other types. We assume the generic case such that the intersections are always generated by reducing the degrees of freedom by one. As the distribution algorithm below only relies on the reduction of dimensions of parameter spaces a detailed discussion is omitted.

The constraints which require a line to lie on a plane or a cylinder, and those which make a line coincident with an axis feature of a surface (central axis of cylinder, cone, etc.; see Table 2.2) relate to issues similar to those for position-on-surface constraints. Constraining a line to lie in a plane reduces the total amount of degrees of freedom of the geometric elements involved by two. If we do not restrict the line, then we have the set of all planes which contain this line, i.e. the plane has one degree of freedom left for the direction of the normal or its position. If we restrict the line to lie on a plane, then the line has two degrees of freedom left, one for the direction and one for the position. Putting a line on a cylinder reduces the degrees of freedom of the involved geometric elements by three. The axis direction of the cylinder has to be parallel to the direction of the line and we can restrict either of the two directions to $\mathbb{S}^0$ (it is completely determined by the other direction). In addition, the position of the line or the position of the cylinder has to lie on a circle, i.e. it is parameterised by $\mathbb{S}^1$. Forcing a line to be an axis of a surface reduces the total number of degrees of freedom by four. We have two parallel directions where either of the two directions can be restricted and the position of the cell is fixed to lie on the line (for the torus and cone we still have one positional degree of freedom left, for cylinder and axes of planar polygonal loops we have no positional degree of freedom left). To distribute these types of constraints we select appropriate sub-manifolds of any of the involved parameter spaces such that the overall reduction sums up to the amount required by the constraint type as mentioned above.

The geometric elements in the constraint graph are considered to be described by parameter product spaces. But this is not always the case. For instance, consider the space of all lines in $\mathbb{E}^3$. A line can be described a position p and a direction d. We can form the

moment vector $l = p \times d$, which is independent of the position p on the line as it remains the same when p is replaced by a point $p' = p + \lambda d$, $\lambda \in \mathbb{R}$. Then the pair (d, l) represents the *normalised Plücker coordinates* of the line. The pair (d, l) fulfils the normalisation equation $\|d\| = 1$ and the *Plücker relation* $d^t l = 0$. Conversely, any six-tuple which fulfils these two conditions represents a line in $\mathbb{E}^3$. Hence, the set of all lines in $\mathbb{E}^3$ forms a four-dimensional manifold [106]. For our purposes we describe a line by a positional and a directional parameter, where the positional and the directional parameter each has two degrees of freedom. We consider the two parameters independently and hence they are interpreted as a product manifold $\mathbb{R}^2 \times \mathbb{S}^2$. This product manifold is locally homeomorphic to the manifold of all lines in $\mathbb{E}^3$ (they are both four-dimensional), but it is not the same manifold. We discuss some of the underlying structures and problems related to this in Section 6.4. For degrees-of-freedom analysis we are mainly concerned with topological dimensions, and so the local homeomorphism property is sufficient. In particular, as we have an existing model and seek a small, local change, we do not expect a problem to arise because of this. However, in general, this can be a cause for singularities and special, non-generic cases.

In general, when intersecting two subsets of the parameter space of a node to compute the new degrees of freedom we only consider the generic case. If we have two subsets M and N of a d-dimensional space with dimensions m and n respectively, the intersection is of dimension $m + n - d$. This is true if we assume that the intersection produces a real reduction of the dimensions, i.e. M is not a subset of N nor is N a subset of M, and the intersection of M and N is not empty. Furthermore, the intersection has to be possible, i.e. $m + n \geq d$. Equivalent reasoning is used in degrees-of-freedom analysis. Note that for the labelling we only indicate the topological type of the intersection which could be interpreted as a parameter space for the intersection rather than the intersection itself.

In summary, the complete constraint distribution algorithm is listed in Algorithm 6.1. We say that we can distribute a constraint directly if one of the distribution options listed in Table 6.1 can be applied using the reasoning in the previous paragraph without doing any redistribution (step I). Otherwise, starting at the constraint which should be distributed, we do a breadth-first search of the constraint graph until a constraint has been found which can be redistributed (steps II and III do the initialisation and step IV does the search). The breadth-first search moves from edge to vertex to edge, and so on, following the directed edges backwards in the graph. By remembering the search sequence with predecessor markers we can backtrack to the original constraint to find the redistribution path and apply the redistributions of the constraints along the path accordingly. Then we try to distribute the new constraint directly in the graph. If this is possible we have found a way to add the constraint and we report success. Otherwise, we try to find another

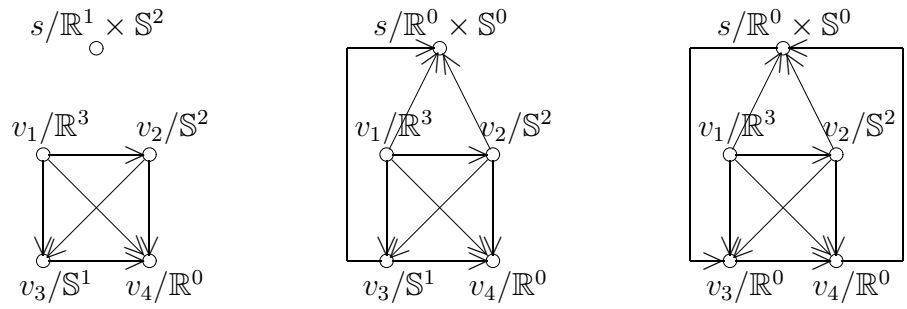
Algorithm `distribute` $(\mathtt{c}, \mathtt{g})$
Try to distribute the constraint `c` in a constraint graph $\mathtt{g} = (\mathtt{nodes}, \mathtt{edges})$. `g` is updated if distribution is successful, and `success` or `failure` of distribution is reported.

I. If `c` can be distributed directly without any redistribution, do so and return `success`.
II. Mark all `nodes` and `edges` as not-visited and set the predecessor of all `nodes` and `edges` to `empty`.
III. Initialise a set `activeE` of edges (constraints) to `c` and a set `activeN` of nodes (geometric elements) to `empty`.
IV. While `activeE` is not `empty`:
    A. Consecutively remove all constraints `e` from `activeE`:
        1. Mark `e` as visited.
        2. Add all not-visited nodes connected by `e` to `activeN` and set their predecessors to `e`.
    B. Consecutively remove the nodes `n` from `activeN` and for all constraints `e` restricting `n` which have not yet been visited do:
        1. Check whether we can find a redistribution `r` of `e`, which can be applied directly to the graph, such that `n` is less restricted than before. Remember the redistribution `R` which creates the largest increase of degrees of freedom in `n` over all `e` and `n`.
        2. Add `e` to `activeE` and set the predecessor of `e` to `n`.
        3. Mark `n` as visited
    C. If `R` is not empty, a redistribution path has been found:
        1. Apply the redistribution in `R`.
        2. Follow the predecessor markers and redistribute each constraint along this path.
        3. If `c` can be distributed directly, do it and return `success`.
        4. Otherwise, clear all marks in `g`, set `activeE` to `c`, `activeN` to `empty` in order to restart the search for a new redistribution path within the loop of step IV.
V. No distribution has been found, return `failure`.

Algorithm 6.1: Constraint Distribution Algorithm.

redistribution path. For this we restart the search at the original edge in the graph modified by the previous redistribution. This ensures that we always find the redistribution option closest to the original edge. If we do not find a path which allows the distribution of the original constraint, we report failure.

To see how the algorithm works, consider the constraint graph in Figure 6.3 linking four



(a) (b) (c)

Figure 6.3: Example Constraint Graph of Distances Between Four Points on a Plane.

points v_l and one plane s. Graph (a) has been created by adding an additional point v_4 to the graph in Figure 6.1 and adding three distance constraints from the other points to v_4. All new distance constraints can be distributed directly such that v_4 is now completely constrained. The plane s is described by a distance and a direction indicated by listing the domains $\mathbb{R}^1$ and $\mathbb{S}^2$ in the graph. In graph (b) we add three constraints placing v_1, v_2 and v_3 on the plane s. Each of these constraints can be distributed directly and each time the degrees of freedom of the plane are reduced by one. This means the plane is now completely determined, i.e. it is labelled $\mathbb{R}^0 \times \mathbb{S}^0$. So far it was possible to add all the constraints by direct distribution.

In graph (c) we distribute a constraint placing v_4 on s as well. We search for a redistribution path in step IV of the algorithm starting at the new constraint edge. Step IV.A adds s and v_4 to `activeN`. In step IV.B we find three direct redistribution options to v_1, v_2, v_3 for s, and three direct redistribution options to v_1, v_2, v_3 for v_4. All the edges for these options are added to `activeE` to continue the search. In step IV.C we choose the redistribution with the maximal increase of degrees of freedom for s or v_4. In our case this can be any of the constraints between s and any of v_1, v_2 or v_3, or between v_4 and either v_1, v_2 or v_3. We choose to redistribute the constraint between s and v_3, initially reducing the degrees of freedom of s by one. We redistribute the constraint for this edge, reducing the degrees of freedom of v_3 by one and increasing the degrees of freedom of s by one. Now the original constraint can be distributed directly and we report success with the distribution as shown in graph (c).

Any redistribution path which does not result in adding the new constraint does not change the solvability properties of the constraint system. While redistribution changes the distribution of degrees of freedom in the graph, the dependency sub-graphs still contain sufficient degrees of freedom, and only the distribution of the constraints in the graph changes.

If the new constraint *can* be distributed, the number of degrees of freedom changes, so we must check if the system is still solvable as described in the following section.

6.3 Solvability Test

Distributing a constraint adds a constraint to the constraint graph. This does, however, not determine if the constraint system remains solvable. The solvability criterion for the general case is similar to the one for the distance constraints. For a three-dimensional object embedded in $\mathbb{E}^3$ we must have at least six degrees of freedom for the system to be solvable in the generic case. For zero-dimensional points we must have three and for collinear points we must have five degrees of freedom. As different types of geometric objects are present, other special cases are possible. Constraints between directions only, for instance, relate to arrangements on the unit sphere. For a zero-dimensional direction set (i.e. one distinct direction) we have only two degrees of freedom. The direction space is only two-dimensional which means if only directions are involved, the minimum number of degrees of freedom required for two or more distinct directions is three. Single surfaces and curves may also have less degrees of freedom.

Other constraints set values for variable angular and length parameters. The presence of variable scalar parameters does not affect the minimum number of degrees of freedom in the three-dimensional case. The scalar parameters do not specify the location or orientation of the object in $\mathbb{E}^3$ and no additional degrees of freedom (e.g. a linear scaling factor) have to be considered as we have constraints setting absolute parameter values.

After a constraint has been successfully distributed in the graph, we have to check whether the new graph still represents a solvable system. The new constraint is distributed amongst some nodes. If the dependency sub-graphs of these nodes over the constraint have sufficient degrees of freedom we say the graph remains solvable under the assumption that we only have generic intersections. The dependency sub-graphs can be detected efficiently by a greedy algorithm following the directions of the edges backwards. To get the complete dependency sub-graph we also have to relabel the start node (only for the sub-graph). For this we collect all edges between the start node and the rest of the sub-graph and compute the new degrees of freedom for the node considering only these edges. We can then easily check the degrees of freedom in the sub-graph. But recall that special cases may arise where the constrained objects require less than six degrees of freedom in order to be solvable (see above).

If the solvability test is successful, then the constraint can be added to the graph without

destroying the generic solvability of the constraint system. Note that with this process we can only check for generic solvability as we do not have any additional information about non-generic cases and how the degrees of freedom are affected by them.

For example, consider the graph in Figure 6.3(c). Distributing the constraint between p_4 and s created this graph. We have to check the dependency sub-graph of s over p_4. This graph is identical with the complete graph and has five degrees of freedom. In order for the graph to be solvable it has to have six degrees of freedom, i.e. the constraint between s and p_4 has made the system unsolvable. Indeed, the constraints in the graph in Figure 6.3(b) are sufficient to determine the four points and the plane up to location and orientation.

As already noted, there are many similarities between our method and the successful `dense` algorithm [121]. The main purpose of `dense` is to detect solvable sub-systems of a constraint system in order to solve the system symbolically. The constraint systems it is intended to handle usually contain only a few over- or under-constrained cases. Our method detects the solvability of a constraint system and has to handle many cases where the system is over-constrained. It creates a close relation between the constraint graph, the flow distribution approach and the actual constraint system. We do not have a rigorous proof to show that the solvability properties detected by the algorithm describe the exact solvability properties of the constraint system under our assumptions. The theoretical issues are still being investigated. However, we believe the approach to be sound because of its close relation to the successful `dense` algorithm. Furthermore, experiments with real constraint systems (see Chapter 7) show that the method is successful when applied to real, albeit simple, problems.

6.4 Manifolds and Geometric Constraints

In this section we discuss some of the mathematical background for our solvability test related to degrees-of-freedom analysis. We start with describing the topological structures present in geometric constraint systems. We show how this relates to the degrees of freedom of the geometric elements and the reduction of degrees of freedom by geometric constraints. This leads to an improved understanding of the relations between the equation system describing the geometric constraints and degrees-of-freedom analysis of geometric constraint systems employing constraint graphs.

A geometric constraint system consist of geometric elements and geometric constraints. The constraints limit the combinations of values the elements are allowed to have at the same time. Typically the constraints do not impose arbitrary limitations, but select sub-

sets of all possible geometries with proper topological structures. We first describe the structures of the geometries with some examples and show how these are restricted by geometric constraints.

A geometric element is described by a geometry type and a vector of real parameters. The geometry type can be regarded as a mapping from the parameter vector space to $\mathbb{E}^3$ representing the geometric realisation of the geometric element. The vector can usually be split into small vectors describing positional, directional, etc. properties of the geometric element. Some of these may be features used for our regularities, but not all of them are. For instance, consider a geometric element of the type plane. It can be described by six real parameters, where three give a position on the plane and the other three give a direction for the normal. Only the normal is a feature of the plane, the position is not (it does not change in a similar way to the cell under isometries). Grouping the six-dimensional vectors describing planes by their function, we can map an element of $\mathbb{R}^3 \times \mathbb{R}^3$ to a plane in $\mathbb{E}^3$. But this mapping is not unique. In order to get the manifold of all planes in $\mathbb{E}^3$, we have to identify the elements of $\mathbb{R}^3 \times \mathbb{R}^3$ which give the same plane. For this we use the equivalence relation

$$(p_1, d_1) \sim (p_2, d_2) :\Leftrightarrow {p_1}^t d_1 = {p_2}^t d_1 \wedge {d_1}^t d_2 = \|d_1\|\|d_2\|. \tag{6.1}$$

on $\mathbb{R}^3 \times \mathbb{R}^3$. This yields a three-dimensional manifold of the form $\mathbb{R} \times \mathbb{S}^2$ of oriented planes in $\mathbb{E}^3$ (note that we require oriented planes for the boundary representation; for unoriented planes the manifold would be $\mathbb{R} \times \mathbb{P}^2$). Thus, for the plane we have a two-dimensional direction space and a one-dimensional position space.

Now consider a conical geometric element. We can parameterise it with an apex parameter in $\mathbb{R}^3$, a direction parameter in $\mathbb{R}^3$ and a semi-angle parameter in $\mathbb{R}$. The parameter product space $\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}$ can be mapped onto the space of all conical surfaces in $\mathbb{E}^3$. Obviously the mapping is not unique and we can factor $\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}$ by identifying parameters which give the same cone. This is described by the equivalence

$$(p_1, d_1, \alpha_1) \sim (p_2, d_2, \alpha_2) :\Leftrightarrow p_1 = p_2 \wedge {d_1}^t d_2 = norm d_1 \|d_2\| \wedge \alpha_1 = \alpha_2 \bmod \frac{\pi}{2}. \tag{6.2}$$

The positional parameters remain three-dimensional while the directional parameters become a two-dimensional space $\mathbb{S}^2$ (in the case of an oriented surface) and the semi-angle space remains one-dimensional ($\mathbb{S}^1$). We get a product manifold of the type $\mathbb{R}^3 \times \mathbb{S}^2 \times \mathbb{S}^1$ for (oriented) cones.

Recall the representation of lines in $\mathbb{E}^3$ using Plücker coordinates mentioned in Section 6.2. We can represent a line uniquely by a pair $(d, l) \in \mathbb{R}^3 \times \mathbb{R}^3$ which fulfils the normalisation equation $\|d\| = 1$ and the Plücker relation $d^t l = 0$. Thus, the set of all lines

in $\mathbb{E}^3$ forms a four-dimensional sub-manifold of $\mathbb{R}^3 \times \mathbb{R}^3$. We can also describe a line by a position and direction pair in $\mathbb{R}^3 \times \mathbb{R}^3$ and then identify pairs which describe the same line by the equivalence

$$(p_1, d_1) \sim (p_2, d_2) :\Leftrightarrow (p_2 - p_1)^t d_1 = 0 \wedge {d_1}^t d_2 = \|d_1\|\|d_2\|. \tag{6.3}$$

This yields the same four dimensional manifold. However, in this case the position and direction are not independent anymore, so we cannot represent it as a product manifold of a positional and directional space. In Section 6.2 we argued that we can still treat it as a product manifold of a two-dimensional direction space and a two-dimensional position space, i.e. choose a position in a plane and then choose a direction from that position. But this only works if the direction does not lie in the plane. As we can choose the plane arbitrarily we can always make sure that this is the case locally. However, globally using this product manifold we treat lines as being different even if they have the same geometric realisation. But for degrees-of-freedom analysis we solely argue about the dimension of the spaces, which is not directly affected by this — especially if we only consider small local modifications.

In general each geometric element is described by a parameter tuple $(p'_1, \ldots, p'_m) \in \mathbb{R}^{d'_1} \times \cdots \times \mathbb{R}^{d'_m}$ where each parameter p'_l is an element of a d'_l-dimensional vector space over $\mathbb{R}$. This tuple is mapped to a set in $\mathbb{E}^3$ depending on the geometric type of the element, which is not necessarily unique. We assume that each parameter tuple can be identified with a tuple $(p_1, \ldots, p_m)$ as an element of a feature space product which describes a unique geometry. Note that all p_l have to be elements of feature spaces (path-wise connected, smooth, abstract manifolds), but not all of them are features. Each of the parameters p'_l in $\mathbb{R}^{d'_l}$ relates to an element p_l in a d_l-dimensional feature space P_l. We say that the *topological type* of p'_l is P_l. We refer to the topological dimension of P_l as the *degrees of freedom* of p'_l.

However, not all geometry manifolds are homeomorphic to product manifolds of feature spaces. By definition, manifolds of the same dimension are locally homeomorphic, but their global topological structure can be different. This means that the parameters are locally independent from each other, but globally they are not. In the context of degrees-of-freedom analysis this is ignored, but may lead in general to singularities where degrees-of-freedom analysis is not able to determine the structure of the constraint system. We are only arguing generically and ignore any such singularities which may be introduced by this system for certain geometric types. We must solve equation systems to determine such situations, as they cannot be detected by arguing solely about the topological type. Note that if we are arguing about local modifications to a valid model, the global structure of the space is less important. The topological types of the parameters we require for our

geometry types are directional feature spaces $\mathbb{S}^2$ (used instead of $\mathbb{P}^2$ as the geometries also have an orientation), positional feature spaces $\mathbb{R}^3$, $\mathbb{R}^2$, $\mathbb{R}^1$, angle feature spaces $\mathbb{S}^1$ and length feature spaces $\mathbb{R}_+$. The shape of a geometric element is then described by a product of parameter spaces which are associated with a feature space.

We consider a single geometric constraint to be between geometric elements limiting the values certain parameters can have at the same time. This selects a subset of the product space of all parameter spaces involved in the geometric constraint system. As it only limits the values of the parameters directly involved in the constraints we can limit the discussion of a single constraint to the product of the parameter spaces of the involved elements. For our constraint systems we consider incidence constraints between features and cells, distance constraints between features of the same type and subset constraints between geometric elements. The constraint types we use to describe the improved model are basically covered by these or can be analysed in a very similar way. For simplicity, we only consider the basic types here.

First consider an incidence constraint between two features p_1, p_2 of the same type represented by the parameters p'_1, p'_2. We only have to consider the parameter spaces P'_1, P'_2 and the related feature spaces P_1, P_2 for these two features (as the features are of the same type we have $P'_1 = P'_2$ and $P_1 = P_2$, but we keep separate parameter space instances for each parameter). With respect to the feature space, the constraint selects the sub-manifold $S = \{(p_1, p_2) \in P_1 \times P_2 : p_1 = p_2\}$ of the product feature space $P_1 \times P_2$. Using the mapping from the product parameter space to the product feature space, S is the image of $S_l = \{(p'_l, p'_l) : p'_l \in P'_l\}$ for l either 1 or 2. Note that S_l is a sub-manifold of $P'_1 \times P'_2$. Thus, if we choose to use S_1, we can express this by setting the degrees of freedom of P'_2 to 0. This indirectly represents the condition that p_1 and p_2 are mapped onto the same feature.

If we only consider the local structure of the d-dimensional feature spaces P_1 and P_2, we get additional options to distribute the degrees of freedom. Locally P_1 and P_2 are like $\mathbb{R}^d$. Thus, locally an element of these spaces can be represented as a d-tuple and we get the local version of S above as $S_{\text{local}} = \{((p_{1,1}, \ldots, p_{1,d}), (p_{2,1}, \ldots, p_{2,d})) \in \mathbb{R}^d \times \mathbb{R}^d : p_{1,l} = p_{2,l} \text{ for } l = 1, \ldots, d\}$. This means that we can arbitrarily reduce the degrees of freedom of both P'_1 and P'_2 under the condition that the overall reduction is d.

Incidence constraints between cells can be interpreted as incidence constraints between the relevant parameters. For each corresponding parameter pair we have the option to set one of the two degrees of freedom to 0. Considering only the local structure in a similar way as for the feature incidences, we can even distribute the dimension reduction between the feature pairs arbitrarily.

Next consider a distance constraint between two features p_1, p_2 of the same type. The constraint selects a sub-manifold $S = \{(p_1, p_2) \in P_1 \times P_2 : d(p_1, p_2) = c\}$. In a d-dimensional feature space this can be interpreted as requiring that feature p_1 lies on a $(d-1)$-dimensional sphere with centre p_2 or vice versa. So S can be parameterised by $S_1 = \{(p'_1, p'_2) : p'_1 \in D^{d-1}, p'_2 \in P'_2\}$ where D^{d-1} is the inverse image of a unit sphere in the feature space under the mapping from the parameter space to the feature space (for positional features this is $\mathbb{S}^2$, for directional features, this is the inverse image of a unit circle in $\mathbb{P}^2$ which is a cone in $\mathbb{R}^3$). We can express this in terms of degrees of freedom by reducing the degrees of freedom of P'_1 by 1. Alternatively we can also do this for P'_2 in a similar way.

Subset constraints are the only type of constraints which relate different feature spaces. For instance, consider a constraint requiring that a point p lies on a plane q. Let p be parameterised by $p'_1 \in \mathbb{R}^3$ (topological type $\mathbb{R}^3$) and q be parameterised by a position $p'_2 \in \mathbb{R}^3$ (topological type $\mathbb{R}^1$) and a direction $p'_3 \in \mathbb{R}^3$ (topological type $\mathbb{S}^2$). Note that, because as not all parameters are features, we cannot argue in terms of the feature spaces. The parameter product domain for the two geometric elements is $\mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$ corresponding to the topological types $\mathbb{R}^3$, $\mathbb{R}^1$ and $\mathbb{S}^2$ respectively. If we interpret the constraints as putting the point in the plane, we can parameterise the selected subset by $S_1 = \{(q'_1 + Mp'_1, q'_1, q'_2) : p'_1 \in \mathbb{R}^2, q'_1 \in \mathbb{R}^3, q'_2 \in \mathbb{R}^3\}$ where M is a 3×2 matrix containing two linearly independent directions in the plane. This reduces the degrees of freedom of the parameter space for p_1 by one. It can now only be chosen from a space which is homeomorphic to $\mathbb{R}^2$. We may call $\mathbb{R}^2$ the *reduced topological type* of p_1.

We can also use $S_2 = \{(p'_1, q'_1, q'_2) : p'_1 \in \mathbb{R}^3, q'_1 \in \mathbb{R}^3, d(q'_1, 0) = d(p'_1, 0), q'_2 \in \mathbb{R}^3\}$ which makes p'_1 determine the distance of the plane from the origin and sets the reduced topological type for q'_1 to $\mathbb{R}^0$. Alternatively we can use $S_3 = \{(p'_1, q'_1, q'_2) : p'_1 \in \mathbb{R}^3, q'_1 \in \mathbb{R}^3, q'_2 \in \mathbb{R}^3, (q'_1 - p'_1)^t q'_2 = 0\}$ which makes p'_1 determine part of the normal direction of the plane under the condition that $p'_1 \neq q'_1$ (ignoring such singularities in the generic case). This sets the reduced topological type of q'_2 to $\mathbb{S}^1$.

Other subset constraints can be interpreted similarly, such that the dimensions of the involved parameter spaces are reduced. As we do not solve any equation system, we have to ignore singularities, special arrangements, etc. This means that the overall reduction of dimensions introduced by constraints can be distributed over any combination of involved parameter spaces by reducing their degrees of freedom. As locally the geometry spaces for the geometric elements behave like independent product manifolds of vector spaces over $\mathbb{R}$, this works properly in many cases.

In general for a geometric constraint there are multiple options for selecting some of the

involved parameter manifolds and requiring that the parameters corresponding to these manifolds have to lie in a sub-manifold. This also selects sub-manifolds of related feature spaces for the parameters. The reduction of the dimensions of the feature spaces by these sub-manifolds can be expressed by reducing the degrees of freedom of the involved parameters. For the solvability analysis we only argue in terms of degrees of freedom left for the involved geometric elements. This avoids having to solve any equation systems, but also means that we have to make certain assumptions about the constraints. The main restriction is that we are arguing about locally homeomorphic manifolds rather than globally homeomorphic manifolds. This means we consider manifolds of the same dimension to be the same or at least similar. As long as only local properties are important this is true. But when global properties are relevant, it will fail.

Enforcing multiple constraints means that we intersect the allowed parameter value sets. We assume that this can be done separately for each parameter, and the intersections are always generic. As discussed in Section 6.2 this means we always get a dimension reduction when intersecting the sets of allowed parameter values. Furthermore, the reduction represents the usual, most likely case, and not any special cases requiring special parameter values and constraints.

In this section we have only given a brief overview of the issues involved in degrees-of-freedom analysis of geometric constraint systems. We have presented the degrees-of-freedom analysis in a topological context and believe that this shows more clearly the structures it can handle and where its assumptions do not apply. Neither are the intersections of the sub-manifolds always generic, nor do we always have product manifolds of independent parameters. But in many cases these assumptions apply in a local sense and thus allow us to determine the solvability of a large number of geometric constraint systems. The topological interpretation also gives a direct approach to the constraint distribution algorithm which is similar to the successful `dense` algorithm [121] and constraint decomposition schemes based on it [51, 52]. However, more work on the theory is required to better understand the involved structures. A more detailed analysis of the topological interpretation may eventually improve our understanding of geometric constraint systems and help to find ways to expand existing methods to more general cases.

6.5 Summary

In this chapter we have presented an algorithm to detect the solvability of a geometric constraint system. By adding constraints consecutively to a constraint graph using the distribution algorithm, and checking if the expanded constraint graph remains solvable,

we can determine the solvability of a constraint system without computing a solution. We simply check the topological dimensions of the involved parameter spaces to verify if a constraint can be imposed on a set of geometric objects which may already be constrained. In order to do this without solving any equations we have to make certain assumptions about the type of constraints. Each constraint limits the values the parameters can have at the same time. There are multiple options to describe this limitation by restricting different parameters to subsets of lower dimension. When enforcing multiple constraints, the subsets have to be intersected. Without solving any equation systems, we assume that this yields certain generic intersections, which is the most likely situation. The topological interpretation of constraint systems allows us to investigate the underlying structures of degrees-of-freedom analysis in more detail and identify special cases for which it does not apply. Future research based on this may enable us to extent the methods to more general cases.

In order to eventually solve the equation system, we use the numerical optimisation method presented in Section 5.6. Degrees-of-freedom analysis is usually combined with a symbolic solver and the constraint system is decomposed into small solvable sub-systems. However, there may be sub-systems which cannot easily be solved symbolically and for which no explicit solution is known. Furthermore, for beautification the geometric model may not always be described completely by our geometric constraints (it is only highly likely as we detect many regularities). Thus, using an optimisation solver starting at the parameter values for the initial model is likely to find the desired solution. It would, however, be possible to replace the optimisation solver with a symbolic or any other type of solver.

Chapter 7

Experiments

In the previous chapters we discussed the analyser, hypothesiser and rebuilder components of our beautification system. In this discussion we have concentrated on robust detection of approximate geometric regularities, and the solvability of a constraint system describing the model and the selected regularities. The constraint system should represent the likely, original design intent of the reverse engineered object. In this chapter we present the results of experiments of using this system on real and simulated reverse engineering data. In general, we have to be careful in interpreting results from testing reverse engineering methods with simulated data, as the actual physical measurement may create problems of a type not present in synthetic data. However, as we will see, the beautification problem is quite similar for real and simulated data, mainly as the beautification methods work solely on the initially reconstructed boundary representation models. The experiments show that our beautification system is able to improve both types of initial model with respect to design intent. As the initial models are approximate, there is always some uncertainty about the actual design intent. Depending on the tolerance for the initial model, specific parameter values and minor regularities are not always reconstructed according to the original design. But major regularities, like global symmetries, major orthogonal systems, etc. representing the global structure of the model, are imposed exactly on the improved model in these test cases.

In Section 7.1 we give a brief overview of the implementation and platform used to test our beautification algorithms. In Section 7.2 we present the results of experiments with simulated data, and in Section 7.3 we present the results of experiments with real data. Finally, we discuss the results of the experiments in Section 7.4

7.1 Test Platform and Implementation

In this section we briefly present details of the platform used to implement and test our beautification algorithms. We also discuss the basic setup for the experiments.

The algorithms were implemented using the modular structure described in Section 1.3. For the analyser we used the algorithms described in Chapter 3. Unless otherwise noted, we used a minimum distance tolerance ΔT_A of 1° for angles and a minimum distance tolerance ΔT_L of 1 unit for lengths (relating to millimetres for the real objects). These tolerances are used to indicate when two features may be different. All other parameters were chosen as described in Chapter 3. For the hypothesiser, we used selection priorities with parameters as described in Chapter 4. By fine-tuning these parameters for each model, the results could certainly be improved as noted later. For testing solvability of the constraint systems we only tested the topological approach as described in Chapter 6 for all models used in this chapter. Experiments with the numerical solvability test showed that this is clearly too slow for practical purposes (see below). The constraint systems were solved using the numerical constraint solver from Section 5.6. Their solutions were computed to within an error of less than 10^{-5} for each equation, which formed the convergence condition for the solver. Note that the equations (see Table 5.3) do not indicate an error in consistent units (not even in any angle or length units), but they were chosen for efficient evaluation. This error bound strongly influences the time required for the numerical solver. Using larger values, say 10^{-2} or 10^{-3}, is sufficient in many cases to construct a reasonably accurate model, and reduces the time for solving the constraint system numerically.

Employing the numerical constraint solver to test for solvability is far too computationally intensive for practical purposes. In [72] we present some timing results. For instance, beautifying the cube model described in Section 7.2 took about 2 hours on a computer with a single AMD Athlon MP 1200MHz with 512MB RAM. For more complex models it could easily take about 24 hours. These very long running times are solely caused by running the numerical solver multiple times to check whether a solution of a constraint system exists. As most of the constraint systems are not solvable, the solver has to handle numerical instabilities and thus the convergence rate is very slow. Furthermore, the constraint system has to be solved to a low error to ensure that a solution actually exists. Beautification of the cube employing the topological solvability test took less than 4 seconds on the same platform with the same tolerance for the solution of 10^{-3} using the optimisation solver. While most of this time is also used for solving the equation system, only one consistent equation system had to be solved, which requires considerably less time.

By specifically tuning the priority parameters for the regularity selection and the constants employed by the numerical constraint solver, the running time for beautifying the cube with the numerical solvability test could be reduced to 10 minutes, which is still considerably longer than employing the topological test. Hence, the numerical solver is

not suitable in practice due to either very long running times or the requirement of time consuming fine-tuning of parameters. Moreover, we have a considerably faster method using the topological solvability test.

For the implementation we employed the ACIS solid modelling kernel from Spatial, version 7.0. It was used mainly for creating and displaying models, and for standard geometric modelling algorithms like surface-surface intersection when rebuilding an improved model. The beautification system itself was build separately with a model representation based on the boundary representation discussed in Section 2.1. This allowed a simpler, more efficient implementation of our algorithms without having to deal with the internal structure of ACIS. The algorithms were implemented in C++ on a GNU/Linux platform (Debian GNU/Linux 3.0 woody) using the GNU GCC C++ compiler, version 3.0.4.

For the timing results of the experiments a dual processor Athlon MP 2600+ 2.12GHz computer with 1.5GB RAM was used. The algorithms ran without any threading or parallel processing on a single processor. We ran the algorithms at least five times to obtain timings where no time variations of more than one tenth of a second were noted for any given data set.

7.2 Experiments with Simulated Data

In this section we present the results of beautifying models reverse engineered from simulated data. By rotating and translating faces of exact models, and generating a point cloud from these perturbed objects, we created simulated point sets. From the point cloud we obtained initial reverse engineered models using the reverse engineering system described in Section 1.1.1. These models were then beautified with our algorithms.

Simulated data was mainly used for initial tests of our methods, as it was easier and faster to obtain. There were no problems related to scanner calibration, accessibility of the object, or physical properties of the surface (colour, reflection, etc.), which make it hard to probe real objects with a laser. We also did not have any occlusion problems caused by deep cavities, etc. which cannot be probed by a laser scanner, and thus we could choose from a large set of test models. In general, for reverse engineering, tests with simulated data may yield results considerably different from results of tests with real data. Randomly distorting a model and creating a point set by introducing some Gaussian noise to point sets obtained from the model surface does not actually account for issues related to physical measurements. Our simulation introduced relatively evenly distributed noise over the whole point set. Real data often is generally very exact, but is disturbed

locally by undesired effects causing peaks or holes in the data. Registering different views of real objects may also introduce errors not present in simulated data. Often it is easier to generate an acceptable initial model from simulated data than real data. Nevertheless, simulation was useful to determine whether the reverse engineering software would be able to handle data from a real object. If it does not work with simulated data, it is highly unlikely to work with real data. As expected, however, the simulation never predicted all problems arising in real data.

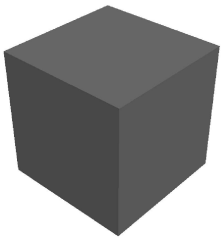
Our aim, however, was not to test the reverse engineering software. Instead, we wished to test our beautification algorithms. Here we worked solely with a boundary representation model. This model was already in a relatively high level representation and problems relating to local inaccuracies in the point data were less important. Once we have a valid initial model, the noise in the point data and the noise introduced by the reconstruction process are represented by noise in the features derived from the model. While such noise is the cause for inaccuracies in the representation of the initial model's surfaces and thus the cause for inaccurate features, for beautification we only deal with the inaccurate features and not the point data. Hence, the problems in the models generated from real and simulated data were quite similar.

In the following, we first discuss the experiments on simulated data from a simple cube model to show the general behaviour of our system. We then present results for simulated data from more complex models. Experiments with real data are discussed in Section 7.3.

7.2.1 A Simple Example with Simulated Data

We illustrate the general behaviour of our method using data simulated from the model of a cube shown in Figure 7.1. This model is simple enough for us to be able to list the regularities detected and explain the behaviour of the system in more detail. A cube with edge length 2 was perturbed by randomly changing the plane normals by at most 3 degrees and face positions by at most 0.1 length units. From the resulting model, a point set was generated, and an initial model was reverse engineered.

As listed in Figure 7.1 the analyser detected 21 approximate regularities. The complete list has been simplified for this discussion. We list the number of regularity types detected at different tolerance levels (the tolerance levels in the table are only indicative, not precise). The algorithms clearly detected the orthogonal system of the cube, but at a high tolerance level due to the perturbation. This also caused the detection of two parallel direction pairs at a lower tolerance level and a third one at a higher tolerance. We detected six angles between the plane normals with a set of various special values close to $\pi/2$ at different



Time taken in

Analysing:	0.02 sec.
Selecting:	1.21 sec.
Solving:	4.54 sec.
Total:	5.77 sec.

Detected Regularities	**Tol. Level**
1 orthogonal system	3°
2 pairs of parallel directions	1°
1 pair of parallel directions	3°
6 sets of special angles	0.1° to 3°
2 pairs of aligned axes	1°
1 pair of aligned axes	3°
1 intersection of 3 axes	0.15
▶ 1 intersection of 2 axes	0.05
1 equality of 12 lengths	0.1
▶ 2 equalities of 3 and 5 lengths	0.04
3 sets of special edge lengths	0.001 to 0.1

Figure 7.1: A Simple Model from Simulated Data with Test Results.

tolerances as an alternative to the orthogonal system. Furthermore, two of the plane axis pairs generated by the centres of opposite planar faces and the plane normals are quite closely aligned. The third aligned axis pair is at a higher tolerance level.

The intersection of two of the aligned axis pairs is at a low tolerance value. At a higher tolerance level this intersection is expanded by the third axis pair. This creates a parent/child relation in our regularity structure, indicated by ▶ in the list. Similarly we find two groups of three and five edge lengths close to each other which are combined at a higher tolerance level to a single group as indicated again by ▶. The group at the higher tolerance level also contains four additional length values, which were not part of any group at a lower tolerance. For each of the groups we also found sets of special edge length values at various tolerance levels.

For the priority values of the regularities, which determine which regularities are chosen in the final model, there are two basic options. We can emphasise the quality of the regularities such that the orthogonal system and all regularities following from it are selected. Alternatively we can favour small tolerances, which results in the selection of alternative regularities where in particular one of the approximately aligned axis pairs is not orthogonal to the other two.

By setting the priorities to favour an orthogonal system, a cube with edge length 2 was created as the improved model. Besides emphasising parallel directions and orthogonal systems, we also had to take care that the special values for edge lengths strongly emphasised integer values to avoid choosing a different edge length.

In contrast, if we favoured precise tolerances, integer angles different from $\pi/2$ were chosen for all relations between the directions. The equal edge length groups were completely rejected due to solvability problems and various different values for the edge lengths were selected.

We also tried a third set of priorities which neither emphasised tolerance nor quality. In this case we obtained three aligned axis pairs where two were orthogonal to each other, but the third one had an angle of $88°$ to the other two. The group of three equal edge lengths was accepted with a special value 1.9. The other edges all had separate special length values or none set specifically by a regularity due to solvability issues.

However priorities were adjusted, the selected constraint system was solvable and the numerical solver found a solution within a predetermined numerical tolerance (10^{-5} for the experiments presented here).

Depending on our assumptions about the initial model, we can either accept high quality regularities or regularities with small tolerances. Only a regularity for which both is true is likely to be accepted in both cases. In this example, the third model may actually be the most likely one if we ignore our knowledge about the original perturbation of the model. There is some evidence in the initial model that the third axis is not part of an orthogonal system as regularities relating to it are at a higher tolerance level. In all three cases, selecting particular special values for lengths and angles is hard. While we can enforce all edges to have the same length this also means that we have to accept a high tolerance level and so various special values are possible.

The overall time required to improve the model from the initial data was 5.77 seconds on average. The breakdown of this time for the various components of our system is shown in Figure 7.1. Note that the time for rebuilding the model was under 0.1 seconds and is accounted for in the solving time.

7.2.2 More Experiments with Simulated Data

We now discuss the results of experiments with more complex models shown in Figure 7.2 generated from simulated data. Models (d) and (e) were obtained from [148]. Table 7.1 lists the number of regularities detected in each case, how many of them were selected to improve the model, and the times required for analysing the initial model, selecting consistent regularities and solving the constraint system. Note that due to the hierarchical clustering, counting the number of regularities found is not trivial. We did not count clusters as regularity which consist only of a single element, but are required for clusters at

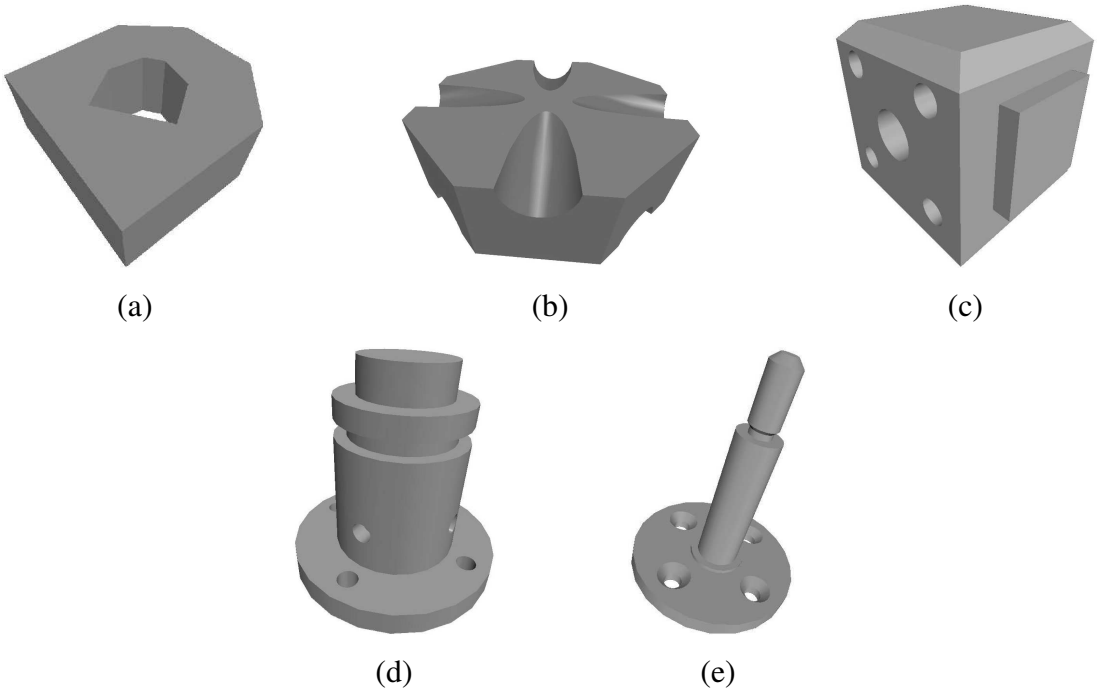


Figure 7.2: Example Models Reverse Engineered from Simulated Range Data.

larger tolerance levels. Furthermore, the multiple special values suggested for an angle or length value in the model were counted as a one regularity. This is justified as the selection rules (see Section 4.3) ensure that only the most likely special value can be selected. Note that we can either set a special value for a scalar parameter consistent with the other regularities or not. In case of an inconsistency caused by one special value we do not get a generically consistent constraint system by choosing an alternative special value for the same scalar parameter (it would at least lead to redundant constraints and more likely to inconsistent constraints). We also list the number of equations required to describe the constraint system in the numerical solver. These equations include those required to describe the topology and the auxiliary cells as well as the regularities detected. In the following we only discuss the major results relating to the main structure of the model.

Model (a) has two planar angle-regular arrangements of planes with base angle $\pi/4$ which have been detected. However, an alternative arrangement combining both into a single regularity with base angle $\pi/18$ has also been detected at a larger tolerance level. As two angle-regular arrangements based on $\pi/4$ are at a lower tolerance level with a more desirable angle, they were selected for inclusion in the model. But note that changing the priorities or increasing the distortion of the object might favour the other regularity. Four orthogonal systems consistent with the planar arrangement were also detected and imposed on the model. For the angle between the two groups of planes, several special values as angles between individual plane pairs were detected. Only one of these angles

Model	(a)	(b)	(c)	(d)	(e)
Faces	14	18	25	23	21
Regularities					
Total	255	255	275	46	39
Selected	31	97	49	30	17
Sel. Equations	366	638	698	306	124
Time taken in					
Analysing	0.19 sec.	0.17 sec.	0.35 sec.	0.04 sec.	0.04 sec.
Selecting	7.01 sec.	7.87 sec.	11.53 sec.	2.83 sec.	2.02 sec.
Solving	37.90 sec.	174.62 sec.	329.54 sec.	28.49 sec.	32.14 sec.
Total	45.10 sec.	182.66 sec.	341.32 sec.	31.36 sec.	34.20 sec.

Table 7.1: Number of Total and Selected Regularities, Selected Equations and Computing Times for Example Models from Simulated Data.

was selected and imposed on the model; problems related to the exact angle value similar to special value problems in the cube example again arose. Additional regularities detected were conical angle-regular arrangements with cones with semi-angles close to $\pi/2$ created by the planar faces. As they were inconsistent with the planar arrangement and had a low priority they were not considered. Further regularities included aligned axes, axis intersections and equal edge lengths.

For model (b) slanted cylinders were cut out at the top and the bottom face of the block where the bottom arrangement was rotated by an angle of $\pi/4$ relative to the arrangement at the top. The planar angle-regular direction sets of the planar faces and the conical angle-regular sets of the cylinder axes were detected and imposed exactly on the improved model. Additional conical angle-regular sets of the directions of cylinder and plane axes were detected as well. Some of these were consistent with the planar and conical angle suggesting the same angle values and were thus selected. Those suggesting angle values inconsistent with the main arrangements were not considered. Unfortunately, the intersection point of the axes generated from the centre of the convex hulls of the side faces was not detected correctly, as the perturbation of the side faces moved some of the axes too far apart. This also caused one aligned axis pair to be missed. Instead regularities specifying individual angles and edge lengths were selected. However, by adjusting the priority and detection parameters these problems could be fixed. The angle between the planar arrangement of the planes and the conical arrangement of the cylinder axes was specified by individual angle values similar to the angle between the two planar angle-regular arrangements in model (a).

In model (c) the orthogonal arrangements of the planes and the conical angle-regular arrangements of the planes at the top were detected and selected for beautification. Additional conical angle-regular arrangements of plane normals were detected, but not all were consistent with the above regularities. Further regularities included various aligned axes and regular arrangements of these axes. The regular arrangement of the cylinder axes was only partly detected with one cylinder missing from the regular axis arrangement on another cylinder and two cylinders missing from the grid. Hence, in the improved model, the positions of these axes was only partly determined by the regular arrangement on a cylinder and the grid arrangement was not considered. The axis positions remained close to their positions in the initial model, though.

Analysing model (d) resulted in detecting that all directions form an orthogonal system, with one main axis. All axes of the main cylinders were found to be aligned, where the axis of the top-most cylinder was aligned with the others at a higher tolerance level. These regularities were imposed exactly on the model. Even though one of the alignments was at a higher tolerance level, no alternative was available and so it was selected as well. The axes of the two pairs of opposite holes in the sides of the large cylinder were also found to be parallel and to lie in a plane with an intersection point on the central axis. The planar angle-regular algorithm detected their symmetrical arrangement. Of the four intersection points of the axes of the holes in the large cylinder, and the axes of the holes in the planar surface at the bottom, only one was detected and imposed on the model. This was sufficient to align the hole pairs properly as the angles between the axes were determined by other regularities. However, note that this result is relatively unstable and modifications of the parameters used could easily break it. The symmetrical arrangement of the cylinder axes around a cylinder aligned with the main central axes was also imposed on the model.

Model (e) is similar to model (d). It has a single main axis which was detected, but no orthogonal system is present. The regular arrangements of the holes on a circle was found. The remaining regularities were similar radii and angle parameters and special values for them. It was also suggested that two cylinder radii which were close to each other may be equal. As this required a change in the topology of the final model, it was not selected. The main problems in the regularities selected for this model were special values. In particular the radii and cone semi-angles for the holes were selected by separate special value regularities, where two of them suggested the same cylinder radii. The equal length and angle parameter regularities were consequently rejected. Only tricky adjustments of the priorities could resolve this problem.

Further discussion of these results will be given in Section 7.4.

7.3 Experiments with Real Data

In this section we present results of experiments using real data. For these experiments, we scanned real objects made from an ACIS model using a simple three-axis milling machine (with an accuracy of 0.05mm and a resolution of 0.025mm). The objects had an approximate size of 70mm $\times$ 70mm $\times$ 40mm. The model discussed in Section 7.3.1 is an exception, as originally we only had a real object and no CAD model (it is an accurately machined object for the RECCAD project of an approximate size of 60mm $\times$ 60mm $\times$ 150mm).

As already indicated in Section 7.2, the results of tests on real data are similar to the results of experiments on simulated data (This was further confirmed by using the exact ACIS models for the real objects to create simulated data). While the inaccuracies were different, the general behaviour of the algorithms was similar. In particular, problems with ambiguous regularities and selection between them were the same. The main difference is that the errors in the simulated data were more evenly distributed. Thus, for simulated data it is more likely that the errors in the desired regularities are at about the same tolerance level. For real data we cannot rely on consistent tolerances levels for the complete model, which further shows the advantages of not using a maximal tolerance (see Chapter 3). (However, note that sometimes simulated data may also have desired regularities at different tolerance levels, as we saw in Section 7.2 for the relatively large error for the third axis of the cube).

In the following, we first discuss a relatively simple model in more detail, and then give the results for some more complex objects.

7.3.1 A Simple Example with Real Data

Figure 7.3(a) shows a simple model used to test our beautification algorithms with real reverse engineering data. To analyse the model we used $\Delta T_A = 1^\circ$ and $\Delta T_L = 2$ length units (millimetres; approx. 1% of the length of the model), which created a good set of regularities. We used parameters different from the ones listed in Section 7.1 as this model was an accurately machined model from the RECCAD project, which should be accurate to a much higher tolerance (although, the initial model from it might not be). In the following we discuss the detected approximate regularities and which of these were selected to improve the initial model.

The object has a single central axis, several planes with normals parallel to this axis and two parallel planes with normals orthogonal to it. This resulted in the detection of two

main parallel direction clusters, where one was a simple cluster representing the main axis at the lowest tolerance level. The other one consisted of two directions representing the plane axes orthogonal to the main axes. Even if parallel in the ideal model, the angle between the two directions in the initial model was sufficiently far apart (about $3°$) such that the two directions were sub-clusters of another cluster. It was also found that these two planes were orthogonal to the main axis.

The central axis regularity was detected and imposed on the model. The two blue parallel planes (only the one in front is visible in Figure 7.3), however, could only be made parallel by allowing large angular tolerances (about $3°$) due to an error in the initial model caused by the registration step. The angular error was already present after registration of the two views for opposite sides of the object, each containing one blue plane. The common points used to register the two views came only from cylindrical and conical surfaces, making the rotation angle hard to determine.

Special values for the cone angle and edge lengths as well as equal edge lengths created similar problems to those for the cube discussed above. As the two plane normals were considered not to be parallel at a small tolerance level, some special values for the angle between the two normals were reported. There was one suggesting some special values for the angle between the directions, and a conical angle regularity consisting of the two plane normals and the main axis. Both regularities would only be realisable if the two directions were not parallel. These regularities were detected as our methods consider all sub-clusters at different tolerance levels, in order to detect possible regularities created by them. These inconsistencies were identified by the solvability test such that only one of the regularities could be selected depending on the priorities. By favouring high quality regularities, we could make the two planes parallel and also orthogonal to the main axis. Alternatively we could look for regularities nearly exactly present in the model, which would result in selecting some special value between the two plane normals. In that case the normals are still made orthogonal to the main axes.

Some of the surface root point positions and vertex positions were also considered to be equal in the position cluster hierarchy. All these regularities had tolerances above 20 length units and would require changes to the topology of the model which would remove large surfaces. As the priorities for these regularities were quite low, they were not considered in any case. Similarly two pairs of approximately equal cylinder radii were detected at the lowest tolerance level, which were considered to be equal at a higher tolerance level. Making them equal would require a change in the topology, and again due to low priorities they were not considered. But instead, a special value for each of the radius clusters was selected.

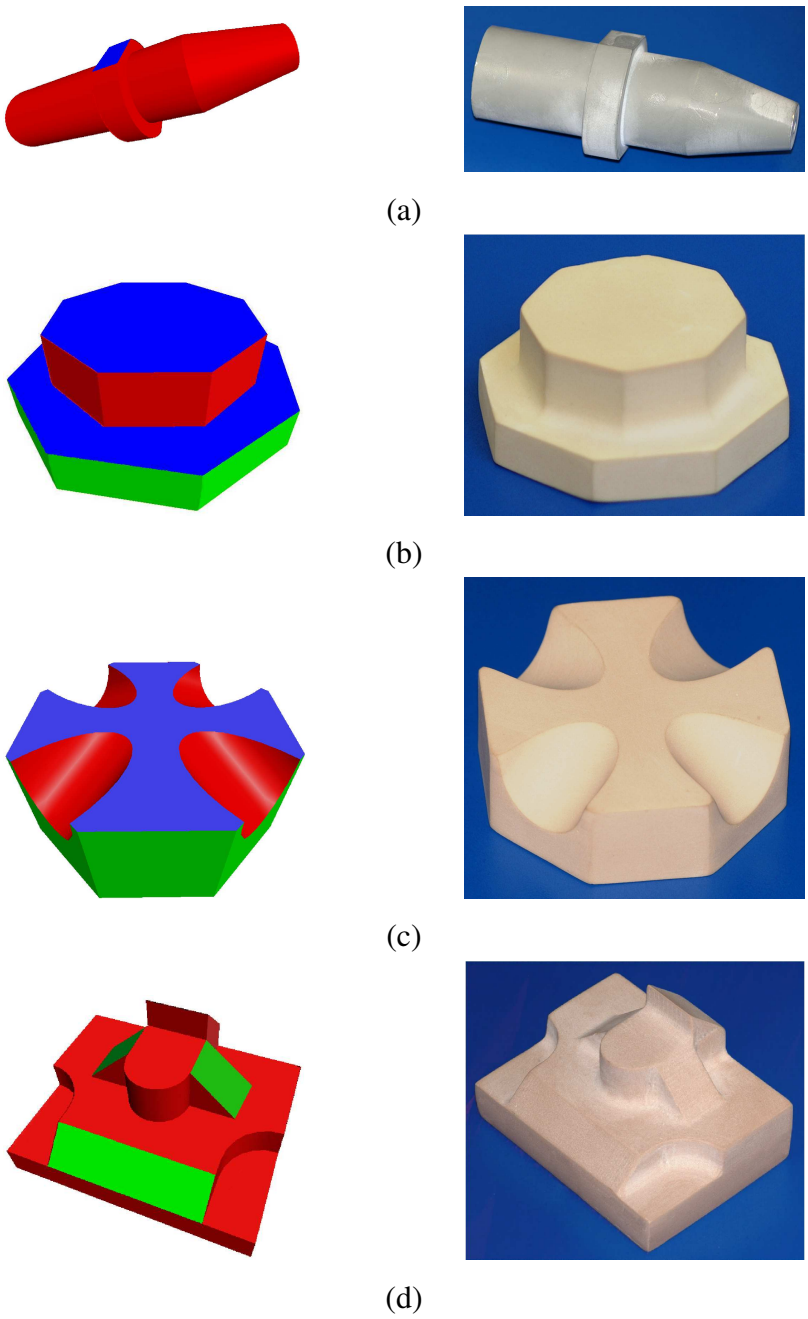


Figure 7.3: Example Models Reverse Engineered from Real Range Data.

7.3.2 Other Real Reverse Engineered Objects

We now discuss the results of improving the remaining three models in Figure 7.3 which were reverse engineered from real data. Due to the large number of regularities found in these more complex models, we do not present them in detail: only how the major regularities were handled by the system is discussed. Table 7.2 lists the number of regularities detected, and how many of each were selected for use to improve the model, in a similar

Model	(a)	(b)	(c)	(d)
Faces	11	19	14	25
Regularities				
Total	25	329	152	287
Selected	14	43	33	45
Sel. Equations	84	469	334	586
Time taken in				
Analysing	0.03 sec.	0.32 sec.	0.89 sec.	0.94 sec.
Selecting	0.11 sec.	9.13 sec.	3.75 sec.	12.05 sec.
Solving	28.52 sec.	178.37 sec.	57.29 sec.	263.89 sec.
Total	28.66 sec.	178.82 sec.	61.93 sec.	276.88 sec.

Table 7.2: Number of Total and Selected Regularities, Selected Equations and Computing Times for Example Models from Real Data.

way to Table 7.1.

The priority values listed in Section 4.2 were used representing the case where quality is only slightly emphasised over the tolerance. Thus, we prefer regularities which are either present very accurately in the initial model or represent a regular arrangement of high quality (as determined by certain priority parameters). If we have contradictory regularities where some are very accurate and others are of high quality, those of high quality are preferred even if they are present at a larger tolerance. In general this resulted in the best overall regularity selections. Fine-tuning the values for particular models could of course improve the selection results. Changing these values to favour quality or tolerance had effects comparable to those discussed in Sections 7.2.1 and 7.3.1.

Model (b) has two symmetrically arranged, planar direction sets based on the angle $\pi/4$. Together with the orthogonal relation between the symmetrically arranged red and green planes, and the blue planes, these regularities have the highest priority and were imposed exactly on the model. Edge lengths caused similar problems to those for the cube. Even by adjusting the priorities, only the group of short edges could be forced to have the same length. The values in the other two groups of lengths were close to each other, but different special values were favoured for the two groups. Special ratios between these values also supported undesired values. The solvability test correctly determined that only one angle between the groups of red and blue planes can be set. However, for the value of this angle there was again a choice between choosing a special value close to the value in the initial model and one of high quality. With our method a particular plane pair was

chosen to select this angle. If we favoured integer degrees, the closest integer to this angle was chosen. The angle in the original design was $10°$. The plane pair chosen had an angle of $45° + 10.8°$ and thus the angle between the two plane sets was chosen to be $11°$. Note that to select a special value of $10°$ between the two plane sets would require to select a special value of $55°$ between the chosen plane pair. In order to prefer this special value, the special value priorities would have to be adjusted to favour multiples of 5 rather than 10. With a more detailed analysis it might be possible to detect that all the angles between the plane pairs should be considered to find an average value for that single angle, but even in this case we have to consider the uncertainty in the average value and select a likely special angle value with a tolerance.

In model (c) the normals of the green planes are arranged symmetrically in a plane, and the axes of the red cylinder are arranged symmetrically on a cone. These regularities were well preserved in the initial model (to within about $1°$) and are also of high quality, so they were selected. The edge lengths and the angle chosen for the conical arrangement had the same problems as for the other models. In addition, in this case we had no regularity specifying a direct relation between the group of cylinders and the planes. Hence, there was a small angle between the cylinder axis directions and the plane normals when projected on the same plane. The edge length regularities and the topological constraints ensured that the lack of a precise relation did not change the topology, i.e. the cylinders could not be rotated in a way that they would intersect with more than one green plane.

The directions in model (d) form one orthogonal system formed by the normals of the red faces and one conical angle regularity formed by the normals of the green faces based on angles of $\pi/4$. The regularities were present in the model to within about $2°$. As they were of high quality, they were selected. The $\pi/4$ rotation between the two direction sets was slightly more ambiguous as it was represented by individual special angle values between various direction pairs from the two sets. The relation was preserved on average to within about $3°$. As our priority parameters favour $\pi/4$ angles, the relation was imposed exactly on the improved model. Further regularities relate to equal edge length and cylinder radii with problems similar to the other models. Regularity selection, however, ensured that the two corner cutouts were congruent.

7.4 Discussion

In the following we discuss the general results of the experiments and the observed behaviour of the beautification algorithms. Our methods are able to improve initial reverse engineered models, but are limited by the ambiguity caused by the fact that we only have

approximate models. Major regularities of the model could be handled quite robustly and were usually exactly enforced in the improved model, but minor regularities did not always represent the intended design. In this context we refer to major regularities as regularities which involve a large number of faces of the model and usually relate to a highly symmetric arrangement (with respect to the features). Minor regularities relate to only a few faces in the model and usually have less symmetry.

As we have to handle an approximate model we have to accept certain tolerance levels. If the tolerance level is small enough (say, at the machine precision level) such that the features of interest are sufficiently distinct, we are able to identify them precisely. For instance, if the only angle values of interest are integers in degrees, then any tolerance smaller than 0.5° allows us to exactly distinguish between the values. However, for beautifying reverse engineered models we usually have to work at tolerance levels where the features cannot be clearly distinguished anymore. Hence, we get some ambiguity with respect to the regularities. We try to eliminate some of this by looking for tolerance levels at which certain properties of the features are present unambiguously in a local sense (see Chapter 2). However, in the case of inconsistencies between these regularities, it is not always possible to make a clear decision as there are always cases of inconsistencies between regularities which are all about equally desirable. This applies in particular to minor regularities relating to a small number of cells in the boundary representation, e.g. multiple special angle values between two planar faces. This is a principal property related to approximate models, and while our methods were designed for this, the ambiguity cannot be avoided.

In general the observed tolerances at which the regularities are present in reverse engineered objects were slightly smaller than those used for the simulated data. The initial angular errors were usually in the range of 1° to 2° and the positional error was about 0.5 to 1 length units (millimetres; scanned points were about 1 millimetre apart). The general behaviour of our beautification algorithms were quite similar for real and simulated data. This is mainly due to the fact that we are working solely on the boundary representation model. The major difference between the two data sources is that for real data it is less likely to find consistent tolerance levels for the complete model. However, this was expected and allowed for in the design of the analyser methods.

The analyser was able to detect approximate regularities for which clear, unambiguous evidence was present in the initial model. It reported the regularities at tolerance levels at which there was no ambiguous interpretation of the data. Most of the intended regularities were detected by the analyser in the initial models. As no maximum tolerance value is used, and the tolerance levels for the regularities are detected automatically (see

Chapter 3), the differences in the tolerances of intended regularities could be handled. However, this also resulted in a larger number of regularities which had to be considered for selection. Trying to devise detection algorithms which only detect intended approximate regularities appears to be considerably harder. While we seek unambiguous evidence in the initial model for a regularity being present, we cannot make a decision whether the regularity is intended without having additional information about the model, such as other regularities, consistency with respect to design intent, and solvability of the related constraint system.

The decision about which regularities should be used to improve the model is made in a separate process. Major regularities could usually be identified easily by the selection process, but minor regularities, special values, etc. related to the particular instance of a model were not always selected correctly. This is directly caused by the ambiguity between inconsistent approximate regularities discussed above. In all cases, independently of the chosen priority parameters, the numerical solver was able to solve the selected constraint system up to the given numerical tolerance. This suggests that the selected constraint systems did not contain any inconsistencies. It also provides evidence that the topological solvability test is sufficient for the kind of models we considered (see Chapter 6).

Major regularities were usually present at relatively small tolerance levels and the quality priorities for them were high. This made selection of major regularities quite robust to changes in the priority parameters, and they were usually selected to improve the model. The priority order for the selection of special values and local relations between faces is more unstable and closely related to the selected priority parameters. The larger the tolerances required to detect and select the major regularities, the higher the uncertainty for minor regularities like special values. This makes it hard to determine the intended minor regularities and often special values different from the ones in the original model were selected.

It is expected that for more complicated models, the ambiguity between the regularities will increase, and selection of correct design intent will be harder. Furthermore, this may create situations where the topological solvability test is not sufficient. Thus, we may get an inconsistent constraint system, for which no solution can be found. But unless regularities at very large tolerances are accepted the regularities still relate to a model close to the initial model and usual engineering models are quite robust with respect to small changes. As we are using a numerical optimisation method for solving the constraint system, we are also able to still find parameters describing some model, which may be close to the original parameters. Thus, it is still likely that we will rebuild a valid model.

But this model may not be better than the initial model with respect to design intent.

The same parameters for regularity detection and priority computation were used for all the experiments. In all cases, adjusting the parameters to suit a particular model improved the results. While there are no regularity detection parameters which only select desired regularities, it is possible to adjust the parameters individually for each model such that the number of undesired regularities is minimised. Similarly for priority parameters, the type of selected regularities could be adjusted between values favouring quality and values favouring high accuracy. We may prefer high quality models which exhibit high symmetry (with respect to their features), but which may require a relatively large change to the initial model. Alternatively, we may try to create improved models very close to the initial model, which may be less regular, however. Furthermore, the priority computation can be adjusted to favour different regularity types. While we tried to keep the number of parameters small, there are still probably too many, and finding optimal values is non-trivial and time consuming. Using the same set of parameters for all experiments, however, was suitable to improve the models. This suggests that there is a set of parameter values for a particular reverse engineering system which can improve the models, even if the results are not optimal. Of course this requires a robust reverse engineering system for creating initial models with consistent tolerance levels.

The time required to improve a model is no more than a few minutes. This is acceptable considering the time required for the whole reverse engineering process, particularly the initial data acquisition. Most of the time spent in beautification is used in numerically solving the constraint system.

The evidence gathered so far suggests that our beautification system can be used to improve reconstructed models even if there has not yet been sufficient testing for a final evaluation. In particular, our regularity selection method creates consistent geometric constraint systems. Additional experiments would require improvements of the robustness and consistency of the reverse engineering system used to obtain the initial models in order to handle more complex models.

The tests show that major regularities, if they are only weakly related to other major regularities, can easily be identified and imposed on the model. However, ambiguities between major approximate regularities which cannot be imposed on the model at the same time (usually involving similar sets of faces) are hard to resolve. No clear decision can be made if the tolerances and the quality of such regularities do not differ distinctively. Only in cases where there is a clear major regularity relating to many faces in the object a clear decision can be made. In other cases there may be alternative models which are about as likely as the chosen one. This is caused by the current selection strategy to

select likely, intended regularities based on priorities. A more intelligent method instead of simple priorities may improve this situation. To make more consistent choices, the decision method should also consider the global structure of the model and the relations between the regularities.

We cannot guarantee to produce specific instances of a model having the desired special values for lengths and angles. In general there is always a choice between high quality regularities and relations close to those in the initial model. Note that a chosen special value is always subject to a tolerance. If the special value is within the tolerance chosen in the original design this should not cause a major problem. Our system allows the setting of various tolerances for the precision present in the initial model. Higher precision can be achieved only by creating a more exact initial model.

Finally, there may be hidden relations in the model which are broken if they are not detected explicitly as a regularity. For instance, for the model in Figure 7.3(c), the rotational angle between the directions of the cylinder axes in the conical angle-regular arrangement and the plane normals in the planar angle-regular arrangement is zero. This was not considered by our regularity detection and consequently not enforced in the improved model. To fix this, additional regularity detection methods are needed.

7.5 Summary

In this chapter we have presented various experimental tests of our beautification algorithms using real and simulated reverse engineering data. The experiments show that our algorithms are able to improve the initial reverse engineered models. But as discussed in detail in Section 7.4, approximate models contain a certain degree of ambiguity between inconsistent approximate regularities, which depends on their complexity and the involved tolerances. This means that minor regularities and special values, etc. are not always selected as originally intended. Our algorithms may also have to be expanded for more complex models with many interacting major regularities. However, in order to handle such models, the robustness of the other reverse engineering algorithms also has to be improved.

Chapter 8

Conclusions

The aim of beautification is to improve the quality of reverse engineered geometric models such that they exhibit exact intended geometric regularities. In this thesis we have presented an approach to beautification as a post-processing step which tries to improve an initial boundary representation model generated by a state-of-the-art reverse engineering system using only that model. For this we have investigated detection of approximate geometric regularities and selection of consistent regularities such that the improved model is more likely to represent the original design intent. Furthermore, the selection requires an efficient method to determine whether a constraint system describing the regularities is solvable. We also presented methods for solving geometric constraint systems numerically and rebuilding an improved model from the solution of a constraint system.

In Section 8.1 we give an overview of the issues involved in beautification as a post-processing step and evaluate our particular approach. Then we discuss the research contributions made in this thesis (see Section 8.2) and give some recommendations for future work (see Section 8.3).

8.1 Evaluation

We have split beautification into three main modules: analyser, hypothesiser and rebuilder (see Section 1.3). In the following we discuss the different issues involved in these three modules and evaluate our particular approach.

The analyser's task is to detect approximate geometric regularities in the initial model generated by the reverse engineering software. In Chapter 2 we introduced a new definition of approximate geometric regularities. It is based on approximate symmetries of discrete feature sets derived as special properties of the boundary representation elements. By linking combinatorial symmetries (distance preserving permutations of features in a metric feature space) with geometric symmetries that induce them, an effective concept

for approximate symmetries has been derived. The concept does not rely on arbitrary tolerance values, but allows efficient computation of exact answers to the question at what tolerance levels certain symmetries are present. By surveying a range of engineering parts we have confirmed that our chosen types of regularities are those commonly present in such parts. Our experiments with engineering objects have also shown that many such regularities can be detected, and that they represent the intended design of the objects. The concept is capable of being expanded to more general types of parts and different surface types. But for parts of completely different natures such as artistic objects, our approach may have to be altered substantially or a completely different regularity concept may have to be developed.

Our general concept of geometric regularities leads to various efficient regularity detection algorithms in Chapter 3. The algorithms were designed to allow for efficient detection of particular types of approximate symmetries of particular feature types. Our experiments have shown that the regularities covered by our definition were detected. Most of the time these were the same regularities as those expected by an engineer. However, additional, undesired regularities were found, and some regularities were not detected. The additional, undesired regularities can be explained by the fact that we have an approximate model. Even for relatively simple approximate models, there are ambiguous approximate regularities, and which regularities are actually part of the intended design has to be determined separately. While our methods give a robust, exact answer to the question of whether an approximate geometric regularity is present in a model, they cannot determine whether it is intended. This decision is made by the hypothesiser later in the process.

The intended regularities which we did not detect are mainly due to errors in the initial model. As our detection methods seek regularities which are present at certain tolerance levels in an exact sense, there are also other approximate regularities which cannot be determined. For instance, consider a random collection of a large number of points in a limited volume of space. With some probability this collection will contain eight points that exhibit approximate cubic symmetry (in some not exactly defined way). If these eight points are not sufficiently distinct from the other points, we are unable to detect this symmetry reliably, i.e. it is equally likely that some of the eight points should actually be matched with other points. In the case of random points the symmetry is also very unlikely to be intended. Our particular definition of approximate symmetries does not include such cases. Instead, we require that at least in a local sense the approximate regularity is present in an exact sense without equally likely alternative interpretations at a similar tolerance. Allowing a more ambiguous definition of approximate regularity would mean that the detection problem would be computationally more intense. See Chapter 2 for a detailed discussion. Overall, however, most intended major regularities were actually

detected. Only cases for which there is no clear evidence for the regularity in the model were missed.

There are certain regularities relating to general partial and incomplete symmetries for which we did not develop detection methods. The problem of general partial symmetries is that there are many ways to define partial symmetries of a feature set, and the situation is considerably more ambiguous than for general global symmetries. For incomplete symmetries, it is often hard to decide when to expand a feature set to be fully symmetric.

The task of the hypothesiser module is to select a subset of all detected approximate geometric regularities. We consider this to be a separate issue from approximate regularity detection. The selected subset should be likely to represent the original design intent, and the related constraint system has to be solvable. Thus, we split this module into a constraint selection subtask responsible for selecting likely regularities, and a constraint solvability subtask responsible for testing the solvability of the constraint system. Both subtasks cannot be completely separated from each other as selection depends on intent and solvability. A selection representing a likely design intent which is not solvable is as undesirable as a solvable constraint system which does not represent a likely design intent.

Our general selection strategy in Chapter 4 considers geometric regularities sequentially in order of a priority. It checks in sequence whether adding a regularity to the set of selected regularities would create an inconsistency in the related constraint system. Each priority indicates how likely the regularity is to be part of the original design, based on the accuracy to which it is present in the initial model, and its general desirability with respect to the regularity type and the associated elements of the model. While we do not need to tune the parameters for computing the priorities for each model individually to achieve an improvement to a model, it is not a simple task to find appropriate generally applicable parameter values. Furthermore, to achieve an optimal improvement, the parameters have to be selected carefully for each individual model.

Our experiments show that our selection strategy works reasonably well for simple to medium complexity objects. Major regularities are selected correctly. Minor regularities and relations between individual faces, etc. are less reliably selected. The main reason for this appears to be the fact that we consider the regularities individually. Each regularity is assigned a priority independently of the other regularities. Major regularities relating to many parts in the object are clearly likely to be given high priority values and they are quite likely to be inconsistent with other major regularities. Often, if a given major regularity is selected, all other major regularities cannot be selected. Regularities relating to a small number of elements of the model are harder to handle in this context. We

ought to consider which other regularities have been selected and if the combination is desirable. Sometimes additional information about the structure of the model should also be examined so that we are able to consider the impact of selecting a particular regularity on the whole model. For instance, sometimes an angle between two planes may simply determine that angle without further implications on the model. In other cases it may determine the relative orientation between two symmetrical sub-parts of the object, which can also be determined by other angles (see example model (b) in Figure 7.3). Either having a reason for choosing a particular angle or considering all relevant angles to find the relative orientation between the two sub-parts may improve the selection.

In the case of multiple major regularities, which can be selected simultaneously, it is likely that they are all selected due to having high priorities. However, such regularities usually appear in larger objects, and this makes it likely that we will also detect a lot of additional undesired regularities. These are either caused by relations between the desired major regularities or by relations between individual faces. The larger the model, the harder it is to make a clear decision without considering additional information like the topology of the model. For this reason we limited ourselves to models having only a few independent simultaneous major regularities. Dependent major regularities cause additional problems similar to those caused by additional minor regularities — it is no longer clear which combination of major regularities is actually desirable.

In order to improve the selection of regularities more intelligent methods are required. The selection of regularities has to be considered as a whole, not one by one. It may also be required to consider alternative consistent regularity sets until one of them becomes more likely. As we can only test for solvability consecutively by adding constraints to a constraint system, the regularity sets have to be built sequentially, but it may be possible to avoid considering all combinations using heuristics, geometric reasoning and design intent knowledge.

The other part of the hypothesiser is the solvability test for the constraint system as discussed in Chapters 5 and 6. We considered both a numerical and a topology-based solvability test. As the numerical test has to solve a constraint system numerically for each added regularity, it is clearly too expensive. It was a simple first approach used to test the general system. The solution provided, however, is also still useful for rebuilding the model.

The topological solvability test developed is based on degrees-of-freedom analysis. Rather than arguing in general about degrees of freedom, we have given such an approach a more precise topological background. This has identified some of the problems with degrees-of-freedom analysis and its limits. In particular, without solving any equation systems, we

have to assume that the intersections between the sub-manifolds of the geometry manifolds are always generic. This worked well for our models and the final constraint systems could always be solved. As this solvability test does not have to solve any equation system, and uses an efficient distribution algorithm operating on the constraint graph, it is considerably faster than the numerical approach.

However, we do not yet have a *proof* that the solvability test actually works, even though it seems to in practice. Many questions relating to handling non-generic cases are still open. Nevertheless, it provides the framework to study these cases in more detail. In order to handle non-generic cases and more general constraint types, an algebraic component which can solve certain equation systems cannot be avoided, however.

The final step consists of rebuilding the model from the selected constraint system. If the topological solvability test has been used, we still have to compute a solution to the constraint system. This takes most of the time used by the beautification process, and is done using the numerical constraint solver. A more sophisticated implementation of the optimisation algorithms which makes, for instance, extensive use of BLAS (Basic Linear Algebra Subprograms) [146] and LAPACK (Linear Algebra PACKage) [147] functions optimised for the particular computer architecture may increase the speed of the solver. For further improvement, the numerical solver could be replaced by one which is more closely linked to the structure detected by the topological solvability test. Decomposition-recombination planners as briefly presented in Section 5.4 could be used for this.

As discussed in more detail in Section 7.4, our experiments have shown that the system can improve simple to medium complexity objects. Using our algorithms, weakly dependent major regularities can be identified and imposed correctly on the model. The regularities are always selected in such a way that the corresponding constraint system is consistent in generic terms. Experiments show that generic solvability is sufficient to find solutions to the constraint system which can be employed to successfully improve the model. More complex models with many separate major regularities as well as models with many dependent regularities require a more sophisticated approach. Nevertheless, it may be possible to expand our concepts and methods for this.

Adding the beautification step as a post-processing step instead of integrating it with the other reverse engineering steps did not cause any serious problems. Our regularity detection method created sufficient regularities to specify the complete improved model in terms of geometric constraints. Thus, the improved model could be created simply by solving a constraint system. Whenever insufficient regularities are present, the condition that the solution should be close to the initial model should suffice to avoid large, undesired modifications. The property of the optimisation solver converging towards a

solution close to the initial value has been observed experimentally and appears to be a general property of this type of numerical optimisation. However, an explicit proof or simple condition to ensure that we always stay close to the original model would be desirable.

8.2 Contributions

In this thesis we have presented an approach to automatically improve reverse engineered geometric models with respect to design intent, in a post-processing step working solely with the boundary representation model. To our knowledge this is the first time this particular approach has been investigated. Alternative approaches focus on constrained fitting algorithms where information about the design intent is obtained from the user. In the following we present the contributions to the solution of this problem made in this thesis.

We have developed a theoretical framework for a certain type of approximate geometric regularities. Features were introduced as properties of boundary representation cells, which change in a similar way to the cell under isometric transformations. Symmetries of these features can be used to describe various common geometric regularities. As features can be handled as discrete points in a metric space we were able to combine their combinatorial and geometric properties to devise a novel concept of approximate symmetries to describe approximate regularities. Compared to other approaches to approximate symmetries, our concept allows for their efficient and robust detection. Various sub-types of geometric regularities in terms of which symmetries are involved could be derived, which created a basis for devising different detection algorithms. In general our concept is a strong basis for investigating methods to handle approximate geometric regularities.

For the detection of approximate regularities we considered algorithms for approximate congruences, repetitions, global symmetries and special values. For approximate congruences it was sufficient to modify an existing hierarchical clustering algorithm. As partial and incomplete symmetries are hard to detect we limited ourselves to common repetition regularities. This resulted in a new algorithm for detection of approximate repetitions which can be used for many regularity types. We also presented a novel, efficient algorithm for the detection of approximate global symmetries. All these methods use our general approximate regularity concept to automatically determine tolerance levels at which certain symmetries are present. We also presented an approach to detect special values for a given scalar parameter in terms of simple fractions. It can be interpreted as an expansion of the continued fraction approach where we try to find a few simple fractions rather than a fraction very close to the scalar.

As the regularities are only approximately present in the reverse engineered model it is likely that they are not mutually consistent. Therefore we investigated ways to select an appropriate set of regularities. Selection during detection is hard as we do not have all regularities in order to consider the different options nor are we able to tell if a constraint system describing the regularities is actually solvable. Hence, we introduced a selection step to determine a set of regularities which is likely to represent the original design intent. Moreover, the related constraint system has to be solvable such that an improved model can be created solely from the solution of the constraint system.

To select likely regularities we investigated the use of a sequential selection strategy based on priorities. In experiments we showed that this strategy works for simple to medium complexity models. It is able to select the intended major regularities of these models such that a real improvement of the models with respect to design intent is possible. However, the experiments also showed that more sophisticated strategies are required to handle minor regularities and more complex models which are likely to exhibit more ambiguous regularities.

To ensure that the constraint system for the selected regularities is consistent, we devised a solvability test based on the topological properties of the domains of the geometric elements. The basic approach is quite similar to degrees-of-freedom analysis, but the topological interpretation links the actual equations or constraints closer to the constraint graph. The assumptions made for degrees-of-freedom analysis become more clear, and the types of non-generic cases, which cannot be handled properly, can easily be identified. Our experiments suggest that the method is indeed able to determine generic solvability of constraint systems and that this is sufficient to beautify the types of models considered in this thesis. We expect that the general approach can be used to improve geometric constraint solving techniques.

Overall we presented a framework for beautification of reverse engineered objects with respect to design intent as a post-processing step. There is some evidence that beautification can be used to improve reverse engineered models. Our regularity detection and selection algorithms are suitable for simple to medium complexity objects and provide a strong basis for advanced beautification methods for more complex cases.

8.3 Future Work

Based on the results of this thesis we recommend the following topics for future work.

We have presented some algorithms for the detection of geometric regularities based on

approximate symmetries. We deliberately did not devise algorithms to detect general partial and incomplete symmetries due to the high computational costs involved. It is, however, desirable to have such methods in order to detect further approximate regularities in the models. For partial symmetries, an appropriate notion of interesting subsets which should be considered is required in order to avoid checking all possible (or a very large number of) subsets. This could either be done using heuristics to only consider interesting parts of the object, using its topology and other structural information, or by considering a hierarchy of subsets and ways to expand small symmetric subsets to larger symmetric subsets. Furthermore, it may be interesting to expand the detection method to handle more general surface types with different properties. In particular, regularities of free-form surfaces are of interest, e.g. nice reflection lines or minimal surfaces, as well as special directional, positional or shape features which can be derived from the free-form surface representation. It is unknown at this time how far artistic regularities or simply the human notion of a beautiful shape could be included, too.

Furthermore, methods to handle regularity detection in more complex models are required. Algorithms for partial symmetry detection address this from the viewpoint of finding symmetric subsets, i.e. determining subsets in the model such that it may be decomposed based on the detected regularity. Another approach would be to decompose the model first into sub-parts and then detect regularities of these sub-parts. This could be based on machining feature detection adjusted for approximate models and focused on design intent rather than machining. The sub-parts could then be improved independently. However, further methods are required to combine the sub-parts properly, which includes detection of appropriate regularity types and selection strategies.

Our current beautification algorithms almost certainly could be improved by investigating more sophisticated selection strategies. This may in particular improve the handling of minor regularities and ambiguities between many major regularities in complex models. Rather than considering each regularity individually, different combinations of regularities need to be considered. This could yield a globally more consistent regularity selection with respect to design intent. One approach would be to use belief networks or probability distributions and treat the detected regularities with their tolerances, etc. as evidence to support certain hypotheses about the design intent. Alternatively, more advanced artificial intelligence techniques may be employed, such as neuro-fuzzy networks which combine the ability of neural networks to learn with the explicit rule representation of fuzzy systems. Alternatively, incorporating evolutionary techniques and the explicit representation of design knowledge in terms of rules may also be of use.

We have investigated a topological interpretation of geometric constraints in the context

of degrees-of-freedom analysis. This may provide a basis for a better understanding of degrees-of-freedom analysis and consequently lead to improved algorithms. It may in particular be suitable to expand existing algorithms to consider more than just the topological dimension of the manifolds (i.e. their topological types) and to handle non-generic cases. This cannot be done solely by arguing about the topological dimensions and spaces involved, but requires the consideration of algebraic aspects. Yet solving simple equation systems may be sufficient. It may also be suitable to study inequality constraints. Many types of inequality constraints may be interpreted as limiting the allowed values of the geometric elements involved to a sub-space. This sub-space has the same topological dimension than the parameter space used to describe the geometric element. But the boundary of the sub-space forms a proper sub-manifold of lower dimension. For instance, an inequality constraint on the distance between two points limits one point to lie in a ball around the other point with an $\mathbb{S}^2$ boundary.

Rather than working solely on the boundary representation, it may also be of interest to use the regularity detection and selection algorithms in combination with a constrained fitting algorithm. Currently these methods require the constraints to be specified manually. But after building an initial model using an unconstrained fitting algorithm, regularities could be detected and selected for a constrained fitting step. In particular the selection has to be adapted here to focus on major regularities and avoid specifying the model completely. Otherwise, the model would already be completely determined by the constraints and the point set could not be considered anymore. This may in particular help to resolve some of the ambiguity problems related to selecting minor regularities such as certain special values as these could still be determined directly from the point set rather than from constraints. But it may also take considerably more time to compute than our approach.

In this thesis we also did not consider any issues relating to the topology of the model. One topological issue is to repair a broken topology of a model, e.g. close holes in the boundary description of the model. Another issue is to actually beautify the topology, i.e. change the topology of a model such that it represents the original, intended topology. This may, for instance, mean to detect that a set of faces should intersect in a single vertex, rather than in multiple vertices with short edges between them, which may even create an additional small undesired face (e.g. at the top of a pyramid). An approach which tries to correct the topology of the initial model before it is beautified by the presented system is considered in [40]. However, at this stage it is hard to determine whether the suggested topology can be realised. Often, this is only possible by checking it with our solvability test during the regularity selection step, and we have to restart the process with a different topology. To some extent this problem can be avoided by making only relatively simple topological modifications which are unlikely to cause problems later. Changes in the topology require

to change the geometric elements in the constraint system and thus alternative topologies are harder to handle for the solvability test during regularity selection. But our approach may be expanded for this. Furthermore, a notion of regularities suggesting topological changes and suitable detection and selection methods are required.

Finally, we note that our methods are not necessarily limited to reverse engineering applications. They may in general be useful for CAD applications which involve the design intent of approximate geometric models. Such approximate models may be generated from reverse engineering systems, sketch input systems or other imprecise user input, e.g. in three-dimensional virtual environments. Applications which detect automatically that models have certain intended regularities are able to provide more robust model modification operations and ensure that intended regularities are not broken. Furthermore, approximate regularities can be enforced exactly on the model and thus the application can ensure that they are present for analysis and manufacturing of the model. The ability to automatically detect the design intent in approximate geometric models may allow for user interfaces on a higher abstraction level simplifying the use of such systems.

GNU Free Documentation License

Version 1.2, November 2002

0. Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You

accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the

text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

you may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties — for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © `YEAR YOUR NAME`. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with…Texts." line with this:

> with the Invariant Sections being `LIST THEIR TITLES`, with the Front-Cover Texts being `LIST`, and with the Back-Cover Texts being `LIST`.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliography

[1] P. K. Agarwal, C. M. Procopiuc. Exact and approximation algorithms for clustering. In: *Proc. 9th ACM/SIAM Symp. Discrete Algorithms*, pp. 658–667, 1998.

[2] R. Agarwala, V. Bafna, M. Farach, B. Narayanan. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Computing*, 28(3):1073–1085, 1999.

[3] E. L. Allgower, K. Georg. *Numerical Continuation Methods, an Introduction*. Springer, Berlin, Heidelberg, New York, 1990.

[4] E. L. Allgower, K. Georg. Continuation and path following. *Acta Numerica*, Cambridge University Press, Cambridge, pp. 1–64, 1993.

[5] H. Alt, K. Mehlhorn, H. Wagner, E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete and Computational Geometry*, 3:237–256, 1988.

[6] C. Armstong, A. Bowyer, S. Cameron, J. Corney, G. Jared, R. R. Martin, A. Middleditch, M. Sabin, J. Salmon. *Djinn — A Geometric Interface for Solid Modelling, Specification and Report*. Information Geometers Ltd., Winchester, UK, 2000.

[7] D. H. Bailey, S. Plouffe. Recognizing numerical constants. In: *Proc. Organic Mathematics Project*, Canadian Mathematical Society, Vol. 20, pp. 73–88, 1997.

[8] G. Barequet, C. A. Duncan, S. Kumar. RSVP: a geometric toolkit for controlled repair of solid models. *IEEE Trans. Visualization and Computer Graphics*, 4(2):162–177, 1998.

[9] P. Benkő, G. Kós, T. Várady, L. Andor, R. R. Martin. Constrained fitting in reverse engineering. *Computer-Aided Geometric Design*, 19(3):173–205, 2002.

[10] P. Benkő, R. R. Martin, T. Várady. Algorithms for reverse engineering boundary representation models. *Computer-Aided Design*, 33(11):839–851, 2001.

[11] S. Berchtold, D. A. Keim, H.-P. Kriegel. Using extended feature objects for partial similarity retrieval. *The VLDB Journal*, 6:333–348, 1997.

[12] P. J. Besl, N. D. McKay. A method for registration of 3–D shapes. *IEEE Pattern Analysis and Machine Intelligence*, 14:239–256, 1992.

[13] Å. Björk. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[14] A. H. Borning. The programming language aspects of thinglab, a constraint oriented simulation laboratory. *ACM Trans. Programming Languages and Systems*, 3(4):353–387, 1981.

[15] A. Borodin, R. Ostrovsky, Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. In: *Proc. 31st Ann. ACM Symp. Theory of Computing*, pp. 435–444, 1999.

[16] P. Brass. On finding maximum-cardinality symmetric subsets. *Computational Geometry: Theory and Applications*, 24:19–25, 2002.

[17] B. Brüderlin, D. Roller. *Geometric Constraint Solving and Applications*. Springer, Berlin, Heidelberg, New York, 1998.

[18] B. Brüderlin. *Rule-Based Geometric Modeling*. PhD thesis, Swiss Federal Institute of Technology, 1987.

[19] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal*. PhD thesis, Institute of Mathematics, University of Innsbruck, Austria, 1965.

[20] B. Buchberger. Gröbner bases: an algorithmic method in polynomial ideal theory. In: N. K. Bose, D. Reidel (eds.), *Recent Trends in Multidimensional Systems Theory*, D. Reidel Publishing Company, Dordrecht, Chapter 6, pp. 184–232, 1985.

[21] J. Canny, I. Emiris. An efficient algorithm for the sparse mixed resultant. In: G. Cohen, T. Mora, O. Moreno (eds.), *Proc. 10th Int. Symp. Applied Algebra, Algebraic Algorithms, and Error Correcting Codes*, Springer, Berlin, Heidelberg, New York, pp. 89–104, 1993.

[22] S. C. Cho. An introduction to Wu's method for mechanical theorem proving in geometry. *J. Automated Reasoning*, 4:237–267, 1988.

[23] R. M. Corless. Continued fractions and chaos. In: *Proc. Organic Mathematics Workshop*, Canadian Mathematical Society, 1997. `http://www.cecm.sfu.ca/organics/papers/corless/` [last accessed 18. March, 2003].

[24] D. Cox, J. Little, D. O'Shea. *Ideals, Varieties and Algorithms*. Springer, Berlin, Heidelberg, New York, 1992.

[25] G. Crippen, T. Havel. *Distance Geometry and Molecular Conformation*. John Wiley & Sons, New York, 1988.

[26] R. Dautray, J.-L. Lions. *Mathematical Analysis and Numerical Methods for Science and Technology, Functional and Variational Methods*, Vol. 2. Springer, Berlin, Heidelberg, New York, 1988.

[27] M. de Berg, M. van Kreveld, M. Overmans, O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, Berlin, Heidelberg, New York, 1997.

[28] J. E. Dennis, R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[29] H. Desaulniers, N. F. Stewart. An extension of manifold boundary representations of the r-sets. *ACM Trans. Graphics*, 11(1):40–60, 1992.

[30] J.-F. Dufourd. An OBJ3 functional specification for boundary representation. In: *Proc. Symp. Solid Modeling Foundations and CAD/CAM Applications*, pp. 61–72, 1991.

[31] C. Durand. *Symbolic and numerical techniques for constraint solving*. PhD thesis, Department of Computer Science, Purdue University, 1998.

[32] C. Durand, C. M. Hoffmann. Variational constraints in 3D. In: *Proc. Int. Conf. Shape Modeling and Applications*, pp. 90–97, 1999.

[33] C. Durand, C. M. Hoffmann. A systematic framework for solving geometric constraints analytically. *J. Symbolic Computation*, 30:493–520, 2000.

[34] D. W. Eggert, A. W. Fitzgibbon, R. B. Fisher. Simultaneous registration of multiple range views for use in reverse engineering of CAD models. *Computer Vision and Image Understanding*, 69(3):253–272, 1998.

[35] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *J. Experimental Algorithmics*, 5(1):1–23, 2000.

[36] H. R. P. Ferguson, D. H. Bailey. A polynomial time, numerically stable integer relation algorithm. Technical Report RNR–91–032, NASA Ames Research Center, MS T045-1, Moffett Field, CA 94035-1000, USA, December 1991.

[37] L. D. Floriani, B. Falcidieno. A hierarchical boundary model for solid object representation. *ACM Trans. Graphics*, 7(1):42–60, 1988.

[38] I. Fudos, C. M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Trans. Graphics*, 16(2):179–216, 1997.

[39] C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate congruence detection of model features for reverse engineering. In: M.-S. Kim (ed.), *Proc. Int. Conf. Shape Modelling and Applications*, IEEE Computer Society, Los Alamitos, CA, pp. 69–77, 2003.

[40] C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Local topology beautification for reverse engineered models. *Computer-Aided Design*, 2003. Submitted.

[41] X.-S. Gao, S.-C. Chou. Solving geometric constraint systems (I. A global propagation approach). *Computer-Aided Design*, 30(1):47–54, 1998.

[42] J.-X. Ge, S.-C. Chou, X.-S. Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31:867–879, 1999.

[43] I. M. Gelfand. *Discriminants, Resultants, and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[44] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, New York, Singapore, Sydney, 1989.

[45] A. Gomes, A. Middleditch, C. Reade. A mathematical model for boundary representations of n–dimensional geometric objects. In: *Proc. 5th ACM Symp. Solid Modeling and Applications*, pp. 270–277, 1999.

[46] J. E. Goodman, J. O'Rourke (eds.). *Handbook of Discrete and Computational Geometry*. CRC Press, Boca Raton, FL, 1997.

[47] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[48] F. Hausdorff. Dimension und äusseres Maß. *Mathematische Annalen*, 79:157–179, 1919.

[49] C. M. Hoffmann, A. Lomonosov, M. Sitharam. Planning geometric constraint decompositions via graph transformations. In: M. Nagl, A. Schürr, M. Münch (eds.), *Proc. AGTIVE'99 Applications of Graph Transformations with Industrial Relevance*, Springer, Berlin, Heidelberg, New York, pp. 309–324, 1999.

[50] C. M. Hoffmann. *Geometric and Solid Modeling, an Introduction*. Morgan Kaufmann, San Mateo, CA, 1989.

[51] C. M. Hoffmann, A. Lomonosov, M. Sitharam. Decomposition plans for geometric constraint systems, part I: performance measures for CAD. *J. Symbolic Computation*, 31(4):367–408, 2001.

[52] C. M. Hoffmann, A. Lomonosov, M. Sitharam. Decomposition plans for geometric constraint systems, part II: new algorithms. *J. Symbolic Computation*, 31(4):409–427, 2001.

[53] C. M. Hoffmann, J. Rossignac. A road map to solid modeling. *IEEE Trans. Visualization and Computer Graphics*, 2:3–10, 1996.

[54] C. M. Hoffmann, P. J. Vermeer. Geometric constraint solving in $\mathbb{R}^2$ and $\mathbb{R}^3$. In: D. Z. Zu, F. Hwang (eds.), *Computing in Euclidean Geometry*, 2nd edition. World Scientific Publishing, Singapore, 1994.

[55] S. Iwanowski. Testing approximate symmetry in the plane is NP-hard. *Theoretical Computer Science*, 80:227–262, 1991.

[56] P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200(1-2):185–204, 1998.

[57] P. Jeavons, D. Cohen, J. Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence*, 24(1–4):51–67, 1998.

[58] X. Y. Jiang, H. Bunke. A simple and efficient algorithm for determining the symmetries of polyhedra. *GVGIP: Graphical Models And Image Processing*, 54(1):91–95, 1992.

[59] R. Joan-Arinyo, A. Soto. A correct rule-based geometric constraint solver. *Computer and Graphics*, 21(5):599–609, 1997.

[60] R. Joan-Arinyo, A. Soto. Combining constructive and equational geometric constraint-solving techniques. *ACM Trans. Graphics*, 18(1):35–55, 1999.

[61] M. Kalkbrener. A generalized Euclidean algorithm for computer triangular representations of algebraic varieties. *J. Symbolic Computation*, 15:143–167, 1993.

[62] C. T. Kelley. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999.

[63] A. Y. Khinchin. *Continued Fractions*. Dover Publications, New York, 1997.

[64] F. Klein. *Vergleichende Betrachtungen über neuere geometrische Forschungen* (das Erlanger Programm, 1872). Mathematische Annalen, 43(1):63–100, 1893.

[65] K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design*, 24(3):141–147, 1992.

[66] G. Kós. An algorithm to triangulate surfaces in 3D using unorganised point clouds. *SIAM J. Computing*, Suppl. 14:219–232, 2001.

[67] G. Kós, R. R. Martin, T. Várady. Methods to recover constant radius rolling ball blends in reverse engineering. *Computer-Aided Geometric Design*, 17(2):127–160, 1999.

[68] G. A. Kramer. *Solving Geometric Constraint Systems: a Case Study in Kinematics*. MIT Press, Cambridge, MA, 1992.

[69] G. A. Kramer. A geometric constraint engine. In: *Constraint-Based Reasoning*, MIT Press, Cambridge, MA, pp. 327–360, 1994.

[70] P. Kucera. Registrace prostorovych ploch. Diploma thesis, Faculty of Electrical Engineering, Czech Technical University, 1996.

[71] H. Lamure, D. Michelucci. Solving geometric constraints by homotopy. In: *3rd Symp. Solid Modeling and Applications*, ACM, New York, pp. 263–269, 1995.

[72] F. C. Langbein, A. D. Marshall, R. R. Martin. Numerical methods for beautification of reverse engineered geometric models. In: *Proc. Geometric Modeling and Processing*, IEEE Computer Society, Los Alamitos, CA, pp. 159–168, 2002.

[73] F. C. Langbein, A. D. Marshall, R. R. Martin. Choosing consistent constraints for beautification of reverse engineered geometric models. *Computer-Aided Design*, 2003. To appear.

[74] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Approximate geometric regularities. *Int. J. Shape Modeling*, 7(2):129–162, 2001.

[75] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Finding approximate shape regularities for reverse engineering. *J. Computing and Information Science in Engineering*, 1(4):282–290, 2001.

[76] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Finding approximate shape regularities in reverse engineered solid models bounded by simple surfaces. In: D. C. Anderson, K. Lee (eds.), *Proc. 6th ACM Symp. Solid Modelling and Applications*, ACM, New York, pp. 206–215, 2001.

[77] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Recognizing geometric patterns for beautification of reconstructed solid models. In: *Proc. Int. Conf. Shape Modelling and Applications*, IEEE Computer Society, Los Alamitos, CA, pp. 10–19, 2001.

[78] R. S. Latham. *Combinatorial Algorithms for the Analysis and Satisfaction of Geometric Constraints*. PhD thesis, Department of Computer Science, Brunel University, December 1996.

[79] R. S. Latham, A. E. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer-Aided Design*, 28(11):917–928, 1996.

[80] D. Lazard. A new method for solving algebraic systems of positive dimension. *Discrete Applied Mathematics*, 33(1-3):147–160, 1991.

[81] D. Lazard. On theories of triangular sets. *J. Symbolic Computation*, 28:105–124, 1999.

[82] W. Leler. *Constraint Programming Languages*. Addison Wesley, New York, Amsterdam, Singapore, 1988.

[83] D.-H. Li, M. Fukushima. A modified BFGS method and its global convergence in nonconvex minimization. *J. Computational and Applied Mathematics*, 129(1–2):15–35, 2001.

[84] Y. T. Li. Numerical solutions of multivariate polynomial systems by homotopy continuation methods. *Acta Numerica*, 6:399–436, 1997.

[85] Y.-T. Li, S.-M. Hu, J.-G. Sun. On the numerical redundancies of geometric constraint systems. In: *Proc. Pacific Graphics*, pp. 118–123, 2001.

[86] Y.-T. Li, S.-M. Hu, J.-G. Sun. A constructive approach to solving 3D geometric constraint systems using dependence analysis. *Computer-Aided Design*, 34(2):97–108, 2002.

[87] E. H. Lockwood, R. H. Macmillan. *Geometric Symmetry*. Cambridge University Press, Cambridge, 1978.

[88] E. H. Lockwood, R. H. Macmillan. Geometric symmetry. *Mathematical Intelligence*, 6(3):63–67, 1984.

[89] G. Lukács, R. R. Martin, A. D. Marshall. Faithful least–squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In: H. Budkhadj, B. Neumann (eds.), *Proc. 5th European Conf. Computer Vision*, Springer, Berlin, Heidelberg, New York, Vol. 1, pp. 671–686, 1998.

[90] L. Luksan, E. Spedicato. Variable metric methods for unconstrained optimization and nonlinear least squares. *J. Computational and Applied Mathematics*, 124:61–95, 2000.

[91] R. R. Martin, D. Dutta. Tools for asymmetry rectification in shape design. *Systems Engineering*, 6:98–112, 1996.

[92] J. M. Martínez. Practical quasi-Newton methods for solving nonlinear systems. *J. Computational and Applied Mathematics*, 124:97–121, 2000.

[93] B. I. Mills, F. C. Langbein. Determination of approximate symmetry in geometric models — an exact approach. *Computational Geometry: Theory and Applications*, 2002. Submitted.

[94] B. I. Mills, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate symmetry detection for reverse engineering. In: D. C. Anderson, K. Lee (eds.), *Proc. 6th ACM Symp. Solid Modelling and Applications*, ACM, New York, pp. 241–248, 2001.

[95] B. I. Mills, F. C. Langbein, A. D. Marshall, R. R. Martin. Estimate of frequencies of geometric regularities for use in reverse engineering of simple mechanical components. Technical Report GVG 2001-1, Geometry and Vision Group, Department of Computer Science, Cardiff University, 2001. `http://ralph.cs.cf.ac.uk/papers/Geometry/survey.pdf`. [last accessed 18. March, 2003].

[96] A. Morgan. *Solving Polynomial Systems using Continuation for Engineering and Scientific Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[97] G. Mullineux. Constraint resolution using optimisation techniques. *Computers & Graphics*, 25(3):483–492, 2001.

[98] M. J. Muuss. Understanding the preparation and analysis of solid models. Technical report, Advanced Computer Systems Team, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, Maryland 21005–5066, USA, May 1995.

[99] J. A. Nelder, R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[100] L. I. Nicolaescu. *Lectures on the Geometry of Manifolds*. World Scientific, Singapore, London, Hong Kong, 1996.

[101] J. Ortega, W. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.

[102] J. Owen. Algebraic solution for geometry from dimensional constraints. In: *ACM Symp. Found. Solid Modeling*, ACM, New York, pp. 397–407, 1991.

[103] J. Owen. Constraints on simple geometry in two and three dimensions. *Int. J. Computation Geometry and Applications*, 6:321–434, 1996.

[104] C. Park, Y. C. Chung. A tolerant approach to reconstruct topology from unorganized trimmed surfaces. *Computer-Aided Design*, 27(4):411–425, 2003.

[105] H. O. Peitgen, P. H. Richter. *The Beauty of Fractals, Images of Complex Dynamical Systems*. Springer, Berlin, Heidelberg, New York, 1986.

[106] H. Pottmann, M. Peternell, B. Ravani. An introduction to line geometry with applications. *Computer-Aided Design*, 31(1):3–16, 1999.

[107] W. Press et al. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 1988.

[108] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, part I. *J. Symbolic Computation*, 13(3):255–299, 1992.

[109] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, part II. *J. Symbolic Computation*, 13(3):301–327, 1992.

[110] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, part III. *J. Symbolic Computation*, 13(3):329–352, 1992.

[111] J. F. Ritt. *Differential Equations from the Algebraic Standpoint*. AMS, Providence, Rhode Island, 1932.

[112] J. F. Ritt. *Differential Algebra*. AMS, Providence, Rhode Island, 1950.

[113] J. M. Rojas. Solving degenerate sparse polynomial systems faster. *J. Symbolic Computation*, 28(1–2):155–186, 1999.

[114] O. E. Ruiz, P. M. Ferreira. Algebraic geometry and group theory in geometric constraint satisfaction. In: *Proc. Int. Symp. Symbolic and Algebraic Computation*, New York, pp. 224-233, 1994.

[115] S. Rusinkiewicz, M. Levoy. Efficient Variants of the ICP Algorithm. In: *Proc. 3rd Int. Conf. 3D Digital Imaging and Modeling*, 145–152, 2001.

[116] N. M. Samuel, A. A. G. Requicha, S. A. Elkind. Methodology and results of an industrial part survey. Technical Report TM–21, Production and Automation Project, College of Engineering & Applied Science, The University of Rochester, July 1976.

[117] C. Scheiner. *Pantographice: Seu Ars Delineandi Res Quaslibet Per Parallelogrammum Lineare Seu Cavum, Mechanicum, Mobile*. Ex typographia Ludovici Grignani, Romae, 1631.

[118] W. J. Schroeder, J. A. Zarge, W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26:65–70, 1992.

[119] T. Seidl, H.-P. Kriegel. Optimal multi-step k-nearest neighbour search. In: L. Haas, A. Tiwary (eds.), *Proc. ACM SIGMOD Conf.*, pp. 154–165, 1998.

[120] M. Shiota. *Geometry of subanalytic and semialgebraic sets*. Birkhäuser, Boston, 1997.

[121] M. Sitharam, C. Hoffmann, A. Lomonosov. Finding dense subgraphs of constraint graphs. In: G. Smolka (ed.), *Constraint Programming*, Springer, Berlin, Heidelberg, New York, pp. 463–478, 1997.

[122] P. Spellucci. *Numerische Verfahren der nichtlinearen Optimierung*. Birkhäuser, Basel, Boston, Berlin, 1993.

[123] N. Sridhar, R. Agrawal, G. L. Kinzel. Active occurrence matrix based approach to design decomposition. *Computer-Aided Design*, 25:400–512, 1993.

[124] N. Sridhar, R. Agrawal, G. L. Kinzel. Algorithms for the structural diagnosis and decomposition of sparse, underconstrained design systems. *Computer-Aided Design*, 28(4):237–249, 1996.

[125] B. Sturmfels. Sparse elimination theory. In: *Proc. Computational Algebraic Geometry and Commutative Algebra*, Cambridge University Press, Cambridge, pp. 377–396, 1993.

[126] K. Sugihara. An $n \log n$ algorithm for determining the congruity of polyhedra. *J. Computer and System Science*, 29:36–47, 1984.

[127] S. Tate. *Symmetry and Shape Analysis for Assembly-Oriented CAD*. PhD thesis, School of Industrial and Manufacturing Science, Cranfield University, 2000.

[128] W. B. Thompson, J. C. Owen, J. de St. Germain, S. R. Stark, T. C. Henderson. Feature–based reverse engineering of mechanical parts. *IEEE Trans. Robotics and Automation*, 15(1):57–66, 1999.

[129] T. Várady, R. R. Martin. Reverse engineering. In: G. Farin, J. Hoschek, M.-S. Kim (eds.), *The Handbook of Computer-Aided Geometric Design*, Chapter 26, Elsevier, Amsterdam, pp. 651–681, 2002.

[130] T. Várady, R. R. Martin, J. Cox. Reverse engineering of geometric models — an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[131] A. Verroust, F. Schonek, D. Roller. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24(10):531–540, 1992.

[132] D. Wang. Decomposing polynomial systems into simple systems. *J. Symbolic Computation*, 25:295–314, 1998.

[133] N. Werghi, R. B. Fisher, C. Robertson, A. Ashbrook. Object reconstruction by incorporating geometric constraints in reverse engineering. *Computer-Aided Design*, 31(6):363–399, 1999.

[134] N. Werghi, R. B. Fisher, A. Ashbrook, C. Robertson. Faithful recovering of quadric surfaces from 3D range data. In: *Proc. 2nd Int. Conf. 3–D Digital Imaging and Modelling*, IEEE Computer Society, Los Alamitos, CA, pp. 280–289, 1999.

[135] N. Werghi, R. B. Fisher, C. Robertson, A. Ashbrook. Faithful recovering of quadric surfaces from 3D range data by global fitting. *Int. J. Shape Modelling*, 6(1):65–78, 2000.

[136] N. Werghi, R. B. Fisher, C. Robertson, A. Ashbrook. Shape reconstruction incorporating multiple non-linear geometric constraints. *Constraints*, 7(2):117–149, 2002.

[137] J. Wolter, T. Woo, R. Volz. Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer*, 1:37–48, 1985.

[138] W. Wu. Basic principles of mechanical theorem proving in elementary geometries. *J. Automated Reasoning*, 2:221–252, 1986.

[139] W. Wu. *Mechanical Theorem Proving in Geometries: Basic Principles*. Springer, Berlin, Heidelberg, New York, 1994.

[140] M. Xiaofang, F. R. Y. Kit, X. Chengxian. Stable factorized quasi-Newton methods for nonlinear least-squares problems. *J. Computational and Applied Mathematics*, 129(1–2):1–14, 2001.

[141] I. M. Yaglom. *Felix Klein and Sophus Lie: Evolution of the Idea of Symmetry in the Nineteenth Century*. Birkhäuser, Boston, MA, 1988.

[142] P. N. Yianilos. Data structures and algorithms for nearest neighbour search in general metric spaces. In: *Proc. 4th ACM/SIAM Symp. Discrete Algorithms*, ACM, New York, pp. 311–312, 1993.

[143] H. Zabrodski, D. Avnir. Measuring symmetry in structural chemistry. *Advances in Molecular Structure Research*, 1:1–31, 1995.

[144] J. Zhan, C. Xu. Properties and numerical performance of quasi-Newton methods with modified quasi-Newton equations. *J. Computational and Applied Mathematics*, 137(2):269–278, 2001.

[145] J. Zupan. *Clustering of Large Data Sets*. Research Studies Press, Letchworth, Hertfordshire, England, 1982.

[146] *BLAS — Basic Linear Algebra Subprograms*. University of Tennessee and Oak Ridge National Laboratory. Network software archive at `http://www.netlib.org/blas/`.

[147] *LAPACK — Linear Algebra PACKage*. University of Tennessee and Oak Ridge National Laboratory. Network software archive at `http://www.netlib.org/lapack/`.

[148] *National Design Repository*. Drexel University. Network CAD model archive at `http://www.designrepository.org/`.