

## APPROXIMATE GEOMETRIC REGULARITIES

F. C. LANGBEIN

*Department of Computer Science, Cardiff University,  
PO Box 916, Cardiff, CF24 3XF, UK,  
F.C.Langbein@cs.cf.ac.uk.*

B. I. MILLS

*Institute of Information and Mathematical Sciences, Massey University at Albany,  
Private Bag 102 904 North Shore Mail Centre, New Zealand,  
B.I.Mills@massey.ac.nz.*

A. D. MARSHALL, R. R. MARTIN

*Department of Computer Science, Cardiff University,  
PO Box 916, Cardiff, CF24 3XF, UK,  
{A.D.Marshall,R.R.Martin}@cs.cf.ac.uk.*

International Journal of Shape Modeling,  
7(2): 129–162, 2001.

Author version. Do not distribute, for personal use only.

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. Reverse engineered geometric models may be *beautified* by finding approximate geometric regularities in such a model, and imposing a suitable subset of them on the model by using constraints. Methods to detect suitable regularities for the beautification of B-rep models having only planar, spherical, cylindrical, conical and toroidal faces are presented in this paper. The regularities are described in terms of similarities. Different properties of faces, edges and vertices, and small groups of these elements in a B-rep model are represented as *feature objects*. Similar feature objects, such as directions which are parallel, form one sort of regularities. For each group of similar feature objects, *special* feature objects which might represent the group form further regularities, e.g. an integer value which approximates the radius of similar cylinders. Further regularities arise from symmetries of feature object sets. Experiments show that the regularities found are suitable for beautification such that subsequent steps allow the selection of a consistent regularity set.

*Keywords:* Beautification; Geometric Regularities; Geometric Constraints; Reverse Engineering; Similarity; Solid Modelling.

### 1. Introduction

Reverse engineering the shape of physical objects has a variety of applications in design and manufacturing, like reproduction and redesign. For many of these applications more than a simple copy is required and the information extracted from the object should represent the design intent. We are interested in reverse engi-

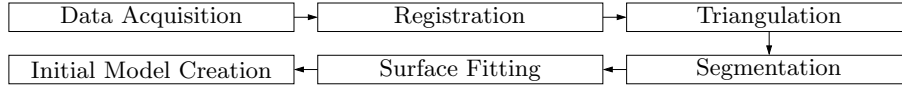


Fig. 1. Main Reverse Engineering Steps

neering a boundary representation (B-rep) model of a particular engineering part from 3D range data, which has all the desired geometric regularities present in the original, ideal design. Our ultimate goal is an intelligent 3D scanning system, which requires a minimum amount of human interaction and is suitable for naive users and non-engineering applications as well as engineers.

We intend to reconstruct models with accurate geometric properties for engineering parts bounded by planar, spherical, cylindrical, conical and toroidal surfaces that either intersect in sharp edges or are connected by fixed radius rolling ball blends. There are reliable surface fitting methods available for these surfaces [1, 2] and many interesting engineering objects can be generated using only these surface types [3, 4]. We assume that blends have been identified in the model [5] and are represented as edge and vertex attributes. Furthermore, the objects should have at most about 200 faces, which is a realistic limit achievable with current technology.

Our particular approach to reverse engineering is illustrated in Figure 1 [6, 7]. Initially multiple views of a physical object are obtained from a 3D laser scanner as dense 3D range data. The views are registered and merged into a single 3D point set [8]. This set is triangulated [9] and then segmented into subsets such that each subset represents a natural face of the object [1, 6]. For each of the point subsets a surface type is determined and a surface of this type is fitted [1]. Finally the surfaces are stitched to form a valid B-rep model, which we call the *initial* model.

The initial model suffers from various inaccuracies due to sensing errors during the data acquisition phase as well as approximation and numerical errors arising from the reconstruction process. Improving the precision of the sensing techniques and the reconstruction methods could reduce the errors, but some errors will always remain. As our intention is to recreate an *ideal* model for a physical object, we also have to take additional errors into account, which were introduced by possible wear of the object and the particular manufacturing method used to make it. To ensure that certain geometric regularities intended in the design are present, such as aligned cylinder axes or orthogonal and parallel planes, they must be identified and enforced at some stage of the reverse engineering process.

Previous approaches augment the surface fitting step by constraint solving methods [10, 11, 12, 13] such that, for instance, two planes are fitted simultaneously under the constraint that they are orthogonal. This requires specialised optimisation techniques reducing the error between the points and the fitted surfaces under desired geometric constraints. Another approach is to identify features like slots and pockets whose approximate location and type is provided manually and use this information to improve the results of the segmentation and surface fitting steps [14].

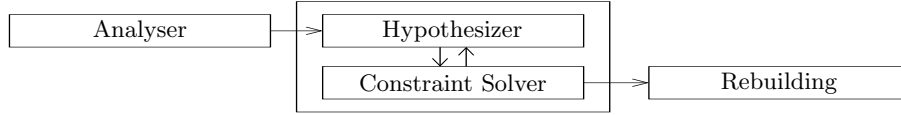


Fig. 2. Beautification Strategy

In our approach we attempt to automatically improve the initial model in a post-processing step which we call *beautification*. Our beautification strategy starts by analysing the initial model to recognize geometric regularities which it satisfies approximately. From this set a hypothesizer has to select a maximal subset of regularities which are likely to be present in the ideal design and can be realized simultaneously. A measure based on how well the regularity is present in the initial model, and how common and important it is, can be used to derive a figure of merit expressing our confidence that the ideal model possesses the regularity. Using this measure and some geometric reasoning, the hypothesizer selects a first set of regularities expressed as geometric constraints. In combination with a numerical and/or graph-based constraint solver the initial selection of constraints is adjusted to find a solvable constraint system, which should contain the desired geometric regularities of the model. Based on the solution of the final constraint system a new model is rebuilt (see Figure 2).

In this paper we present methods to detect approximate geometric regularities in the initial model for the first beautification step. These methods are improved and extended versions of those discussed in [15, 16]. We focus on *local* regularities in the sense that they relate to single properties of one or only a small number of elements in the B-rep model. For a method to detect approximate symmetries as *global* regularities see [17]. In future work we will develop the hypothesizer and constraint solving strategy.

Our regularities are defined as similarities between properties of B-rep model elements, and similarities between these properties and given special properties. For instance, we look for approximately equal cylinder radii, and we also try to find a special value, like an integer, for the average radius of each set of similar radii. We assume that the regularities are sufficiently distinct from approximation errors in the model, so that they can be distinguished from the noise by using appropriate thresholds. The output of the analyser is a list of regularities that the ideal model might possess. Whether a particular regularity is indeed present has to be decided later. Hence, we aim to produce a large set of possible regularities instead of looking for a small set of very likely regularities.

In the following we introduce the types of regularity detected by our methods and the notion of similarity as the fundamental concept used to define approximate regularities. Furthermore, we describe how to detect our chosen regularities in general using a hierarchical clustering algorithm. In the remaining sections we present the individual regularities and particular methods for detecting them. Finally we

Element	Geometry	Feature Object	Type
face	plane	root point normal polygonal loop(s) root points of loops	position direction loop position
	sphere	centre radius	position length
	cylinder	point on axis axis direction radius	position direction length
	cone	apex axis direction semi-angle	position direction angle
	torus	centre axis direction minor radius major radius sum of radii (unless lemon) difference of radii (unless apple)	position direction length length length length
edge (optional)	straight	direction of edge distance between end points	direction length
	circle	radius angle of the circle segment normal of circle plane	length angle direction
	ellipse	normal of ellipse plane	direction
vertex	point	location	position

Table 1. Basic Feature Objects Derived from a B-rep Model

provide results of applying the methods to some example models.

## 2. Approximate Geometric Regularities

We describe certain geometric regularities that are approximately present in a B-rep model in terms of similarities. From a *global* point of view this leads to approximate symmetries as similarities between the model and isometric images of the model, which are discussed in [17]. This can be expanded to partial symmetries requiring that only a subset of the model is approximately symmetric or that the model can be extended in a well-defined way to make it symmetric.

*Local* regularities are based on properties of B-rep model elements like faces, edges and vertices, which are represented by typed *feature objects*. The *type* is defined by the property the object describes. A feature object is handled separately from the B-rep model but refers back to the element(s) which generated it. For instance, we have directional feature objects arising from the normal of a plane, and the axis of a cone or cylinder. The radius of a cylinder and the semi-angle of

a cone form two other types of feature objects. Note that a single model element can generate several different feature objects of various types, which may not be independent of each other. Further feature objects can be derived by combining simple feature objects like the apex of a cone and the direction of its axis to form an axis feature object. Such axis feature objects may generate intersection points as further feature objects.

A list of basic feature objects is given in Table 1. The feature objects obtained from a B-rep model element depend on its geometry and its boundary. Note that we handle the feature objects arising from edges as optional since they do not always provide additional information about the model, and may sometimes create an unnecessarily large number of feature objects. We discuss the feature objects along with related regularities in detail below, and also add additional derived feature objects as appropriate.

We define and detect approximate geometric regularities in terms of similarities between feature objects. For one sort of regularity, we compare feature objects of the same type to derive sets of similar feature objects. For instance, we find parallel directions using directional feature objects. Another sort of regularity identifies special values for feature objects, by comparing them with predefined values, e.g. a length which is an integer. The feature objects are elements of a *feature space* defined by the feature object type. For instance, directional feature objects are represented as points on the unit sphere with antipodal points identified, which is a representation of the real projective plane  $\mathbb{P}^2$ . We seek (partial) symmetries of the feature objects in the feature space, e.g. for the directional feature objects we try to find  $n$  points regularly arranged on a circle in  $\mathbb{P}^2$  such that we have an approximate  $n$ -fold rotational symmetry.

### 2.1. Common Geometric Regularities

We surveyed about 600 mechanical components to determine common geometric regularities which are suitable for beautification [3]. Various objects like small engine parts, fittings and brackets for optical systems, plastic fittings, caps and connectors, sliding fittings for cupboards, a general selection of CAD models from online repositories and company catalogs, and parts from other surveys were reviewed. The parts chosen had low to medium complexity, i.e. less than about 200 faces and their geometric properties were significant for their application. They were also physically small enough to be put on a typical 3D scanner, which means that they fit inside a 50cm cube, and were light enough to man-handle onto the scanner bed. The features were large enough (bigger than about 5mm) to provide sufficient data to be able to properly fit surfaces, and there were no deep cavities that could not be probed by a 3D laser scanner.

About 97% of the parts exhibited important geometric regularities which could be classified using our similarity concept. This justifies our approach of trying to exploit such regularities to improve the quality of reverse engineered models. We list

Directions	Parallel directions.	5
	Directions which have the same angle relative to a special direction.	4
	Symmetrical arrangements of directions.	4
Axes	Aligned axes.	3
	Axes intersecting in a point.	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
	Equal positions under projection.	3
	Regular distances between positions arranged on a line or a grid.	3
Scalar Parameters	Equal scalar parameters.	5
	Special scalar parameter values.	3
	Simple integer relations between scalar parameters.	4
Loops	Equal loops independent of scaling.	4
Surface Types	Surface is approximately a plane or a cylinder.	—

Table 2. Common Geometric Regularities with their Estimated Frequency.

common regularities for which we present analysis methods in Table 2. The number in the last column indicates how common the particular geometric regularity is with 5 being nearly always present to 1 being rare as determined manually (except for surface types, see below).

We look for parallel directions and directions making the same angle to a special direction. For instance, the directions could all be orthogonal to a special direction which means that they lie in a plane, or they could have some other angle to the special direction which makes them lie on a cone. In addition these directions could be arranged symmetrically in the plane or on the cone as indicated in Figure 3.

Some of the directions can be associated with positions, and thus produce axis feature objects. The positions are obtained from vertices, apices of cones, centres of spheres and tori, etc. We also look for aligned axes, and their common intersection points. Furthermore, parallel axes could be arranged along lines and grids with equal distances between them or they could be arranged symmetrically on a cylinder.

For positions alone we seek equal positions, and positions which are equal when projected onto a special plane or line derived from the main directions in the model. In addition positions may be arranged regularly on a line or a grid with equal distances between them.

Scalar parameters from faces and edges are either lengths or angles. For each type separately we seek similar parameter values, and look for special values including integers and simple fractions. We also try to find simple integer relations between pairs of scalar parameters of the same type, i.e. relations of the form

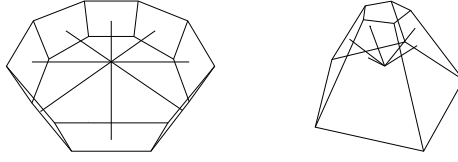


Fig. 3. Symmetrically Arranged Directions

$n_2 p_1 = n_1 p_2$  with the parameters  $p_i$  and some integers  $n_i$ .

Furthermore, we present a method to find similar polygonal loops independent of a scaling factor which could be generalized to detect similarities between loops with curved edges.

Finally we consider changing surface types of some faces. For example, a large radius cylindrical face is similar to a planar face and may have been incorrectly classified as planar. As misclassification is a problem of the model building software and not a regularity of a real part, no frequency has been determined.

### 3. Detecting Similarities as Cluster Hierarchies

Before we discuss specific regularities, we introduce some general methods to find similarities. Simple regularities are determined by finding similar feature objects. Often these similarities are the first step towards more complex regularities. We detect them using a hierarchical clustering algorithm. From the cluster hierarchy we extract distinct tolerance levels and a maximal tolerance level, which we use to simplify it.

Depending on the regularity type, we select a similarity measure  $\delta$  to indicate how close two feature objects are to each other when clustering. For a feature space  $X$ ,  $\delta$  is defined as a symmetric, non-negative function  $\delta : X \times X \rightarrow \mathbb{R}_0^+$ , such that  $x_1 = x_2$  implies  $\delta(x_1, x_2) = 0$ . We call two feature objects  $x_1, x_2 \in X$   $\varepsilon$ -similar (with respect to  $\delta$ ) if  $\delta(x_1, x_2) < \varepsilon$ , ( $\varepsilon \in \mathbb{R}^+$ ). This is sufficient for a measure to decide if a feature object is close to a special feature object. In order to get stable results from the clustering algorithm, the similarity measure should be a similarity metric, i.e. it should also fulfil the triangle inequality and  $\delta(x_1, x_2) = 0$  should imply that  $x_1 = x_2$ .

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

Given a set of feature objects, a similarity metric  $\delta$ , an averaging method  $\mathbf{avg}$ , and some tolerance value  $\Delta T$ , the hierarchical clustering algorithm should create a hierarchical structure of clusters such that each cluster is either the union of its sub-clusters or a simple set of feature objects and the top-level cluster is the whole

set. We call a cluster which consists only of feature objects a *base cluster*. Each cluster  $C_k$  should be represented by an average feature object  $c_k$  and a tolerance  $t_k$ , which is the maximum distance of the elements in the cluster from the cluster average with respect to  $\delta$ . Furthermore, the average feature objects representing two different clusters  $C_l, C_k$  must be at least  $\Delta T + 2 \max\{t_l, t_k\}$  apart, i.e. the clusters should be at least  $\Delta T$  apart from each other. Each cluster in the cluster hierarchy contains feature objects that are close to each other at some tolerance level and at the same time are sufficiently distinct from other clusters. Note that we do not limit the number of clusters, but generate as many clusters as required.

A variety of approaches exist to the clustering problem [18, 19]. The most common approach is the agglomerative (bottom-up) technique. Initially each element represents a cluster with tolerance 0. We start with the smallest value of  $\delta(c_l, c_k)$  representing the closest pair in the current set and combine the two elements to give a new element  $c_j$  representing a cluster which replaces  $c_l$  and  $c_k$ . This is repeated until only one element remains. A brute force solution searching for the closest elements each time requires  $O(n^3)$  time for  $n$  elements. Note that there are alternative numerical [20] and top-down [21] techniques.

To improve the brute force approach we use a distance matrix containing the distances between the feature objects and maintain a quad-tree like data structure to keep track of the closest pair [22]. 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 elements are grouped arbitrarily into pairs  $(c_l, c_{l+1})$  and the distance between two pairs is defined to be the minimum distance between the four elements of the two pairs. These pairs define a new closest pair problem in a matrix of half the size. Continuing this recursively we get the closest pair in  $D$  as the element left in the last matrix at the root of this structure. From there we can extract it and update the matrices. After initializing we have to update at most two rows and two columns of each of the matrices for each update operation. Hence, this clustering method requires  $O(n^2)$  space and time assuming that  $\delta$  and  $\text{avg}$  require constant time and space.

In order to get a hierarchical structure fulfilling our special requirements for the distance between the clusters we have to take care when combining two elements  $c_l, c_k$  representing clusters with the tolerances  $t_k \geq t_l$ . If  $d_0 = \delta(c_l, c_k) - 2 \min\{t_l, t_k\}$  is smaller than  $\Delta T$ , both clusters are merged into a single cluster. Otherwise, if  $d_1 = \delta(c_l, c_k) - t_l - t_k$  is larger than  $\Delta T$ , both clusters are sufficiently far apart from each other and they become two sub-clusters of a new cluster. If  $d_1$  is smaller than  $\Delta T$ , then, while  $d_0$  indicates that the clusters are sufficiently distinct, they are too close with respect to their tolerances and thus we make  $C_l$  a sub-cluster of  $C_k$ .

### 3.1. *Simplifying Cluster Hierarchies*

As all feature objects are combined into a single top-level cluster, the hierarchy contains clusters at high tolerance levels which are not likely to represent a desired regularity. While we require the average feature objects of the clusters to be suf-



ficiently distinct from each other, we do not create clusters at distinct tolerance levels. Experiments with setting a maximum tolerance lead to reasonable results, but only for simple objects was there a tolerance level which distinguished exactly between desired and unwanted regularities, and a large number of tolerances to detect different regularity types were required [15, 16]. In general the number of unwanted regularities could only be minimised, but not avoided unless desired regularities were dropped as well. As this means we have to make a decision about which regularities are used at a later stage in most cases, we drop the idea of using maximum tolerance levels and instead simplify the cluster hierarchy by determining the distinct tolerance levels within the cluster hierarchy and discard clusters above the tolerance level where the largest jump between the tolerance levels occurs. Using merit functions and geometric reasoning the subsequent steps can employ the simplified hierarchies to make consistent decisions about which regularities to include.

For simplification we use the constant  $\Delta T$  from above to distinguish between tolerances. Let  $\{t_l\}$  be the set of cluster tolerances in the hierarchy such that  $t_0 < t_1 \leq t_2 \leq \dots \leq t_n$  with  $t_0$  the largest cluster tolerance smaller than  $\Delta T$  or 0, if no such tolerance exists. Let the first tolerance level  $T_0$  be  $t_0$ . For  $l = 0, 1, 2, \dots$ , we remove  $t_l$  from the list of tolerances if its distance to the last tolerance level  $T_k$  is smaller than  $\Delta T$ . Otherwise, it is used as a new tolerance level  $T_{k+1}$ . This creates distinct tolerance levels  $\{T_k\} \subset \{t_l\}$  that represent tolerance jumps larger than  $\Delta T$ . For each cluster in the hierarchy we determine the smallest  $T_k$  which is larger than the cluster tolerance. If this  $T_k$  is the same as the one determined for its sub-clusters, the sub-clusters are merged with the parent cluster. This results in a cluster hierarchy where there is at least one  $T_k$  between the tolerances of the sub-clusters and their parent.

To detect the largest jump between the distinct tolerance levels we need an approximation  $T_m$  for the largest possible tolerance level, which is automatically derived from the initial model (see below).  $T_m$  is required since all feature objects may be similar to each other, in which case the largest tolerance jump is above the largest tolerance of the clusters. We add  $T_m$  as the last element to the ordered set  $\{T_k\}$  unless one of the  $T_k$  is larger than  $T_m$ , and find the index  $j$  of the maximum of tolerance jumps  $\delta_k = T_{k+1} - T_k$ . Then clusters with tolerances larger than  $T_j$  are removed from the hierarchy.

Figure 4 shows an example for clustering points in the plane (a) and the simplified cluster hierarchy (b). The simplification process removed some of the distinctions at small tolerance levels as well as an intermediate cluster and the top-level cluster.

For hierarchical clustering we only need one tolerance  $\Delta T$ , which is either a length tolerance  $\Delta T_L$  or an angular tolerance  $\Delta T_A$  for most regularities discussed below. By increasing  $\Delta T$ , feature objects are more likely to be judged the same.  $T_m$  is automatically derived from the model. Let  $L_{\max}$  be the largest length present in the model. Then an appropriate  $T_m$  for lengths is  $L_{\max}/2$ . Since we identify

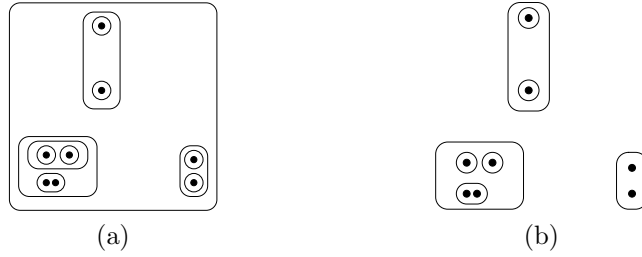


Fig. 4. Example for Hierarchical Clustering and Simplifying the Cluster Hierarchy

opposite directions, our angles are in the interval  $[0, \pi/2]$  and thus  $T_m = \pi/4$  is appropriate for angles.

To find regularities as similarities we first create a hierarchical cluster structure of appropriate feature objects and then simplify the hierarchy so that the remaining clusters are at distinct tolerance levels and clusters at very large tolerance levels are removed. The simplified hierarchy represents possible similarities between the feature objects at different tolerance levels.

Depending on the feature object type, further consistency checks can be used to decide if a cluster represents a regularity. For instance, an intersection point of  $n$  axes should be a cluster of  $n(n-1)/2$  intersections of axis pairs. For some feature object types we can also recompute the average feature objects for the clusters to better fit the elements of the cluster before the simplification step.

We discuss clustering below for each of the feature object types, together with specialised methods for special values and symmetries in the feature spaces.

#### 4. Directions

We extract directions such as normals of planes and directions of cylinder and cone axes from faces and edges of a B-rep model as indicated in Table 1. The directions are represented as points on the unit sphere with antipodal points identified, i.e. opposite directions are considered to be equal. This direction space is the real projective plane  $\mathbb{P}^2$ . In this context we can describe regular arrangements of directions as points and circles in  $\mathbb{P}^2$ . Further regularities are derived from symmetrical arrangements of the points on the circles in  $\mathbb{P}^2$ .

Approximately parallel directions are approximately equal points in  $\mathbb{P}^2$ . We find them by clustering the directions with a similarity measure that compares the (smaller) angle between two directions  $d_1$  and  $d_2$ ,  $\delta(d_1, d_2) = \arccos(|d_1^t d_2|)$ . Furthermore, we define the weighted average between two directions  $d_1$  and  $d_2$ . If  $d_1^t d_2$  is non-negative, i.e.  $d_1$  and  $d_2$  point roughly in the same direction, the weighted average is  $(\omega_1 d_1 + \omega_2 d_2)/(\omega_1 + \omega_2)$  with the positive weights  $\omega_1, \omega_2$ . If  $d_1^t d_2$  is negative, the weighted average is  $(\omega_1 d_1 - \omega_2 d_2)/(\omega_1 + \omega_2)$ . We create the cluster hierarchy using  $\Delta T_A$ .

Before the cluster hierarchy is simplified we improve the average directions repre-

senting the clusters in a least squares sense. Let  $d_l$ , ( $l = 1, \dots, n$ ), be the directions of a cluster including all directions from its sub-clusters. We find a new average direction  $x$  by minimizing the error of the linear system  $d_l^t x = 1$ , ( $l = 1, \dots, n$ ), in a least squares sense using the singular value decomposition of the column vector matrix  $[d_1, \dots, d_n]$ . Note that the  $d_l$  all must point in the same half space as the original average direction  $x_0$ , which can be achieved by replacing  $d_l$  with  $-d_l$  if  $d_l^t x_0 < 0$ .

Finally, the cluster tolerances are recomputed and the cluster hierarchy is simplified using  $T_m = \pi/4$ . Even if the number of directions extracted is large, we expect to find only a limited number of *different* directions. The cluster hierarchy could still contain clusters at large tolerance levels which would usually require a change of the combinatorial topological structure.

In the following sub-sections we discuss the arrangements of directions represented by circles in  $\mathbb{P}^2$  and regular arrangements on these circles. Directions that are on a great circle of the sphere represent directions that are orthogonal to another direction and thus lie in a plane, which we call *direction plane*. Directions that are on a small circle of the sphere represent directions that lie on a cone, which we call *direction cone*. In addition the arrangement of the directions in a plane or a cone may be symmetric. We call these *planar* or *conical angle-regular* (see Figure 3).

#### 4.1. Regularly Arranged Directions

We first try to find direction sets that lie on circles in  $\mathbb{P}^2$ . A set of directions  $\{d_i\}$  on a circle satisfies the equation system  $|d_i^t x| = a$ , where  $x \in \mathbb{P}^2$  is the centre of the circle. If  $a = 0$ , we have a direction plane with normal  $x$  and if  $a \in (0, 1)$ , we have a direction cone with axis  $x$ . Note that for a plane, taking the absolute value of  $d_i^t x$  is not required, and we can drop it for a cone if we ensure that all directions have the same orientation relative to  $x$ .

To find the sets of directions lying in a direction plane, we cluster the normals representing all direction planes generated from each pair of linearly independent directions. The clustering is done in the same way as clustering parallel directions, but we employ the average directions from the parallel direction base clusters instead of all directions. This not only reduces the number of normals generated, but also avoids cases where the directions are approximately linearly dependent.

Before we simplify the cluster hierarchy, we recompute the average direction plane normals using a least squares error method. For directions  $d_l$ , ( $l = 1, \dots, n$ ), in a plane with normal  $x$ , we get the linear system  $d_l^t x = 0$ , ( $l = 1, \dots, n$ ), under the constraint  $\|x\| = 1$ . Thus we minimize  $\|Dx\|$  for  $x \in \mathbb{P}^2$  with the column vector matrix  $D = [d_1, \dots, d_n]$ . Let  $USV^t$  be the singular value decomposition of  $D$ . As  $U$  and  $V$  are unitary they do not change the norm of a vector. Thus  $\|Dx\|$  is optimal if  $V^t x = e_l$  where  $e_l$  is the  $l$ -th standard basis vector corresponding to the smallest singular value of  $D$ , i.e. the solution to the least squares problem is  $x = Ve_l$ . After the average and the tolerances have been recomputed for each cluster, the cluster

hierarchy is simplified using  $T_m = \pi/4$ .

A direction plane cluster is consistent if it contains  $n(n-1)/2$  plane normals generated from  $n$  parallel direction base clusters. If a cluster is not consistent, we either combine the cluster with its parent if it has one or remove it completely. However, we also have to consider the parallel direction cluster hierarchy. Consider a direction plane cluster consisting of  $n$  plane normals  $p_{lk}$  generated by the parallel direction base cluster pairs  $d_l$  and  $d_k$ . This cluster is consistent if there are  $n(n-1)/2$  different  $p_{lk}$ . If this is not the case, we count the number  $m$  of direction pairs  $d_l, d_k$  for which there is no  $p_{lk}$  in the cluster, but which are considered to be parallel in the parallel direction cluster hierarchy at a higher tolerance level, still smaller than the tolerance of the direction plane cluster. We still consider the cluster to be consistent if there are  $n_1(n_1-1)/2$  different  $p_{lk}$  for  $n_1 = n - m$ , as there are  $m$  parallel direction pairs.

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

One way to find sets of directions on a cone would be to cluster the generated direction cones as pairs  $(c_l, \alpha_l)$  [16]. Better results can be achieved if we consider the angles separately creating a cone angle hierarchy and use this hierarchy to guide the clustering of cone directions. To cluster the angles, we generate a cone angle feature object for each direction cone and cluster them using  $\Delta T_A$  and the similarity measure  $\delta(\alpha_1, \alpha_2) = |\alpha_1 - \alpha_2|$ . The cluster hierarchy is simplified using  $T_m = \pi/4$ .

We do a depth first traversal of the angle cluster hierarchy, clustering the cone directions at the lowest level of the angle hierarchy first and reporting the results to higher hierarchy levels. At the lowest level in the hierarchy we simply collect all cone directions belonging to the angles combined in that cluster and cluster them in the same way as the direction planes. For each of the clusters we check if they are consistent. For direction cones this means that  $n$  directions cones have to be generated by  $n(n-1)(n-2)/6$  different parallel direction base clusters. Analogously to the direction plane consistency check, we also check if directions on the cone are approximately parallel at higher tolerance levels in the parallel direction hierarchy. Consistent clusters are preserved and marked as consistent. Other clusters which do not fulfill the consistency condition are marked as such and kept at the top level in the current hierarchy.

When moving upwards in the angle cluster hierarchy the clustering results of lower levels are combined into a single set of clusters and the clustering of the cone directions is continued using them. At each level the clusters are checked for consistency and only the consistent ones are preserved, while the others are simply

reported to the next higher angle cluster level until the top-level angle clusters are reached. When combining clusters at higher levels, those which are marked consistent are always added as sub-clusters to new clusters.

This results in a direction cone cluster hierarchy which is created by only comparing the directions of the cones. To avoid mixing cones with different angles, the angle clusters are used to ensure that only cone directions with similar angles are combined at different tolerance levels. The direction cone cluster hierarchy is simplified by considering the direction tolerances of the clusters using  $T_m = \pi/4$ .

We finally recompute the average direction cone representing a cluster built from the direction base clusters  $d_l$  by solving the linear system  $d_l^t x = 1$ , ( $l = 0, \dots, n$ ), in a least squares sense using singular value decomposition to give the average semi-angle and the average cone direction.

In general, direction cones alone do not represent an important regularity, especially as a model can easily create many different direction cones. The main reason for detecting direction cones is to find symmetrical arrangements of directions as discussed in the next sub-section.

#### 4.2. Symmetrically Arranged Directions

Exploring direction planes and cones in more detail may reveal symmetries in the arrangements of the directions (see Figure 3). Given a set of directions in a plane or on a cone we look for subsets such that the angles between the directions in this subset are integer multiples of a base angle  $\beta$ , i.e. the directions as points on the unit sphere are arranged equispaced around a small or great circle. The subsets can be incomplete in the sense that not all multiples of  $\beta$  need be present. We present an algorithm to detect these symmetries in direction planes and later discuss modifications for direction cones and other similar symmetries.

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

At present we only look for base angles which are (approximately) present as an angle between directions. For instance, if we have two approximate angles  $2\pi/6$  and  $5\pi/6$  relative to some direction, the underlying base angle  $\pi/6$  is not detected. However, the angles between the involved directions of such an arrangement are found as special angle values. An efficient implementation of such cases is left as future work. One might, for instance, employ approximate integer relations.

We check the directions  $d_l$  from the direction plane clusters for angle-regular arrangements. As the direction planes were created using the parallel direction

- I. Compute the angles  $\alpha_{lk}$  between directions  $d_l$  and  $d_k$  for  $l < k < m$ , where  $m$  is the number of directions  $d_l$ .
- II. For all reference directions  $d_k$  with  $k < m$  and for all angles  $\alpha_{lk}$  with  $k < l \leq m$ :
  1. Find the candidates  $\beta_{kn} = \pi/n$  such that  $n < N_{\max}$  and  $|\beta_{kn} - \alpha_{lk}| < t_1$ .
  2. Add  $\beta_{kn}$  to the list of candidates for  $d_k$  unless it is an integer multiple of one of the  $\beta_{kn_0}$  already in the list.
  3. If  $\beta_{kn}$  has been added to the list, remove any  $\beta_{kn_0}$  from the list which is an integer multiple of  $\beta_{kn}$ .
- III. For each reference direction  $d_k$  with  $k < m-1$  consider the subset  $\{d_k, \dots, d_m\}$  and for each candidate  $\beta_{kn}$  with  $n = 1, \dots, N_k$ , where  $N_k$  is the number of base angle candidates for  $d_k$ :
  1. For each object  $d_j$  with  $k < j \leq m$ :
    - A. If for  $f = \alpha_{jk}/\beta_{kn}$ , we have  $t = |\text{round}(f)\beta_{kn} - \alpha_{jk}| < t_1$ , then record  $d_j$  to be a round( $f$ ) multiple of the base angle. If there is already a direction  $d$  as the multiple round( $f$ ), then only replace  $d$  by  $d_j$  if the tolerance of  $d_j$  is smaller than the tolerance of  $d$ .
  2. If the planar angle-regular subset found has at least two elements:
    - A. If we already found a subset with a base angle which is a divisor or a multiple of the base angle of this subset, the two sets have common directions and the error between the two initial directions is smaller than  $t_1$ , then merge the two subsets and adjust the base angle.
    - B. Otherwise, create a new planar angle-regular set with base angle  $\beta_{kn}$ .

Algorithm 1. Finding Planar Angle-Regular Direction Sets

clusters, the  $d_l$  are the average directions of these clusters. To avoid ambiguous angle-regular arrangements due to the tolerances involved, we derive a maximum  $N_{\max}$  for the allowed  $n$  values of the base angles. Let  $t_1$  be the sum of  $\Delta T_A$  and the the maximal tolerance of the parallel direction clusters and the direction plane cluster. If  $n$  is larger or equal than  $N_{\max\_tol} = \text{floor}(\pi/(4t_1))$ , angle regularities with base angles  $\pi/n$  and  $\pi/(n+1)$  cannot be distinguished at tolerance level  $t_1$ . Thus we set  $N_{\max} = \min\{N_{\max\_tol}, N_{\max\_user}/2\}$ , where  $N_{\max\_user}$  is provided by the user to compensate for small tolerance values. Even if the tolerances are small an  $n$  larger than about 36 (a base angle of  $10^\circ$ ) may not be of interest. We divide  $N_{\max\_user}$  by two so that we can also use it for the conical case where opposite directions in the plane cannot be considered equal and thus we have twice as many directions to consider (see below).

Given a set of  $m$  directions  $\{d_l\}$ , we look for a minimum number of subsets which are approximately angle-regular with respect to  $t_1$ . The algorithm for this consists of three main steps. First we compute all angles  $\alpha_{lk}$ , ( $k < l \leq m$ ), between the directions. From these angles we derive a set of possible base angles  $\beta_{kn}$  for each  $d_k$ , which we call the reference direction for the angles  $\beta_{kn}$ . Note that more than

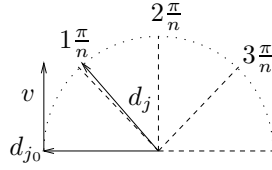


Fig. 5. Planar Angle-Regular

one base angle candidate can be close to a single  $\alpha_{lk}$  depending on the tolerances. In the third step we try to find angle-regular subsets by checking the angles  $\alpha_{jk}$  for each reference object  $d_k$  and all base angle candidates  $\beta_{kn}$  (see Algorithm 1).

To compute the angles between the directions in step I, the reference object  $d_k$  is used to choose the angle that lies consistently to the right of the reference object  $d_k$  with respect to the direction plane normal  $q$  (see Figure 5). With  $v = q \times d_k$  we have

$$\alpha_{lk} = \begin{cases} \arccos(d_k^t d_l) & \text{if } v^t d_l \geq 0, \\ \pi - \arccos(d_k^t d_l) & \text{if } v^t d_l < 0. \end{cases} \quad (1)$$

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

In step II we look for base angle candidates  $\beta_{kn}$  for each set  $\{d_k, \dots, d_m\}$  referenced by  $d_k$ . We have to check the relations between the base angles added for a single reference object to ensure that we use the smallest base angles and eliminate multiples.

In step III we check each set  $\{d_k, \dots, d_m\}$  for  $k < m - 1$  for angle-regular subsets with respect to the candidates  $\beta_{kn}$  found in the previous step. The reference object is always an element of an angle-regular subset of the set. Thus, for each  $\beta_{kn}$  we seek approximately angle-regular subsets of  $\{d_{k+1}, \dots, d_m\}$ . A  $d_l$  for  $l \in \{k + 1, \dots, m\}$  belongs to the angle-regular subset, if, for  $f_l = \alpha_{lk}/\beta_{kn}$ , we have  $|\text{round}(f_l)\beta_{kn} - \alpha_{lk}| < t_1$ . However, we only allow one direction for each multiple of  $\beta_{kn}$ , as the directions have already been clustered into parallel directions. Hence, if we have two objects  $d_{l_1}$  and  $d_{l_2}$  for which  $p = \text{round}(f_{l_1}) = \text{round}(f_{l_2})$ , we use the one that is closer to  $p$ , i.e. the one for which  $|p - f_{l_j}|$  is smaller.

Once we have found an angle-regular subset for a particular base angle and reference direction, we first check if that subset can be merged with a set found before. This is the case if one of the base angle candidates of the two subsets is a multiple of the other, the subsets have common elements and the error between the two reference directions is smaller than  $t_1$ . If there is no such set, we create a new regularity. While the algorithm has been designed to avoid cases where sets have to be merged, they cannot be avoided completely due to the approximate nature of the problem. Comparing the sets rather than the relations between the base angle candidates is more robust and therefore preferred.

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

If any of the computed  $\alpha_{lk}$  are not involved in an angle-regular subset, we check whether we can find a special value for this angle, using the algorithm to detect special angle parameters described in Section 7.

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

As opposite directions are no longer identified in this case, the maximum  $n$  for the base angles is  $N_{\max} = \min\{N_{\max\_tol}, N_{\max\_user}\}$  with  $N_{\max\_tol} = \text{floor}(\pi/(2t_1))$ . The tolerance  $t_1$  is derived in the same way as for the planar case. Note that three axis directions forming an orthogonal system generate a special conical angle regularity with semi-angle  $\arccos(1/\sqrt{3})$  and angle  $\pi/2$  between the axis directions.

## 5. Axes

For all directions associated with a position, i.e. which represent axes, we consider positional arrangement as well as the direction information. We seek approximate intersections of axes, and regular arrangements of parallel axes.

For cylinders, cones and tori we combine the directions with the *root point* of the surface to get the axis. 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 could be defined specifically as the point on its axis closest to the origin or the centroid of the model. For our purposes it is enough to choose an arbitrary point on the axis. For directions derived from edges, we use a special point of the edge. For elliptical edges we choose the centre. For straight edges we could choose the mid-point of the edge. Again for our purposes it is enough to choose an arbitrary point on the edge.

For planar faces we do not have an obvious root point as for other surface types, but we can define one by considering its boundary loops. Suitable root points are



the average position of the vertices around each loop, and the centre of the convex hull of each loop. Note that using several such root points leads to multiple axes for each planar face. Other possibilities exist for defining root points of planar surfaces, or for directions defined by more general curve types.

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

The intersection points are clustered using the Euclidean distance as a similarity measure and the length tolerance  $\Delta T_L$ . The averaging method used for clustering is the weighted average of two points in  $\mathbb{R}^3$ . The resulting intersection point cluster hierarchy is simplified using  $T_m = L_{\max}/2$ . After this we check if the distances between the axes are consistent with the cluster tolerances. We start with the clusters lowest in the hierarchy and move upwards from there. If a cluster contains an intersection point feature object with an axis distance larger than the sum of the cluster tolerance and  $\Delta T_L$ , the intersection point is moved to the parent of the cluster, if there is one, or it is removed from the hierarchy.

Next, we check if each of the clusters in the hierarchy is consistent. If it is not consistent the cluster is removed from the hierarchy and either merged with its parents, or only its sub-clusters are left, if there are any. An axis intersection cluster which contains  $n$  different axes is consistent if it has  $m = n(n - 1)/2$  intersection points of axis pairs. However, as some of the axes might be parallel, some intersection points might not be present, so the cluster is still consistent if it has only  $m - p$  intersection points where  $p$  is the number of parallel axis pairs in the cluster with respect to  $\Delta T_A$  in the parallel direction cluster hierarchy.

We are also interested in regular arrangements of parallel axes. Each cluster of parallel directions is considered separately, and the directions for which we have a root point are extracted. In addition we consider the axes from conical and planar angle-regular arrangements if an intersection position for angle-regular arranged axes has been found. We start by exploring the axes in the parallel direction base cluster and move up to the parents in the hierarchy. If a regular arrangement of axes has been found in a parent cluster, it is only accepted if it is not completely inside a single sub-cluster. Otherwise it should have been detected earlier in the sub-cluster.

Finding regular arrangements of axes can be interpreted as finding regular arrangements of the axis positions projected on a plane orthogonal to the average direction of the parallel direction cluster. To find a position for this plane we project the axis positions on a line through the origin in the average direction and take the average of these projections. By clustering the projected points using  $\Delta T_L$ ,

we create a cluster hierarchy of approximately aligned axes. If the parallel direction cluster used to detect the parallel axes has sub-clusters, we only accept aligned axis clusters which contain axes from different sub-clusters. The resulting clusters represented as 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 in following sub-sections.

### 5.1. *Parallel Axes along Lines and Grids*

We detect parallel axes along lines and grids using the projected points from above. For each pair of projected points, we create a line. These lines are clustered into approximately parallel lines and in each cluster of parallel lines we group the points generating the lines into point sets lying approximately on the same line. Then we check whether the points on the average lines are regularly spaced. By further examining pairs of approximately orthogonal groups of regularly spaced, parallel lines, we find the grids.

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

$$\text{avg}_{\text{pl}}(d_1, \omega_1, d_2, \omega_2) = w \left( \frac{d_1}{\|d_1\|} \omega_1 + \text{sign}(d_1^t d_2) \frac{d_2}{\|d_2\|} \omega_2 \right) \quad (2)$$

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

For each parallel line cluster, we solve the linear system  $d_l^t x = 1$ , ( $l = 0, \dots, n$ ), in a least squares sense where the  $d_l$  are the normalised direction vectors of the parallel lines in the cluster. The solution  $x$  is used as the average direction for the parallel line cluster and the cluster tolerance is updated. After this the cluster hierarchy is simplified using  $T_m = L_{\text{max}}/2$ .

The clusters represent sets of parallel lines, but we still have to group the points which generate these lines to find distinct lines on which the points lie approximately. If we have two approximately parallel lines each generated by two points, we can consider these points to lie approximately on the same line if (at least) one of the points is used to generate both lines. In general we can use this to find the distinct lines by detecting sets of points connected by lines in a parallel lines cluster. We start with pairs of points each one representing a single parallel line in the cluster. As long as we can find two sets of points which have at least one point in common we merge the sets.

Each of the point sets representing points that are approximately on a line is further examined for regular arrangements of the points on the line, i.e. we look for base distances such that the distances between a subset of the points on the line can be represented as integer multiples of a base distance, analogously to the method used for angle-regular arrangements (Algorithm 1). The main difference is that we do not have a special value such as  $\pi/n$  for the base distance, but any distance could be a base distance.

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

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

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

$$d_l D_l \approx \text{round} \left( \frac{d_l D_l^t D_1}{d_1} \right) d_1 D_1 + \text{round} \left( \frac{d_l D_l^t D_2}{d_2} \right) d_2 D_2. \quad (3)$$

Another grid with base distances  $d_3$ ,  $d_4$  and the corresponding directions  $D_3$  and  $D_4$  is a diagonal of the fundamental grid if its reference point is on the grid and the distances are compatible. Without loss of generality, we assume that  $d_1^2 + d_2^2 < d_3^2 + d_4^2$ , i.e. the diagonal of the second grid is longer than the diagonal of the first one. Then the distances of the two grids are compatible, if Equation (3) holds for  $l = 3$  and  $l = 4$ . The grid with the shorter diagonal is the fundamental grid and we eliminate the one with the longer diagonal. Figure 6 illustrates a diagonal grid

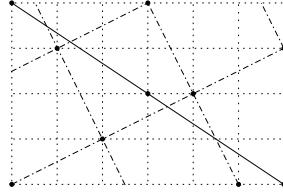


Fig. 6. Combining a Fundamental Grid with Diagonal Lines and Grids

marked with dot-dashed lines and a distance-regular diagonal line drawn solid on a fundamental grid marked with dashed lines. Only the positions of the fundamental grid which are also on the illustrated diagonals are marked.

Any distance-regular line that is not combined to give a grid or removed as a diagonal of a grid is noted as a regularity. In addition we also check whether distance-regular lines and grids have a base distance which is a special value (see Section 7).

### 5.2. *Parallel Axes on Cylinders*

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

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

Each triple of remaining clustered points generate a circle. We only consider circles with radii larger than  $\Delta T_L$  and smaller than  $L_{\max}$ , and the smaller arc of the circle described by the three points should be larger than  $\Delta T_A$ . For  $n$  points there are  $n(n-1)(n-2)/6$  possible circles, which can easily become too many for our hierarchical clustering algorithm. Thus, we first group the circles by clustering their positions and stop as soon as the distance between the pairs of average positions is larger than some  $d_{\max}$ . Note that we do not create a cluster hierarchy.  $d_{\max}$  determines how many different average positions we consider. The larger it is,

the more possible relations between circles we take into account. We set it to  $\Delta T_L \min\{L_{\max}/\Delta T_L, D_{\max}\}$  with some user-defined limit  $D_{\max}$ , which should be as large as possible with respect to the memory available.

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

Once we have the circle cluster, we recompute a circle that fits the points in a least squares sense. For this we minimize the function  $F : (c, r) \mapsto \sum_i (\|p_i - c\| - r)^2$  for the circle centre  $c$  and the radius  $r$  with the points  $p_i$ . As  $r = 1/n \sum_i \|x_i - c\|$  for a minimum of  $F$ , we actually minimize  $c \mapsto \sum_i (\|p_i - c\| - 1/n \sum \|p_i - c\|)^2$  using the Nelder-Mead downhill simplex method [23].

In a similar way to finding symmetrical arrangements of directions in a plane, we can use Algorithm 1 to find symmetrical arrangements of points around the circle. Additionally we check for special radius values.

## 6. 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 tolerance  $\Delta T_L$ , the Euclidean distance as a similarity measure and the weighted average between points, we seek approximately equal positions. The clusters are simplified using  $T_m = L_{\max}/2$ .

The average positions of the resulting clusters could be examined further for distance-regular arrangements on lines and on two- and three-dimensional grids in a similar way to the previous section. However, as the points are now in  $\mathbb{R}^3$ , not in  $\mathbb{R}^2$ , many additional possibilities are present, which makes a general search for such arrangements expensive. Furthermore, finding approximately regular arrangements which are only there by chance and are not intended is likely to happen much more often. Thus, we limit the search to special lines and grids. The directions of these special lines and grids are taken from orthogonal systems identified as conical angle-regular directions, and parallel direction clusters which contain more than 1/3 of the total number of direction features from faces.

The directions are further used to find partially equal positions. Positions are considered to be partially equal in 2D with respect to a direction if they are equal when projected onto a plane orthogonal to the direction. Further, they are considered to be partially equal in 1D with respect to a direction if they are equal when projected onto a line in the given direction. Both regularities can be detected by clustering the projected points using  $\Delta T_L$ . We remove clusters from the hierarchy

which contain projected points which are as close together as the original points, as these represent approximately equal positions detected earlier. Finally the cluster hierarchy is simplified using  $T_m = L_{\max}/2$ .

As all directions in the parallel direction cluster hierarchy are considered, there may be positions considered to be partially equal with respect to different directions. To eliminate such ambiguous cases we compare clusters from the same partially equal cluster hierarchy to find clusters with the same positions and remove the cluster with the larger tolerance. Note that there is also an ambiguity between 1D and 2D partial equality. As this does not create a contradiction, we do not explicitly check for it.

Also note that we could look for distance-regular grids and circles in the planes of 2D partially equal positions and for distance-regular arrangements on the lines of 1D partially equal positions.

## 7. Scalar Parameters

Table 1 lists length and angle parameters as two types of feature objects describing the geometry of B-rep model elements independently of their location or orientation. Note that we could also add blend radii as a third parameter type derived from the blend attributes of the edges.

We find similar parameters of the same type using the hierarchical clustering algorithm with the angular or length tolerance  $\Delta T$ , and simplify the hierarchy according to the parameter type. The average parameter value found for a cluster may be close to a special value. We present methods to determine if parameter values should be replaced by special values, and to seek integer relations between parameter values below.

Note that we do not consider dependencies between the parameters to find special values. However, the hypothesizer and constraint solver have to consider dependencies, as consistency rules limiting which regularities can be selected at the same time. In a similar way, dependencies between edge parameters and surface parameters have to be taken care of. 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.

### 7.1. Special Parameter Values and Ratios

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

The functions  $f_l$  are usually of the form  $f_l : v \mapsto vK_l$ , where  $K_l$  are base units for length or angular measurements. For length units  $K_l$  depends on the measurements in the model and could for instance be 1.0, 0.1 or 2.54 (cm to inch conversion). For

angles we use base units  $\pi$  and  $\pi/180$  (degrees) and in addition the special function  $f_t : v \mapsto \arctan(v)$ .

To find special values for some  $v$ , we seek integer pairs  $n_k, m_k$  for each relevant  $f_l$  such that  $|f_l^{-1}(v) - n_k/m_k| < t_0$ .  $t_0$  depends on the cluster tolerance  $t_c$  and the appropriate  $\Delta T$ . It also depends on the scale represented by  $f_l$ . For the linear  $f_l$  we set it to  $t_0 = \max\{t_c, \Delta T\}/K_l$  to have consistent tolerance values independent of the constant  $K_l$ . As  $\arctan(x) = x + O(x^3)$ , we use  $t_0 = \max\{t_c, \Delta T\}$  for  $f_t$ , which is a good approximation as long as  $t_c$  is small. Note that for  $f_t$  we have the condition  $||v| - \pi/2| > \Delta T_A$  assuming that  $v \in [-\pi, \pi]$ , which can easily be achieved as  $v$  is an angle. The algorithm to find the integer pairs approximating some  $w = f_l^{-1}(v)$  is described in the next sub-section. Note that we may have to eliminate duplicates, if we consider multiple functions  $f_l$ , e.g.  $\arctan(1) = \pi/4$

Furthermore, we look for simple ratios between the average cluster parameters. For each pair of average parameters  $v_l$  and  $v_k$ , we seek two integers  $n_j, m_j$  such that  $|v_l/v_k - n_j/m_j| < t_r$ , where  $t_r$  is a separate user-defined tolerance, and the clusters have been ordered such that  $v_l/v_k \geq 1.0$ . 1/1 ratios are avoided as they represent similarities in the cluster hierarchy. We also require that the difference between the two parameters is larger than the maximum of the two cluster tolerances. To find the integers we use the algorithm mentioned above with  $w = v_l/v_k$ .

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

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

## 7.2. Finding Simple Fractions

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

Finding an integer relation between real numbers, using Euclid's algorithm for two numbers, or the PSLQ algorithm for more than two numbers [24], and recognising numerical constants [25] are related problems, but they assume that a close

- I. The function has been called as `simple_frac(w,n,m,s,M,neg)` where  $w$  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 with the sign indicated by  $s$ ,  $M$  is the maximum denominator and `neg` indicates whether  $w$  has to be added or subtracted from  $n/m$ .
- II. Let  $a = 1$ .
- III. While the denominator  $b = \text{round}(a/w)$  is not larger than  $M$ :
  1. If  $b > a$ :
    - A. Let  $r = w - a/b$ .
    - B. If  $r$  is negative, set `neg_r` to `true` and  $r = |r|$ . Otherwise `neg_r` is `false`.
    - C. If `neg` is `false`, the new numerator is  $p = nb + ma$ . Otherwise:
      - i. If  $nb < ma$ , the new numerator is  $p = ma - nb$  and  $s = -s$ .
      - ii. Otherwise the new numerator is  $p = nb - ma$  and `neg_r` = `not(neg_r)`.
    - D. The new denominator is  $q = ma$ .
    - E. Reduce the fraction  $p/q$  to simplest terms.
    - F. If  $r < t_0$ , add  $sp/q$  to the special values list, if it is not already in it.
    - G. If  $r > t_1$  and  $sp/q$  was not already in the special values list, call `simple_frac(r,p,q,s,M_0M,neg_r)`.
    - H. Let  $a = a + 1$ .

Algorithm 2. Recursive Algorithm for Finding Simple Fractions.

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

A straight forward method for finding fractions within a tolerance is to determine the numerators  $n$  for a given denominator  $m$  by multiplying  $w_0$  by all integers  $m = 1, \dots, M_0$ , where  $M_0$  is some maximum allowed denominator. If  $|w_0 - n/m| < t_0$  with  $n = \text{round}(w_0m)$ , i.e. if  $n/m$  is within the tolerance limit, then  $n/m$  is a suitable special fraction. However, as we also require to find fractions closer to  $w_0$ , we must use larger values for  $M_0$ , which makes this method expensive.

If  $1/(2t_0) < m$ , then more than one  $n/m$  for a fixed  $m$  could be in the interval  $(w_0 - t_0, w_0 + t_0)$ . To avoid this ambiguity we set the maximum allowed value for  $m$  to  $M_0 = \text{floor}(1/(2t_0))$ . Additionally we limit  $M_0$  by two configuration parameters  $M_{\min}$  and  $M_{\max}$ , say 3 and 10, to avoid too small or too large an  $M_0$  if we have a large or small tolerance  $t_0$ .

An alternative to the simple method is to approximate real numbers using continued fractions [26]. We approximate  $w_0$  by  $a_1 = \text{floor}(1/w_0)$  such that  $w_0 = 1/(a_1 + w_1)$  and continue approximating  $w_1$  and so on until  $w_l$  is smaller than some tolerance  $t_1$ . While this quickly computes a good approximation, it does not generate all special values close to  $w$ , especially as the iterative process based



0.59	= 1/2	+	<u>0.09</u>			
			= 1/11	[→ <b>13/22</b> ]	-	0.000909
			= 2/22		-	0.000909
	= 2/3	-	<u>0.076667</u>			
			= 1/13	[→ <b>23/39</b> ]	-	0.000256
	= 3/5	[→ <b>3/5</b> ]	-			0.01

Table 3. Finding special values for **0.59** with  $t_0 = 0.05$ ,  $t_1 = 0.01$  and  $M_0 = 5$ .

on the function  $c : x \mapsto 1/x - \text{floor}(1/x)$  behaves chaotically [27].

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

The complete recursive algorithm considering the signs of all involved values is listed in Algorithm 2. Due to the behaviour of the function  $c$  from above, it is more likely to find fractions with small denominators if  $|w_0| > 0.5$ . As  $w_0$  is the remainder to the closest integer, we have  $|w_0| \leq 0.5$ . Thus, we start with  $w = 1.0 - |w_0|$ ,  $n = 1$ ,  $m = 1$  and the flag `neg = true` (otherwise we would start with  $w = |w_0|$ ,  $n = 0$ ,  $m = 1$ , `neg = false`).  $M$  is initially  $M_0$  and  $s$  is  $-1$  if  $w_0$  is negative and  $1$  otherwise.

Table 3 contains an example for 0.59. The special values generated in the process are marked with  $\rightarrow$ . For the intermediate result  $1/2$ , which is too far away from 0.59 to be accepted as a special value, we do one recursion step to find more special values for the error 0.13.

Note that the algorithm can only miss special values whose denominator is larger than  $M_0$ . The precision increases with the depth of recursion. With appropriate  $t_1$  and  $M_{\max}$  we get the simple method as a special case.

### 8. Similar Polygonal Loops

As another regularity, we seek approximately equal edge loops in the B-rep model. Our current method only considers polygonal loops. A more sophisticated system handling all loops by considering the polygons formed from the vertices but taking the geometries of the edges into account could be based on this system. The basic

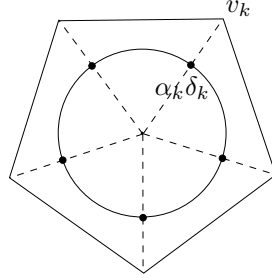


Fig. 7. Representing Polygons as Distributions on the Unit Circle

idea of our method is to represent each polygon as a function and then cluster the Fourier coefficients of these functions [28].

There are various ways to represent a polygon as a function. One way is to define a curvature of a polygon. Let  $\alpha_k$  be the angle at the  $k$ -th vertex of the polygon with  $m$  vertices  $v_k$  such that  $v_0 = v_m$  and let  $l_k$  be the accumulated length of the line segments from  $v_0$  to  $v_k$ . In our particular approach we use a *distribution* (see [29] for details)  $f$  of the form  $f = \sum_{k=0}^{m-1} \alpha_k \delta_k$ .  $\delta_k$  is the Dirac distribution defined on the unit circle  $\mathbb{T}$  with the singularity at  $2\pi l_k / l_m$  such that  $\langle \delta_k, \phi \rangle = \phi(2\pi l_k / l_m)$  for  $\phi \in \mathbf{C}^\infty(\mathbb{T})$  where  $\mathbf{C}^\infty(\mathbb{T})$  is the space of infinitely differentiable functions on  $\mathbb{T}$  in the complex plane identified with the infinitely differentiable,  $2\pi$  periodic, complex functions on  $\mathbb{R}$ . This means that we project the polygon on the unit circle and represent the projection as a sum of distributions  $\alpha_k \delta_k$ , which represent the  $\alpha_k$  at the projected position of  $v_k$  on the unit circle indicated by  $\delta_k$  (see Figure 7). The complex Fourier coefficient of order  $j$ , ( $j \in \mathbb{Z}$ ), of the distribution  $f$  is

$$u_j = \frac{1}{2\pi} \langle f, \exp(-ij \cdot) \rangle = \frac{1}{2\pi} \sum_{k=0}^{m-1} \alpha_k \exp\left(-ij 2\pi \frac{l_k}{l_m}\right). \quad (4)$$

For each polygon we compute the first  $n$  Fourier coefficients of  $f$ , which gives us a complex vector  $u$  of Fourier coefficients with  $n$  components  $u_j$ . When comparing two of these vectors  $u, v$  we use the similarity measure  $\delta(u, v) = \sum_{j=1}^d ||u_j| - |v_j||$ , comparing only the amplitude and not the phase. The averaging method for clustering is the usual weighted average of complex vectors. The vectors are clustered with a tolerance  $\Delta T_P$  and the resulting cluster hierarchy is simplified. By using the Fourier transform of  $f$ , we compare loops independently of scale. Note that  $f$  could be chosen differently and that we could also split the polygons into small sections and compare those [28]. While there are other methods to compare polygons, our method is efficient and simple, and provides a natural similarity measure.

Let  $p$  be the minimum of  $(l_{k+1} - l_k) / l_m$ , i.e. the smallest ratio between an edge length and the circumference of the polygon. Let  $P$  be the minimum  $p$  from all polygons being compared. Then at least  $\text{ceil}(1/P)$  Fourier coefficients have to be

computed to ensure that all relevant frequencies are considered. As the Fourier coefficients of  $f$  can be computed exactly, only a small number of coefficients is sufficient to characterize the shape of the polygon. Typically we use the first 10 coefficients.

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

## 9. Surface Types

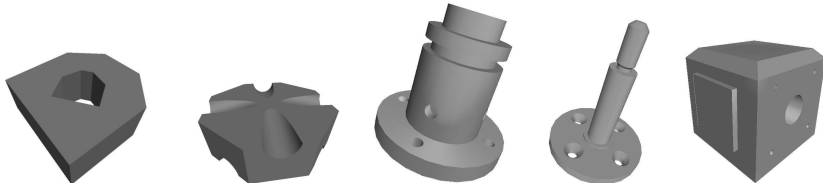
A different kind of error also needs to be corrected during beautification. For some surfaces, the surface fitting software might not have found the correct geometric type. For example, part of a cone with a small apex angle could be interpreted as a cylinder.

To determine if a surface type might need to be changed during beautification, we have to check whether each piece of surface present in the model could be well approximated by another surface type. We check if a surface is approximately a plane or a cylinder. Note that a cylinder might also be approximately a cone, or a plane might be approximately a sphere, etc., but for such cases, we do not have any indication of parameter values to use for the alternative surface. Furthermore, for beautification our aim is to replace more complex surfaces by simpler ones, not vice versa.

To decide if a piece of surface might need replacing, we use the curvature of the surface, which is also used for determining the surface type in the model building software [1]. For planar, cylindrical, spherical and toroidal surfaces the principal curvatures  $k_1$ ,  $k_2$  are constant. If  $\sqrt{k_1^2 + k_2^2}$  is approximately zero as indicated by some tolerance  $t_c$ , the surface is approximately a plane. If only one  $k_i$  is approximately zero, the surface is approximately a cylinder with radius  $1/k_j$ , where  $k_j$  is the second curvature.

We have to take special care for a cone, as one of the principal curvatures is not constant. Let  $k_2(p)$  be the non-constant principal curvature at a point  $p$  on a conical face. Conical faces for which the surface type can be changed do not contain the apex and are always finite, so  $k_2$  has a minimum and maximum. If  $\max_p k_2(p) - \min_p k_2(p)$  is approximately zero with respect to  $t_c$ , then the cone is approximately a cylinder or a plane. It is approximately planar if  $K = (k_2(p_{\min}) + k_2(p_{\max}))/2$  is approximately zero, otherwise it is approximately a cylinder with radius  $1/K$ .

However, it is not sufficient to only check the curvature to decide if a surface should be of a different type. It has to be possible to change the surface within the combinatorial topological structure of the model. For this we check if all the boundary loops of the surface are also approximately on the alternative surface. This depends on the surface types involved and basically requires fitting a surface



	(a)	(b)	(c)	(d)	(e)
<b>Time (sec):</b>	0.83	0.89	0.59	0.14	2.2
<b>Desired:</b>	56	28	31	25	67
<b>Unwanted:</b>	8	5	1	1	16
<b>Missed:</b>	0	2	4	0	9

Fig. 8. Experiments with Simulated Range Data.

of an appropriate type to special points taken from the boundary in a least-squares sense. In addition, closed boundary curves which cannot be transformed continuously in the face to a point are an indication that a surface cannot be a plane, but it could be a cylinder.

## 10. Experiments

We now give some experimental results to demonstrate how our methods work in practice. The methods were implemented\* on a GNU/Linux platform (Pentium III, 450 Mhz, 256MB memory), and were tested on real reverse engineered and simulated data from various engineering objects. We present some results of the test runs with simulated data, and detailed results of testing the methods with a real data set, and discuss the general results.

### 10.1. *Experiments with Simulated Data*

By rotating and translating faces of exact models and generating a point cloud from these perturbed objects, we created simulated range data. From the point cloud we used reverse engineering software to obtain reconstructed models (see Section 1 and [6]). The resulting initial models were then analysed by our methods.

Some of our test objects are shown in Figure 8, where (c), (d) were obtained from [30]. Faces were perturbed by  $2^\circ$ . The length perturbation was about 1% of the corresponding  $L_{\max}$  (about 0.1 to 0.2 length units). In order to consider the effects of the hierarchical structures, the tolerance values for clustering were set to half of these amounts, i.e.  $\Delta T_L = 0.005L_{\max}$  and  $\Delta T_A = 1.0^\circ$ . For simplicity, feature objects from edges were not considered. As for the simulated data we have a known upper tolerance limit, we ignored any clusters which had tolerances larger than  $2^\circ$  and  $0.01L_{\max}$  length units. We only counted regularities generated by clusters with tolerances just under these limits and not any of their sub-clusters. This

\*The sources are available under the GNU General Public License at <uri: <http://www.langbein.org/software/sil/>>.

gives a better picture of the actual desired regularities found and takes into account that subsequent steps have to identify regularities at high tolerance values which would require big changes to the model. Furthermore, we did not count the special values found as regularities for the results. In Figure 8 we also list the number of desired, unwanted and missed regularities found. An unwanted regularity is one that is not part of the design and conflicts with the desired regularities. Missed regularities represent major regularities not detected by our methods rather than all valid relations between the feature objects which were undetected.

Object (a) consists of two planar angle-regular arrangements of planes with base angle  $\pi/4$  which have been detected. However, an alternative arrangement combining both into a single regularity with base angle  $\pi/18$  has also been detected at a larger tolerance level. Further unwanted regularities were conical angle-regular arrangements with cones with semi-angles close to  $\pi/2$  created by the planar faces. Further regularities included aligned axes, axis intersection points and similar polygonal loops.

In Object (b) the planar angle-regular direction sets of the planar faces and the conical angle-regular sets of the cylinder axes were detected. Additional conical angle-regular sets of cylinder and plane axes were detected, some of which were unwanted. Unfortunately the intersection point of the axes generated from the centre of the convex hulls of the side faces was not detected correctly, as the perturbation of the side faces moved some of the axes too far apart. This was also the cause that one aligned axis pair was missed.

Analysing (c) resulted in a number of equal cylinder radii with special values. The methods detected that all the directions in the model are part of an orthogonal system, with one main direction of parallel axes. Furthermore, it was found that all axes of the main cylinders are aligned except for one. The axes of the opposite holes in the sides of the large cylinder were also found to be parallel and to all lie in a plane with an intersection point on the central axis. The planar angle-regular algorithm detected their symmetrical arrangement. Of the four intersection points of the axes of the holes in the large cylinder and the axes of the holes in the planar surface at the bottom only one was detected. The symmetrical arrangement of the cylinder axes on a cylinder aligned with the main central axes was also detected.

Object (d) has a single main axis which was detected. Furthermore, the regular arrangements of the holes on a circle was found. The remaining regularities were similar length and angle parameters. The methods erroneously suggested that two cylinder radii which were close to each other may be equal, which would require a change in the combinatorial topological structure.

In object (e) the orthogonal arrangements of the planes and the conical angle-regular arrangements of the planes at the top were detected. One planar angle-regular arrangement was missed as it required a larger tolerance setting. Further regularities included aligned axes, of which a few were not detected. The regular arrangement of the cylinders was only partly detected with one cylinder missing from the circular arrangement and two cylinders missing from the grid.

By adjusting the tolerance values it was possible to include nearly all of the missed regularities mentioned above. However, this also added further unwanted regularities at higher tolerance values while the desired ones were still present at lower tolerance values. The decision about which regularities are desired has to be made by subsequent beautification steps and not by the analyser.

The experiments with simulated data showed that when using small tolerance levels we find a few, very accurate, and thus also very likely regularities. At larger tolerance levels, we detect more desired regularities and eventually all desired regularities are found. However, by increasing the tolerance levels we also increase the likelihood of detecting unwanted regularities. Only for simple objects are there tolerance levels which distinguish exactly between desired and unwanted regularities.

Because of the latter observation we do not use upper limits for the regularity tolerances, but instead we simplify the cluster hierarchy. The subsequent beautification steps have to select a suitable subset of all the regularities detected, using geometric reasoning. Often, regularities at high tolerance levels are inconsistent with regularities at lower levels. If this is detected the regularities at the higher levels can be identified as unwanted (see, for instance, the two angle-regular arrangements in object (a) which were combined to give a single angle regularity at a higher tolerance level). Besides the tolerance levels, the subsequent steps can also consider the kind of each regularity, and, for instance, prefer an orthogonal system at a higher tolerance level to setting the angles to  $89^\circ$ . To help make such decisions, we could also consider if a regularity would require a change in the combinatorial topological structure of the model. Additionally, we could avoid accepting regularities at extremely different tolerance levels in different parts of the object when taking a global view. However, especially if the object has been created by combining many individual views with possibly different scanner settings, the tolerance levels may not necessarily be consistent.

## 10.2. *An Experiment with Real Data*

Next we discuss in detail the results of analysing a model reverse engineered from real range data. The reverse engineered model is shown in Figure 9. For simplicity we did not consider feature objects generated by edges. To analyse the model we used  $\Delta T_A = 1^\circ$  and  $\Delta T_L = 0.2$  length units (2% of the length of the model), which created a good set of regularities. In the following we present the regularities and the cluster hierarchies in detail. Note that due to the hierarchies, we cannot easily count the regularities in a consistent way like we did for the simulated data, since we did not remove regularities above certain tolerance levels.

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

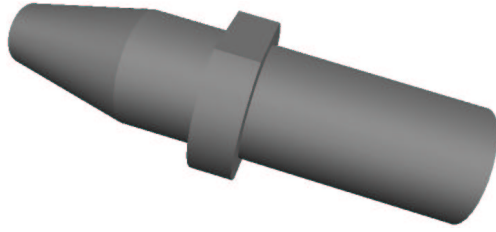


Fig. 9. A Model Reverse Engineered from Real Range Data.

clusters of another cluster. It was also found that these two planes were orthogonal to the main axis.

As the two plane directions were considered not to be parallel at a small tolerance level, some unwanted regularities were reported. There was one suggesting some special values for the angle between the directions, and a conical angle regularity consisting of the two plane directions and the main axis. Both regularities would only be realisable if the two directions are not parallel. These regularities were detected as our methods consider all sub-clusters at the different tolerance levels, in order to detect possible regularities created by them. Dependencies like this can easily be found in the subsequent steps by detecting that different values for the same angles are required by the regularities. We can either make the two planes parallel and then also orthogonal to the main axis or we accept the two unwanted regularities mentioned above. In the first case we accept a regularity at a larger tolerance level, as a parallel regularity appears to be desirable and allows other desirable regularities to be realised.

From the parallel direction cluster representing the main axes the methods further detected that the axes of the cylinders and the cone are aligned. The axis of the larger cylinder was slightly further away from the average of the aligned axes (about 0.3 length units) such that it was a sub-cluster of the main regularity.

Some of the surface root point positions and vertex positions were considered to be equal in the position cluster hierarchy. All clusters had tolerances above 2 length units and would require changes to the combinatorial topological structure of the model which would remove large surfaces.

Two pairs of similar cylinder radii were detected at the lowest tolerance level, which were considered to be equal at a higher tolerance level. Making them equal would require a change in the combinatorial topological structure.

In general the results for real data are similar to those for simulated data. The major difference is that we cannot rely on consistent tolerance levels for the complete model, which further shows the advantages of not using a maximal tolerance. It appears that the desired regularities can be separated from the unwanted regularities with the methods already mentioned briefly for the simulated data. Further experiments with methods developed for the subsequent beautification steps have

to be conducted, though.

## 11. Conclusions

We have presented algorithms based on similarities to find approximate geometric regularities in inaccurately reverse engineered B-rep models. Instead of using various thresholds to decide which regularities are present in the model, the methods use cluster hierarchies, and list which regularities are present at which tolerance level. Tests with various perturbed objects were satisfactory in the sense that most desired geometric regularities were found and appear to be suitable for the subsequent beautification steps. Unwanted regularities, especially at larger tolerance levels, are also reported and will have to be identified in the subsequent beautification steps using geometric reasoning.

The methods given here could be expanded to find additional types of regularities based on the same principles. The methods could also be modified to handle other face and edge types by defining feature objects for them. Furthermore, a machining feature recognizer could be employed to partition the model into interesting subsets which could be analysed and beautified separately before they are combined using higher level beautification on the whole model. Especially for more complex models this might improve the results.

In future work we intent to develop a system which tries to find a maximal, consistent set of constraints describing the main regularities, which will be used to generate an ideal model. This includes developing methods to detect inconsistencies between the constraints based on geometric reasoning, and using optimization and graph-based methods to solve geometric constraints, while detecting inconsistencies between the constraints. We also intent to develop decision methods to resolve conflicts between contradictory regularities.

## Acknowledgements

This project is supported by the UK EPSRC Grant GR/M78267. The authors would like to thank T. Várady from the Hungarian Academy of Sciences and CAD-MUS Consulting and Development Ltd. for providing reverse engineering software.

## References

1. G. Lukács, R. R. Martin, A. D. Marshall. Faithful Least-Squares Fitting of Spheres, Cylinders, Cones and Tori for Reliable Segmentation. In: H. Budkhadj, B. Neumann (eds), *Proc. 5th European Conf. Computer Vision*, 2–6 June, Albert-Ludwigs Universität, Freiburg, Germany, Vol. 1, pp. 671–686. Springer, Berlin, Heidelberg, New York, 1998.
2. N. Werghe, R. B. Fisher, A. Ashbrook, C. Robertson. Faithful Recovering of Quadric Surfaces from 3D Range Data. In: *Proc. 2nd Int. Conf. on 3D Digital Imaging and Modelling*, 4–8 October, Ottawa, Canada. IEEE Computer Society, Los Alamitos, CA, USA.



3. 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>>.
4. N. M. Samuel, A. A. G. Requicha, S. A. Elkind. Methodology and Results of an Industrial Part Survey. Technical Report TM-21, Production and Automation Project, College of Engineering & Applied Science, University of Rochester, July 1976.
5. G. Kós, R. R. Martin, T. Várady. Methods to Recover Constant Radius Rolling Ball Blends in Reverse Engineering. *Computer-Aided Geometric Design*, 17(2):127-160, 1999.
6. P. Benkő, R. R. Martin, T. Várady. Algorithms for Reverse Engineering Boundary Representation Models. *Computer-Aided Design*, 33(11):839-851, 2001.
7. T. Várady, R. R. Martin, J. Cox. Reverse Engineering of Geometric Models - an Introduction. *Computer-Aided Design*, 29(4):255-268, 1997.
8. D. W. Eggert, A. W. Fitzgibbon, R. B. Fisher. Simultaneous Registration of Multiple Range Views for Use in Reverse Engineering of CAD Models. *Computer Vision and Image Understanding*, 69(3):253-272, 1998.
9. G. Kós. An Algorithm to Triangulate Surfaces in 3D Using Unorganised Point Clouds. *Computing*, Suppl 14:219-232, May 2001.
10. P. Benkő, G. Kós, T. Várady, L. Andor, R. R. Martin. Constrained Fitting in Reverse Engineering. To appear in *Computer-Aided Geometric Design*, 2001.
11. C. Robertson, R. B. Fisher, N. Werghi, A. P. Ashbrook. Fitting of Constrained Feature Models to Poor 3D Data. In: *Proc. Adaptive Computing in Design and Manufacture*, pp. 149-160, Plymouth, UK, April 2000.
12. 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.
13. N. Werghi, R. B. Fisher, A. Ashbrook, C. Robertson. Improving Model Shape Acquisition by Incorporating Geometric Constraints. In: A. F. Clark (ed), *Proc. British Machine Vision Conference*. BMVA, September 1997.
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. 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, 2001.
16. 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, Los Alamitos, CA, USA, 2001.
17. B. I. Mills, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate Symmetry Detection for Reverse Engineering. In: D. C. Anderson, K. Lee (eds), *Proc. 6th ACM Symp. Solid Modelling and Applications*, pp. 241-248. ACM Press, 2001.
18. J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.
19. J. Zupan. *Clustering of Large Data Sets*. Research Studies Press, Letchworth, Hertfordshire, England, 1982.
20. R. Agarwala, V. Bafna, M. Farach, B. Narayanan. On the Approximability of Numerical Taxonomy (Fitting Distances by Tree Metrics). *Computing*, 28(3):1073-1085,

- 1999.
21. P. N. Yianilos. Data Structures and Algorithms for Nearest Neighbour Search in General Metric Spaces. In: *Proc. 4th ACM-SIAM Symp. Discrete Algorithms*, 25–27 January, Austin, TX, USA, pp. 311–321. ACM Press, 1993.
  22. D. Eppstein. Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs. *ACM Journal of Experimental Algorithmics*, 5(1):1–23, 2000.
  23. J. A. Nelder, R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.
  24. H. R. P. Ferguson, D. H. Bailey. A Polynomial Time, Numerically Stable Integer Relation Algorithm. RNR Technical Report RNR–91–032, NASA Ames Research Center, MS T045–1, Moffett Field, CA 94035–1000, USA, December 1991.
  25. D. H. Bailey, S. Plouffe. Recognizing Numerical Constants. In: *Proc. Organic Mathematics Workshop*, Vol. 20, pp. 73–88. Canadian Mathematical Society, 1997. <uri: <http://www.cecm.sfu.ca/organics/papers/bailey/>>.
  26. A. Y. Khinchin. *Continued Fractions*. Dover Publications, New York, 1997.
  27. R. M. Corless. Continued Fractions and Chaos. In: *Proc. Organic Mathematics Workshop*, Vol. 20. Canadian Mathematical Society, 1997. <uri: <http://www.cecm.sfu.ca/organics/papers/corless/>>.
  28. S. Berchtold, D. A. Keim, H. P. Kriegel. Using Extended Feature Objects for Partial Similarity Retrieval. *The VLDB Journal*, 6:333–348, 1997.
  29. R. Dautray, J. L. Lions. *Mathematical Analysis and Numerical Methods for Science and Technology, Functional and Variational Methods*, Vol. 2. Springer, Berlin, Heidelberg, New York, 1988.
  30. Drexel University. National design repository. <uri: <http://edge.mcs.drexel.edu/repository/>>.