Topological and Geometric Beautification of Reverse Engineered Geometric Models

F. C. Langbein A. D. Marshall R. R. Martin
School of Computer Science, Cardiff University, PO Box 916, Cardiff, CF24 3XF, UK,
{F.C.Langbein,Dave.Marshall,Ralph.Martin}@cs.cf.ac.uk.

B. I. Mills
Inst. Information and Mathematical Sciences, Massey University, Auckland, Albany, NZ, b.i.mills@massey.ac.nz.

C. H. Gao
School of Manufacturing Science and Engineering, Sichuan University, Chengdu, China, chunhua_gao88@hotmail.com.

Abstract

Boundary representation models reverse engineered from 3D range data suffer from various inaccuracies caused by noise in the measured data and the model building software. Beautification aims to improve such models in a post-processing step solely working with the boundary representation model. The improved model should exhibit exact geometric regularities representing the original, ideal design intent. We present a set of algorithms for a complete beautification system which improves the topology and geometry of a reverse engineered boundary representation model with respect to design intent. This includes practical methods to correct localised topological defects such as holes and pinched faces, and algorithms to reliably detect approximate regularities such as approximately aligned cylinder axes, approximate symmetries and congruences. Furthermore, as the detected regularities are likely to be mutually inconsistent, an efficient method to select and impose a consistent set of regularities is discussed. Experiments show that simple to medium complexity models can be improved with respect to design intent.

Keywords: Beautification; Topological Defects; Approximate Geometric Regularities; Reverse Engineering.

1 INTRODUCTION

Reverse engineering extracts sufficient information from physical objects to reconstruct CAD models for a particular purpose like redesign, reproduction or quality control. Ideally, for applications like redesign, the reconstructed model should exhibit exactly the same geometric properties present in the original, ideal design. Rather than trying to create a description of the exact measured physical object, suitable for creating an exact copy or for inspection purposes, we are interested in reverse engineering the shape of an engineering object such that the description represents the original design intent. For this purpose we use a state-of-the-art reverse engineering system which can create a valid boundary representation (B-rep) model of the object's natural surfaces from dense 3D range data. However, due to inaccuracies in the measured data, approximation and numerical errors during the reconstruction process, and possible wear of the scanned object, the model is approximate in the sense that it exhibits intended regularities such as symmetries only approximately. We present a system to automatically improve such models in a post-processing step, which we call *beautification*.

For an overview of reverse engineering systems see [28]. Firstly, 3D range data from multiple views of the object is collected. In a registration step these views are combined to give a single 3D point set [5, 24]. The most crucial phase of the process is segmentation and surface fitting, where the natural surfaces of the object have to be determined and surfaces of suitable geometric types have to be fit. We first create a triangular mesh for the object surface [12]. The mesh is segmented into subsets representing natural surfaces of the object. For each of the subsets, one or multiple analytic surfaces are fitted [3, 29]. However, segmentation and surface fitting cannot be separated completely from each other, and the methods have to be carefully chosen to work together. During segmentation, we already have to consider the surface types to be fitted to the subsets. Finally, a complete B-rep model is created by stitching the fitted surfaces.

The system described can only handle certain object types robustly. Therefore, we consider here engineering objects composed only of planar, spherical, cylindrical, conical and toroidal surfaces that either intersect at sharp edges or are connected by fixed radius rolling ball blends. There are reliable surface fitting methods available for these surfaces [3, 29] and many realistic engineering objects can be described using only these surface types [22, 25]. We ignore blends in this paper, but Kós et al. [13] present a method to determine the radius of fixed-radius rolling ball blends. We can store such information as edge attributes ,and instruct the modelling kernel to construct appropriate blends after the beautification step.

Instead of fitting surfaces individually, there are alternative approaches which fit surfaces simultaneously under certain conditions. Thompson et al. [27] consider feature-based reverse engineering of mechanical parts. In their system a human identifies features like slots and pockets in the 3D point set interactively. The system requires user identification of the type and the approximate location of each feature. This information can then be used to reconstruct the model by fitting parametric feature models instead of simple surfaces to the 3D point set, which improves the accuracy of the generated models.

Alternatively we can fit multiple surfaces to 3D point sets under geometric constraints [2, 30]. So, rather than fitting surfaces individually, they are fitted simultaneously using the constraints as a set of conditions which the surface parameters have to fulfil in addition to providing a good fit to the 3D point data.

Both approaches require human interaction because low-level information about an object's surface in form of point sets is often not enough to make decisions about higher-level design intent. We try to avoid the necessity for human interaction by first extracting a higher-level B-rep model with analytic, natural surfaces. From this representation, further information about the actual design intent can be derived automatically. By trying to improve the B-rep model without further reference to the point data we also significantly reduce the computing time.

Beautification is a final step in producing a solid model from the

range data. It has to address defects in the topology as well as adjust the geometry. For example, if we reverse engineer a four-sided pyramid, we expect all four sloping faces to meet at a single vertex at the top. Equally important is that the sloping faces are arranged symmetrically to form a regular pyramid. We address both problems in sequence. First we detect and fix topological defects. Then we seek potential geometric regularities. Typically, a large number are found, not all of which need be mutually consistent. Thus, a consistent subset of these regularities, which is likely to represent the original design intent and describes the complete improved model, is selected. Our overall beautification process consists of the following main steps:

I. **Detecting topological defects**: small faces, sliver faces, short edges, gaps in the model arising from missing scan data, etc., are identified.
II. **Adjusting the topology**: isolated small faces and short edges are replaced by a single vertex, and surrounding topology is adjusted to meet it; existing faces are extended to cover gaps left by missing data, by removing the edges and loops bounding gaps; etc.
III. **Detecting approximate geometric regularities**: symmetric arrangements of faces, other regular arrangements, etc. which are approximately present in the geometry are detected; exact conditions for approximate regularities are used rather than arbitrary tolerances, and the methods aim to detect sufficient regularities to determine the improved model.
IV. **Selecting geometric regularities**: a consistent set of geometric regularities is selected which is likely to describe the design intent of the model; to do this, the regularities are expressed by constraints, and methods to determine the solvability of constraint systems and the likelihood of a regularity being part of the ideal design are employed.
V. **Rebuilding an improved model**: from the solution of the selected constraint system, an improved model is rebuilt also using the updated model topology; here remaining topological defects have to be detected and either fixed immediately or the process has to be restarted at step II.

Using this approach, the topological changes desired may not be geometrically realisable; this is only determined when the geometry is adjusted after the topology. Hence, model rebuilding may fail. In this case we can either try to fix the model during rebuilding, e.g. fill holes with additional faces, or return to the topological beautification phase and choose an alternative topology. For typical reverse engineered objects, the topological defects are localised in the sense that the interaction between multiple topological defects is limited to local faces rather than the global structure. This allows us to fix topological defects of different types in a well-defined sequence, and limits the possible inconsistencies between topological and geometric adjustments. Also note that we aim to change the model by a relatively small amount—just enough to impose approximate geometric and topological regularities on the model which are present within a small tolerance. But unless we have an unstable model that can only be realised by a very particular set of parameters, small topological and geometric adjustments are likely to be compatible.

In the following we describe our algorithms for topological and geometric beautification. We then present the results of some experiments and conclude with a discussion of the abilities of the current system. As will be shown, our system is able to improve models of simple to medium complexity with respect to design intent.

2 TOPOLOGICAL BEAUTIFICATION

We first detect and fix topological defects occurring in reverse engineered B-rep models. The following tasks are carried out for fixing defects arising during earlier model building processes:

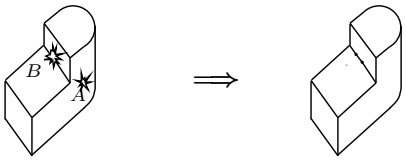


Fig. 1: Repairing a Face Gap and an Edge Gap

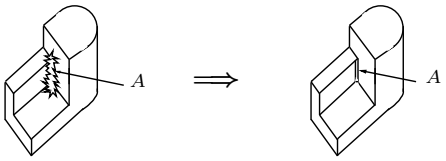


Fig. 2: Repairing a Complex Multiple Face Gap

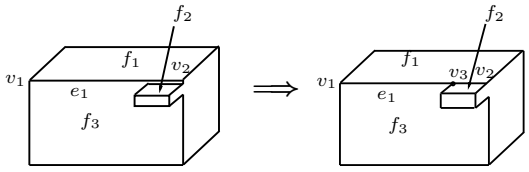
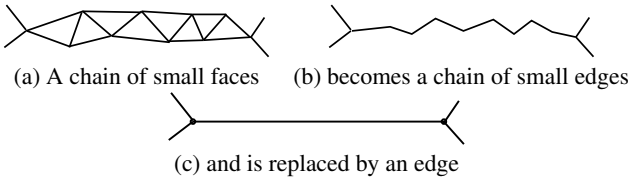


Fig. 3: Repairing a *Spiking* Pinched Face



(a) A chain of small faces (b) becomes a chain of small edges

(c) and is replaced by an edge

Fig. 4: Removing a Chain of Small Faces

- **Removing gaps in a single face**: A loop of half-edges may exist in the interior of a face, with nothing on the other side of the loop. Such a case may arise, for example, where the scanner did not collect any data from within a deep concavity in the face. Here the loop of half-edges should be removed, extending the face (see A in Fig. 1).
- **Removing gaps crossing an edge**: A loop of half-edges may span two faces, with nothing on the other side of the loop. The edge between the faces is divided into two pieces by the gap. The gap should be removed, the existing faces extended, and the two edge pieces joined (see B in Fig. 1).
- **Removing gaps spanning multiple faces**: A loop of half-edges may span multiple faces, with nothing on the other side of the loop. Existing faces and edges must be extended to fill the gap, and new vertices and edges must be added as needed (see A in Fig. 2).
- **Adjusting pinched faces**: If a face narrows to a very thin part it is *pinched*. Other parts of the model should be adjusted to remove the thinning, resulting in a change in connectivity of the face (see Fig. 3).
- **Merging adjacent faces with the same geometry**: Two adjacent faces may share the same geometry across a contiguous edge sequence. Edges and vertices as appropriate should be removed, and the faces merged.
- **Merging adjacent edges with the same geometry**: Two consecutive edges may share the same geometry, and are the only edges meeting at a vertex. The vertex should be removed, and the edges merged.
- **Removing chains of small faces**: Faces should meet in an edge, but instead a chain of small faces may separate them.

Modification order	1	2	3	4	5	6	7	8	9	10	11
Fixing → can cause ↓	Face gaps	Edge gaps	Multi-face gaps	Pinched faces	Small face chains	Sliver faces	Small edge chains	Adjacent faces	Adjacent edges	Small faces	Small edges
Face gaps	No	No	No	No	No	No	No	No	No	No	No
Edge gaps	No	No	No	No	No	No	No	No	No	No	No
Multi-face gaps	No	No	No	No	No	No	No	No	No	No	No
Pinched faces	No	No	No	No	No	No	No	No	No	No	No
Small face chains	No	No	Yes	Yes	No	No	No	No	No	No	No
Sliver faces	No	No	Yes	No	No	No	No	No	No	No	No
Short edge chains	No	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
Adjacent faces	No	No	No	Yes	Yes	Yes	No	No	No	No	No
Adjacent edges	No	Yes	No	Yes	Yes	Yes	Yes	No	No	No	No
Small faces	No	No	Yes	Yes	Yes	No	No	Yes	No	No	No
Small edges	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No

Table 1: Sequence of Topological Repair

The chain of small faces should be replaced by an edge (see Fig. 4, where the first step is to reduce a chain of small faces to a chain of short edges).

- **Removing chains of short edges**: Several consecutive short edges may need to be replaced by a single long edge (see Fig. 4).
- **Removing isolated small faces**: Several edges should meet in a single vertex, but instead they meet at several distinct vertices, joined by several short edges which surround a small face. The small face should be replaced by a vertex connected to the existing edges.
- **Removing isolated short edges**: Several edges should meet at a single vertex, but instead they meet at several distinct vertices, joined by one or more short edges. These short edges should be replaced by a vertex.
- **Removing sliver faces**: Two faces should meet in an edge, but instead a long very thin face (a *sliver* face) may separate them. The sliver face should be replaced by an edge.

Topological beautification may thus involve the local addition or removal of faces, edges and vertices, and other updates to nearby topology to ensure that a correct, valid model results. For example, edges may need to be disconnected from an existing vertex, and connected to a new vertex.

Topological beautification as outlined above has some similarities to, but also some differences from, CAD model healing [4, 21, 23]. Healing, like topological beautification, aims to improve the topology, but the differences are that (i) it starts with invalid models, not valid ones, and (ii) it makes changes whose main aim is to ensure a valid model is the result. For example, healing may have to cope with problems such as physically impossible geometry, incorrectly oriented surfaces, faces with no defined geometry, self-intersecting edges, faces whose boundary is not a closed loop, and incomplete topology even though all individual faces are present. Note that such problems of validity are *not* expected to occur in beautification. Approaches to healing usually distinguish between topological and geometric problems and often involve user interaction to resolve difficult cases.

2.1 Tolerances and Processing Order

In our algorithm, we use a tolerance to determine which faces are small or pinched, which edges are short, etc. This tolerance can be detected automatically by a consistent clustering method as described in Section 3. It may also be provided by the user based on the magnitude of errors expected in the model (for example, from knowledge of the scanner and reverse engineering algorithm characteristics). All tolerances we require can be computed from a tolerance for vertex distances. In the following we implicitly assume that phrases like "a face is small" mean that an appropriate tolerance is used to make the corresponding decision.

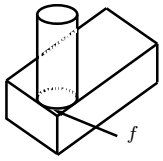


Fig. 5: A necessary Small Face

When choosing a tolerance, we should be careful that small but significant parts of the model are not deleted. For example, in Fig. 5, face f may be small, but is necessary, and we should not attempt to remove it. Thus, the length tolerance should be larger than the size of any small face or short edge which is to be deleted, but smaller than any part of the model which is to be retained. For simplicity, we assume here that a single global tolerance value is used, satisfying this requirement. However, if, for example, different regions of the object were scanned at different resolutions, or if large features exist for which the scanner could not capture high quality data, a more sophisticated approach with an adaptive tolerance might be needed.

In a raw reverse engineered model, multiple topological defects of multiple types will usually coexist. To efficiently resolve these problems, we need to detect and modify the defects in the right order. In particular, to avoid having to use a loop which considers different defect types repeatedly, we need to fix some types of defects earlier than others. For example, fixing a gap spanning multiple faces may produce a sliver face, but fixing a sliver face can never produce a gap.

Table 1 lists all the cases in which solving a topological defect of a given type may create further topological defects of a different type. The columns are topological defects to be solved in order, and the rows are new problems which may arise from solving them, *assuming the ordering given*. As items above the diagonal in Table 1 are all "No", no repair in the sequence shown can cause a problem of a type already fixed earlier. Note carefully the logic. Given *this* particular ordering, certain defect types are known not be present at each stage, having been fixed earlier. Thus, certain potential complex interactions between multiple defect types can be ignored as "cannot arise". Table 1 shows just one self-consistent ordering in which defects of the various types can be solved sequentially. Other orderings may also be possible.

2.2 Fixing Topological Defects

We detect and fix the topological defects in the order given by Table 1. We first detect the topological defects of a given type, then immediately fix these problems, and continue with the next type of topological defect. For efficiency, the information gathered when detecting earlier problems may be reused when detecting later prob-

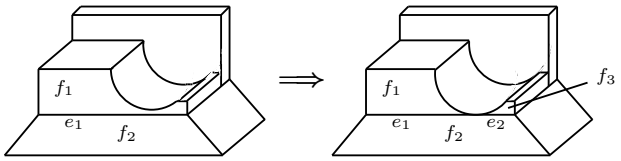


Fig. 6: Fixing a Pinched Face Creates a Small Face

lems. For instance, to find face gaps, we seek edges bounding only a single face. Such edges are relevant for face, edge and multiple face gaps, depending on how many faces are involved.

In detail, topological fixing is carried out as follows; brief justification of the order used is also given:

1. **Face gaps:** Fixing all types of gaps is done first—no other topological changes introduce new gaps. Fixing face gaps also decreases the number of edges in the model, speeding up subsequent processing. Fixing face gaps does not produce any new problems, and fixing a face gap may also prevent a face from being considered to be pinched at a later stage.
2. **Edge gaps:** When we remove a gap lying across an edge, it may create a short edge, as shown in Fig. 1: removing gap B produces an edge e which may be short. Other considerations are similar to those for face gaps.
3. **Multiple face gaps:** Fixing a gap across multiple faces can create a small face or a sliver face, as shown in Fig. 2. Fixing such gaps adjacent to a small face may also create new small faces and short edges. New sliver faces may also arise, e.g. when gap A in Fig. 2 is fixed.
4. **Pinched faces:** Fixing pinched faces may produce problems such as small faces (see f_3 in Fig. 6(b)), short edges (see e_2 in Fig. 6(b)), adjacent faces with the same geometry (see f_1 and f_2 in Fig. 3(b)), and adjacent edges with the same geometry (see e_1 and e_2 in Fig. 3(b)). Thus, we fix pinched faces after removing gaps, but before fixing other problems.
5. **Chains of small faces:** Chains of small faces should be processed as a whole, rather than face by face. Removing a chain of small faces results in a chain of short edges (see Fig. 4). Processing the resultant chain of short edges can lead to adjacent faces with the same geometry. We also need to remove chains of small faces before single small faces, and short edges. Thus, chains of small faces are handled now.
6. **Sliver faces:** Removing a sliver face can produce adjacent faces or edges with the same geometry. It may also produce short edges, or even a chain of short edges in some cases.
7. **Chains of short edges:** Chains of short edges should be processed as a whole, rather than edge by edge (see Fig. 4). Replacing a chain of short edges by an edge may produce a new short edge, or adjacent edges with the same geometry.
8. **Adjacent faces with same geometry:** Merging adjacent faces with the same geometry creates adjacent edges with the same geometry. It may also result in new small faces and edges.
9. **Adjacent edges with same geometry:** Fixing adjacent edges with the same geometry can produce a new short edge if, e.g., the adjacent edges are short, and the new edge formed is still short.
10. **Small faces:** Removing small faces can create new short edges.
11. **Short edges:** Fixing short edges is the final step, which cannot produce any further problems.

The input to our topological beautification algorithm is a raw, reverse-engineered B-rep model, together with a tolerance value. The output is a B-rep model with modified topology. The geometry

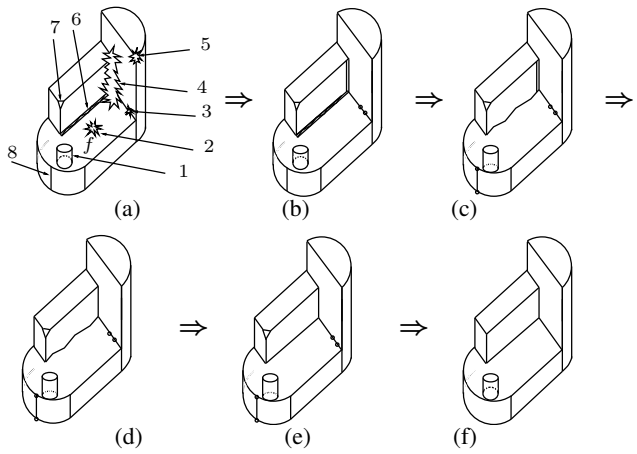


Fig. 7: Modifying a Complex Model

is adjusted to meet the new topology (and geometric regularities enforced) in the geometric beautification phase.

Detecting the topological defects can in principle be done by a linear scan of the faces and/or edges in the model by checking a simple condition which depends on the defect type. Note that this only applies to local topological defects, which are typical for reverse engineered models. Defects relating to the global structure, e.g. chains of small faces along all natural edges of the model, cannot be handled in this way. However, such cases arise due to poor range data, and the problem should be solved by obtaining more accurate data.

Fixing topological defects requires relatively simple local adjustments of the topology like adding/removing some vertices, edges or faces and connecting the resulting structure appropriately. Hence, overall the linear scan of the model determines the time order of the algorithm which increases linearly with model complexity, as justified in detail in [8].

2.3 Topological Beautification Example

A raw model illustrating topological beautification is shown in Fig. 7(a). In this model, cylindrical boss 1 is near the edge of face f, causing it to be pinched. Problems 2–5 are gaps of various kinds. Problem 6 is a chain of several small faces. 7 is another small face, while the faces adjacent to edge 8 have the same geometry.

We first detect gaps by finding edges with only one coedge. We then find loops of such edges, which gives four loops for gaps 2, 3, 4 and 5. There are no edges adjacent to 2 so it is a face gap. There are two edges connected to 3, so it is an edge gap. 4 and 5 are multiple face gaps. Each face gap is removed by extending the face. Each edge gap is removed by extending the faces and connecting the edges meeting the gap. For each multiple face gap, we extend faces adjacent to the gap and intersect them as appropriate, possibly adding a new face as well as new edges and vertices. For gap 4, a new long and thin face (a sliver face) is created. For gap 5, the faces intersect in a single point, so we simply insert a new vertex, as shown in Fig. 7(b). Next, we fix pinched faces addressing problem 1 and remove chains of small faces, 6, as shown in Fig. 7(c). Then we replace the sliver face arising from gap 4 by an edge as shown in Fig. 7(d). The chain of short edges arising from the chain of small faces 6 is replaced by a single edge (see Fig. 7(e)). Next we seek adjacent faces having the same geometry; faces adjacent to edge 8 with approximately the same geometry are detected. Afterwards, we find and merge adjacent edges with the same geometry, which were produced by merging the edges on either side of edge 8. Finally, the remaining small face 7 is removed (see Fig. 7(f)).

3 GEOMETRIC BEAUTIFICATION

After topological beautification we still have to adjust the geometry to agree with the new topology. Furthermore, we also wish to adjust the geometry so that the model exhibits exact intended geometric regularities which were possibly only approximately present in the raw model. For this we generate a constraint system which completely describes the geometry of the beautified model for rebuilding.

We present a general concept for approximate geometric regularities based on an exact notion of approximate symmetry. As we aim to detect a large number of regularities, it is likely that they are not all mutually consistent. Hence, we have to select a set of consistent regularities which completely describes the likely design intent of the model. In our current approach we consider adding constraints describing the regularities one-by-one, in order of a priority, to a constraint system. A regularity is only selected if the constraint system remains solvable after adding the regularity. We give here an overview of the core concepts of geometric beautification; for details see [14].

3.1 Approximate Geometric Regularities

A large class of common regularities can be described in terms of *feature* symmetries. By features we do not mean machining features like slots, but properties of B-rep elements (called *cells* in the following) which change in a similar way to the element under isometric transformations. Each feature has a type such as position or direction. For instance, the radii of a torus are length features, and the direction of the axis is a direction feature. We require that the relations between features of the same type from different cells are preserved when transformed by the same isometry. For instance, length features do not change under isometries at all, so the relations are automatically preserved. While rotations and reflections may change the direction of symmetry axes, isometries do not change the angle between directions, so the relations between such directions are preserved.

Following Klein's Erlanger Program, a geometric property of a cell (or a group of cells) is any property which remains invariant under isometric transformations. For a straight edge, the length, for a sphere, the radius, are such properties. Now, for instance, consider two orthogonal planes. The $\pi/2$ angle between the plane normals does not change when we transform both planes by an isometry. However, as we intend to detect such arrangements for all directions from the cells, this approach is not suitable. We would have to generate features for each plane pair, and more general for each pair of cells with suitable directions. Instead, we choose to define direction features (e.g. plane normals) for individual cells. These are no longer invariant under isometries, but they change in the same way than the cell when transformed by an isometry. This means the relations between the features of different cells is preserved, and we can use such features to determine regular arrangements between cells (e.g. symmetrically arranged directions where the angles between the directions are integer multiples of some angle π/n).

Table 2 lists some examples of features we consider. One principle for defining features is to concentrate on basic properties of cells. For instance, for finite cylinders the sum of the radius and the length in the direction of the axis is a length feature, but it is unlikely to have any useful meaning. For similar reasons we do not consider features like the area of a face, which may depend on functional purposes.

Table 3 lists the types of common regularities which we can determine in terms of feature symmetries. The regularities are mainly distinguished by the type of symmetry involved. Note that we also consider special values (e.g. a length exactly equal to an integer) not related directly to symmetries. In the following we first describe exact symmetries in this context, and then consider approximate symmetries.

Cell	Feature	Type
Vertex	Location	Position
Straight Edge	Edge length	Length
Planar Face	Normal	Direction
Spherical Face	Centre	Position
	Radius	Length
Cylindrical Face	Axis direction	Direction
	Axis	Axis
	Radius	Length
Conical Face	Apex	Position
	Axis direction	Direction
	Axis	Axis
	Semi-angle	Angle
Toroidal Face	Centre	Position
	Direction	Direction
	Axis	Axis
	Major radius	Length
	Minor radius	Length
	Sum of radii	Length
	Difference of radii	Length

Table 2: Some Features

Feature Type	Regularity	Symmetries
Direction	Parallel directions	Identity
	Symmetries of directions	Isometries
	Rotational symmetries of directions like in regular prisms and pyramids	Rotations
Axis	Aligned axes	Identity
	Parallel axes arranged equi-spaced along lines and grids	Translations
	Parallel axes arranged symmetrically on cylinders	Rotations
	Axes intersecting in a point	Identity
Position	Equal positions	Identity
	Point set symmetries	Isometries
	Equi-spaced positions arranged on a line or a grid	Translations
	Positions arranged symmetrically on a circle	Translations
	Equal positions when projected on a special line or plane	Identity
Length/ Angle	Equal scalar parameters	Identity
	Special scalar parameter values	(special value)
	Simple integer relations between scalar parameters	(special value)

Table 3: Some Common Geometric Regularities

The simplest regularity type is formed by cells with identical features, e.g. parallel directions. In this case we say the features remain invariant under *identity* of the feature space (the space of all values for a feature type). More generally, a feature set which remains invariant under isometries or a sub-group of isometries in the feature space represents a regularity. Usually it is not the whole set of features of a particular type which remains invariant. We have to find appropriate, maximal subsets. For instance, we may have eight positional features from a model. These may exhibit the symmetry of a cube and we cannot add additional positional features from the model such that the expanded set still has cubic symmetry. So we distinguish between *global* and *partial* symmetries.

Let $f_1, \ldots, f_n$ be a set of n mutually distinct features which remains invariant under a group G of isometries. The isometries are associated with permutations of the features. Each $g \in G$ induces a permutation σ of the labels $1, \ldots, n$: for $g(f_k) = f_l$ we get $\sigma(k) = l$. In order to avoid ambiguities we have to assume that the features are mutually different. Otherwise g induces more than one

permutation. Hence, identity regularities are a special case, and must be detected first. We can then replace identical features by a single feature, which can be used to find non-trivial symmetries as permutations. Note that the permutations preserve the distances between the features.

For instance, for n points $p_1, \ldots, p_n$ arranged in sequence symmetrically around a circle in $\mathbb{E}^2$, the permutation $\sigma : l \mapsto (l + 1) \bmod n$ is induced by a $2\pi/n$ rotation around an appropriate point. By detecting all distance preserving permutations of the points we find these rotations.

There are also feature sets which would be symmetric if certain additional features, which are not present, are added. We call such cases *incomplete*. However, note that any set can be expanded to be invariant under any transformation group, so we require additional conditions. The basic idea is that there is sufficient information in the feature set such that sufficient isometries can be detected.

In particular, repetitions are a suitable basis for incomplete and partial symmetries. We say a symmetry is a repetition if all isometries can be constructed by concatenating a single transformation g, i.e. $G = \{g^r : r \in \mathbb{Z}\}$. For instance, a π/n rotation around a point generates the group of $k\pi/n$ rotations. Given n points $p_1, \ldots, p_n$ arranged in sequence symmetrically on a circle, the regularity is created by $k2\pi/n$, $k \in \mathbb{Z}$ rotations. After removing one of the points, a $2\pi/n$ rotation still maps some of the remaining points onto each other and we still detect some $k2\pi/n$ rotations between the points. The more points we remove the less often we find these rotations.

For beautification, we require a concept of *approximate* geometric regularities. In the approximate case the features are only matched approximately by an isometry. This generates a relation on the feature set which is reflexive and symmetric, but not necessarily transitive. The lack of transitivity generates ambiguous situations where global information is required for algorithms to work correctly. But under certain conditions such a relation can be transitive when restricted to the relevant features, and we can use concepts similar to the exact case for efficient detection algorithms. Essentially we detect tolerance levels at which a transitive matching of the features is possible.

There is a variety of approaches to approximate symmetry. One approach is to compute a measure of asymmetry [31]. Such measures suffer from calibration problems and the precise threshold at which an object is declared to be symmetric is arbitrary, based on supposition and subjective evaluation. Another approach is to find a symmetric object with a small distance (e.g. using the Hausdorff metric) from the object in question. Alt et al. [1] determine a group of transformations such that the images of the set under the transformations remain approximately equal. The time order for this approach is high-polynomial depending on the precise context; from the transformation alone we also do not immediately know which features are matched. Iwanowski [11] detects a symmetric set approximately equal to the original set, which is potentially more useful. However, as a consequence of producing more information, the computation is NP-complete.

In the exact case, symmetries relate to distance preserving permutations which can be detected by checking whether the features match locally, which automatically implies a global match. We define approximate symmetries such that we retain this behaviour for the features in question. We detect approximately distance-preserving permutations at tolerance levels where a local match implies a global match.

Let $R = \{r_1, \ldots, r_n\}$ be a set of n features of the same type. Let $s =_\varepsilon t$ if and only if $|s - t| < \varepsilon$ for $s, t \in \mathbb{R}$ and let $D(R) = \{d(r_l, r_k) : l, k \in L\}$ for $L = \{1, \ldots, n\}$. A permutation σ is an approximate symmetry of R at tolerance level ε, if $=_\varepsilon$ is an equivalence relation on $D(R)$, and $|d(f_l, f_k) - d(f_{\sigma(l)}, f_{\sigma(k)})| < \varepsilon$ for all $l, k \in L$ (see [19]).

Another type of regularity can be detected if we have a structure in the feature space which explicitly selects special values. For lengths we have a special point 0 and from there we can detect special distances from this point as special length values, similarly for angles and the special angle 0. In the approximate case we look for some special values close to the approximate feature value.

3.2 Detecting Approximate Regularities

The first, basic type of regularities we consider are identity regularities like approximately parallel directions. Based on these regularities, we determine regular arrangements either as global symmetries or partial and incomplete symmetries. For partial regularities we have to first determine appropriate subsets and for incomplete regularities we require rules telling which sort of sets we accept. Due to the computational costs of detecting such regularities we only consider repetitions like equi-distant positions on a line. In the following we only give a brief overview of our regularity detection (for details see [7, 15, 16, 17, 19, 20]).

For identity regularities we have to find relations $=_\varepsilon$ on the feature set which are an equivalence. This can be done using a hierarchical clustering algorithm. We have to ensure that each of the clusters represent an equivalence class—at least for a subset of all features. Hence, we call a cluster *consistent* if all the distances between its elements are smaller than a tolerance and the distances to other features is larger than this tolerance.

By seeking consistent clusters at multiple tolerance levels we avoid the requirement of setting a maximum tolerance which is hard to find. Under the condition that all desired regularities are detected, the number of unwanted regularities can in general only be minimised, but not completely avoided, if we use a maximum tolerance. A hierarchical structure of consistent clusters more accurately represents the structures present in the raw model.

For clustering we use Eppstein's closest pair algorithm [6] which requires $O(n^2)$ time. The clusters are formed by starting with each feature in its own cluster. Then clusters are merged in sequence from the smallest distance between two features to the largest one. Depending on whether the clusters fulfil the consistency condition from above or not, they are added as a sub-cluster or merged to form a single cluster. If we keep track of how many distances between the features in a cluster have been considered, we can easily determine if all distances have been considered (consistency condition). From the cluster hierarchy we get different tolerance levels at which the clusters are either locally or globally consistent.

The cluster hierarchy describes identity regularities (depending on the feature type) and it forms the basis for global symmetries and repetitions. We replace the clusters by their centroids at each consistent tolerance level and compute with these new features to detect further regularities.

First consider global symmetries (see [19, 20]). Following our definition of approximate symmetries, we seek approximately distance preserving permutations at consistent tolerance levels. Checking all permutations of the features would be computationally too expensive. But note that an isometry in d-dimensional space is determined by specifying the mapping for only $d + 1$ features (a simplex). Hence, it is sufficient to select $d + 1$ features and try all possible mappings of these features to all available features in turn. One of the $d+1$ features can be the centroid of the feature set which has to be mapped onto itself. The remaining d points are selected in sequence starting with the feature furthest away from the centroid. The third point is the one giving the largest triangle area with the first two, and so on. As the centroid is always mapped onto itself, we only have to check all possible mappings from the d remaining features to all features in the set.

Once we have found a valid mapping for the simplex, mapping it onto another feature simplex within tolerance, we know the isometry within tolerance. Due to the consistent clustering we know that

a match between features within this tolerance is unique. Hence, for our purposes we have sufficient information about the transformation. Thus it is sufficient to compute distance preserving permutations for the $d+1$ features, and then check directly if the remaining features are mapped approximately onto another feature in the set. Given a non-degenerate simplex, the distances between the features in the simplex and another feature f uniquely determine the position of f. Thus, given a distance preserving permutation between two simplices of points in the feature set, we can check whether it induces a distance preserving permutation on all points by matching the distances from the two simplices. The matching has to be unique as we are only checking this for consistent tolerance levels.

By choosing a large simplex as described above, relations between features close together may not be detected immediately. For long thin prisms, the algorithm may not detect the rotational symmetry, but only detect the mirror symmetry between the end-points. In retrospect, as a first analysis this is justified. The rotational symmetry can be detected in a second pass after the object has been expanded orthogonally to its longitudinal axis. In a similar manner an object whose primary structure is planar can be expanded orthogonally to the plane to examine its secondary symmetries.

The time complexity of the symmetry detection method for a single set of n consistent clusters is $O(n^{2.5}\log^4 n)$ in $\mathbb{E}^3$ [19]. The clustering method requires $O(n^2)$ time. Each time the symmetry detection is called the clustering method has made the partition of points coarser. Thus, an immediate limit to the number of calls is n, which can be reached by a collection of points built up one at a time, adding each point a bit further away each time. So the order of global symmetry detection in the worst case is $O(n^{3.5}\log^4 n)$. For practical objects with reasonably limited complexity we actually expect it to take $O(n^2\log^4 n)$.

A similar approach for detecting approximate congruences by matching a simplex from one feature set with a simplex in the other feature set is described in [7].

In order to efficiently detect partial, incomplete repetitions, we first have to determine suitable feature subsets fulfilling some condition. For instance, for rotational symmetries of directions as induced by a prism, we compute an orthogonal vector for each direction pair and then create a consistent cluster hierarchy for such vectors which are approximately parallel. We can then look for the repetitions in each consistent cluster separately. For symmetrically arranged directions, we check the angles between the directions, and determine sets of directions where the angles are approximate integer multiples of some base angle. The base angle itself has to be present as an angle between two directions. However, not all integer multiples of the base angle have to be present for us to be able to detect incomplete repetitions. For a detailed discussion of the methods for different feature types see [17].

The overall regularity detection process starts by extracting all features from the cells of the raw model. Then for each feature type separately, we first create a consistent hierarchical clustering structure. This is used to determine global symmetries and repetitions at different tolerance levels. This gives us geometric regularities at different tolerance levels describing regular arrangements of features. Due to the consistency condition, we only determine regularities for which we have unambiguous evidence in the raw model. The regularities are distinguished by the symmetry type and the features involved. In addition we can determine special values for any scalars involved in these regularities (see [17]). The expected time for regularity detection is roughly $O(n^2)$ for typical reverse engineered objects, where n is the number of faces in the model.

3.3 Selecting Geometric Regularities

Regularity detection determines a large set of approximate regularities present in the raw model at various tolerance levels. Various relations between the cells are considered to find potential regularities, and not all of these are likely to be mutually consistent. As we do not use a strict tolerance limit, we get some regularities which only exist at rather large tolerances. Moreover, the detection methods seek multiple relations between the same features. Hence, the set of detected regularities is likely to contain many inconsistencies and a suitable subset has to be selected. The selected regularities must be mutually consistent and should be likely to represent the original, ideal design intent.

The regularities in terms of symmetries can easily be expressed by sets of geometric constraints. Essentially the constraints relate the features, which in turn describe the geometry of the cells. In order to use only simple constraints, additional auxiliary objects may have to be included. These are only used to express the regularities, e.g. an auxiliary direction for a parallel direction cluster. In this section we assume that we have a solvability test available to determine if a given set of regularities described by constraints is solvable (see Section 3.4).

Using a sequential selection method which tries to add regularities in order of a priority is suitable as a means for improving simple to medium complexity models. We build a constraint system in sequence by adding constraints describing the regularities in order of a priority. For each constraint, we check if adding it preserves the solvability. If not, the regularity it describes is rejected and the corresponding constraints are removed from the constraint system. Instead of a method to determine the overall desirability of a selected set of regularities, we use priorities only to decide which regularity to choose in case of an inconsistency. This way we get a selection of regularities with high priorities by at most considering each regularity once for the solvability test.

Even if the topology has been adjusted earlier, we require a description of the topology in order to enforce the dependencies between the features, and to ensure that a valid model is generated from the regularities. The features not only have to fulfil the regularity constraints, but also have to describe the cells such that the geometric intersections of the cells are consistent with the topology, e.g. a vertex position must lie on all faces the vertex belongs to. Such regularities must always be present in the constraint system and thus they are added first.

The priority of a regularity is computed by taking a weighted average of: a measure of the numerical accuracy to which the regularity's constraints are satisfied in the raw model, a figure of merit expressing the quality or desirability of the regularity depending on specific arrangements and constants involved, and a constant describing a minimum desirability for each regularity type. This average is weighted by a merit function which indicates how common the regularity is (determined by surveying a range of engineering components). All weights and merits have values in the interval $[0, 1]$. For a detailed description of the priority computation, see [14, 18].

While the order of the regularities can be adjusted by varying the constants in the priority function, their large number makes it hard to predict the effect of changes. Currently the priorities used are sufficient to improve the model, but a sophisticated decision process considering more complex relations between regularities and the model, globally, could improve regularity selection with respect to design intent.

3.4 Solvability of Constraint Systems

Besides the priority computation, an efficient constraint system solvability test is crucial for the selection process. Interpreting constraints in a topological context leads to a method similar to the usual degrees-of-freedom analysis [9] where the degrees of freedom become the topological dimensions of the involved spaces. For more details of the solvability test see [14, 18]. While it is desirable to ensure that we have a unique solution or at most a discrete set of solutions, we only check whether at least one solution exists.

In particular we accept cases where there are infinitely many solutions. We can seek a solution for the geometry of the final model close to that of the raw model. The large number of regularities we detect makes under-constrained systems very unlikely.

To verify the solvability of a constraint system, we consecutively add constraints to a constraint graph and check whether the system remains solvable. Our topological interpretation leads directly to an efficient algorithm for representing geometric constraints as directed edges in a constraint graph indicating the dependencies and restrictions created by the constraints. By analysing the dependencies in the constraint graph we quickly determine whether a constraint system expanded by an additional constraint remains solvable in a generic sense.

From a set-theoretical point of view we can say that an unconstrained feature may have any value in the feature space. By adding a constraint we limit the allowed values the involved features can have at the same time, i.e. we select a subset of the direct product of feature spaces. When adding multiple constraints the features have to lie in the intersection of these subsets. For instance, an unconstrained point in $\mathbb{E}^3$ can be at any location in space, i.e. its parameter domain is $\mathbb{R}^3$. A distance constraint between two points v_1, v_2 limits the allowed values the two points can have at the same time. One way of enforcing this is by allowing v_1 to be parameterised by an arbitrary value in $\mathbb{R}^3$ and requiring that v_2 is on a sphere of fixed radius with centre v_1. This means that v_2 can be described by a parameter on the unit sphere $\mathbb{S}^2$ in combination with the position of v_1. Thus, for v_2 we select a (lower-dimensional) sub-manifold of the parameter manifold $\mathbb{R}^3$ which is essentially $\mathbb{S}^2$. Obviously, the roles of v_1 and v_2 can be exchanged. When a 3D point is constrained by two such distance constraints, it has to lie in the intersection of two spheres. In general this intersection may be empty, a point, a circle or a sphere.

We can interpret a distance constraint as a reduction of the parameter space $\mathbb{R}^3$ to $\mathbb{S}^2$ for one of the two points involved. Essentially this reduces the topological dimension of the parameter domain by one. Also when intersecting the different sub-manifolds, in most cases we get a dimension reduction. In the context of distance constraints the most likely case is that the intersection is a circle, i.e. each constraint takes one degree of freedom away and the point has one degree of freedom left. To get the other intersection cases, the distances have to fulfil special conditions. Hence, a circular intersection is the generic case. Without solving any equation systems, we cannot in general determine any of the non-generic cases. We use a similar reasoning for all types of intersections. If a feature with d degrees of freedom has to lie in the intersection of an m- and an n-dimensional sub-manifold, the dimension of the generic intersection is $m + n - d$.

This basic principle can be applied to other geometric constraints as well. We have constraints between the features of a model. Each feature can initially have an arbitrary value in a feature space which is a smooth manifold. The constraints between the features select sub-manifolds of the feature spaces. There is usually more than one way of selecting such sub-manifolds. This makes constraints edges in a hyper-graph between the feature nodes. We can interpret a constraint as a directed hyper-edge depending on how the constraint selects sub-manifolds. We refer to selecting the direction of an edge in a graph as *distributing* the constraint in the graph. Distributing the constraint adds a directed edge to the graph, which represents a particular way of enforcing the restrictions of the constraint on the features. When we choose another option for an already distributed constraint, we call this *redistributing* the constraint, i.e. changing the direction of the edge.

Obviously it is not always possible to distribute a new constraint in a given constraint system. If both points involved in a distance constraint are already $\mathbb{R}^0$, we cannot generically intersect either of them with $\mathbb{S}^2$. However, as for each distance constraint there are

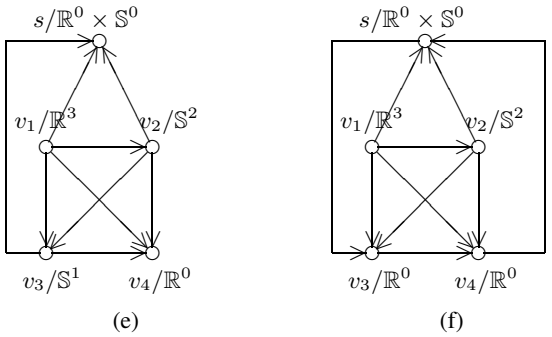


Fig. 8: Constraint Graph of Distances between 4 Points on a Plane

two ways in which it can be added to the constraint graph, it may be possible to choose a different distribution of constraints along some of the edges in the graph such that we can add the constraint. Thus, we have to do a graph search starting at the edge of the new constraint. A breadth-first search backwards along the directed edges is used to find the first edge which can be changed closest to the new edge. We do not have to consider all paths starting with the new constraint, but only find the shortest paths to the edges referring to constraints which can be redistributed. Whenever a constraint can be redistributed, we redistribute the constraints along the whole path to the edge of the original constraint. We then have to repeat the breadth-first search until we can distribute the new constraint, or all redistribution options have been exhausted. Finding a redistribution path is similar to the `distribute` method used in the `dense` algorithm [10].

However, a method to distribute a constraint does not reveal if the resulting constraint system is solvable. The directions of the edges in a constraint graph define the dependencies between the nodes. Given an arbitrary node n in the constraint graph and an edge e directed towards this node, we can follow edges backwards to determine the sub-graph $S(n, e)$ of all nodes on which n depends due to e. To detect $S(n, e)$ we mark n as visited and follow the edge e backwards marking its starting point as visited. From there we continue following all (directed) edges backwards which lead to unvisited nodes; we use a greedy algorithm for this. We stop when no further unvisited nodes can be reached by following the edges backwards. n and all edges between n and the detected sub-graph are added to $S(n, e)$, which we call the *dependency sub-graph* of n due to e. In this sub-graph we change the parameter space of n so that only the edges in $S(n, e)$ are considered. The resulting $S(n, e)$ represents a solvable sub-graph if the sum of the remaining degrees of freedom of the nodes in $S(n, e)$ is at least six, five or three depending on the dimensionality of the object involved. As our constraints only specify *relative* relations between the geometric elements it is not possible to fix the location or the orientation of the resulting object. In $\mathbb{E}^3$ this means that there should be in general six degrees of freedom (three for location and three for orientation) left, e.g. consider three or more points which do not all lie on a line. If we have a lower-dimensional object in 3D (e.g. two points on a line), we only have five degrees of freedom left to fix the position and the orientation of the object. Similarly for a single point we require three degrees of freedom.

When we have successfully distributed a new constraint as a directed edge e in the graph, we must test if the new graph is solvable. If distribution failed, we already know that the constraint system is not solvable. To test for solvability, we only have to consider the changes made during distribution. Let n be the node the constraint has been distributed to, i.e. the directed edge e points towards n. We claim that the graph remains (generically) solvable if the depen-

dency sub-graph $S(n, e)$ is solvable, i.e. it has sufficient degrees of freedom left. Originally the number of degrees of freedom of a node is the dimension of its parameter space. By distributing constraints we select sub-manifolds of the parameter space and intersect those sub-manifolds. This reduces the dimension of the sub-manifold of allowed parameter values for a node. The dimension of this sub-manifold is the amount of degrees of freedom left after constraint distribution. In general in the sub-graph $S(n, e)$ the sum of degrees of freedom has to be at least six (three rotational and three positional degrees of freedom in $\mathbb{E}^3$) in order for it to represent a solvable system. However, note that there are special cases of lower dimensional subsets embedded in $\mathbb{E}^3$ which have to be handled separately, e.g. two points on a line with only five degrees of freedom.

As an example, consider the constraint graph in Fig. 8 linking four points v_l and one plane s. Graph (a) represents six distance constraints between four points, three of which are constrained to lie in a plane. Each of these constraints could be distributed directly by taking one degree of freedom away as indicated by the directed edge. The plane s is described by a distance and a direction, i.e. its domain is $\mathbb{R}^1 \times \mathbb{S}^2$ which has been reduced to $\mathbb{R}^0 \times \mathbb{S}^0$ by the constraints. In graph (b) we distribute a constraint placing v_4 on s as well. We search for a redistribution path starting at the new constraint edge. For s We find three direct redistribution options to v_1, v_2, v_3, and three direct redistribution options to v_1, v_2, v_3 for v_4. Suppose we choose to redistribute the constraint between s and v_3, initially reducing the degrees of freedom of s by one. We redistribute the constraint for this edge, reducing the degrees of freedom of v_3 by one and increasing the degrees of freedom of s by one. Now the original constraint can be distributed directly and we report success with the distribution as shown in graph (b).

Distributing the constraint between v_4 and s creates the graph in Fig. 8(b). We have to check the dependency sub-graph of s over v_4. This graph is identical with the complete graph and has five degrees of freedom. In order for the graph to be solvable it has to have six degrees of freedom, i.e. adding the constraint between s and v_4 has made the system unsolvable. Indeed, the constraints in the graph in Fig. 8(a) are sufficient to determine the four points and the plane up to location and orientation.

4 MODEL REBUILDING

The regularity selection process results in a list of consistent regularities as determined by the solvability test which in general have high priorities. They are represented by constraints which describe the improved model. As the final beautification step we have to rebuild a beautified model from this constraint system and some additional information from the updated model topology. The first step is to find a solution to the constraint system. (Note that the topological solvability test does not actually compute a solution and thus we still have to solve the system).

We do not decompose the constraint system which would allow a symbolic solution. Instead, we simply call a numerical solver to find a solution. Note that for beautification the geometric model may not always be described completely by our geometric constraints (it is only highly likely as we detect many regularities), this making a symbolic approach unsuitable. Furthermore, not all constraint systems can be readily solved symbolically. We use a least-squares optimisation method on an error target function based on a numerically robust implementation of the BFGS quasi-Newton method [26]. We start at the feature values from the raw model, which makes it likely that we find a solution close to it.

From the numerical solution of the constraint system, an improved model is rebuilt using the beautified topological information from the raw model with the feature values obtained from the solution. We create new faces using the solution of the constraint system and re-intersect them to obtain the complete model. The solution of the constraint system gives the vertex positions and the faces. We get sharp edges by using a standard surface-surface intersection algorithm for the adjacent surfaces. If the result is a *closed* curve, the intersection gives the edge immediately. Otherwise we have to limit it with vertices to get the edge. If multiple solutions arise for the intersection, the information in the raw model is used to determine which part of the intersection curve is required, e.g. to distinguish between a convex and concave cylindrical surface. Smooth intersection edges (i.e. ones at which the adjacent faces meet tangentially) have to be handled separately as special cases; this is straightforward as we only consider certain analytic surface types. Note that in general it may also be useful to explicitly compute other certain intersections in order to get analytic representations of the intersections rather than free-form curves.

4.1 Consistency between Topology and Geometry

In our beautification system we make decisions about adjusting the topology before we adjust the geometry of the model. Because during selection, the topology is included by special constraints, the resulting constraint system cannot be inconsistent unless it is necessary to consider non-generic cases. These non-generic cases are unlikely for usual engineering objects. In order to avoid selecting invalid topologies, we can already check the topological constraints during topological beautification. This avoids creating inconsistent topology.

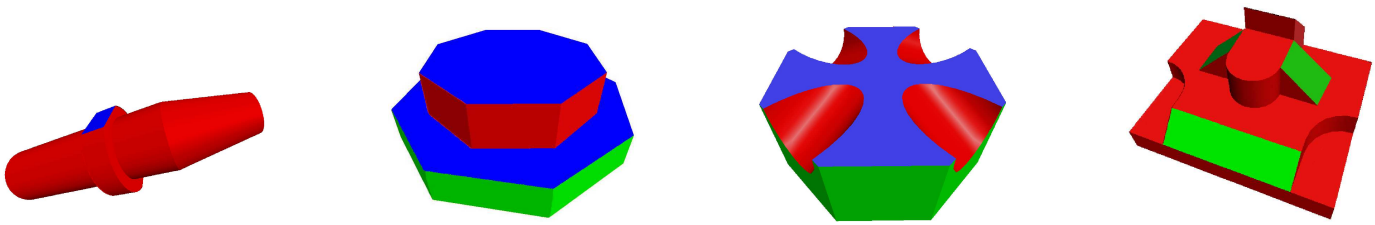
However, note that we do not use general constraints requiring two surfaces to simply intersect without specifying the desired relation (this cannot be done with our interpretation of constraint systems). As we consider a large variety of geometric relations between intersecting faces, the relation between the surfaces is specified by the geometric regularities. Hence, it is unlikely that we have to handle cases where two faces do not intersect. However, should this happen we can always insert another face in a similar way to that used for filling gaps.

The more important problem of splitting topological and geometric beautification is that we do not consider whether the topological and geometric changes are consistent with respect to design intent. Topologically, we may decide to remove a small face, but this small face may be required in order to realise a complicated geometric regularity. If we remove the face first, then we cannot realise this regularity later. Such issues rarely arise in practice, and methods to handle topological and geometric decisions at the same time are left as future work.

5 EXPERIMENTS

The system presented has been tested using a variety of simple to medium complexity reverse engineered models. Our experiments show that it is able to improve raw models with respect to design intent. As the raw models are approximate, there is always some uncertainty about the actual design intent. Depending on the tolerance settings, specific parameter values and minor regularities are not always reconstructed according to the original design. However, major regularities, like global symmetries, major orthogonal systems, etc. representing the global structure of the model, *are* imposed exactly on the improved model. Depending on our assumptions about the raw model, we can *either* accept high quality regularities, *or* regularities with small tolerances. Only a regularity meeting both requirements is likely to be accepted in both cases.

In the following we briefly discuss the results of beautifying the models shown in Fig. 9. Due to the large number of regularities found, we do not present them in detail: only how the major regularities were handled by the system is discussed. For more tests

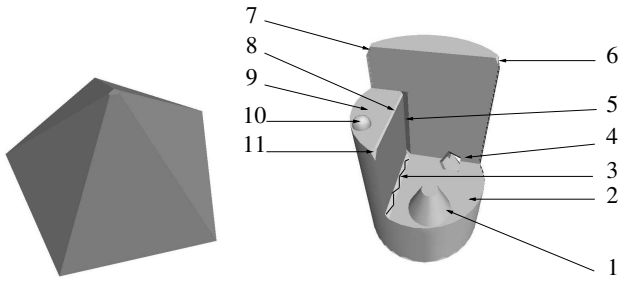


(a) (b) (c) (d)

Fig. 9: Geometric Beautification Examples

Model	(a)	(b)	(c)	(d)	(e)	(f)
Faces	11	19	14	25	6	10
Regularities						
Total	25	329	152	287	55	144
Selected	14	43	33	45	17	23
Sel. Constraints	84	469	334	586	108	153
Time in sec. taken by						
Detecting	0.03	0.32	0.89	0.94	0.09	0.05
Selecting	0.11	9.13	3.75	12.05	1.21	2.28
Solving	28.52	178.37	57.29	263.89	24.37	39.23
Total	28.66	178.82	61.93	276.88	25.67	41.56

Table 4: Experimental Results for Geometric Beautification



(e) (f)

Fig. 10: Topological Beautification Examples

with more detailed results see [8, 14, 15, 18]. Table 4 lists the number of regularities detected, and how many of each were selected for use to improve the model in the geometric beautification phase.

In general, when selecting regularities we have a choice between regularities with small tolerances or regularities which are of high quality. Regularities at small tolerances represent arrangements which are nearly exactly present in the raw model. Regularities of high quality relate to common regularities or very desirable regularities, e.g. a completely symmetric arrangement of faces. Note that some regularities may be present at a small tolerance and be of high quality. The priorities for selection were chosen to slightly prefer high quality to small tolerance regularities. If we have contradictory regularities where some are very accurate and others are of high quality, those of high quality are more likely to be selected even if they are present at a larger tolerance. In general this resulted in the best overall regularity selections. Fine-tuning the values for particular models could of course improve the selection results.

The example models in Fig. 9 all had the correct topology and only geometric beautification was required. The topological beautification phase took under 1 second for all the models and in the following we only describe the issues relating to geometric beautification.

For object (a) the single central axis, several planes with normals parallel to this axis were correctly detected and imposed on the improved model. There are also two parallel planes with normals orthogonal to the central axis (coloured in blue, only one is visible). The orthogonal relation was exactly imposed on the model. However, the angle between the normals of the two planes was set to 3°. The normals of the two planes were combined into a cluster at a high tolerance level. Even if parallel in the ideal model, the angle between the two directions in the raw model was sufficiently far apart (about 3°) such that the two directions were sub-clusters of this cluster. Using our standard priority setting this regularity was not selected, but rather a special value for the angle between the normals was chosen.

Model (b) has two symmetrically arranged, planar direction sets based on the angle $\pi/4$. Together with the orthogonal relation between the blue planes, and the symmetrically arranged red and green planes, these regularities have the highest priority and were imposed exactly on the model. Choosing proper special values for the edge lengths and clustering them in sets of edge lengths was considerably harder. While the equalities were detected in the cluster hierarchy, they were not selected due to the priorities. Even by adjusting the priorities, only the group of short edges could be forced to have the same lengths. The values in the other two groups of lengths were close to each other, but different special values were favoured for the two groups. As we only have an approximate average edge length for each selected cluster of approximately equal edge lengths, it is hard to find the original special value for the length. Only a special value which appears to be desirable with respect to the priorities can be selected (e.g. the closest integer if there is one within acceptable tolerance).

The solvability test correctly determined that only one angle between the groups of red and blue planes can be set. However, for this angle, there was a choice between a special value close to the value in the raw model, and one of high quality. With our method a particular plane pair was chosen to select this angle. If we favoured integer degrees, the closest integer to this angle was chosen. The angle in the original design was 10°. The plane pair chosen had an angle of $45^\circ + 10.8^\circ$ and thus the angle between the two plane sets was chosen to be 11°. Note that to select a special value of 10° between the two plane sets would require the selection of a special value of 55° between the chosen plane pair. In order to prefer this special value, the special value priorities would have to be adjusted to favour multiples of 5 rather than 10 as at present. With a more detailed analysis it might be possible to detect that all the angles between the plane pairs should be considered to find an average value for that single angle, but even in this case we would have to consider the uncertainty in the average value and select a likely special angle value with a tolerance.

In model (c) the normals of the green planes are arranged symmetrically in a plane, and the axes of the red cylinders are arranged symmetrically on a cone. These regularities were well preserved in the raw model (to within about 1°) and are also of high quality, so they were selected. The edge lengths and the angle chosen for the symmetrical cylinder arrangement had the same problems as for the

other models. In addition, in this case we had no regularity specifying a direct relation between the group of cylinders and the planes. Hence, there was a small angle between the cylinder axis directions and the plane normals when projected on the same plane. The edge length regularities and the topological constraints ensured that the lack of a precise relation did not change the topology, i.e. the cylinders could not be rotated in a way that they would intersect with more than one green plane.

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

The models shown in Fig. 10 also exhibit topological defects besides approximate geometric regularities. Object (e) was originally a five sided, regular pyramid, but the sloping faces do not properly intersect at the top. This was detected and corrected by topological beautification in 0.92 seconds, which reduced the 7 faces and 15 edges to 6 faces and 10 edges. Geometric beautification consequently determined the structure of the regular pyramid which was imposed exactly in the improved model.

The raw model (f) contains 21 faces and 40 edges (and is described further in [8]). It contains pinched faces (2 and 9), a chain of small faces (3), an edge gap (4), two face gaps (5 and 7), two small faces (6 and 11) and a sliver face (8). In the model, 1 is a conical boss and 10 is a spherical boss. Note that sliver face 8 is adjacent to multiple face gap 5 and small face 11. The topologically improved model has only 10 faces and 15 edges left. It took 1.59 seconds to improve the model. Geometric beautification detected and correctly imposed the central axis of the object and an orthogonal system of planes in combination with the central axis.

5.1 Discussion

Our methods are able to improve reverse engineered models, but are limited by the ambiguities caused by the fact that we only have approximate models. Major regularities of the model can be handled quite robustly and are usually exactly enforced in the improved model, but minor regularities do not always represent the intended design. In this context we refer to major regularities as regularities which involve a large number of faces of the model and usually relate to a highly symmetric arrangement (with respect to the features). Minor regularities relate to only a few faces in the model and usually have less symmetry.

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

Our system is able to detect approximate regularities for which clear, unambiguous evidence is present in the raw model. It reports the regularities at tolerance levels at which there is no ambiguous interpretation of the data. Most intended regularities are detected in the raw models. As no maximum tolerance value is used, and the tolerance levels for the regularities are detected automatically, the differences in the tolerances of intended regularities can be handled. However, this also results in a larger number of regularities which have to be considered for selection. Trying to devise detection algorithms which only detect intended approximate regularities appears to be considerably harder. While we seek unambiguous evidence in the raw model for the presence of a regularity, we cannot make a decision whether the regularity is intended without having additional information about the model, such as other regularities, consistency with respect to design intent, and solvability of the related constraint system.

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

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

The same parameters for regularity detection and priority computation were used for all the experiments. In all cases, adjusting the parameters to suit a particular model improved the results. While there are no regularity detection parameters which only select desired regularities, it is possible to adjust the parameters individually for each model such that the number of undesired regularities is minimised. As for the priority parameters, the type of selected regularities could be adjusted between values favouring quality and values favouring high accuracy. We may prefer high quality models which exhibit high symmetry (with respect to their features), but which may require a relatively large change to the raw model. Alternatively, we may try to create improved models very close to the raw model, which may be less regular. Furthermore, the priority computation can be adjusted to favour different regularity types. While we tried to keep the number of parameters small, there are still probably too many, and finding optimal values is non-trivial and time consuming. Using the same set of parameters for all experiments, however, did allow us to improve all of the models. This suggests that there is a set of parameter values for a particular

reverse engineering system which can improve the models, even if the results are not optimal. Of course, this presupposes a robust reverse engineering system for creating raw models with consistent tolerance levels.

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

6 CONCLUSION

The aim of beautification is to improve the quality of reverse engineered geometric models so that they exhibit exact intended geometric regularities. We have presented an approach to beautification as a post-processing step which tries to improve a raw model generated by a state-of-the-art reverse engineering system, using only that model as input.

Topological beautification can be done in a straightforward approach in linear time. The new topology is enforced in combination with geometric beautification, which requires roughly quadratic time for detection and selection of geometric regularities. Most of the time for beautification is spent on rebuilding the model. In particular solving the constraint system numerically is computationally expensive.

Our experiments show that our system works reasonably well for simple to medium complexity objects. The topology is adjusted appropriately and major regularities are detected and selected correctly. Minor regularities and relations between individual faces, etc. are less reliably selected. The main reason for this appears to be the fact that we consider the regularities individually: each regularity is assigned a priority independently of the other regularities. Major regularities relating to many cells of the object are clearly likely to be given high priority values and they are quite likely to be inconsistent with other major regularities.

In future work we will address handling more complex models. This will include decomposing the model into suitable sub-parts, expanding the regularity detection to more general regularities and in particular regularities between individual sub-parts. We will also investigate more intelligent selection methods which consider combinations of regularities, and can combine topological and geometric beautification in the context of the decomposed model.

ACKNOWLEDGEMENTS

This project was supported by the UK EPSRC Grant GR/M78267. F. C. Langbein is supported by by the NUF-NAL Grant 00638/G. We wish to thank T. Várady and P. Benkő of the Hungarian Academy of Sciences and CADMUS Consulting and Development Ltd. for providing reverse engineering software and helpful discussions.

REFERENCES

[1] H. Alt, K. Mehlhorn, H. Wagner, E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete and Computational Geometry*, 3:237–256, 1988.

[2] P. Benkő, G. Kós, T. Várady, L. Andor, R. R. Martin. Constrained fitting in reverse engineering. *Computer-Aided Geometric Design*, 19(3):173–205, 2002.

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

[4] G. Butlin, C. Stops. CAD data repair. In: *Proc. 5th Int. Meshing Roundtable, Sandia National Laboratories*, 7–12, 1996.

[5] 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.

[6] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *J. Experimental Algorithmics*, 5(1):1–23, 2000.

[7] C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate congruence detection of model features for reverse engineering. In: M.-S. Kim (ed), *Proc. Int. Conf. Shape Modelling and Applications*, IEEE Computer Society, 69–77, 2003.

[8] C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Local Topological Beautification of Reverse Engineered Models. Submitted, 2003.

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

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

[11] S. Iwanowski. Testing approximate symmetry in the plane is NP-hard. *Theoretical Computer Science*, 80:227–262, 1991.

[12] G. Kós. An algorithm to triangulate surfaces in 3D using unorganised point clouds. *SIAM J. Computing*, Suppl. 14:219–232, 2001.

[13] 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.

[14] F. C. Langbein. Beautification of Reverse Engineered Geometric Models. PhD thesis, Dept. Computer Science, Cardiff University, 2003. `http://www.langbein.org/research/BoRG/beautification.pdf`.

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

[16] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Finding approximate shape regularities in reverse engineered solid models bounded by simple surfaces. In: D. C. Anderson, K. Lee (eds), *Proc. 6th ACM Symp. Solid Modelling and Applications*, ACM, 206–215, 2001.

[17] F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Recognizing geometric patterns for beautification of reconstructed solid models. In: *Proc. Int. Conf. Shape Modelling and Applications*, IEEE Computer Society, 10–19, 2001.

[18] F. C. Langbein, A. D. Marshall, R. R. Martin. Choosing consistent constraints for beautification of reverse engineered geometric models. *Computer-Aided Design*, to appear, 2003.

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

[20] B. I. Mills, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate symmetry detection for reverse engineering. In: D. C. Anderson, K. Lee (eds), *Proc. 6th ACM Symp. Solid Modelling and Applications*, ACM, 241–248, 2001.

[21] A. A. Mezentsev, T. Woehler. Methods and algorithms of automated CAD repair for incremental surface meshing. In: *Proc. 8th International Meshing Roundtable*, 299–309, 1999.

[22] B. I. Mills, F. C. Langbein, A. D. Marshall, R. R. Martin. Estimate of frequencies of geometric regularities for use in reverse engineering of simple mechanical components. Technical Report GVG 2001-1, Geometry and Vision Group, Department of Computer Science, Cardiff University, 2001. `http://ralph.cs.cf.ac.uk/papers/Geometry/survey.pdf`.

[23] J. C. Park, Y. C. Chung. A tolerant approach to reconstruct topology from unorganized trimmed surfaces. *Computer-Aided Design*, 35(9):807–812, 2003.

[24] S. Rusinkiewicz, M. Levoy. Efficient Variants of the ICP Algorithm. In: *Proc. 3rd Int. Conf. 3D Digital Imaging and Modeling*, 145–152, 2001.

[25] N. M. Samuel, A. A. G. Requicha, S. A. Elkind. Methodology and results of an industrial part survey. Technical Report TM–21, College of Engineering & Applied Science, The University of Rochester, 1976.

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

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

[28] T. Várady, R. R. Martin. Reverse Engineering. In: G. Farin, J. Hoschek, M. S. Kim (eds), *Handbook of Computer Aided Geometric Design*, Elsevier, Ch. 26, 2002.

[29] N. Werghi, R. B. Fisher, C. Robertson, A. Ashbrook. Faithful recovering of quadric surfaces from 3D range data by global fitting. *Int. J. Shape Modelling*, 6(1):65–78, 2000.

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

[31] H. Zabrodski, D. Avnir. Measuring symmetry in structural chemistry. *Advances in Molecular Structure Research*, 1:1–31, 1995.