Choosing Consistent Constraints for Beautification of Reverse Engineered Geometric Models

F. C. Langbein * A. D. Marshall R. R. Martin

Department of Computer Science, Cardiff University, PO Box 916, 5 The Parade, Cardiff, CF24 3XF, UK

Abstract

Boundary representation models reconstructed from 3D range data suffer from various inaccuracies caused by noise in the data and the model building software. Such models can be improved in a *beautification* step, which finds geometric regularities approximately present in the model and imposes a consistent subset of them on the model. Methods to select regularities consistently such that they are likely to represent the original, ideal design intent are presented. Efficiency during selection is achieved by considering degrees of freedom to analyse the solvability of constraint systems representing the regularities (without actually solving them). Priorities are used to select regularities in case of inconsistencies. The selected set of constraints is solved numerically and an improved model is rebuild from the solution. Experiments show that the presented methods can beautify models by selecting consistent regularities and enforcing major intended regularities.

Key words: Reverse Engineering; Beautification; Geometric Constraints; Degrees of Freedom Analysis.

1 Introduction

Reverse engineering geometric models uses information from a physical object to reconstruct a CAD model for applications like redesign, reproduction and quality control. Our particular goal is to reconstruct a boundary representation

---

* Corresponding author.
  Email adresses: `F.C.Langbein@cs.cf.ac.uk` (F. C. Langbein), `A.D.Marshall@cs.cf.ac.uk` (A. D. Marshall), `R.R.Martin@cs.cf.ac.uk` (R. R. Martin).
   .

(B-rep) model which exhibits the *exact* geometric properties present in the original, ideal design, using minimal human interaction, starting from 3D range data. Such a system should provide a simple, high-level user interface for inexperienced and non-engineering users, as well as engineers.

Introductions to reverse engineering are given by Várady et al. in [23,24]. Our approach starts with a data acquisition phase where multiple views of a physical object are obtained using a 3D laser scanner. The views are registered to form a single point set, which is then triangulated. Local properties derived from the triangulation are used to segment the point set into subsets which represent the natural surfaces of the object. To each of the subsets a surface (or surfaces) of appropriate type is fitted separately [2,26]. The resulting faces are stitched to create an *initial* B-rep model.

In this paper we consider engineering parts with 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 [2,26] and many realistic engineering objects can be generated using only these surface types [18,19]. We assume that the blends have been identified in the model and are represented as edge and vertex attributes. Hence, we ignore them in the rest of this paper.

State-of-the-art reverse engineering systems can create valid initial B-rep models approximating physical objects. However, these initial models suffer from various inaccuracies resulting from sensing errors arising during data acquisition as well as approximation and numerical errors in the reconstruction process. Improving the precision of sensing techniques and numerical methods may reduce the errors, but some errors will always remain. Other sources of error may also be present, such as wear of the object and the particular manufacturing method used to make it. Certain intended regularities, e.g. aligned cylinder axes, are an important part of engineering designs. They must be present for reverse engineered CAD models to have the greatest usefulness to applications like redesign. To ensure their presence, they have to be enforced at some stage of the reverse engineering process.

Previous approaches for enforcing regularities are based on constrained surface fitting methods [1,25,27], which, for example, fit two planes simultaneously under the constraint that they are orthogonal. Another approach is to drive the segmentation and surface fitting phases using features like slots and pockets whose approximate locations and types are provided by a user [22].

Our approach is to improve the initial model in a separate *post-processing* step which we call *beautification*. Improving the model without further reference to the point data avoids the computational expense of constrained fitting. The process consists of three main steps. First we *analyse* the initial model to detect approximate geometric regularities, in terms of geometric constraints:

e.g. we look for approximately aligned cylinder axes. As these regularities are only approximately present, the detected regularity set is likely to contain inconsistencies. Hence, in a *hypothesizer* step we select an appropriate consistent subset which represents the likely original design intent. Finally we *reconstruct* an improved model based on the selected regularities.

In [10–12] we presented methods to detect approximate regularities based on *local* relations in a B-rep model, such as parallel or orthogonal planes. An overview of the regularities and how they are expressed using geometric constraints is given below. Algorithms for processing *global* regularities such as symmetries of a B-rep model are discussed in [5,16,17], and are not considered here, but could be added to our beautification system without major changes.

To enforce a consistent subset of the potential regularities on the initial model, we require algorithms to detect inconsistencies and resolve them intelligently so that the resulting model is likely to represent the original design intent. This paper concentrates on presenting an efficient method for detecting inconsistencies between regularities. By representing the regularities as constraints, we can detect inconsistencies by considering the solvability properties of constraint systems, using degrees of freedom analysis [8]. While many constraint solving methods are available, most are aimed at finding the solution of well constrained systems [3]. In our case many constraints contradict each other, and relatively few are chosen for the final system. During regularity selection we are more interested in the solvability of a constrained system than a solution.

In order to prefer likely regularities we prioritize the regularities and consecutively add them to a constraint system. We only accept a regularity if the expanded constraint system remains solvable. The priorities are determined by merit functions indicating the desirability and likelihood of the regularity in the ideal design, and the accuracy to which the regularity is satisfied in the initial model. This approach was first used in [9], where solvability of the constraint system was decided by trying to solve it numerically using least-squares optimization algorithms. This successfully created improved models and the selected constraint systems were consistent up to a numerical tolerance. However, for each added regularity, the whole constraint system had to be solved numerically, so the time required (many hours) to find a consistent constraint system was too long for practical purposes.

In this paper we determine the solvability of a constraint system without solving it. This dramatically reduces the computing time (to a few minutes at most). It guarantees the consistency of the constraint system in a generic sense. Certain special situations are not detected as will be explained later, but our experiments show that this is not a problem for typical objects. By employing priorities in combination with our solvability test, reverse engineered models can be improved such that the major regularities are enforced exactly. Minor

regularities relating to the exact instance of the model, e.g. edge lengths, are selected consistently, but due to uncertainty represented by tolerances in the initial model, they may not exactly represent the original design.

We first overview the regularities we consider, how we find them, and how we represent them with geometric constraints. Then we present the top level algorithm for selecting regularities. The solvability test and priorities employed by this algorithm are discussed in detail in the following sections. After explaining the constraint selection process, we describe how we numerically solve the constraint system and construct an improved model. Finally we present some experimental results.

2   Geometric Regularities

We first give an overview of the geometric regularities used in this work and how they can be expressed in terms of geometric constraints. This is a summary of earlier work on representing [9] and detecting regularities [10].

Approximate geometric *regularities* are defined in terms of *similarities* and *regular arrangements* which lead to efficient algorithms for detecting them [5,10–12,16,17]. Reconstructed B-rep models are represented by elements like faces and vertices together with information about the topology of the model. We describe each of the elements by a type and a set of appropriate positional, directional, length and angular *features* (here, features describe shape properties, not machining features like slots or pockets). E.g. an element of the type *spherical face* is described by a positional (centre) and a length (radius) feature. A feature is either a scalar or a 3D vector. Directions are always represented by unit vectors. There are basic features required to describe the object (e.g. the centre, radii and direction for a torus) and extended features dependent on other features for additional properties (e.g. the sum and the difference of the radii of a torus) as listed in Table 1. Polygonal loop root points (extended features of planar faces) are the centroids of their edge loops. Regularities are detected as similarities between the features, e.g. approximately parallel directions. We also look for regular arrangements such as symmetries of positions or symmetrically arranged directions (using the Gaussian sphere).

We use *auxiliary* objects to describe certain regularities. E.g., for a set of parallel directions, we constrain each element to be parallel to an auxiliary direction rather than constraining them to be pairwise parallel. This avoids the introduction of complex constraint types which are hard to handle during the regularity selection. We have auxiliary lines, planes, cylinders, positions, directions, angles and lengths with the same basic features as the corresponding geometric objects from the model. Auxiliary objects do not have extended

4

Geometric Object	Basic Features / Extended Features
Plane	Position, direction
	Polygonal loop root points
Sphere	Position, radius
Cone	Position, direction, semi-angle
Cylinder	Position, direction, radius
Torus	Position, direction, major radius, minor radius
	Radii sum, radii difference
Straight	Position, direction
	Length
Circle	Position, direction, radius
	Circle segment angle
Ellipse	Position, direction, major direction, major radius, minor radius
Vertex	Position

Table 1. Geometric Objects and Their Features.

features as we do not have regularities relating directly to them.

Regularities are described by sets of geometric constraints. They specify relations between geometric objects of the B-rep model, and the auxiliary objects. Any regularities between geometric objects which can be expressed by constraint types in Table 2 can be handled by our system. Note that we do not allow topological changes of the model in this paper—such changes will be considered in a separate paper. We do, however, need to include constraints describing the topology of the model, to ensure that, for instance, faces intersect in a vertex. Hence, we have two classes of constraint sets: required constraint sets, which (mainly) enforce the correct topology on the model, and regularity constraint sets, which (mainly) determine the geometry of the model.

2.1 Required Constraints

To impose the correct topology, 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

Geometric Constraint	Equation
Parallel directions d_1, d_2	$d_1^t d_2 = 0$
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	$\|p_1 - p_2\|_2 = 0$
Constant distance l between two positions p_1, p_2	$\|p_1 - p_2\|_2 = l$
Distance between two position p_1, p_2 is a constant multiple ν of a variable length l	$\|p_1 - p_2\|_2 = \nu l$
Position p_0 is the average of n positions p_k	$n p_0 = \sum_{k=1}^{n} p_k$
Constant value α for angle/length parameter s	$s = \alpha$
Equal angle/length parameters s_1, s_2	$s_1 - s_2 = 0$
Linear relation between lengths/angles s_k with constants α_k	$\sum_k \alpha_k s_k = 0$
Position p on geometric object O (surface/curve)	$p \in O$

Table 2. Geometric Constraints.

faces at vertices of the model. In the case of two intersecting planes, two 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, this only ensures that the intersection is not empty (the vertices have to be in it), but does not specify the type of the intersection.

Instead of adding inequality constraints or alternative systems to ensure proper intersections, we use regularities. As the regularity detection phase considers all possible relations between face features, at least one regularity is present which specifies the exact relation between two adjacent faces. Thus, the regularities determine an exact relation between adjacent faces, e.g. constraining a cylinder axis to be parallel to a plane normal. If every regularity specifying a precise relation between a pair of adjacent faces is rejected, the relation is determined indirectly from other regularities. In such cases the direct relation between the face pair cannot be constrained without making the constraint system unsolvable or over-constrained. In the constraint system we consider edges of the model to be independent of the faces and they are only used to express certain regularities. Note that it is possible to add additional 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.

To enforce necessary dependencies between the basic and extended features, we use various required constraints. Feature dependencies include constraints setting the loop root points to be the centroid of the vertices in the loop, the length of a straight line to be the distance between its end-points, the circle

Parallel directions	Aligned axes
Symmetric directions	Axis intersections
Orthogonal system	Axes regularly on grid
Special angle between directions	Axes equi-spaced on line
Equal positions	Axes symmetrically on cylinder
Partially equal positions	Special ratio between lengths/angles
Equal lengths/angles	Special values for lengths/angles

Table 3. Geometric Regularities.

segment angle to be the angle between two auxiliary lines joining the centre to two vertices on the circle, the major direction of an ellipse to be orthogonal to the ellipse plane normal, and appropriate constraints for the radii sum and difference of tori.

2.2 Regularity Constraints

The particular regularities we consider are listed in Table 3. In [11,12] we gave various methods for detecting them. Regularities describing similar features like approximately parallel directions are detected by a hierarchical clustering algorithm where each cluster represents a regularity. Using the clusters, we seek regular arrangements of the features. Instead of setting maximum tolerances, we only use two tolerances giving the minimum value for when two angles or lengths should be considered as potentially different [10]. For instance, we look for approximately equal lengths by creating a cluster hierarchy, and also try to find possible special values (like an integer) close to the average length of members of each cluster. The hierarchies are truncated by detecting a large jump in the tolerance values between the clusters. In the following we give a brief overview of the regularities. We indicate how they can be expressed with constraint sets using the constraint types in Table 2 (for details see [9]).

The regularities in a cluster hierarchy are arranged in a tree. 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 a particular one, e.g. a parallel direction regularity must be present before a corresponding aligned axes regularity is added. Separate regularity hierarchies are used for parallel directions, equal positions, and equal length and angle parameters. For each cluster of similar features we create a corresponding auxiliary object, and constrain the features in the set to be equal to this object. To handle hierarchies, we constrain the auxiliary object for children to be equal to the auxiliary object for their parent.

Aligned axes and axis intersections can be found by clustering features where a parallel direction cluster is used to determine if axes should be aligned or intersect. For parallel aligned axes we further look for regular arrangements of axes on grids, lines and cylinders. Auxiliary lines are used to add appropriate constraints. For regular arrangements of axes we create additional auxiliary objects. For instance, given some bolt holes arranged around a circle, we create an auxiliary cylinder and require that the axes of the holes have directions parallel to the cylinder's axis. Auxiliary planes are constructed containing the cylinder axis, with normals constrained to be orthogonal to the cylinder axis. 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.

We recognise planar and conical cases of symmetrically arranged directions (e.g. plane normals around a prism, and a pyramid). The first case comprises a set of directions orthogonal to a direction d_0. 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 onto the plane defined by d_0 are integer multiples of $2\pi/n$. 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.

We also look for cluster hierarchies of equal positions when projected onto special planes (2D partially equal) and lines (1D partially equal) derived from important 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. Finally, there are regularities specifying special ratios between pairs of angle or length features and special values for these features. For all these regularities appropriate constraint sets and dependencies are created.

3   Choosing Consistent Regularities

Given a set of potential regularities, we wish to select a consistent subset which can be used to improve the initial B-rep model so that it more closely represents the original, ideal design intent. For this we add constraint sets corresponding to regularities, in order of priority, to a constraint system. We reject regularities which make the system generically unsolvable. The method is summarised in Algorithm 1. It employs a method `distribute` to add a

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

  I. Create a constraint (hyper-)graph g from the set of regularities R and T. Mark all regularities and constraints as inactive.

  II. Detect multiple constraints between the same geometric objects in g:

    1. Replace multiple identical constraints by a single constraint and appropriate references in the corresponding regularities.

    2. Wherever inconsistent constraints exist between the same geometric objects, find the corresponding regularity with the highest priority and remove all other regularities from R.

  III. Call `distribute`(t, g) for all constraints t in T to add them to g and mark them as active.

  IV. Repeat until R is empty:

    1. Remove the regularity r from R with the highest priority, for which all regularities it depends on are marked active.

    2. Add all i in the set of inactive constraints I for r to the graph by calling `distribute`(i, g) and check if the graph remains solvable:

      a. If a constraint made the graph unsolvable, reject r by removing all inactive constraints I from the graph and by removing all regularities depending on r from R.

      b. Otherwise, mark all inactive constraints I and r as active.

  V. Report the active regularities / constraints as the set of selected, consistent regularities.

Algorithm 1. Select Consistent, High Priority Regularities.

constraint to a constraint graph and a solvability test which are described in Section 4. Priorities are used to determine the order in which the regularities are checked, as discussed in Section 5.

Initially we have a list of potential regularities described by sets of constraints. They are marked as inactive to indicate that they have not yet been added to the constraint system. A regularity is marked *active* when it becomes one of the regularities selected to improve the model. If it cannot be added to the constraint system it is completely removed from the list.

We generate a (hyper-)graph representation of the constraints as described in Section 4. The nodes are the geometric objects and the edges are the constraints on them. A constraint is marked *active* if it has been added to the graph and *inactive* otherwise.

The constraint sets describing different regularities may contain several constraints between the same geometric objects. We detect these during graph creation. 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

9

from each involved regularity. If the constraints involve different constants, the related regularities are mutually inconsistent and only one regularity can be added to the constraint system. Only the regularity with the highest priority is kept, and the others are discarded. This is justified in that if a constraint cannot be added to a system without making it over-constrained or unsolvable, then another constraint between the same objects with different constants has the same property. Constants may exist for which the system *is* (non-generically) solvable, but it will still be over-constrained. However, note that the regularity with the highest priority may be rejected due to some inconsistency unrelated to the inconsistency between the constraints on the same geometric objects. In this case it may still be possible to add one of the other regularities. Such situations are unlikely, but not impossible. An algorithm which checks if alternative regularities have to be *reactivated* is given in [9].

A more sophisticated algorithm could be employed to detect additional inconsistencies in this pre-selection process. We could, for instance, 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. This creates complex selection rules (see [9]). However, as such cases are subsumed in the general cases which are detected by our symbolic solvability test, such methods are not used here.

Following the creation of the constraint graph, we first add the constraints describing the topology of the model to the graph. As we start with a valid B-rep model, the resulting constraint system must be solvable, so we do not need to check for solvability.

The next step selects the regularities. We take the regularity with the highest priority from the list for which all regularities it depends on are already active. The constraints for this regularity are added to the constraint graph unless they were previously added as part of a regularity made active earlier. For each added constraint we check if the resulting graph represents a generically solvable constraint system. If this is not the case, the regularity is rejected, and we remove all constraints *newly* added by the regularity to the graph. Also, any regularity in the list of potential regularities which depends on the rejected regularity is removed as well. These ideas are explained further in Section 4.

The selection process ensures that in the case of inconsistencies, regularities with the highest priorities are selected in preference to lower priority regularities. It does not, however, maximize the priority sum of selected regularities. Priorities are only used to compare the regularities in case of an inconsistency. They are not absolute measures for quality or desirability.

4 Solvability of Geometric Constraint Systems

To choose between regularities, we must be able to detect inconsistencies. We now present an efficient method to determine the solvability of a constraint system, so that we can detect inconsistencies between already accepted, consistent regularities and a new regularity.

We use a method which can decide if a given constraint system is solvable, i.e. has at least one solution. While it is desirable to ensure that we have a unique solution or a discrete set of solutions, we only check if a solution exists. We also 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. Note that the solvability test does not find a numerical solution to the constraint system. This is done later (see Section 6).

Various methods exist for handling geometric constraints (see, for instance, Brüderlin et al. [3]). Many of these methods are specific to 2D constraints and cannot be easily generalised to 3D. Our approach is based on a topological approach to degrees of freedom analysis [8] and analysing dependencies between geometric objects [14]. Its structure is similar to the `dense` algorithm [20], but we do not aim to decompose the constraint system as do Hoffmann et al. [4,7].

A geometric constraint system can be expressed as a hyper-graph, from which we can determine certain generic properties of the constraint system. To verify if a constraint system is solvable we consecutively add constraints to the graph and verify if the system remains solvable. The method for adding a constraint to the graph and the solvability test are now considered.

Our geometric objects are fully described by a *type* and a set of basic *features*. For unconstrained geometric objects, the values of the features can be chosen freely. Adding a constraint limits the values the features can have simultaneously. As the domains of the features are infinite, we cannot use methods that explicitly list the allowed values, but we can still list their domains. Geometric constraints usually limit the allowed values in these domains to subsets of lower dimension. When adding multiple constraints, the solutions are represented by the intersection of all these subsets. By reasoning symbolically about the properties of these intersections, especially their dimensions, we can determine the solvability properties of the constraint system. As we do not compute the solution of the constraint system, we can only check for generic solvability. Our experiments show that this does not cause problems for typical reverse engineered models.

To illustrate the concepts of our method for determining solvability of constraint systems, let us consider systems only including constant distance constraints between points in 3D 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 $\mathbb{R}^3$. Consider a distance constraint between two points p_1, p_2. It limits the allowed values the two points can have at the same time. One way of enforcing this is by allowing p_1 to have an arbitrary value in $\mathbb{R}^3$ and requiring that p_2 is on a sphere of fixed radius with centre p_1. This means p_2 can be described by a parameter on the unit sphere $\mathbb{S}^2$ in combination with the position of p_1. Clearly, the role of p_1 and p_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. This gives two choices for reducing the dimension of the parameter spaces.

A distance constraint is represented in the constraint graph by a directed edge indicating which of the two points is constrained to $\mathbb{S}^2$. To keep track of the reductions, we label each point with its parameter space. We use an example to illustrate what happens as further constraints are added. Consider the constraint graph for three points p_l in $\mathbb{R}^3$ shown in Figure 1. Initially all three points are unconstrained and their domain is $\mathbb{R}^3$ (Figure 1(a)). After adding the first distance constraint $dist_1$ between p_1 and p_2 we limit, for instance, p_2 to $\mathbb{S}^2$ (Figure 1(b)). Next we add a constraint $dist_2$ between p_1 and p_3 and decide to limit p_3 to $\mathbb{S}^2$ (Figure 1(c)). Finally we add a constraint $dist_3$ between p_2 and p_3. Again we have a choice to limit p_2 or p_3 to $\mathbb{S}^2$. However, both points are already reduced to $\mathbb{S}^2$. If we choose to put p_3 on another sphere, p_3 has to be in the intersection of two spheres. In the generic case two spheres intersect in a circle, so p_3 is now given by a parameter on the unit circle $\mathbb{S}^1$ (Figure 1(d)). In general 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 $d(p_1, p_2) > d(p_1, p_3) + d(p_2, p_3)$. The third case corresponds to a coincidence constraint, not a distance constraint. In all of these cases additional conditions are present which cannot be determined directly from the constraint graph. Hence, for the solvability test we assume that we always have the generic case.

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 $\mathbb{S}^2$, assuming 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 choose one of them, i.e. we get $\mathbb{R}^0$. Having two points indicates that
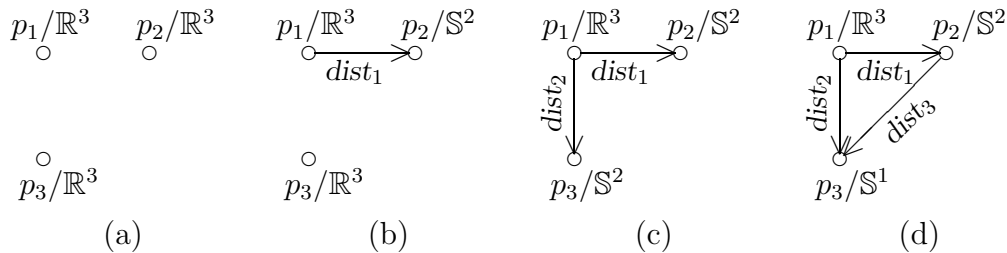
Figure 1. Distance Constraint Graph Between Three Points.

there are 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 is justified as we can take the solution which is closer to the initial model. We refer to adding a constraint to a constraint graph using a directed edge as above as *distributing* a constraint in the graph.

The dimensions of the domains of the points represent the degrees of freedom of these objects (3 for 3D points). The reduction of these domains to lower-dimensional subsets indicates the number of degrees of freedom removed by a constraint (1 for distance constraints). Our approach is similar to degrees of freedom analysis [8], but 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 intersect either of them with $\mathbb{S}^2$. However, because each constraint can be added to the constraint graph in two ways, it may be possible to choose a different distribution for 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. The depth-first approach follows one particular path in the graph backwards along the directed edges to its end before we consider any other paths. 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 search all paths from the new constraint, but only find the shortest paths to the edges which can be redistributed. Whenever an edge can be redistributed, we redistribute the whole path to that edge. 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 [20]. In that method the constraint graph is converted to a bipartite graph between nodes representing constraints and other nodes representing geometric objects. Edges in this graph connect the constraint nodes with the geometry nodes that they constrain. The graph is interpreted

13

as a flow network from a source (connected to constraint nodes) to a target (connected to object nodes). The capacity of an edge from the source to the constraint node indicates the degrees of freedom removed by the constraint. The capacity of an edge from an object node to the target indicates the degrees of freedom of the object. The capacities of other edges 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 the 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. Note that in our approach the algorithm on the graph is more closely linked to the constraint system without considering a flow network.

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.

For the solvability test consider the following simple example. If we want to determine point p_3 in the constraint system in Figure 1, we can do this by setting an arbitrary location for p_1, choose p_2 on a sphere around p_1 and then determine p_3 by choosing a parameter in $\mathbb{S}^1$, using the locations of p_1 and p_2. This specifies all three points up to location and orientation in $\mathbb{E}^3$. As distance constraints cannot specify a 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 it, we can follow edges backwards to determine the sub-graph $S(n, e)$ of all nodes on which n depends due to e. 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 on 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. This is checked whenever we distribute an
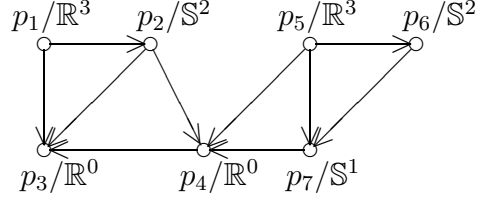
Figure 2. Example Constraint Graph for Dependency Sub-Graphs for Point p_4.

edge and compute the resulting degrees of freedom (each constraint removes a generic number of degrees of freedom from a node; we do not remove more degrees of freedom from a node than it has 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 2. Node p_4 has three dependency sub-graphs. The first graph consists of the nodes p_1, p_2, p_4 with p_4 relabelled to $\mathbb{S}^2$. This sub-graph has 7 degrees of freedom left, so it is solvable. Without relabelling p_4 it would only have 5 degrees of freedom which are not sufficient. The second sub-graph consists of p_4 and p_5 with p_4 relabelled to $\mathbb{S}^2$. Without relabelling p_4 to $\mathbb{S}^2$ in the second sub-graph the graph would not be solvable (it would have 3 degrees of freedom, whereas two points on a line require 5 degrees of freedom). The third sub-graph consists of p_4, p_7, p_5, p_6 with p_4 relabelled to $\mathbb{S}^1$, with 7 degrees of freedom.

When we successfully distribute a new constraint edge e in the graph, we must test if the new graph is solvable; if distribution fails, we already know that the constraint system is not solvable. To test for solvability we only have to consider changes made during distribution. Let n be the node the constraint e has been distributed to. We assert 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. However, note that there are special cases of lower dimensional subsets embedded in $\mathbb{E}^3$ which must be handled separately.

Suppose no redistributions are required to distribute e. In this case only the node n is changed. The dependency sub-graphs which do not contain e did not change. Thus, we only have to check $S(n, e)$. Assuming that all intersections are generic, we only have to check if there are sufficient degrees of freedom in $S(n, e)$. If redistributions *are* needed, we can consider each redistribution separately. Assume an edge exists between two points n_1 and n_2 which initially constrains n_2. When we redistribute this edge, n_1 is constrained by n_2. The degrees of freedom are moved from n_1 to n_2 and this is indicated by 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.

Geometric Constraint	Distribution
Parallel directions d_1, d_2	d_1 or d_2 in $\mathbb{S}^0$
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 $\mathbb{R}^0$
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 $\mathbb{R}^0$
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 4. Distribution of Geometric Constraints.

4.2 Distributing Constraints

We now present the distribution algorithm for each constraint type in detail. We describe the geometric objects in terms of positions, directions, lengths, and angles. Constraints limit the allowed combinations of values for these features. The domain for positions is $\mathbb{R}^3$, for directions, $\mathbb{S}^2$, for lengths, $\mathbb{R}^1_+$ and for angles, $\mathbb{S}^1$. Any constraint can be interpreted as selecting a lower-dimensional subset of these domains. This may usually be done in more than one way. We assume that all intersections are generic, as is usual in degrees of freedom analysis.

In Table 4 we list the different options for distribution of constraints. Note that we do not consider all possible distributions of the dimensions, but only those which have a simple geometric meaning. E.g., for two parallel directions we only consider the cases where one of the two directions is equal to the other and thus fully determined. We do not allow both d_1 and d_2 to have one degree of freedom. We distinguish between constraints with constant and variable parameters, as only variable elements can be used during distribution, e.g., 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 set the value of the distance.

Constraints requiring positions to lie on a surface or curve are omitted from Table 4. Surfaces and curves are usually described by a combination of po-

sitional, 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 directions. However, 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 3D positions. For planes we only have a 1D position space (its distance from the origin), for cylinders and straight lines we have a 2D position space (the position of the axis). All other objects we consider have 3D position spaces.

We do not discuss the issues relating to faces in the constraint graph in detail, but only enough to understand how they are handled by the distribution algorithm. Whenever we put a position on a surface or curve we reduce the dimension of one of the parameter spaces of the surface or curve, or of the position, by one. We can choose any of the parameter spaces involved for the distribution. The types of the parameter spaces may vary, and intersections between them can be complicated. We assume that the intersections are always generated by reducing the degrees of freedom by one.

Generically, 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 assumes 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, the intersection of M and N is not empty, and that $m + n \geq d$. 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.

Algorithm 2 is the overall constraint distribution algorithm. A constraint can be *distributed directly* if one of the distribution options in Table 4 can be applied using the previous reasoning 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 is found which can be redistributed (steps II and III are initialization; step IV does the search). The breadth-first search moves from edge to vertex to edge, and so on, following directed edges backwards in the graph. By remembering the search sequence with predecessor links, we can backtrack to the original constraint to find the redistribution path, and apply the redistributions along the path accordingly. We then try to distribute the new constraint directly in the graph. If this is possible we do so and report success. Otherwise, we try to find another redistribution path, i.e. 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.

Method `distribute` (`c`,`g`): Try to distribute the constraint `c` in a constraint graph `g` = (`nodes`, `edges`). `g` is updated according to the distribution, and success or failure of distribution is reported.

I. If `c` can be distributed directly without redistribution, do so, and return success.

II. Mark all `nodes` and `edges` as unvisited and set the predecessor for all `nodes` and `edges` to empty.

III. Initialise a set `activeE` of edges (constraints) to `c` and a set `activeN` of nodes (B-rep 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 unvisited nodes connected by `e` to `activeN` and set their predecessors to `e`.

  B. Consecutively remove nodes `n` from `activeN`, and for all constraints `e` restricting `n` which have not yet been visited do:
    1. Seek a redistribution `r` of `e`, which can be applied directly to the graph, with `n` is less restricted than before. Remember the redistribution `R` which gives 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 links and redistribute each constraint along this path.
    3. If `c` can be distributed directly, do it and return success.
    4. Otherwise, reset all flags in `g`, set `activeE` to `c` and `activeN` to empty and restart the search for a new redistribution path within the loop of step IV.

V. No distribution has been found, return failure.

Algorithm 2. Constraint Distribution.

To see how this works, consider the constraint graph in Figure 3 linking four points p_l and one plane s. Graph (a) was created by adding an additional point p_4 to the graph in Figure 1 and adding three distance constraints from the other points to p_4. All new distance constraints can be distributed directly such that p_4 is now completely constrained. The plane s is described by a distance and a direction indicated by $\mathbb{R}^1$ and $\mathbb{S}^2$ in the graph. In graph (b) we add three constraints placing p_1, p_2 and p_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 all the constraints could be added by direct distribution.
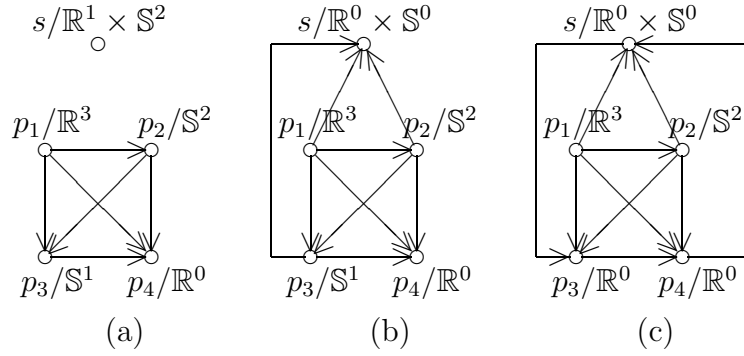
Figure 3. Example Constraint Graph of Distances Between Four Points on a Plane.

In graph (c) we distribute a constraint placing p_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 p_4 to `activeN`. In step IV.B we find three direct redistribution options via p_1, p_2, p_3 for s, two direct redistribution options via p_1, p_2 for p_4 and one additional path via p_4 to p_3. All the edges for these options are added to `activeE` to continue the search. In step IV.C we choose the redistribution with maximal increase of degrees of freedom for s or p_4. Here, this can be any of the constraints between s and p_1, p_2 or p_3 or between p_4 and p_1 or p_2. We choose to redistribute the constraint between s and p_3 initially reducing the degrees of freedom of s by one. We redistribute this edge reducing the degrees of freedom of p_3 by one and increasing the degrees of freedom of s by one. Now the original edge can be distributed directly and we report success with the distribution as shown in graph (c).

Any redistribution path which does *not* succeed in adding the new constraint does not change the solvability properties of the constraint system. While the redistribution changes the distribution of degrees of freedom in the graph, the dependency sub-graphs still contain sufficient degrees of freedom. Only the distribution of the edges 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 next.

4.3 Solvability Test

The solvability criterion for the general case is similar to the one for the distance constraint example. A 3D object embedded in $\mathbb{E}^3$ must have at least six degrees of freedom for the system to be generically solvable. For zero-dimensional points we must have three and for collinear points we must have five degrees of freedom. When other 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 2D which means if only directions are involved, the minimum number of degrees of freedom required for two or more directions is three. Single surfaces and curves may also have less degrees of freedom.

We also have constraints setting 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 3D case.

After a constraint has been successfully distributed in the graph we must check if 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. Recall that special cases may arise where the constrained objects require less than six degrees of freedom in order to be solvable.

If the solvability test is successful, the constraint can be added to the graph without destroying the generic solvability of the constraint system. We 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 3(c), created by distributing the constraint between p_4 and s. 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 makes the system unsolvable. (The constraints in Figure 3(b) already 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 [20]. Its main purpose is to detect solvable sub-systems of a constraint system, to solve it symbolically. The constraint systems it handles usually contain only a few over- or under-constrained cases. Our method 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 `dense` algorithm. Furthermore, experiments (see Section 7) show that the method is successful when applied to real (albeit simple) problems.

5   Prioritizing Geometric Regularities

To resolve inconsistencies between regularities, we use a mechanism to select regularities more likely to be present in the original design [9]. This is based on priorities computed using merit functions. Deciding which regularities to choose is non-trivial and often there is more than one sensible choice.

The priority $w(r)$ of a regularity r is computed by taking a weighted average of: a measure $w_e(r)$ of the numerical accuracy to which the regularity's constraints are satisfied in the initial model, a merit $w_q(r)$ for the quality or desirability of the regularity depending on specific arrangements and constants involved, and a constant $w_b(r)$ describing a minimum desirability for each regularity type. This average is weighted by $w_c(r)$ indicating how common the regularity is (determined by surveying a range of engineering components). Thus,

$$w(r) = w_c(r)\left(c_e w_e(r) + c_q w_q(r) + c_b w_b(r)\right),\tag{1}$$

where all constants and functions are in $[0, 1]$ and $c_q + c_e + c_b = 1$, e.g. $c_e = 3/6$, $c_q = 2/6$, $c_b = 1/6$. The maximum of $w(r)$ is $w_c(r)$ and the minimum for an undesirable regularity with high error is $w_c(r)w_b(r)c_b$.

To find $w_e(r)$, we combine the average angular error e_r and the average length error e_l of the regularity's constraints. w_e should be close to 1 for small errors and drop quickly towards 0 when the errors become too large. We convert angular errors to length units using the maximum length L_m in the model:

$$w_e(r) = \frac{1}{1 + c_l(L_m \sin(e_r) + e_l)}\tag{2}$$

where c_l is a constant indicating the base length unit for the model, e.g. $c_l = 1$ or 2.54. L_m can be chosen to be the diagonal of the bounding box of the model, or the maximum edge length.

$w_q(r)$ describes the desirability or quality of the regularity if enforced exactly on the model, computed by considering the regularity type and geometric objects, their arrangement and special values involved. We first define some quality factors used to compute $w_q(r)$.

All special values involved in the regularities have the form $v = \pm(n/m)^{1/(r+1)}b$ with n, m, and r integers and b some base value like π or 1. We evaluate the
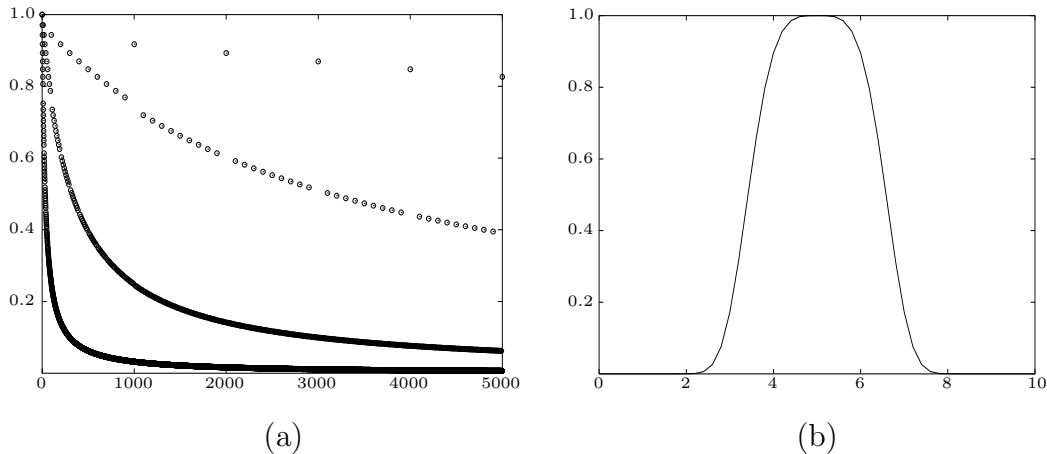
(a)   (b)

Figure 4. Graph of Quality Merit Functions for (a) Special Values of Denominators $w_{sv}(m, 0, 1)$ over m and (b) Common Boundary Elements $w_a(X, 0.5)$ over $n(X)$.

quality of special values using the function

$$w_{sv}(m, r, b) = \frac{3q(b)}{3 + c_0 l + c_1(m/M^K - 1) + c_2 r} \tag{3}$$

where $q(b)$ is a constant in $[0, 1]$ evaluating the desirability of the base value b (e.g. $q(\pi) = 1$, $q(\pi/180) = 0.8$, $q(1) = 1$, ...), 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 m, c_1 is a constant indicating the importance of the non-zero part of m and c_2 indicates the importance of the root r, e.g. $c_0 = 0.01$, $c_1 = 0.005$, $c_2 = 0.7$. The graph of $w_{sv}(m, 0, 1)$ is shown in Figure 4(a) for special values of the form n/m. One can identify separate curves for values for m of the type $p * 1000$, $p * 100$, $p * 10$ and p for $p \in \mathbb{N}$ representing the non-zero part of m. Increasing r moves these curves closer towards the m-axis. The formula for w_{sv} has been chosen to favour special values with a small non-zero part m/M^K, small roots r and short representations in the base M.

Another quality factor is the number $n(X)$ in a set X of B-rep model elements 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

$$w_a(X, p) = \exp\left(-(c_w(p|X| - n(X)))^{c_p}\right) \tag{4}$$

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 geometric elements 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(b) shows the graph of $w_a(X, 0.5)$ for some 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 in which more of the possible positions are occupied. If all consecutive positions are occupied, the quality factor $w_{ra}(r)$ is set to $w_{sv}(m, 1, b)$. Otherwise, we make a list of smallest integers k for which all positions (starting at an arbitrary position) with the distance $k(n/m)b$ between them are occupied. For each k we add $m/(kn)w_{sv}(m/\gcd(m, kn), 1, b)$ to w_{ra}. For axes arranged regularly on a grid we have two orthogonal directions; for each direction we project the occupied positions in the grid onto a line and proceed as for arrangements on a line. The average of the quality for both lines 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

$$w_t(O) = \frac{1}{|O|} \sum_{t \in \text{ObjectTypes}} c(t) \exp\left(-(t_w(|O| - c(t)))^{t_p}\right) \qquad (5)$$

with constants t_w and t_p, e.g. $t_w = 0.05$, $t_p = 2$. The graph of $w_t(O)$ is similar to $w_a(X, p)$ in Figure 4(b). It favours regularities having repeated objects of the same geometric type.

We select appropriate quality factors and compute their weighted average to give $w_q(r)$ for each type of regularity (see Table 5). E.g. for parallel directions the quality $w_{sv}(0, 1, \pi)$ of the parallel angle is most important, but we also prefer regularities with objects of the same geometric type. For planar symmetrically arranged directions, the main emphasis is on the number of occupied positions and faces arranged in a loop as computed by $w_{ra}(r)$ and $w_a(F, 1)$, but we also consider the special angle value for the planar arrangement and the geometric types involved. Table 5 also lists the values used for w_c and w_b. These, and w_q, were derived from a part survey estimating the frequency of regularities in simple mechanical components [18], and were further refined to produce desired priorities in various example models. These values could be changed for differing application domains.

While the order of the regularities can be adjusted by varying the constants, their large number makes it hard to predict the effect of changes. Choosing

Regularity r	$w_c(r)$	$w_b(r)$	$w_q(r)$
Parallel directions	1.00	1.00	$0.8w_{sv}(0,1,\pi) + 0.2w_t(O)$
Symmetric directions (planar)	1.00	1.00	$0.2w_{sv}(2,1,\pi) + 0.3w_a(F,1)$ $+ 0.1w_t(O) + 0.4w_{ra}(r)$
Symmetric directions (conical)	0.90	0.70	$0.3w_{sv}(m,r,b) + 0.3w_a(F,1)$ $+ 0.1w_t(O) + 0.3w_{ra}(r)$
Orthogonal system	1.00	1.00	$0.6 + 0.3w_a(F,1) + 0.1w_t(O)$
Special angle between directions	0.90	0.60	$0.8w_{sv}(m,r,1) + 0.2w_a(0,0.5)$
Equal positions	0.80	0.55	$w_t(O)$
2D partially equal positions	0.85	0.65	$0.5w_a(V,0.5) + 0.5w_t(O)$
1D partially equal positions	0.83	0.60	$0.5w_a(V,0.5) + 0.5w_t(O)$
Aligned axes	0.97	0.85	$w_t(O)$
Axis intersections	0.90	0.80	$0.1w_a(F,1) + 0.9w_t(O)$
Axes regularly on grid	0.90	0.85	$0.3w_t(O) + 0.7w_{ra}(r)$
Axes equi-spaced on line	0.88	0.75	$0.2w_t(O) + 0.8w_{ra}(r)$
Axes symmetrically on cylinder	0.95	0.90	$0.3w_t(O) + 0.7w_{ra}(r)$
Equal lengths / angles	0.90	0.75	$w_t(O)$
Special ratios between lengths / angles	0.80	0.55	$w_{sv}(m,r,1)$
Special values for lengths / angles	0.85	0.70	$w_{sv}(m,r,b)$

Table 5. Constants and Merit Functions for Regularity Priorities.

priorities based on multiple-choice questions presented to a user might improve this. Currently the priorities are sufficient to improve the model, but a sophisticated decision process considering more complex relations between regularities and the model, globally, could improve regularity selection with respect to design intent.

6   Constructing an Improved Model

Regularity selection results in a list of generically consistent constraints describing the improved model. A numerical method is then used to find a so-

lution of the constraint system, followed by a reconstruction process which generates an improved model from this solution.

A numerical approach is justified as we already have a valid B-rep model which can be used to provide a good initial value for the numerical solver. As the initial value is close to the solution, only a few iterations are required to converge; convergence occurs if a discrete set of solutions exists, which was the case in all examples we tested, due to the large number of regularities involved. Symbolic methods based on the results of the solvability test using the graph-representation of the constraint system provide an alternative approach. It may be possible to modify the solvability test to create a decomposition plan for a symbolic constraint solver [7].

The constraint system contains equations of the types in Table 2. We also add one equation per direction vector to ensure that it is a unit vector. We solve the constraint system using numerical optimization methods based on quasi-Newton (variable metric) methods [6,15,21]. These need an approximation method for the Hessian matrix of second partial derivatives and a line-search method. For the latter we considered the Goldstein-Armijo and PWS methods [21]. While both perform well, PWS is more stable and more suitable for the BFGS quasi-Newton update. 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 with a condition guard initiating restarts of the iteration [21]. Further numerical stability was achieved by using a damped version of BFGS [13], at the cost of an increased number of iterations. Using a hybrid method switching between BFGS and Gauss-Newton steps improved convergence rates while still giving acceptable numerical stability [15].

After a numerical solution to the constraint system has been found, an improved model is rebuilt using the topological information from the initial model and the feature values from the numerical solution. We create new faces using the solution of the constraint system and re-intersect them to obtain the complete model. The solution of the constraint system gives vertex positions and the faces. Sharp edges are found using a surface-surface intersection algorithm for adjacent surfaces, guided by the initial model to determine which part of the intersection curve is required. For smooth edges, the intersection is tangential and cannot be computed in this way. Such special cases are computed separately for all combinations of the considered surface types.

Additional adjustments include moving the object to a special position and orientation with respect to certain determined vertex positions, orthogonal systems and main axes.
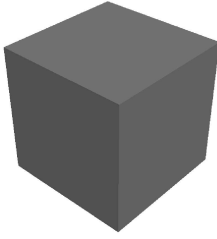
	Detected Regularities	Tol. Level
	1 orthogonal system	3°
	2 pairs of parallel directions	1°
	1 pair of parallel directions	3°
	6 sets of special angles	0.1° to 3°
	2 pairs of aligned axes	1°
Time taken in	1 pair of aligned axes	3°
	1 intersection of 3 axes	0.15
Analysing: 0.1 sec.	▶ 1 intersection of 2 axes	0.05
Selecting: 1.6 sec.	1 equality of 8 lengths	0.1
Solving: 2.1 sec.	▶ 2 equalities of 3 and 5 lengths	0.04
Total: 3.8 sec.	3 sets of special edge lengths	0.001 to 0.1

Figure 5. A Simple Model from Simulated Data with Test Results.

7   Experiments

We have tested our ideas using models reverse engineered from both real and simulated data, and we now provide some of the results. The system was implemented on a GNU/Linux system having a Pentium III 700MHz with 256MB RAM. We first discuss a simple model reverse engineered from simulated data to show the general properties of our system in detail. Then we discuss examples using models reverse engineered from real data.

7.1   A Simple Example Using Simulated Data

We illustrate the general behaviour of our method using simulated data from a model of a cube with edge length 2, perturbed by randomly changing the plane normals by up to 3 degrees and face positions by up to 0.1 length units. This model is simple enough to be able to list the regularities detected and explain the behaviour of the system in more detail. An initial model was reverse engineered from a point set generated from this object.

The analyser detected 21 approximate regularities (see Figure 5). The complete list has been simplified for this discussion. We list the number of regularity types detected at different tolerance levels (given approximately). We clearly detect the orthogonal system of the cube, but at a high tolerance level due to the perturbation. This also caused the detection of two parallel di-

rection pairs at a lower tolerance level and a third one at a higher level. We detected six angles between the plane normals with a set of various special values close to 90° at different 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 adds the third axis pair. This creates a parent/child relation in our regularity structure (see ▶ in the list). Similarly we find two groups of 3 and 5 edge lengths close to each other which are combined at a higher tolerance level (again see ▶). For each group we also found sets of special edge length values at various tolerance levels.

For the priorities, we have two basic choices. We can emphasise *regularities* so that the orthogonal system and all regularities following from it are selected. Alternatively we can favour *small tolerances*, resulting in the selection of alternative regularities where e.g. one of the approximately aligned axis pairs is not orthogonal to the other two. However priorities were set, the numerical solver found a solution to within a predetermined numerical tolerance. By setting the priorities to favour an orthogonal system a cube with edge length 2 was created as the improved model. Besides emphasizing parallel directions and orthogonal systems, we also had to ensure that integer values for edge lengths were strongly favoured to achieve this. In contrast, if we favoured precise tolerances, integer angles different from 90° were chosen for all relations between directions. The equal edge length groups were completely rejected due to solvability problems and varying values for edge lengths were selected. We also tried a third set of priorities which neither emphasized 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.

Whether to accept high quality regularities or regularities with small tolerances depends on assumptions about the initial model. Only regularities satisfying both requirements are likely to be always accepted. In the above example, the third model may actually be the most likely one if we ignore our knowledge of how we perturbed 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 force all edges to have the same length, this also means that we have to accept a high tolerance level and so various special values become possible.

The time taken to improve this model was 3.8 seconds; a detailed breakdown

Model	(a)		(b)		(c)		(d)	
	Reg.	Cons.	Reg.	Cons.	Reg.	Cons.	Reg.	Cons.
Total	229	743	382	1808	216	1263	487	2555
Selected	67	217	164	683	156	910	227	1453
Faces	11		19		14		25	
Time in								
Analysing	0.2 sec.		0.6 sec.		0.5 sec.		0.9 sec.	
Selecting	12.9 sec.		27.3 sec.		18.9 sec.		38.5 sec.	
Solving	24.4 sec.		58.2 sec.		45.6 sec.		113.3 sec.	
Total	37-.5 sec.		86.1 sec.		64.9 sec.		152.7 sec.	

Table 6. Number of Total and Selected Regularities/Constraints and Computing Times for Example Models.

is given in Figure 5. The time for rebuilding the model was under 0.1 seconds and is included in the solving time.

7.2 Improving Reverse Engineered Objects

We now discuss the results of improving four models 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. Our example models are shown in Figure 6. Table 6 lists the number of regularities detected and the number of constraints required to describe them, and how many of each were selected to improve the model. Note how large these numbers are even for relatively simple objects.

The priority values from Section 5 were used, resulting in quality only being slightly emphasized over tolerance — in general this resulted in the best overall regularity selections (Fine-tuning the values for particular models improved the results). Changing these values to favour quality or tolerance had effects comparable to those discussed in Section 7.1.

Model (a) has a central axis, several planes with normals parallel to this axis and two parallel planes with normals orthogonal to it. These regularities were detected and imposed on the model. The two blue parallel planes (only the one in front is visible), however, could only be made parallel by allowing large angular tolerances (about 3°) due to an error in the initial model. This angular error arose during registration of the two scanner views for the opposite sides of the object, containing the blue planes. (The common points in the range data used for registration of the two views only belonged to cylindrical and
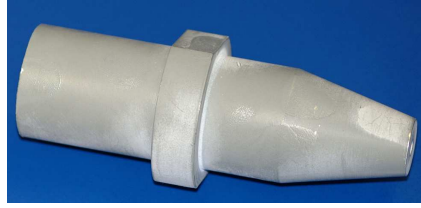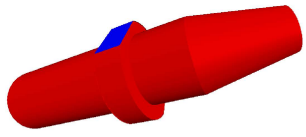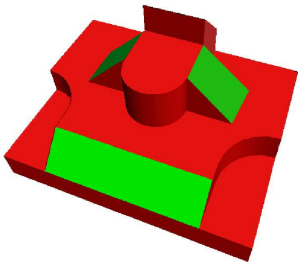
(a)
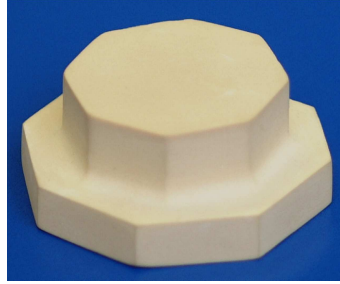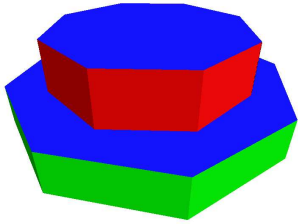
(b)
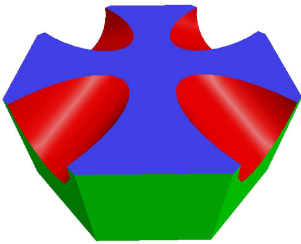
(c)

(d)

Figure 6. Example Models.

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.

Model (b) has two symmetrically arranged, planar direction sets based on the angle $\pi/4$. Together with the orthogonal relation between the symmetrically arranged 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 after adjusting the priorities, only the two groups of short edges could be forced to be of the same length. The values in the other two groups of lengths were similar, 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 fixed. However, for the value of this angle there was again a choice between a special value close to the value in the initial model and one of high quality. Our method used a particular plane pair to select this angle. If we favoured integer degrees the closest integer to this angle is chosen. The angle in the original design was $10°$ degree; the plane pair chosen had an angle of $45° + 10.8°$ degrees and thus the angle between the two plane sets was chosen to be $11°$. A more sophisticated analysis might be able to use all angles between the plane pairs to find an average value for this angle, giving a greater chance of success.

In model (c) the green normals of the planes are arranged symmetrically in a plane, and the axes of the red cylinders 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 were not rotated in a way that they would intersect with more than one green plane.

The directions in model (d) form two orthogonal systems: the normals of the green planes and the remaining directions from the red faces. The regularities were present in the model to within about $2°$. As they are also of high quality, they were selected. The $45°$ degree relation 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 in average to within about $3°$. As our priority parameters favour $45°$ 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. The regularity selection, however, ensured that the two slots are congruent.

30

Usually, the observed tolerances for the reverse engineered real objects were slightly smaller than those used for simulated data. Initial angular errors were usually about 1°–2° and positional errors were about 0.5–1 length units (millimetres; scanned points were about 1 millimetre apart). This made the priorities of major regularities quite robust to changes in the priority parameters, and they were usually exactly imposed on the model. The order of selection of special values and local relations between faces is less stable and has greater dependence on the choice of priority parameters. Especially for models where high tolerances have to be accepted in order to select the major regularities, the uncertainty for special values is relatively large. We can only make a guess within the tolerance depending on which kind of values we prefer.

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 is strong evidence that the selected constraint systems did not contain any inconsistencies. It also provides evidence that the generic solvability test is sufficient for the kind of models we considered.

It is expected that in 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 generic solvability test is not sufficient. As we are using a numerical optimization method it is still likely that a solution which generates a valid model will be found, but it will not exactly (within numerical tolerance) impose the regularities on the model.

The time required to improve a model is up to a few minutes. This is acceptable, especially considering the time required for the whole reverse engineering process, and particularly data acquisition. Most of the time spent in beautification is used in numerically solving the constraint system.

Our research so far suggests that our beautification system can be used to improve reconstructed models, even if more testing is needed (However, in order to handle more complex models the robustness of the other reverse engineering phases also has to be improved). In particular, our regularity selection method creates consistent geometric constraint systems. The tests show that major regularities, when only weakly related to other major regularities, are easily identified and imposed on the model. However, major approximate regularities which cannot be imposed on the model at the same time because they involve similar sets of faces are harder to resolve. No clear decision can be made if neither the tolerances nor the quality of these regularities differ distinctively. Only in cases were there is a clear major regularity relating to many faces in the object can a clear decision be made. In other cases there may be alternative models which are as plausible as the chosen one. A more

intelligent selection method than simple priorities, which considers the global structure of the model, may improve this situation.

A new model having specific special values for lengths and angles cannot be guaranteed. 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 [10]. Higher precision can be achieved only by creating a more exact initial model.

8   Conclusion

We have presented an efficient system to beautify simple reverse engineered geometric models. The core components of this system are regularity detection methods reported in earlier work [5,10–12,16,17] and methods to select consistent geometric constraints described here. Using this system, weakly dependent major regularities can be identified and imposed correctly on the model. The regularities are always selected such that the corresponding constraint system is consistent in a generic sense. Experiments show that generic solvability is sufficient to find solutions of the constraint system which can be employed to successfully improve the model.

Future work will consider topological changes to the model and alternative methods for selecting regularities in the presence of inconsistencies.

References

[1] 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.

[2] P. Benkő, R. R. Martin, T. Várady. Algorithms for reverse engineering boundary representation models. *Computer-Aided Design*, 33(11):839–851, 2001.

[3] B. Brüderlin, D. Roller. *Geometric Constraint Solving and Applications*. Springer, Berlin, Heidelberg, New York, 1998.

[4] I. Fudos, C. M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Trans. Graphics*, 16(2):179–216, 1997.

[5] C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate congruence detection of model features for reverse engineering. Accepted for *Int. Conf. Shape Modelling and Applications*, 2003.

[6] J. X. Ge, S. C. Chou, X. S. Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31:867–879, 1999.

[7] C. M. Hoffmann, A. Lomonosov, M. Sitharam. Decomposition plans for geometric constraint systems. *J. Symbolic Computation*, 31(4):367–427, 2001.

[8] G. A. Kramer. *Solving geometric constraint systems: a case study in kinematics*. MIT Press, 1992.

[9] F. C. Langbein, A. D. Marshall, R. R. Martin. Numerical methods for beautification of reverse engineered geometric models. In: H. Suzuki, R. R. Martin (eds), *Proc. Geometric Modeling and Processing 2002*. IEEE Computer Society, Los Alamitos, CA, 159–168, 2002.

[10] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Approximate geometric regularities. *Int. J. Shape Modeling*, 7(2):129–162, 2001.

[11] 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 Press, 206–215, 2001.

[12] 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, 10–19, 2001.

[13] D. H. Li, M. Fukushima. A modified BFGS method and its global convergence in non-convex minimization. *J. Computational and Applied Mathematics*, 129(1–2):15–35, 2001.

[14] 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.

[15] L. Luksan, E. Spedicato. Variable metric methods for unconstrained optimization and nonlinear least squares. *J. Computational and Applied Mathematics*, 124:61–95, 2000.

[16] B. I. Mills, F. C. Langbein. Determination of approximate symmetry in geometric models – an exact approach. Submitted to *Computational Geometry*, 2002.

[17] 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 Press, 241–248, 2001.

[18] 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, Dept. Computer Science, Cardiff University, 2001. `http://ralph.cs.cf.ac.uk/papers/Geometry/survey.pdf`.

[19] 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, University of Rochester, July 1976.

[20] M. Sitharam, C. M. Hoffman, A. Lomonosov. Finding dense subgraphs of constraint graphs. In: G. Smolka (ed). *Constraint Programming*. Springer, Berlin, Heidelberg, New York, 463–478, 1997.

[21] P. Spellucci. *Numerische Verfahren der nichtlinearen Optimierung*. Birkhäuser, Basel, Boston, Berlin, 1993.

[22] W. B. Thompson, J. C. Owen, J. Germain, S. R. Stark, T. C. Henderson. Feature-based reverse engineering of mechanical parts. *IEEE Trans. Robotics and Automation*, 15(1):57–66, 1999.

[23] T. Várady, R. R. Martin, J. Cox. Reverse engineering of geometric models – an introduction. *Computer–Aided Design*, 29(4):255–268, 1997.

[24] T. Várady, R. R. Martin. Reverse engineering. Chapter 26 in: G. Farin, J. Hoschek, M.-S. Kim (eds). *The Handbook of Computer-Aided Geometric Design*. Elsevier, Amsterdam, 651–681, 2002.

[25] N. Werghi, R. Fisher, C. Robertson, A. Ashbrook. Object reconstruction by incorporating geometric constraints in reverse engineering. *Computer-Aided Design*, 31(6):363–399, 1999.

[26] 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.

[27] N. Werghi, R. B. Fisher, C. Robertson, A. Ashbrook. Shape reconstruction incorporating multiple non-linear geometric constraints. *Constraints*, 7(2):117–149, 2002