Finding Approximate Shape Regularities In Reverse Engineered Solid Models Bounded By Simple Surfaces

F. C. Langbein B. I. Mills 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,B.I.Mills, Dave.Marshall, Ralph.Martin}@cs.cf.ac.uk.

Abstract

Current reverse engineering systems are able to generate simple valid boundary representation (B-rep) models from 3D range data. Such models suffer from various inaccuracies caused by noise in the input data and algorithms. The quality of reverse engineered geometric models can potentially be improved by finding candidate shape regularities in such an initial model, and imposing a suitable subset of them on the model by using constraints, in a postprocessing step called *beautification*. Finding such candidate regularities is a necessary first step, and is discussed in this paper. Algorithms for analysis are presented which use *feature objects* to describe properties of faces, edges and vertices, and small groups of these elements in a B-rep model with only planar, spherical, cylindrical, conical and toroidal faces. The methods seek similarities between feature objects, e.g. axes which are parallel, for each property type. For each group of similar feature objects they also try to find a special feature object which might represent the group, e.g. an integer value which approximates the radius of similar cylinders. The feature objects used represent shape parameters, directions, axes and positions present in the model. Experiments show that the regularities found by these algorithms include the desired regularities. Although other spurious regularities which must be discarded in subsequent beautification steps are also produced, their number can be reduced by appropriate choice of tolerance values.

Keywords: Beautification; Shape Regularities; Similarity; Reverse Engineering; Geometric Interrogations and Reasoning.

1 INTRODUCTION

Reverse engineering the shape of physical objects has a variety of applications in design and manufacturing like reproduction or redesign. Our goal is to create a system that, for simple engineering objects, reconstructs a boundary representation (B-rep) model from a physical part with a minimum of human interaction suitable for naive users and non-engineering applications as well as engineers. The generated model should have all the intentional shape regularities present in the original design of the part.

Sixth ACM Symposium on Solid Modelling and Applications,
Ann Arbor, Michigan, June 4 – 8, 2001.

A valid B-rep model can be generated from dense 3D range data obtained from multiple views of an object using a laser scanner [3, 15]. The multiple views are merged into a 3D point set, which is triangulated and segmented into subsets representing the faces of the object [10]. For each subset, a surface approximating the points is fitted. These surfaces are then stitched to form a valid B-rep model. For this project we only consider planar, spherical, cylindrical, conical and toroidal surfaces which either intersect at sharp edges or are connected by fixed-radius rolling ball blends. There are reliable surface fitting methods available for these surfaces [11] and many interesting engineering objects can be generated using only such surfaces [13]. For our current project we focus on reverse engineering objects with up to about 200 primary faces, which is a realistic limit achievable with current technology.

Sensing errors from the data acquisition phase and numerical errors arising from the successive algorithmic steps distort the created model. As our intention is to recreate 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 the object. Even if we increase the accuracy of the sensing techniques and the surface fitting methods, some errors will remain.

Our approach to creating an ideal model from an approximated initial model is based on geometric reasoning. This means shape regularities of the model which are approximately present are determined and imposed on the model. One way of doing this is to use constrained surface fitting methods [16, 2]. This might, for instance, require that two planes are fitted simultaneously under the constraint that they are orthogonal. A second approach is to identify features like slots and pockets whose approximate location and type are provided by a human being and use this information to improve the results of the segmentation and surface fitting phase [14].

Our method attempts to improve the approximate models without human assistance in a postprocessing step, which we call *beautification*. Instead of improving the model during the surface fitting stage, we aim to analyse the generated B-rep model to find approximate regularities and adjust the model accordingly (see Figure 1). Given sufficient constraints generated from the regularities, an ideal model may be deduced from the constraints without further reference to the measured point data.

In this paper we present methods to find approximate shape regularities in B-rep models. Later work will address the constraint imposition strategy. Our shape regularities are defined in terms of similarities between properties of B-rep model elements, and similarities between these properties and given special properties. For instance, we look for approximately equal radii of cylinders and we also try to find a special value, like an integer, which is approximately equal to the average radius. The regularities are local in the sense that they relate to single properties of one or only a small number of B-rep model elements. A different approach taking a global view of looking for symmetries of the complete model is presented in a companion paper [12].

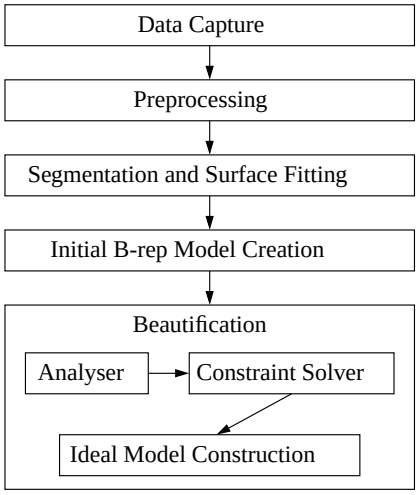


Figure 1: Reverse Engineering Phases

We assume that the regularities are sufficiently distinct from the approximation error of the solid model, so that they can be distinguished from the noise by using various thresholds. Our methods produce a large set of possible regularities the ideal model might possess instead of looking for a small set of very likely regularities. As the potential regularities are likely to contain mutually inconsistent requirements, a subsequent decision process will have to choose a maximal, consistent set of regularities which are most likely present in the ideal model.

The following section introduces the types of regularity detected by our methods. Then we discuss the notion of similarity as the fundamental concept for our methods. In the remaining sections we describe our algorithms for finding regularities and show how they perform using several examples.

2 SHAPE REGULARITIES AND FEATURE OBJECTS

We express shape regularities in terms of properties of face, edge and vertex groups of the B-rep model. A property of a group of one or more of these elements is represented by a *feature object*, which is handled as a typed object separate from the B-rep model, but refers to the elements of the group. The *type* of a feature object is given by the property it represents. The feature objects are stored as vectors of some dimension d depending on the type. For instance, the axis of a cone or cylinder forms a feature object of type *axis* represented by a direction and a position. The radius of a cylinder and the semi-angle of a cone are two other types of feature object. Note that a single B-rep model element can generate several different feature objects of various types. Also note that further feature objects can be created from groups of feature objects, or from groups containing feature objects and B-rep model elements. For instance, axes as feature objects may generate intersection points as further feature objects.

The regularities we look for are derived from a survey [13] identifying which regularities are commonly present in simple engineering parts. An overview of the shape regularities we seek is given in Table 1. The number in the last column indicates how common the regularity is, from 5 for being nearly always present to 1 for rare, as determined manually. In particular, we look for approximately equal shape parameters from surfaces and edges such as radii, lengths and angles. We try to find special values for these parameters, like integers and integer multiples of π, and simple integer relations of the form $n_2 p_1 = n_1 p_2$ between shape parameters p_l with integers n_k.

The directions in a B-rep model such as surface normals and directions of axes, referred to as axis directions in this paper, are clustered into parallel sets. In addition we look for special directions with respect to which all elements of a set of axis directions have the same angle. The sets generated in this way can be interpreted as axis directions in a plane or on a cone, and we check if they are arranged symmetrically (see Figure 2).

For axes, i.e. for axis directions which are associated with a position, we look for common intersection points and aligned axes. For parallel axes, we seek regular arrangements of these axes along a line or a grid with regular spacing. Parallel axes may also be arranged symmetrically on a cylinder.

Furthermore, we examine positions obtained from vertices, centres of spheres and tori, apices of cones, axis intersection points, etc. for approximately equal locations and for regular arrangements on a 2D or 3D grid. We also seek positions which are equal after projection onto a special plane or line derived from the main axis directions present in the model.

3 SIMILARITY AND SPECIAL FEATURE OBJECTS

We define a similarity measure indicating how close two feature objects are to each other. This is used in the hierarchical clustering algorithm described below to find similar feature objects and to find special feature objects similar to a given feature object. For a set X of feature objects of the same type, a similarity measure is defined as a symmetric, non-negative function $\delta : X \times X \to \mathbb{R}_0^+$, such that $x_1 = x_2$ implies $\delta(x_1, x_2) = 0$. We call two feature objects $x_1, x_2 \in X$ ε-similar (with respect to δ) if $\delta(x_1, x_2) < \varepsilon$, ($\varepsilon \in \mathbb{R}^+$).

This definition suffices for a measure to decide if a feature object is close to a special feature object. However, for the clustering algorithm δ should be a similarity metric, i.e. it should also fulfil the triangle inequality and $\delta(x_1, x_2) = 0$ should imply that $x_1 = x_2$. Depending on the types of feature object and regularity, appropriate similarity measures and metrics are defined later.

To represent groups of similar feature objects by a single feature object, we need an averaging method `avg` to combine two similar feature objects of the same type. Given two ε-similar feature objects x_1 and x_2 of the same type and two positive weights ω_1, ω_2, it generates a new average feature object $x_{\text{avg}} = \texttt{avg}(x_1, \omega_1, x_2, \omega_2)$, which represents x_1, x_2 such that $\delta(x_{\text{avg}}, x_l) < \varepsilon$, ($l = 1, 2$).

We use a hierarchical clustering algorithm to find similar feature objects of the same type. Given a set of feature objects $X = \{x_l\}$, a similarity metric δ, and an averaging method `avg`, we wish to create a partition $\{C_k\}$ of X such that each cluster C_k contains feature objects which are t_{a}-similar for some tolerance t_{a}. C_k is then represented by some feature object c_k and a tolerance t_k as the maximum distance between the elements of C_k and c_k. Note that we do not limit the number of clusters C_k, but generate as many clusters as required by the tolerance t_{a}. The width of each cluster is limited by t_{a}.

To form a hierarchical structure, we create a nested subset structure for the clusters C_k. We partition a C_k further into disjoint sub-clusters S_j represented by feature objects s_j and with a tolerance t_j such that $\delta(s_j, x_l) < t_j$ for $x_l \in S_j$. The sub-clusters have to be sufficiently distinct from each other, i.e. for each pair S_{j_1}, S_{j_2}, ($j_1 \neq j_2$), the condition $\delta(s_{j_1}, s_{j_2}) - t_{j_1} - t_{j_2} \geq t_{\text{d}}$ has to be fulfilled for some tolerance t_{d} satisfying $0 < t_{\text{d}} \leq t_{\text{a}}$. The sub-clusters are again partitioned if we can find further subsets fulfilling a similar condition.

Shape Parameters	Equal shape parameters.	5
	Special values for shape parameters.	3
	Simple integer relations between shape parameters.	4
Axis Directions	Parallel axis directions.	5
	Sets of axis directions which have the same angle relative to a special direction (axis directions on planes and cones).	4
	Symmetrical arrangements of axis directions.	4
Axes	Axes intersecting in a point.	3
	Aligned axes.	3
	Parallel axes arranged along lines and grids with regular distances between them.	3
	Parallel axes arranged symmetrically on cylinders.	2
Positions	Equal positions.	2
	Regular distances between positions arranged on a line, a 2D grid or a 3D grid.	3
	Equal positions under projection.	3

Table 1: Shape Regularities

The hierarchical clustering reveals groups of feature objects in a cluster which are considerably closer to each other than to the other elements of the cluster. This can be used in the subsequent steps to decide which regularities should really be enforced and makes the tolerances maximal rather than optimal. Maximal tolerances ensure that all intended regularities are found, but they also cause the detection of regularities which are not present in the ideal model and which could be avoided by an optimal choice of tolerances.

A variety of approaches to the clustering problem exist [8]. The most common approach is the agglomerative technique, which starts with the smallest value of $\delta(c_l, c_k)$, and combines the two elements to form a new element $\hat{c}$ which replaces c_l and c_k. Clusters and sub-clusters are formed accordingly as this is repeated until only one cluster remains, or the distance between the clusters is too large. A brute force solution searching for the closest elements each time requires $O(n^3)$ time.

To improve the brute force approach we use a distance matrix containing the distances between feature objects of the same type and maintain a quad-tree like data structure to keep track of the closest pair [6]. We store the distances $\delta(c_l, c_k)$ between the elements c_l and c_k representing clusters in a matrix D for $l < k$. The clusters are grouped arbitrarily into pairs (C_l, C_{l+1}) and the distance between two pairs is defined to be the minimum distance between the four clusters of the two pairs. These pairs define a new closest pair problem of half the size, which can be solved recursively. The closest pair in D is at the root of this data structure. After initializing we have to update at most two rows and two columns of each of the matrices defining the closest pair problems for each update operation. This method requires $O(n^2)$ space and time assuming that δ and `avg` require constant time and space.

Additionally we need methods to find special feature objects similar to an average feature object, e.g. the average radius of a group of similar cylinders could be approximately an integer. Note that there might be more than one feature object close to it within a given tolerance. In the following sections details of clustering and finding special values are given for different types of feature object.

4 SHAPE PARAMETERS

The first feature objects we consider are shape parameters (see Table 2(a)). These describe the shape of an element independently of its location and orientation. Each shape parameter is treated as a separate feature object and more than one shape parameter feature object may arise from a single element.

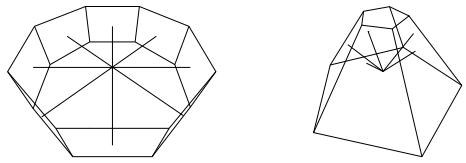


Figure 2: Planar and Conical Angle-Regular Directions

To find similar parameters we use our hierarchical clustering algorithm. As it is obviously not very useful to compare certain parameters like the semi-angle of a cone with the radius of a sphere, we assign a type to each parameter (see Table 2(b)) and only cluster parameters of the same type. Note that we could add further parameters dependent on the parameters listed, like the sum and the difference of the major and minor radius of a torus. Since the values for angle and length parameters are on different scales, the tolerances used for clustering depend on the parameter type. We use t_{la} and t_{ld} for lengths and t_{aa} and t_{ad} for angles. The averaging method for all parameter types computes the weighted average of real numbers.

We assume that fixed-radius rolling ball blends are identified in the B-rep model by the surface fitting methods and thus we handle them as edge attributes and not as surfaces. But we can treat blend radii as separate shape parameters of type *blend* and find similar blend radii and special values for blend radii.

Note that the parameters from edges might depend on surface parameters. For example, the radius of a circle which is the boundary of the top of a cylinder has to be equal to the cylinder radius. Such relationships are better handled in the constraint solver phase where they should become obvious.

The result of clustering the shape parameters is a hierarchical structure of similar shape parameters sorted by type. In the following subsections we present methods to find special parameter values and special integer relations between parameter values. Both problems can be reduced to finding a simple fraction approximating a real number.

4.1 Special Parameter Values And Integer Relations

For each of the average shape parameters of the clusters and subclusters we try to find a simple fraction as special value approximat-

Element	(a) Shape Parameter	(b) Type	(c) Axis Direction
plane	—	—	normal
sphere	radius	length	—
cylinder	radius	length	direction of the axis
cone	semi-angle	angle	direction of the axis
torus	major radius	length	direction of the axis
	minor radius	length	
straight edge	distance between end points	length	direction of the line between the end-points
circular edge	radius	length	normal of plane in which the circle lies
	angle of the circle segment	angle	
elliptical edge	—	—	normal of plane in which the ellipse lies

Table 2: Shape Parameters and Axis Directions

ing its value. As there might be more than one special value within a given tolerance, we create a list of appropriate special values for each shape parameter.

Let v be an average shape parameter from a parameter cluster or sub-cluster. Depending on the parameter type we also have a tolerance t_{a} (either t_{la} or t_{aa}) and we choose a family of functions $f_l : \mathbb{R} \to \mathbb{R}$ which represents the scales on which we look for simple fractions. For angle parameters we use the two functions $f_\pi : v \mapsto v\pi$ and $f_{\text{t}} : v \mapsto \arctan(v)$. For length parameters the family is defined by $f_{K_l} : v \mapsto vK_l$ where the K_l are base units for length measurements like 1.0, 0.1 or 2.54 (cm to inch conversion).

To find a special parameter value for v, we look for integer pairs n_k, m_k such that $v \approx f_l(n_k/m_k)$ for all specified functions f_l. $s_k = f_l(n_k/m_k)$ is a valid approximation of v if $|s_k - v| < t_{\text{a}}$. Hence, to find a special parameter value for v, we try to find fractions n_k/m_k which approximate $w_l = f_l^{-1}(v)$ for each function f_l within a tolerance t derived from t_{a}. For linear $f_l : v \mapsto vK_l$ we use the tolerance $t = t_{\text{a}}/K_l$. For f_{t}, we use t_{aa} as tolerance with the condition $||v| - \pi/2| > t_{\text{aa}}$ assuming that $v \in [-\pi, \pi]$. The algorithm for determining simple fractions for w_l with tolerance t is described in Section 4.2. Note that if we consider multiple functions f_l, we have to compare the results for different functions and eliminate duplicates, e.g. $\arctan(1) = \pi/4$.

The algorithm to find simple fractions approximating a real number can also be applied to finding integer relations between shape parameters. For each pair of clusters C_k and C_j represented by c_k and c_j respectively we look for a set of integer relations n_l/m_l which approximate $w = c_k/c_j$. To have a consistent input for the algorithm, we order the clusters such that $c_k > c_j$. If $|n_l/m_l - w| < t_{\text{ra}}$ for some tolerance t_{ra}, we accept n_l/m_l as a valid approximation for w. To avoid finding equality integer relations, i.e. $1/1$ relations, we only generate relations between clusters and sub-clusters which do not belong to the same cluster.

4.2 Finding Simple Fractions

In the previous subsection we reduced finding special parameter values and special integer relations between parameters to finding a list of special fractions for a real number w in the open interval $(w - t, w + t)$ with tolerance t. We assume that integer values are always special for w and that w is non-negative.

Although finding integer relations between real numbers [7] or recognizing numerical constants [1] are well studied related problems, such methods assume that the real numbers are exact or that a very close approximation is required. We try to find special fractions which lie in an interval containing w.

If $1/(2t) < m$, then more than one n/m for a fixed m could be in the interval $(w - t, w + t)$. To avoid this ambiguity we set the maximum allowed value for m to $M_0 = \text{floor}(1/(2t))$. To limit the number of possible special fractions found, M_0 should not be larger than M_{max}, say 10. Also note that if t is larger than $1/2$, we would actually ignore the integers, so M_0 should at least be 1. Optionally, if we have to work with large t, we can set a minimum M_{min} for M_0 which is larger than 1.

If M_0 is small, we could simply multiply w by each $m \leq M_0$ and check if $|w - n/m| < t$ with $n = \text{round}(wm)$, i.e. check if the relation n/m is within the tolerance limits. However, for large M_0, this becomes expensive as we have to check many fractions for a large number of real numbers w.

To overcome this problem, we derive a method for larger M_0 by combining this simple method with continued fractions [9] as listed in Algorithm 1. We start by approximating w by $a_0 + x_0$ where $a_0 = \text{round}(w)$. The error x_0 is approximated by using the simple method recursively. Starting with $b_l = 1$ in recursion level l, we get $x_l = b_l/a_l + x_{l+1}$ for $a_l = \text{round}(b_l/x_l)$ and error x_{l+1}. We get additional approximations by increasing b_l by one, until a_l is larger than some limit M. The process is repeated recursively for each error x_{l+1} until the error is smaller than some tolerance t_{rmin}. Table 3 contains an example for 0.63 with $t = 0.05$, $M_0 = 5$ and $t_{\text{rmin}} = 0.01$ where the special values generated are marked with $[\to \cdot]$. For the intermediate result $1/2$ we do one recursion step to find special values for the error 0.13.

In order to use this method for an arbitrary real number, we must do some preprocessing. First the sign and the closest integer a_0 to w have to be found. If $x_0 = |w - a_0| < t_{\text{rmin}}$, the integer is the only special value for w and we stop immediately. Otherwise we continue with x_0. To process small denominators, we seek simple fractions for $1 - x_0$ if $x_0 < 0.5$, as functions of the type $b/x_0 - \text{round}(b/x_0)$ used in the recursive algorithm are more likely to generate small denominators for $x_0 \in (0.5, 1)$. Based on the tolerance t for w, we also generate an initial limit M_0 for the allowed denominator within the bounds explained earlier.

To find simple fractions for x_0, we call the recursive algorithm `rec_frac` listed in Algorithm 1, where we have to consider all the results of the preprocessing steps to generate the final special values. In recursion step l we solve the problem to find simple fractions p/q for a real number x_l with $q < M$ using the simple method, where the limit M for the denominator depends on the recursion level starting with $M = M_0$. For x_0 we already have a simple fraction n/m such that $x_0 = n/m \pm x_l$. Whether x_l has to be added or subtracted from n/m is indicated by the flag `neg_frac`. For each $b = 1, 2, \ldots$ we check the denominator $a = \text{round}(b/x)$ until $a > M$. The fraction b/a is an approximation for x_l. Depending on `neg_frac` we add it to n/m or subtract it, generating a new approximation p/q for x_0. If this approximation is close enough, it is

I. The function has been called as `rec_frac(`x`,`n`,`m`,neg_frac,`M`)` where x is the value which has to be approximated by a fraction, n, m are the two integers representing the fraction n/m found so far, `neg_frac` indicates if x has to be added or subtracted from n/m and M is the maximum denominator allowed at this recursion level with the initial limit being M_0.
II. Let $b = 1$.
III. While the denominator $a = \text{round}\,(b/x)$ is not larger than M:
   1. If $a > b$:
      A. Let $r = x - b/a$.
      B. If r is negative, set `neg_r` to `true` and $r = -r$. Otherwise set `neg_r` to `false`.
      C. If `neg_frac` is `true`, then the new numerator is $p = na - mb$, otherwise $p = na + mb$.
      D. The new denominator is $q = ma$.
      E. If $r < t$, then add p/q to the list of special values, if it is not already in it.
      F. If $r > t_{\text{rmin}}$ and p/q was not already in the list of special values, call `rec_frac(`r,p,q`,neg_r,`M_0M`)`.
   2. Let $b = b + 1$.

Algorithm 1: Recursive Algorithm `rec_frac` for Finding Simple Fractions

0.63	$= 1/2$		$+$	0.13	$= 1/8$	$[\rightarrow \mathbf{5/8}]$	$+$	0.005
					$= 2/15$	$[\rightarrow \mathbf{19/30}]$	$-$	0.003333
					$= 3/23$	$[\rightarrow \mathbf{29/46}]$	$-$	0.000435
	$= 2/3$	$[\rightarrow \mathbf{2/3}]$	$-$	0.036667				
	$= 3/5$	$[\rightarrow \mathbf{3/5}]$	$+$	0.03				

Table 3: Finding Special Values for 0.63 with $t = 0.05$, $M_0 = 5$ and $t_{\text{rmin}} = 0.01$

added to the list of special values. If the error $r = |x - b/a|$ is still larger than some tolerance t_{rmin}, and p/q was not already in the list of special values, we call `rec_frac` for r recursively with a new limit M_0M for the denominator. By multiplying M with the initial limit M_0, we ensure that we can still find fractions for a smaller real number within the denominator limit. Note that we always reduce fractions n/m by their greatest common denominator, in order to keep the values small and to ensure that we add *simple* fractions to the list of special values.

Also note that the algorithm can only miss special values close to w if their denominator is larger than M_0. The precision increases with the depth of the recursion.

5 AXIS DIRECTIONS

Directions, like normals of planes, arising from the elements of a B-rep model, provide the basis for another class of regularities. Note that some of these directions are also associated with a position. In this section we are only interested in the angular arrangement of the directions. Directions with positions are covered in Section 6.

We extract directions from B-rep model elements as feature objects represented by unit vectors (see Table 2(c)). Opposite directions, i.e. the unit vectors d and $-d$, are identified. This space of directions is the real projective plane P_2 and can be represented by the unit sphere with identified antipodal points. In this context we call its elements *axis directions*. Regular arrangements of the axis directions correspond to points and circles in P_2. The way in which the directions are arranged on the circles might create further regularities.

By using the hierarchical clustering algorithm, we can find parallel axis directions represented by points in P_2. For the simliarity measure, we define the angle between two axis directions d_1 and d_2 as $\angle(d_1, d_2) = \arccos(|{d_1}^t d_2|)$, i.e. the smaller angle between d_1 and d_2. The weighted average between two axis directions d_1 and d_2 with weights ω_l is $\left(\omega_1 d_1 + \text{sign}({d_1}^t d_2)\omega_2 d_2\right) / (\omega_1 + \omega_2)$. The two tolerance values required for clustering are t_{aa} and t_{ad} as defined earlier. The resulting parallel axis direction clusters represent the main directions present in the solid model. Even if the number of directions extracted is large, we expect to find only a limited number of *different* directions.

In the following subsections we discuss the arrangements represented by axis directions lying on circles in P_2 and regular arrangements on these circles. Axis directions that are on a great circle of the sphere represent directions that are orthogonal to another direction and thus lie in a plane. Axis directions that are on a small circle of the sphere represent axes that are on a cone. The arrangement of the axis directions in the plane or the cone can be symmetric. We call these *planar* or *conical angle-regular* (see Figure 2).

5.1 Regular Arrangements Of Axis Directions

A set of axis directions $\{d_l\}$ on a circle satisfies the equation system $|{d_l}^t x| = a$, where $x \in P_2$ is the centre of the circle. For $a = 0$, the directions lie in a plane with normal x, which we call an *axis plane*. For $a \in (0, 1)$, we have a cone with axis x, which we call an *axis cone*. Note that for a plane, taking the absolute value of ${d_l}^t x$ is not required, and we can drop it for a cone if all axis directions have the same orientation relative to x.

To find the sets of axis directions d_l lying in an axis plane, we cluster the normals representing all axis planes generated from each pair of linearly independent axis directions. The clustering is done in the same way as the clustering of parallel axis directions, but we employ the clustered parallel axis directions instead of all axis directions. This not only reduces the number of normals generated, but also avoids approximately linearly dependent axis directions. Therefore, we also only consider the main clusters of parallel axis directions and not the sub-clusters.

A similar method is used to find axis cones. For each triple d_1, d_2, d_3 of axis directions representing parallel axis direction clusters, we generate an axis cone with axis direction c and semi-angle α by solving the linear equation system ${d_l}^t x = 1$, $(l = 1, 2, 3)$, from which we get $\alpha = \arccos(|x^t d_1|/\|x\|)$ and $c = \alpha x$. As the d_l are subject to errors, we avoid flat axis cones that actually represent axis planes or axis cones that represent parallel axis directions by rejecting axis cones for which $\alpha < t_{\text{aa}}$ or $|\pi/2 - \alpha| < t_{\text{aa}}$.

I. Let $\{d_l\}$ be a set of m objects for which distance-regular subsets with respect to the condition $\texttt{reg}_n$ are sought.
II. Compute the distances α_{lk} between objects d_l and d_k for $l < k < m$.
III. For all reference objects d_{j_0} with $j_0 < m$ and for all distances α_{lj_0} with $j_0 < l < m$:
   1. Find the candidates $\beta_{j_0 n}$ for some $n \in \mathbb{N}$ such that $\texttt{reg}_n(\beta_{j_0 n}, \alpha_{lj_0})$ is `true`.
   2. Add $\beta_{j_0 n}$ to the list of candidates for d_{j_0} unless it is an integer multiple of one of the $\beta_{j_0 n_0}$ already in the list.
   3. If $\beta_{j_0 n}$ has been added to the list, remove any $\beta_{j_0 n_0}$ from the list which are integer multiples of $\beta_{j_0 n}$.
IV. For each reference object d_{j_0} with $j_0 < m - 1$ consider the subset $\{d_{j_0}, \ldots, d_m\}$ and for each candidate $\beta_{j_0 n}$ with $n = 1, \ldots, N_{j_0}$:
   1. For each object d_j with $j_0 < j \leq m$:
      A. If for $f = \alpha_{lj_0}/\beta_{j_0 n}$, we have $|\operatorname{round}(f)\,\beta_{j_0 n} - \alpha_{lj_0}| < t_{\text{a}}$, then do:
         a. Record object d_j to be the multiple $\operatorname{round}(f)$ of the base distance. It is stored as a multiple $\operatorname{round}(f)$ under the following conditions.
         b. If there is already an object o as the multiple $\operatorname{round}(f)$ and o is the parent of d_j, then only replace o by d_j if d_j is sufficiently closer to the multiple as indicated by t_{pd}.
         c. If there is already an object o as the multiple $\operatorname{round}(f)$ and d_j is the parent of o, then only replace o by d_j if o is not sufficiently closer to the multiple as indicated by t_{pd}.
         d. If there is already an object o as the multiple $\operatorname{round}(f)$ and d_j is not related to it in the hierarchical structure, then replace o by d_j if d_j is closer to the multiple.
         e. If there is no object o as the multiple $\operatorname{round}(f)$, then make d_j this object.
   2. If the arrangement in the distance-regular subset for $\beta_{j_0 n}$ is regular (see text), note it as a regularity. In addition remove integer multiples of $\beta_{j_0 n}$ from the base distance candidate list for the reference objects d_l which are in the current regular subset.

Algorithm 2: Finding Distance Regularities

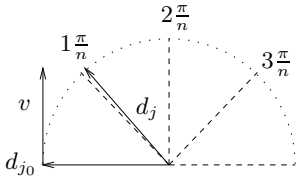


Figure 3: Planar Angle-Regular

The axis cones represented by pairs (c_l, α_l) are clustered using $\sqrt{\angle(c_1, c_2)^2 + (\alpha_1 - \alpha_2)^2}$ as similarity measure and the tolerances t_{aa} and t_{ad}. The averaging method for two cones generates an average cone from the weighted average of the axis directions and the weighted average of the semi-angles.

Note that we cluster the description of the axis planes or cones, and not the axis directions used to create them. For instance, three axis directions d_1, d_2 and d_3 generate three axis planes $\mathrm{p}(d_1, d_2)$, $\mathrm{p}(d_1, d_3)$ and $\mathrm{p}(d_2, d_3)$. If $\mathrm{p}(d_1, d_2)$ and $\mathrm{p}(d_1, d_3)$ are combined into a single cluster, $\mathrm{p}(d_2, d_3)$ is not automatically added to the same cluster. As $\mathrm{p}(d_1, d_2)$ and $\mathrm{p}(d_1, d_3)$ are only approximately the same plane, $\mathrm{p}(d_2, d_3)$ is not necessarily the same plane. Hence, the way the axis planes are generated creates dependencies between them and the clusters might not have a transitive structure with respect to the generating axis directions. We get similar dependencies for the axis cones.

Any reasonable choice of tolerances should ensure that dependent axis planes or cones are in the same main cluster even if they are in a different sub-cluster. If this is not the case, then the tolerances are too restrictive for the errors in the model or they are too relaxed, causing the combination of axis directions which are not related.

Handling these dependencies in the clustering algorithm would be rather expensive as to do so would require adding all other clusters dependent on the new cluster when combining two clusters to form a new one. Hence, combining the clusters could no longer be done in constant time. Instead, the dependencies can be handled by a post-processing step which combines dependent (sub-)clusters.

We can further improve the quality of the approximation of the regular arrangements once we know the sets of axis directions forming them. Given a set of axis directions d_l which are approximately on a circle in P_2, we get a linear equation system ${d_l}^t c = \alpha$ for $l = 1, 2, \ldots, n$. For $\alpha \neq 0$, i.e. for axis cones and parallel axes, we can solve the above system in a least-squares sense using the pseudo inverse D^* of the matrix D formed by the d_l, which improves the approximation quality of the axis cones and parallel axes [4], again solving $Dx = 1$ and then computing α and c as above. Using this approach for axis planes creates a constrained optimization problem $\|[{d_1}^t c; \ldots; {d_n}^t c]\|_2 \rightarrow \min$ for c subject to the condition $\|c\|_2 = 1$ (or at least $\|c\|_2 \gg 0$) which is more difficult to solve [4].

5.2 Angle-Regular Axis Directions

Examining axis planes and cones further can reveal symmetrical arrangements of the directions (see Figure 2). Given a set of axis directions in a plane or a cone, we look for subsets such that the angles between the axis directions in a subset are integer multiples of a base angle β. The subsets can be incomplete in the sense that not all multiples of β are present. The main problem is to identify appropriate subsets of multiples which satisfy this condition approximately. We first describe the exact problem for axis planes, then reformulate it more generally to present an algorithm which can also be used for axis cones and other regularities introduced later.

Let $\{d_j\}$ be a set of m axis directions in an axis plane and let α_{lk} be the angle between d_l and d_k. We call the d_j *angle-regular* if there is a $\beta \in \{\alpha_{lk}\}$ such that $\beta = \pi/n$ for $n \in \mathbb{N}$ and for each α_{lk} there is an integer p such that $\alpha_{lk} = p\beta$. This means that the angle between some reference direction $d_{j_0} \in \{d_j\}$ and each other direction is a multiple of π/n. As we identify opposite directions, we only consider angles in the interval $(0, \pi]$. Based on which multiples of β are present, we decide whether an angle-regular set is considered to be a regularity. We also avoid too small base angles β by setting a maximum N_{max} for n, which should be smaller than $\pi/(2t_{\text{aa}})$. Note that in the approximate case the base angle $\beta = \pi/n$ is only close to some α_{lk} and not necessarily a member of the set.

At this stage we do not look for base angles which are not approximately present as an angle between axis directions. For instance, if we have two approximate angles $2/6\,\pi$ and $5/6\,\pi$ relative to some direction, the underlying base angle $\pi/6$ is not detected. An efficient implementation for these cases is left as future work. One might, for instance, use approximate integer relations.

From a more abstract point of view, the $\{d_j\}$ are objects which can be ordered with respect to an arbitrary reference object d_{j_0} such that we get a set of distances α_{lj_0} between d_{j_0} and d_l for all l. To find a base distance β similar to the base angle in the approximate case, we look for all β fulfilling a regularity condition $\texttt{reg}_n(\beta, \alpha)$ for an $\alpha \in \{\alpha_{lk}\}$. This condition depends on the object type. In this more general case we call the set *distance-regular*.

Given a set of m objects $\{d_j\}$ we look for a minimum number of subsets which are approximately distance-regular with respect to $\texttt{reg}_n$. The algorithm consists of three main steps (see Algorithm 2). First we compute all distances α_{lk}, $(k < l < m)$, between the objects, where a column j_0 in that matrix with elements below the diagonal represents the distances to the reference object d_{j_0}. From these distances we derive a set of possible $\beta_{j_0 n}$ for each d_{j_0} using the condition $\texttt{reg}_n$. Note that as we look for approximate distance regular arrangements, a single α_{lk} can generate more than one candidate α depending on the tolerances. In the third step we try to find distance-regular subsets by checking the distances α_{lj_0} for each reference object d_{j_0} and all base distance candidates $\beta_{j_0 n}$.

Computing the distances between the objects in the first step depends on the type of the objects. For axis directions in a plane, the distances α_{lk} are the angles between the axis directions d_l and d_k. We have to take care to choose the angle that lies consistently to the right of the reference object d_l with respect to the normal q of the axis plane (see Figure 3). With $v = q \times d_k$ we have $\alpha_{lk} = \arccos({d_k}^t d_l)$ if ${v}^t d_l \geq 0$ and $\alpha_{lk} = \pi - \arccos({d_k}^t d_l)$ if ${v}^t d_l < 0$. This also allows us to identify which of the $k\pi/n$, $(k = 0, \ldots, n-1)$, directions a particular d_j occupies for some $n \in \mathbb{N}$.

In the next step, candidates $\beta_{j_0 n}$ for base distances are derived from each α_{lj_0} such that $\texttt{reg}_n(\beta_{j_0 n}, \alpha_{lj_0})$ is satisfied for some $n \in \mathbb{N}$. A particular $\beta_{j_0 n}$ is added to the list of base angle candidates for d_{j_0} unless it is an integer multiple of some other $\beta_{j_0 n_0}$ which is already in the list. If any $\beta_{j_0 n_0}$ in the list is an integer multiple of $\beta_{j_0 n}$, we remove $\beta_{j_0 n_0}$. This ensures that we only use the smallest $\beta_{j_0 n}$ to produce a minimum number of distance-regular subsets. For the axis planes, we use the regularity condition $|\beta_{j_0 n} - \alpha_{lk}| < t_{\text{aa}}$ as $\texttt{reg}_n(\beta_{j_0 n}, \alpha)$ to find all $\beta_{j_0 n} = \pi/n$ such that $n < N_{\max}$.

In the final step we check each set $\{d_{j_0}, \ldots, d_m\}$ for $j_0 \geq m-1$ for distance-regular subsets with respect to the candidates $\beta_{j_0 n}$. The reference object is always an element of a distance-regular subset. Therefore, for each $\beta_{j_0 n}$ we search for objects in $\{d_{j_0+1}, \ldots, d_m\}$ that form an approximately distance-regular subset. A d_l for $l \in \{j_0 + 1, \ldots, m\}$ belongs to the distance-regular subset if $|\operatorname{round}(f_l)\, \beta_{j_0 n} - \alpha_{lj_0}| < t_{\text{a}}$ for $f_l = \alpha_{lj_0}/\beta_{j_0 n}$. However, we only allow one object for each multiple of $\beta_{j_0 n}$ as the objects were already clustered. Hence, if we have two objects d_{l_1} and d_{l_2} for the same multiple, we use the one that is closer to p, unless one of the objects is a parent of the other in the hierarchical clustering structure, whereupon we take the parent, even if the child is closer, if the difference between the two tolerances is smaller than some t_{pd}.

Before we accept a distance-regular subset of objects as a regularity we also check which multiples of $\beta_{j_0 n}$ are actually present. We accept a planar angle-regular subset as a regularity if either all multiples, or at least every second one is present, or at least three consecutive multiples are occupied. If a set is accepted, we have to remove multiples of $\beta_{j_0 n}$ in the list of candidate base distances for all objects d_l in the distance-regular subset, otherwise subsets of the distance-regular subset found will be identified as distance-regular later.

For the axis planes we check in addition if the angle between two axis directions, which are not involved in any angle-regular subset, has a special value (see Section 4.1).

To find angle-regular axis directions on an axis cone, we employ Algorithm 2 again. The definition for a conical angle-regular subset is similar to the planar case. We project the axis directions from the axis cone on the plane through the origin orthogonal to the axis c of the cone. Due to the projection, opposite directions on the plane actually represent different axis directions on the cone. Therefore, we have to use base angles of the form $2\pi/n$. However, each axis direction on the cone can still point in one of two directions. Thus, we always project the direction pointing to the same side of the plane defined by cone normal c, i.e. ${d_l}^t c > 0$.

Note that three axis directions forming an orthogonal system generate a special conical angle regularity with a base angle $\pi/2$ on a cone with semi-angle $\arccos(1/\sqrt{3})$.

6 AXES

For all axis directions which are also associated with a position, i.e. which represent axes, we consider in addition the arrangement of these positions as well as the direction information. We seek approximate intersections of axes and regular arrangements of parallel axes.

For cylinders, cones and tori we use the root point of the surface as the associated position. The root point of a cone is its apex and the root point of a torus is its centre. The root point of a cylinder is an arbitrary point on the central axis. For elliptical edges we choose the centre of the ellipse and for straight edges we choose an arbitrary point on the edge as the associated position.

Planar faces do not have an obvious root point, but we can define such a root point by considering the boundary loops. Reasonable choices for root points are the average of the vertex positions for each loop and the centre of the convex hull of each loop. Note that these define multiple axes for a single planar face. Other possibilities exist for defining root points of planar surfaces and more general types of edges.

First we search for intersection points of the axes. We generate the approximate intersection point of each axis pair by computing the minimum distance between the two axes. If it is smaller than t_{la}, we say the axes intersect and use the mid-point of the shortest line between them as the approximate intersection point. These intersection points are then clustered to find sets of axes intersecting approximately in a point, where we use actual axes only and not the axis direction clusters. But we only intersect axes that belong to different axis direction clusters to avoid trying to intersect approximately parallel axes.

Furthermore, we seek regular arrangements of parallel axes. We check each cluster of parallel axis directions separately and extract the elements for which we have a root point. In addition we consider the axes from conical angle-regular arrangements if an intersection position for such axes has been found. To explore regular arrangements of the parallel axes, we project the locations for the axes onto a plane through the origin orthogonal to the axis direction.

Finding regular arrangements of axes is now reduced to finding regular arrangements of the root points projected onto the plane. By clustering the points using the tolerances t_{la} and t_{ld} we detect approximately equal axes. The resulting clusters represented by points in a plane are examined further to detect if the points lie on a line, a grid or a circle, and are regularly spaced, as described below.

6.1 Regularly Arranged Parallel Axes

We detect regular arrangements along lines and grids by generating a line for each pair of points, clustering these lines into approximately parallel lines and finding approximately equal lines in the parallel lines clusters. The equal line clusters are then checked for regular arrangements of the points. By examining pairs of approximately orthogonal groups of parallel lines, we find the grids.

First we generate a line for each pair of points p_1, p_2 in the plane. To cluster the lines into parallel groups, we represent them by vectors $d = p_2 - p_1$. For two such vectors d_1, d_2 with $\|d_2\| > \|d_1\|$, we use the similarity measure $\delta(d_1, d_2) = \|d_1 - ({d_1}^t u)u\|$ with $u = d_2/\|d_2\|$. By using the distance between d_1 and its projection on d_2 instead of the angle between the two vectors, we also take the distance between the points into account. If we used the angle between the vectors alone, two points which are on different parallel lines in a grid and sufficiently far apart might generate a line which is approximately parallel to the grid lines. For this reason, we also define the weighted average between the directions using a weighted average of the lengths as

$$\begin{aligned}\mathsf{avg}_{\texttt{parallel\_line}}(d_1, \omega_1, d_2, \omega_2) = \frac{\|d_1\|\omega_1 + \|d_2\|\omega_2}{(\omega_1 + \omega_2)^2} \\ \left(\frac{d_1}{\|d_1\|}\omega_1 + \operatorname{sign}({d_1}^t d_2)\frac{d_2}{\|d_2\|}\omega_2\right).\end{aligned} \tag{1}$$

Each of the resulting clusters of approximately parallel lines is again clustered to get groups of equal lines. To do this, the lines in a cluster of parallel lines are represented by the two points p_1, p_2. The tolerance used is t_{la}, and the similarity measure between two lines represented by p_1, p_2 and q_1, q_2 is based on the average distance between two corresponding end-points. If $(q_1 - q_2)^t(p_1 - p_2) \geq 0$, the points p_1 and q_1, and p_2 and q_2 correspond to each other. Otherwise p_1 and q_2, and p_2 and q_1 correspond. Let d_1 and d_2 be the two distance vectors between corresponding points and d be the unit vector representing the direction of the parallel line cluster. Then the similarity measure between the two lines is $(\|d_1 - ({d_1}^t d)d\| + \|d_2 - ({d_2}^t d)d\|)/2$, i.e. the average of the orthogonal distance with respect to d between the corresponding points. The weighted average of these two lines is the line with end-points $r_l = p_l + \omega_2/\omega_1(d_l - ({d_l}^t d)d)$, $(l = 1, 2)$.

For each group of approximately equal lines we seek base distances such that the distances between a subset of the points on the line can be represented as integer multiples of a base distance, analogously to the angle-regular arrangements using Algorithm 2. The main difference is that we do not have a regular value such as π/n for the base distances, but any distance could be a base distance. This simplifies the algorithm as we get at most one possible base distance per distance between the points. Note that we split approximately equal lines into different lines if we find different base values such that each line represents one base value.

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

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

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

$$\begin{aligned}d_l D_l \approx \operatorname{round}\left(\frac{d_l {D_l}^t D_1}{d_1}\right) d_1 D_1 \\ + \operatorname{round}\left(\frac{d_l {D_l}^t D_2}{d_2}\right) d_2 D_2.\end{aligned} \tag{2}$$

Another grid with the base distances d_3, d_4 and the corresponding directions D_3, D_4 is a diagonal of the grid if its reference point is on the grid and the distances are compatible. Without loss of generality, we assume that $d_1^2 + d_2^2 < d_3^2 + d_4^2$, i.e. the diagonal of the second grid is longer than the diagonal of the first one. Then the second grid is a diagonal if (2) holds for $d_l = d_3$, $D_l = D_3$ and $d_l = d_4$, $D_l = D_4$. The grid with the shorter diagonal is the fundamental grid and we eliminate the one with the longer diagonal.

Any distance-regular lines that are not on a grid and are not removed as diagonals of a grid are noted as regularities. In addition we also check whether the base distances have a special value for all distance-regular lines and grids (see Section 4.1).

Parallel axes arranged along a circle can be found by generating all circles formed by triples of points in the plane and clustering the circles.

7 POSITIONS

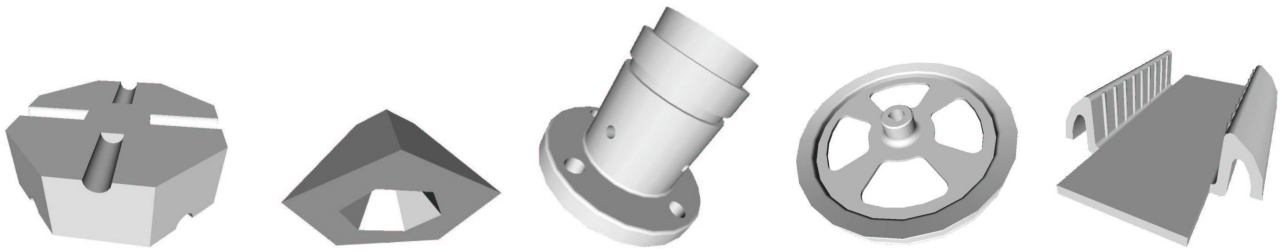
In this section we briefly discuss regularities of positions such as vertices, root points of surfaces, and axis intersection points. Using the hierarchical clustering algorithm for these points with tolerances t_{la} and t_{ld}, the Euclidean distance as similarity measure, and the weighted average between points, we seek approximately equal positions.

The resulting position clusters could be examined further for distance-regular arrangements on lines and 2D and 3D grids using a similar approach to that in the previous section. However, as the points are in $\mathbb{R}^3$ and not in the plane, many additional options are present, which makes a general search for such arrangements expensive. Furthermore, finding unintended approximately regular arrangements is likely to happen often. Thus, we limit the search to special lines and grids. If a B-rep model has one or more orthogonal systems, we search for distance-regular arrangements on lines and grids built from the directions of the orthogonal system(s). If the model has a main axis without an orthogonal system, we search for distance-regular arrangements on lines parallel to the main axis and on 2D grids orthogonal to the main axis.

These directions are used further to find partially equal positions, i.e. positions which are equal under projection. We cluster positions projected on a plane through the origin which is orthogonal to a special axis direction. Adding a second orthogonal axis direction, we project the positions further onto a line to find partially equal positions with respect to two axis directions.

8 EXAMPLES

The methods for finding regularities described in this paper were implemented on a GNU/Linux platform. We have tested the methods with various simple objects and more complex objects from [5]. Distortions induced by the reverse engineering process were simulated by translating and rotating the faces of the exact models and generating a point cloud from the perturbed object. The models reconstructed from the point clouds of both exact models, and models



	(a)	(b)	(c)	(d)	(e)
Faces:	26	14	29	53	88
Time (sec.):	0.145	0.155	0.550	0.621	1.553
R:	33	52	22	104	140
U:	0	11	2	21	27
M:	0	0	0	7	5

Figure 4: Example Objects

perturbed by varying amounts, were analysed. A test was considered to be successful if all the intended regularities in the model were found when using tolerances in agreement with the amount of perturbation.

Some examples are shown in Figure 4. Timings were obtained using several runs for each model on a 450Mhz Pentium III with 256MB of RAM, and the results were averaged with variations smaller than one hundredth of a second. The expected execution times for models with up to 200 faces range from a few seconds up to a few minutes for models with many *different* feature objects. Further we list the the total number of regularities **R** as found by the methods. **U** is the number of unwanted regularities detected, where an unwanted regularity is one that is not part of the design and conflicts with the desired regularities. **M** is the number of important regularities missed by our algorithms as identified by a human being. These values were determined for the objects with angles perturbed by 3 degrees and positions perturbed by 0.3 units, with angle tolerances of 5 degrees and length tolerances of 0.5 units. Further experiments showed that the number of unwanted regularities could be reduced by careful tolerance value selection. It appears that all tolerances for the methods can be expressed in terms of angular and positional tolerances which could be derived for a particular reverse engineering system from the errors in the reconstruction of a known test object. In the following we briefly describe the main results for the example objects.

The planar angle-regular axis directions with base angle $\pi/4$ of object (a) were detected. It was also discovered that the axes of the slots at the top and the bottom, and the axes of the planar faces intersect, and that the intersection points are partially equal. In addition various aligned axes, and two orthogonal systems, were found as conical angle regularities. Object (b) consists of a five-sided truncated pyramid with a six-sided truncated pyramid cut out at the bottom. Both conical angle regular arrangements were found. Some of the planar faces of the two pyramids were considered to be parallel depending on the amount of perturbation and the tolerance values. In the perturbed model additional conical angle-regular arrangements were found involving either the top of a pyramid and its sides or a combination of the faces of the two pyramids. Those caused by the perturbation were easily identifiable as they contradicted the two main conical angle regularities.

The main axis for each of the objects (c) and (d) was detected as a set of parallel axis directions and aligned axes. For object (c), the axes of the holes arranged on a circle and the axes of the holes intersecting on the main axis were detected as well as some other axis intersection points. Some of the cylinder radii were unwantedly considered to be equal. In object (d) two planar angle regularities of the faces orthogonal to the main axis with base angle $\pi/2$ and the angle $\pi/6$ between the two arrangements were detected. A number of vertices and intersection points of axes close to each other were unwantedly considered to be equal. Some of the regular arrangements of axes of planar faces were not detected or incorrectly determined and some parameters were considered to be equal, causing some special parameter values not to be found. Object (e) was found to have a single orthogonal system. Also multiple distance regular arrangements of axes along lines were found. The type of regularities which caused problems were very similar to those for object (d). Nevertheless, these deficiencies must be seen in the context that nearly all expected regularities were correctly found.

We also carried out some preliminary tests of partial models actually reverse engineered from point data. The results were satisfactory in the sense that they were similar to the results for the simulated data. The tolerance values for the real data appear to depend mainly on the reverse engineering system and not on the particular object.

In general all the regularities found were either really present in the model or could be explained by the perturbation of the model in combination with the tolerance values used. By choosing small tolerance values, only a small set of very accurate and thus very likely regularities are found. By increasing the tolerance values, there is a point when we get all the desired regularities. However, by increasing the tolerances, we also increase the number of unwanted regularities such that at the point where all desired regularities are found, we also get some unwanted regularities. An automated decision about which regularities should be used to improve the model has to be made by a subsequent step. For the analyser the user has to supply the tolerances, which should be maximal rather than optimal due to the hierarchical structure of the regularities. Additionally the user can provide parameters specifying which feature objects should be considered regular.

9 CONCLUSIONS

We have presented algorithms to find local shape regularities in terms of similarities between feature objects derived from B-rep model elements. The algorithms have been implemented and tested using various perturbed mechanical components. The results are promising in the sense that the main approximate regularities of the parts were found quickly while few unwanted regularities were reported. Thus, the analysis methods provide a strong basis for our subsequent beautification steps.

The methods presented can be expanded to find additional regu-

larities based on similarities between other types of feature objects and for other types of B-rep model elements. In future work we will develop a strategy to select and solve a maximal set of consistent constraints and generate a new, improved B-rep model, which should in some sense be close to the original model. The constraint solver has to handle a large set of inconsistent constraints, which are likely to form an overconstrained system, but some parts of the model might also be underconstrained. The inconsistencies could be detected explicitly by examining the constraint system as well as implicitly by using a hierarchical order of the constraints. For underconstrained parts of the model, additional conditions can be derived from the original model. The results of the constraint-solver phase will mainly modify the geometry and the location of the faces. To construct the new model, the intersections of the faces could be recomputed and then they could be stitched to form the new model. This might involve topological changes either to enforce a regularity or to ensure the generation of a valid model.

ACKNOWLEDGEMENTS

This project is supported by the UK EPSRC Grant GR/M78267. The authors would like to thank S. G. Schirmer from The Open University for invaluable comments and Tamás Várady from the Hungarian Academy of Sciences and CADMUS Consulting and Development Ltd. for providing reverse engineering software.

REFERENCES

[1] D. H. Bailey, S. Plouffe. Recognizing Numerical Constants. In: *Proc. Organic Mathematics Workshop*, Simon Fraser University, December 1995.

[2] P. Benkő, G. Kós, T. Várady, L. Andor, R. R. Martin. Constrained Fitting In Reverse Engineering. Submitted to *Computer-Aided Geometric Design*, 2001.

[3] P. Benkő, R. R. Martin, T. Várady. Algorithms For Reverse Engineering Boundary Representation Models. To appear in *Computer-Aided Design*, 2001.

[4] Å. Björk. *Numerical Methods For Least Squares Problems*. SIAM, Philadelphia, 1996.

[5] National Design Repository, Drexel University, <uri: http://edge.mcs.drexel.edu/repository/>.

[6] D. Eppstein. Fast Hierarchical Clustering And Other Applications Of Dynamic Closest Pairs. In: *Proc. 9th ACM-SIAM Symp. Discrete Algorithms*, pages 619–628, January 1998.

[7] H. R. P. Ferguson D. H. Bailey. A Polynomial Time, Numerically Stable Integer Relation Algorithm. NASA Technical Report RNR–91–032, December 1991.

[8] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[9] A. Y. Khinchin. *Continued Fractions*. Dover Publications, 1997.

[10] G. Kós. An Algorithm To Triangulate Surfaces In 3D Using Unorganised Point Clouds. To appear in *Computing*, 2001.

[11] G. Lukács, R. Martin, D. Marshall. Faithful Least-Squares Fitting Of Spheres, Cylinders, Cones And Tori For Reliable Segmentation. In: H. Budkhadj, B. Neumann B (eds.), *Proc. 5th European Conf. Computer vision*, vol. 1, Albert-Ludwigs Universität, Freiburg, Germany, pages 671–686, Springer, 1998.

[12] B. I. Mills, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate Symmetry Detection For Reverse Engineering. *These proceedings*.

[13] 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. Submitted to *Computer-Aided Design*, 2000.

[14] 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. on Robotics and Automation*, 15(1):57–66, 1999.

[15] T. Várady, R. R. Martin, J. Cox. Reverse Engineering Of Geometric Models – An Introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[16] 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.