Numerical Methods 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
{F.C.Langbein,A.D.Marshall,R.R.Martin}@cs.cf.ac.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. The quality of such models can be improved in a beautification step, which finds geometric regularities approximately present in the model and tries to impose a consistent subset of these regularities on the model. A framework for beautification and numerical methods to select and solve a consistent set of constraints deduced from a set of regularities are presented. For the initial selection of consistent regularities likely to be part of the model's ideal design priorities, and rules indicating simple inconsistencies between the regularities are employed. By adding regularities consecutively to an equation system and trying to solve it using quasi-Newton optimization methods, inconsistencies and redundancies are detected. The results of experiments are encouraging and show potential for an expansion of the methods based on degree of freedom analysis.

Keywords: *Reverse Engineering; Beautification; Geometric Constraints; Numerical Constraint Solver.*

1. Introduction

Reverse engineering a physical object is the extraction of information from the object that is sufficient for a particular purpose like reproduction or redesign. For an overview see [18]. Our aim is to reconstruct a boundary representation (B-rep) model of an engineering part from 3D range data, which has the desired geometric properties present in the original, ideal design. Our ultimate goal is an automated, intelligent 3D scanning system suitable for naive users and non-engineering applications as well as engineers.

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. The blends are treated separately, and are ignored in the rest of this paper. Valid B-rep models approximating these objects, referred to as *initial models*, can be generated by current reverse engineering systems [2]. However, the generated models suffer from various inaccuracies created by sensing errors arising in the data acquisition phase as well as approximation and numerical errors in the reconstruction process. Improving the precision of the sensing techniques and the reconstruction methods may reduce the errors, but some errors will always remain. As our intention is to create an ideal model for a physical object, we also have to consider additional errors introduced by possible wear of the object and the particular manufacturing method used to make it. To ensure that certain intended geometric regularities, like aligned cylinder axes or orthogonal planes, are present, they have to be enforced at some stage of the reverse engineering process.

Previous approaches augment the surface fitting step by constraint solving methods [1, 19] such that, for instance, two planes are fitted simultaneously under the constraint that they are orthogonal. Another approach is to identify features like slots and pockets whose approximate location and type are provided by a human and use these to drive the segmentation and surface fitting phase [17].

We propose 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. In [8, 9, 10] we presented various methods to find regularities approximately present in the initial model. This methods create a large set of potential regularities, which are not all likely to be consistent with each other. Ideally we wish to enforce a consistent subset of these regularities on the initial model which describes the ideal design of the model.

Numerical methods to select and solve a set of consistent geometric constraints to rebuild an improved B-rep model are presented. We start by summarizing the regularities considered and how they are expressed as constraints. For a general overview of geometric constraint topics see [4]. The regularities are prioritized according to the likelihood of the presence of the specific regularity types and the accuracy to which they are already satisfied in the initial model. The

priority is used to decide which regularities should be removed in case of inconsistencies. Furthermore, we detect simple inconsistencies between the constraints to determine *selection rules*, and select an initial set of regularities which satisfies these rules to reduce the number of inconsistencies.

To detect inconsistencies we add the regularities one at a time to the system according to the priority and also consider dependencies on other regularities. We solve the system by minimizing the error of the constraints for the regularities in a least-squares sense employing quasi-Newton optimization methods. If the system cannot be solved the newly added regularity is rejected together with any regularities dependent on it. Note that if there are rules indicating inconsistencies involving the rejected regularity we may be able to consider previously deselected regularities.

The presented approach is our general framework for beautification. Experiments show that the methods succeed in selecting and solving a constraint system suitable for improving models with simple inconsistencies and little interaction between major regularities. While these results are encouraging the algorithm is rather slow due to the large number of regularities involved. To address this and derive a practical method for beautification we propose modifications of the presented methods to improve the speed of the algorithm and handle more complex cases.

2. Regularities and geometric constraints

We describe the structure of the model and the potential regularities detected in the initial model in terms of geometric constraints. As a single regularity usually requires multiple geometric constraints we create a constraint set for each regularity. The sets can only be added or rejected as a whole in the selection process and we identify them with the regularities in the following. We also have required constraints describing dependencies and the topology of the model. As we do not consider topological changes they always have to be part of the constraint system.

The constraints are handled as relations between geometric objects. The objects are described by a type and a set of appropriate directional, positional, length and angular features represented as scalars and 3D vectors. There are basic features required to describe the object and extended features dependent on other features for additional properties as listed in Table 1. Root points for planes are created by taking the average of polygonal edge loops of planar faces. Furthermore, auxiliary objects (planes, cylinders, lines, directions, positions, lengths, angles) are used to express certain regularities with simple constraints.

The geometric constraints are represented as equations listed in Table 2. We require all direction vectors to be normalized. Distance and angle constraints use either a constant value, or a variable length or angle feature. They include equality, parallelism and orthogonality. The linear relations between angles or lengths also include equality.

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
Auxiliary line	Position, direction
Auxiliary plane	Position, direction
Auxiliary cylinder	Position, direction, radius
Auxiliary position	Position
Auxiliary direction	Direction
Auxiliary angle	Angle
Auxiliary length	Length

Table 1. Geometric objects and their features.

To enforce necessary dependencies between the base and extended features, and the topology of the model, we create various required constraints which must always be part of the constraint system. Feature dependencies include constraints setting the loop root points to be the average of the vertices in the loop, the length of a straight line to be the distance between the end-points, the circle segment angle to be the angle described by the centre and two vertices on the circle (which can be expressed as the angle between two auxiliary lines defined by the two position pairs), 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. Furthermore, for each direction we add a constraint normalizing its 3D vector.

For the topology of the model we add constraints requiring each vertex to lie on appropriate edges and faces. This does not fully specify the relation between adjacent faces. These relations are determined exactly by the regularities, e.g. constraining a cylinder axis to be parallel to a plane normal. As the regularity detection phase considers all possible relations between face features, and suggests multiple options for special relations, the relations between adjacent faces are part of the regularities. Furthermore, in order to solve the constraint system we consider the edges in the

$d^t d = 1$	Normalize direction d.
${d_1}^t d_2 = \cos(\alpha)$	Angle between normalized directions d_1, d_2 is constant or variable angle α.
$\|p_1 - p_2\| = \lambda$	Distance between positions p_1, p_2 is constant length λ.
$\|p_1 - p_2\| = \nu l$	Distance between positions p_1, p_2 is constant multiple ν of length l.
$p = 1/n \sum_{k=1}^{n} p_k$	p is average position of n positions $p_1, \ldots, p_n$.
$\sum_k \alpha_k s_k = 0$	Linear relation between lengths or angles s_k with constants α_k.
$s = \alpha$	Constant value α for length or angle s.
$p \in O$	Position p on geometric object O.

Table 2. Geometric constraints.

model as independent of the intersection between adjacent faces. They become auxiliary objects used to express certain regularities. When rebuilding the model all intersections are recomputed. Note that it is possible to add additional positions to describe the topology, especially for cases where there are no natural vertices. In the following we assume that constraints for the necessary relations between the features and the topology are always included in the constraint system.

In [9, 10] we presented various methods to detect local regularities of reverse engineered geometric models. These regularities are defined in terms of similarities between features derived from the B-rep model elements, and special values for these features. Besides directional, positional, length and angle features, we also use derived features such as axes (a position, direction pair). Similarities between features are expressed as cluster hierarchies 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 setting the minimum value when two angles or lengths should be considered as potentially different [8]. 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 in each cluster. The hierarchies are truncated by detecting a large jump in the tolerance values between the clusters.

The regularities with the priorities (see Section 3) are listed in Table 3. Each regularity is described by a constraint set. For the hierarchies the regularities are arranged in a tree, where a regularity can only be added to the constraint system if its children are also present. Furthermore, we add dependencies requiring other regularities to be present before we can add a particular one, e.g. requiring a parallel direction regularity to be present before a corresponding aligned axes regularity is added.

We have separate regularity hierarchies 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 the hierarchies we constrain the auxiliary object of the children to be equal to the auxiliary object of the parent.

In order to handle regularities for axes of planes (created by the plane normal and loop root points), cones, cylinders, tori, circles and ellipses we create auxiliary lines. These are required to be parallel to the direction of the object and the corresponding position of the object has to lie on the line.

Aligned axes are also represented as clusters describing the distances between the axes with respect to a suitable parallel direction cluster. For each cluster we create an auxiliary line and require that the positions of the axes in the cluster lie on this line. To represent the hierarchy we add an equal-position and a parallel-direction constraint per child requiring that the auxiliary line of the child is equal to the auxiliary line of its parent. The regularities are also marked as dependent on the parallel direction regularity.

For each cluster of axis intersections we create an auxiliary position and constrain it to lie on all axes in the cluster. For the hierarchy we constrain the auxiliary positions of the children to be equal to the parent's auxiliary position.

Furthermore, 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. Hence, we create an auxiliary line for each cluster and constrain the positions in the cluster to lie on this line. To express the hierarchy we require the lines of the children to be incident to the lines of the parent. Positions equal when projected on a line lie in the same plane. Thus, we create a similar structure using auxiliary planes. Both types of regularities are marked as dependent on the parallel direction regularity for the projection direction.

We also consider regular arrangements of parallel axes on grids, equidistant arrangements on lines and (partially) symmetrical arrangements on cylinders. We mark each of these non-hierarchical regularities as dependent on the regularity requiring the axes to be parallel. For a symmetrical arrangement on a cylinder we create an auxiliary cylinder and require that the positions of the axes lie on that cylinder and their directions are parallel to the cylinder direction. For each of the symmetrically arranged axis locations on the cylinder we create an auxiliary plane 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 each of the special value regularity for the cylinder radius we create a separate special value constraint.

For axes equally spaced along a line we create an auxiliary line, a plane p_0 with a normal orthogonal to this line and the direction of the parallel axes, and auxiliary planes q_k orthogonal to this line where the distances between them are integer multiples of some variable length feature. The axes are constrained to lie on p_0 and an appropriate q_l. For axes arranged on a grid we create a second auxiliary line and replace the single plane p_0 by planes p_k arranged in the same way as the q_k on the first line. Each axis is required to lie on a plane pair q_l, p_k. Similarly to the auxiliary cylinder radii we have separate regularities each specifying a special value for the length feature indicating the base distance between the planes along the line.

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 and 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 value and the angles between the directions projected on the plane defined by d_0 are integer multiples of $2\pi/n$. 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.

Furthermore, we also have lists of special values for angles between individual directions. For each special value and direction pair we create a separate constraint.

Finally there are regularities specifying special ratios between pairs of angle or length features and special values for these features. We create appropriate linear relations and constant value constraints.

3. Prioritizing regularities

As we expect to have inconsistent regularities, a mechanism is required to select regularities which are more likely part of the model's ideal design. We base the selection on a priority induced by a merit function. Note that the decision about which regularities to choose is non-trivial and often there is more than one choice.

To compute the priority $w(r)$ of a regularity r we take 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 a constant $w_c(r)$ indicating how common the regularity is. We get

$$w(r) = w_c(r)\left(c_e w_e(r) + c_q w_q(r) + c_b w_b(r)\right) \quad (1)$$

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 is $w_c(r)w_b(r)c_b$.

For $w_e(r)$ we combine the average angular error e_r in radians 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. By converting the angular error to length units using the maximum length L_m in the model we have

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

where c_l is a user-defined constant indicating the base length unit for the model, e.g. $c_l = 1$.

$w_q(r)$ describes the desirability or quality of the regularity if it could be enforced exactly on the model 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 the special values involved in the regularities have the form $v = \pm(n/m)^{1/(r+1)}b$ with integers n, m, and r and some base value b like π or 1. We evaluate the quality of special values using the function

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

where $q(b)$ is a constant in $[0, 1]$ evaluating the desirability of the base value b (e.g. $q(\pi) = 1$, $q(\pi/180) = 0.8$, $q(1) = 1, \ldots$), 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 the 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. $c_0 = 0.01$, $c_2 = 0.005$, $c_3 = 0.7$. w_{sv} favours 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. It is computed as

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

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

Regularity r	$\mathbf{w_c(r)}$	$\mathbf{w_b(r)}$	$\mathbf{w_q(r)}$
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 equispaced 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 ratio 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 3. Regularity priorities.

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.

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. If all consecutive positions are occupied the quality factor $w_{ra}(r)$ for this arrangement is $w_{sv}(m,1,b)$. 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 $m/(kn)w_{sv}(m/\gcd(m,kn),1,b)$ to w_{ra}. For axes arranged regularly on a grid we have two orthogonal directions. For each of the directions we can project the occupied positions in the grid on a line and handle the line like equispaced axes along a line. The sum 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

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

with constants t_w and t_p, e.g. $t_w=0.05$, $t_p=2$.

Depending on the regularity type we select appropriate quality factors and compute their weighted average. E.g. for parallel directions we mainly consider the quality $w_{sv}(0,1,\pi)$ of the parallel angle, but also prefer regularities with objects of the same geometric type. For planar symmetrically arranged directions, we put the main emphasis on the number of occupied positions and faces arranged in a loop as computed by $w_{ra}(r)$ and $w_a(F,1)$, but also consider the special angle value for the planar arrangement and the geometric types involved.

Table 3 lists the constants for w_c and w_b, and the way w_q is computed for the regularity types. w_q, w_c and w_b were derived from a part survey estimating the frequency of regularities in simple mechanical components [15]. The constants were in addition refined by the authors to adjust the priority order of regularities in various example models. Users may adjust these values depending on a particular application or personal preference.

While the ordering can be adjusted by changing the constants, the rather large number of constants makes it hard to predict the effect of the changes for a user who is not aware of the internal relations. A method which could choose the priority depending on a few multiple-choice questions presented to a user might improve this. Alternatively we might use neural networks to compute the priorities or use a belief network to make a decision about which of the inconsistent regularities to include.

4. Initial constraint selection and rules

The first stage of constraint processing attempts to resolve certain simple inconsistencies between the regularities. There are obvious inconsistencies between sets of reg-

ularities of the same type involving the same geometric objects but with different special values. These and similar inconsistencies can easily be detected and eliminated before we attempt to solve the system. Note that not all inconsistencies can be removed in this way as there are more complicated dependencies induced by multiple regularities between different geometric objects. As our regularities often specify multiple potential relations between the same geometric objects, eliminating simple inconsistencies reduces the number of constraint systems we have to check for solvability and thus speeds up the algorithm.

By traversing the list of constraints from all regularities we detect sets of constraints between the same geometric objects. We first note constraints between the same geometric objects with the same constants involved to avoid adding the same constraints more than once to the system. Further we detect regularities associated with constraints between the same objects 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 at the same time.

Further inconsistencies arise from incidence constraints. 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. In the current implementation we only consider incidences described by a single regularity.

Regularities which are part of the currently selected constraint system are marked active, and the others are marked inactive. The rules from the inconsistency detection stage can in general be expressed as selection 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_l elements of R_l are active for all $l = 1, 2$ for which $R_l \neq \emptyset$. 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.

Initially all regularities are marked as active. The rules are enforced one at a time by calling Algorithm 1. The method is called for each rule r with $D = \emptyset$. The algorithm 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 R_1 or 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 R_l such that at most n_l elements are still active. Let A_l be the set of elements we would have to deactivate in R_l to satisfy the rule and w_l be the largest priority in A_l. Then we choose to deactivate the set with the smallest w_l. This way we keep regularities with larger priorities active rather then trying to maximize 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. Alternative selections are possible, but also make it harder to influence the selection by adjusting the constants for the priorities.

The deactivations may allow the activation of other regularities as it may reduce the number of active elements in an R_l set of another, already enforced rule (step II and IV). If we activated any other regularity we have to check al-

Method `enforce` (r, D): Enforce rule r with the regularity sets R_1, R_2 and integers n_1, n_2 on the constraint system without activating any of the regularities in the set D.

I. Deactivate regularities in R_l to satisfy r, if there are more than n_l active regularities in R_l for all $l = 1, 2$ for which $R_l \neq \emptyset$ and note the deactivated regularities in d:
   1. For $l = 1, 2$ find all active sets A_l in R_l and remove the n_l elements with the highest priority.
   2. Find the set A_{l_0} with the smallest largest priority.
   3. Call `deactivate`(c, d) for all $c \in A_{l_0}$.

II. Find a set X of regularities which may be activated due to the deactivations by checking for each $c \in d$ and for each previously enforced rule e:
   1. If before step I for e there were at least $n_1 + 1$ active elements in R_1 and now there are less or R_1 is empty and there were n_2 active elements in R_2 and now there are less, add all inactive regularities from R_2 to X.
   2. If before step I for e there were at least $n_2 + 1$ active elements in R_2 and now there are less or R_2 is empty and there were n_1 active elements in R_1 and now there are less, add all inactive regularities from R_1 to X.

III. $D = D \cup d$.

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

V. Call `enforce`(r, D) for all already enforced rules r which may be affected by the activations in IV.

Algorithm 1. Enforce a selection rule.

The deactivations may allow the activation of other regularities as it may reduce the number of active elements in an R_l set of another, already enforced rule (step II and IV). If we activated any other regularity we have to check al-

ready 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 R_2 and more than n_1 elements of R_1 are active or R_1 is empty. In order of highest to lowest priority, the inactive regularities in R_2 which are in X are removed from X and we try to activate them recursively as long as less than n_2 elements of 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 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 $n_l + 1$ elements of R_l to be active for either $l = 1$ or $l = 2$ is added to the set of rules which have to be checked in step V of Algorithm 1.

For instance, assume that we have four regularities A, B, C, D which are all marked active and for which $w(A) < w(B) < w(C) < w(D)$. First we add a rule with $R_1 = \emptyset$, $n_1 = 0$, $R_2 = \{A, B\}$, $n_2 = 1$, i.e. either A or B, not both, can be active. `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. `Activate` initially succeeds in this, but the recursive check deactivates B again due to the second rule.

5. Numerical constraint solver

We employ numerical optimization methods to solve the constraint system [5]. After the initial regularity selection we solve the system using quasi-Newton methods minimizing the least-squares error of the constraints from Table 2. The main problem is to resolve all inconsistencies to get a solution satisfying the selected regularities exactly and eliminate redundancies to avoid numerical problems.

For quasi-Newton methods [3, 14, 16] we have a choice for the linear search method and for the approximation method for the Hessian matrix of the second partial derivatives. For the line-search method we considered Goldstein-Armijo and PWS [16]. While both methods 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 [16]. Further improvements to numerical stability, especially for inconsistent cases, were achieved by using a damped version of the BFGS method [11]. Using a hybrid method switching between BFGS and a Gauss-Newton step improved the convergence rates and still performed reasonably well with respect to numerical stability [14].

A very simple approach to solving the constraint system would be to find the minimum of the least-squares error. If the selected constraints do not contain any inconsistencies this performs well. However, when inconsistencies are present the solution does not exactly satisfy any constraints dependent on the inconsistencies, and sometimes the new model is worse than the initial model, as the optimization distributes the error over the constraints. One improvement is to use a weighted least-squares error function where the weights are the priorities of regularities. In order for the weights to work well, the errors of all constraints must be in the same error units. This approach causes the constraint equations to become more complicated which slows down the optimization method and also reduces the numerical stability. While the results are biased towards constraints with larger weights, the constraints are still not satisfied exactly as there is still some disturbance from inconsistent constraints with smaller weights. In order to satisfy the constraints exactly the inconsistencies have to be eliminated.

5.1. Detecting numerical inconsistencies

To detect numerical inconsistencies we add the regularities to the system one at a time in order of priority, and check if the optimization method still finds a solution. If so, the regularity did not introduce a (numerical) inconsistency and remains active. Otherwise, we permanently deactivate the regularity (and all regularities depending on it) and check if we can activate additional regularities due to the deactivation. Note that the *required* constraints discussed in Section 2 are always part of the constraint system.

Method `solve`(C, x_0, R): Find a solvable subset of the regularities in C and solve the constraint system it describes. x_0 is the vector of feature values derived from the initial model. R is a set of the required constraints and rules indicating when to use them.

I. Add the topological constraints in R to the set of accepted constraints S and set vector x to the values of features used by the constraints in S.

II. While C contains active regularities:
   1. Remove the active element c from C with the highest priority for which all constraints it depends on are in S.
   2. $S_1 = S \cup \{c\} \cup R(c)$, where $R(c)$ are the constraints from R required by c and set x_1 to the values in x and expand it if new features are added using the values from x_0.
   3. Solve $x_1 = \arg\min_z f(z)$ with the initial value x_1 and the least-squares error function f for C_1 also considering redundancies.
   4. If $f(x_1) \approx 0$, set S to S_1 and x to x_1.
   5. Otherwise, mark c and its dependants as permanently removed and check if we can activate other regularities in C.

Algorithm 2. Select a consistent constraint system and solve it.

When adding the regularities one at a time we have to consider the dependencies between them in addition to the priorities. We add the regularity with the highest priority for which all dependencies are already part of the system. This can be implemented efficiently using a priority queue.

Algorithm 2 is the consistent constraint selection and solving method. It is called with all regularities C marked as active or inactive according to the initial selection. The vector x_0 contains the values for all features involved in the constraint system based on the values in the initial model. When creating the constraint system only some of these values are actually part of it and we choose an appropriate subset of them to form the vector x. In addition we have a set of required constraints R. Most of these constraints describe the topology of the initial model and are always present in the selected constraint system S (step I). Some required constraints are only added if certain features or auxiliary objects are in S, e.g. we only normalize a direction if it is involved in some other constraint as well.

In step II.3, when consecutively testing regularities, 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. To identify numerical redundancies we use the method described in [13]. When a new constraint in $\{c\} \cup R(c)$ is already exactly 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.

The test if additional regularities can be activated in case a constraint was permanently disabled in step II.5 is similar to steps II – V of Algorithm 1 where d consists only of a single element.

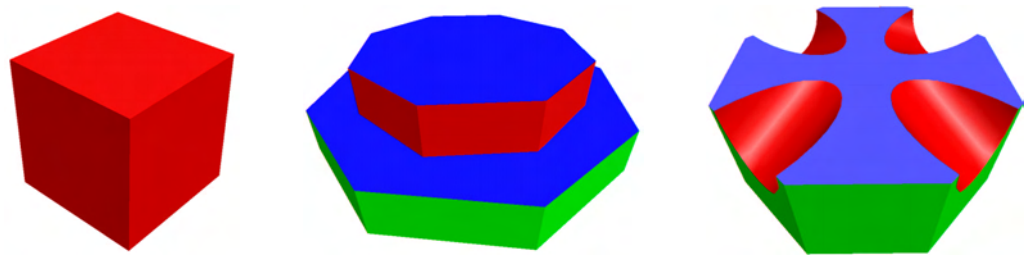
Algorithm 2 selects a numerically consistent constraint system. Only inconsistencies which cause an overall least-squares error larger than the tolerance used in step II.4 for the convergence test are detected. With appropriate parameters for the optimization method such that it always converges, no consistent regularities are removed.

6. Examples

We have tested the methods above using various reverse engineered models with the priority constants given in Section 3. In the following we summarize the results using the models shown in Figure 1 which also lists the number of regularities (including all clusters and special values) and the derived constraints in total detected in the initial model, and those initially selected and accepted for the final system.

The parallel and orthogonal relations between the planes of the cube (model (a)) were favoured by the priorities compared to all other angles only close to 0 or $\pi/2$. Hence, they were added first to the constraint system. Other potential regularities of (a) relate to equal edge lengths and special values for these lengths. As the priority mixed equal-length and special values the resulting object had only approximately equal edge lengths. By adjusting the priorities in favour of equal-lengths and w_q rather than w_e all the equal-length regularities were accepted. The selected special value for the edge length was correctly set to 2.0. But note that if the desired value were only close to 2.0, it is still likely that the improved model would have an exact edge length of 2.0.

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 hence imposed exactly on the model. For the edge lengths there were similar problems as 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. Special ratios between



	(a)		(b)		(c)	
	Reg.	Cons.	Reg.	Cons.	Reg.	Cons.
Total	89	261	382	1808	216	1263
Initial Selection	34	156	263	1384	156	910
Final System	23	132	117	712	93	586

Figure 1. Example models with number of regularities/constraints at different stages.

these values also supported undesired values. The redundancy and inconsistency checks correctly determined that only one angle between the groups of red and blue planes can be set. Choosing its value resulted in problems similar to the special length values.

In model (c) the green planes are arranged symmetrically in a plane and the red cylinders are arranged symmetrically on a cone. Due to the high priorities of these regularities they were imposed first together with the orthogonality relations. The edge lengths and the angle for the conical arrangement had the same problems as for models (a) and (b). 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 directions and the plane normals when projected in the same plane. The edge length regularities and the topological constraints avoided that this relation changed the topology or even broke the model, however.

The tests show that major regularities, independent of other major regularities, are easily identified and imposed on the model. Yet specific instances of each model relating to 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. For a more consistent choice the decision method would have to consider the global structure of the model. Furthermore, there may be hidden relations in the model which are broken if they are not detected explicitly as a regularity.

A major problem is the running time of the algorithm. On a single AMD Athlon MP 1200MHz with 512MB running GNU/Linux the algorithm took about 2 hours for object (a), 25 hours for object (b) and 23 hours for object (c). This is due to the large number of redundant and inconsistent constraints caused either by ambiguities or by larger errors, especially for objects (b) and (c). For each of these regularities the algorithm tries to solve the complete system even if no further regularities can be added. At about half the running time only redundant and inconsistent regularities were left.

To reduce the number of regularities which have to be checked we filtered the regularities by setting lower limits for the priorities and the constraint errors in the initial model. This reduced the time for object (a) to about 10 minutes and about 10 hours were required for object (b) and (c). More detailed interactive selection of constraints could reduce the time further. This, however, requires a lot of user interaction and specific interactive selection of constraints, which is already quite complex for objects (b) and (c) and also not our goal as we aim for minimal high-level user interaction.

Most of the time the algorithm requires is spent on solving the constraint system. As most of the regularities are approximately present in the model they only cause relatively small errors. For the optimization methods this means we have to use a strict convergence test and take a relatively small step in each iteration to avoid accepting or rejecting regularities due to numerical instabilities. This combined with the large number of constraint systems which have to be checked is the main cause for the long time required.

7. Structural inconsistencies

To improve the algorithm we require a fast solvability test when adding a new regularity. We seek a method which could indicate inconsistencies without solving the system. Furthermore, we would like to identify solvable sub-systems, which once a solution for the sub-system has been found can be replaced by a single rigid object such that the remaining constraint system becomes simpler. Degree-of-freedom (dof) analysis methods as, for instance, described in [6, 7, 12] address this problem. They detect struc-

tural inconsistencies and redundancies in a constraint graph derived from the constraint system without solving the system. They also identify solvable sub-systems which can be solved numerically.

Ways to include these methods are currently under investigation by the authors. We intend to replace step II.3 in Algorithm 2 by a dof analysis which only calls the numerical solver if a solvable sub-system has been found. As not all inconsistencies can be detected by a dof analysis a numerical solvability test is still required, but it is not required to call it for each regularity.

8. Conclusions

We have presented a method using some simple geometric reasoning and a numerical constraint solver to select a consistent constraint system from a large set of automatically generated constraints describing desired geometric regularities of a reverse engineered geometric model. Weakly related major regularities of simple models can be identified and imposed correctly on the model. However, the large number of possible combinations of regularities is the cause for slow performance of the method. The presented approach is our general framework for beautification. For a practical system more sophisticated selection methods and faster solvability tests are required.

In future work we will investigate methods to detect structural inconsistencies in the constraint system and combine this with the numerical methods. Furthermore, we will consider topological changes to the model and alternative methods for selecting the regularities in the presence of inconsistencies.

Acknowledgements

This project is supported by the UK EPSRC Grant GR/M78267. We would like to thank T. Várady and P. Benkő from the Hungarian Academy of Sciences and CADMUS Consulting and Development Ltd. for providing reverse engineering software and helpful discussions.

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] Å. Björk. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

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

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

[6] 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.

[7] 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.

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

[9] 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*, pp. 206–215. ACM Press, New York, 2001.

[10] 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, Genova, Italy, 7–11 May*, pp. 10–19. IEEE Computer Society Press, Los Alamitos, CA, 2001.

[11] 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.

[12] 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.

[13] Y.-T. Li, S.-M. Hu, J.-G. Sun. On the numerical redundancies of geometric constraint systems. In *Proc. Pacific Graphics*, pp. 118–123. IEEE Computer Society Press, Los Alamitos, CA, 2001.

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

[15] 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. of Computer Science, Cardiff University, 2001. `<uri: http://ralph.cs.cf.ac.uk/papers/Geometry/survey.pdf>`.

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

[17] 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.

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

[19] 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.