

Constructing Regularity Feature Trees for Solid Models

M. Li, F.C. Langbein, and R.R. Martin

School of Computer Science, Cardiff University, Cardiff, UK
{M.Li, F.C.Langbein, R.R.Martin}@cs.cf.ac.uk

Abstract. Approximate geometric models, e.g. as created by reverse engineering, describe the approximate shape of an object, but do not record the underlying design intent. Automatically inferring geometric aspects of the design intent, represented by feature trees and geometric constraints, enhances the utility of such models for downstream tasks. One approach to design intent detection in such models is to decompose them into *regularity features*. Geometric regularities such as symmetries may then be sought in each regularity feature, and subsequently be combined into a global, consistent description of the model's geometric design intent. This paper describes a systematic approach for finding such regularity features based on recovering broken symmetries in the model. The output is a tree of regularity features for subsequent use in regularity detection and selection. Experimental results are given to demonstrate the operation and efficiency of the algorithm.

1 Introduction

Reverse engineering creates a geometric model from measured 3D data [25]. This model is not necessarily suitable for applications which need to modify or analyse it: it suffers from inaccuracies caused by sensing errors, as well as approximation and numerical errors arising during reconstruction. Such models are approximate in the sense that intended *regularities* like symmetries, congruent sub-parts, aligned cylinder axes, etc. within the model are not exactly represented. Furthermore, even if a regularity *is* preserved to within a sufficiently small tolerance, it can easily be destroyed by later editing operations, if it is not *explicitly* denoted as a property to be preserved. It is thus desirable to explicitly determine the geometric design intent of such models, as embodied by regularities. In this paper we consider the first step of this process, decomposing boundary representation (B-rep) models into *regularity feature trees* (RFTs). Note that such models may have come from reverse engineering, but could also be any other model whose design intent is not explicitly known or has been lost.

Our overall goal is to represent the geometric design intent of a model using a *feature tree* augmented with geometric constraints describing regularities, for processing with a feature-aware constraint solver such as Frontier [23]. Regularities are geometric properties the designer may have desired, e.g., global symmetries of vertices, or congruent sub-parts arranged in a regular manner, or plane normals and other directions forming an orthogonal system. We do not

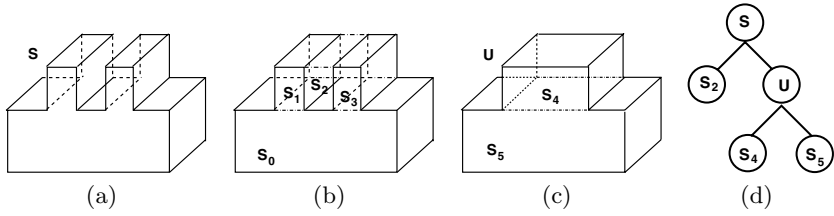


Fig. 1. A simple model and its regularity feature tree

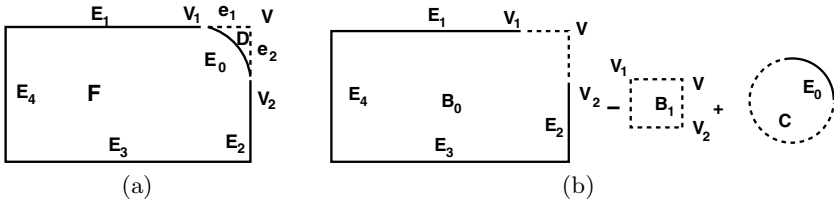


Fig. 2. Expressing a simple face with regularity features and CSG primitives

consider higher-level *functional* or *aesthetic* intent, nor do we look for features specifically useful for purposes such as machining.

Previous work proposed *beautification* of approximate reverse engineered geometric models to improve them with respect to design intent [3,7,8,14]. This approach determines candidate regularities of the *whole model* based on symmetries, and then imposes a consistent subset of these regularities by solving a geometric constraint system. This works well for models with a limited number of ambiguous interpretations in terms of regularities, e.g. models with a major rotational symmetry axis where the rotational symmetry is only broken by a few features. However, complex models often have too many *alternative* plausible approximate regularities for decision methods to be able to determine which regularities represent the original design intent of the whole model. Thus, in this paper we consider the construction of an RFT to simplify the problem by decomposing the whole model into manageable parts, hierarchically. Subsequent work will consider the regularity analysis in each regularity feature *separately*, so these can be combined gradually to form a global description of the intended overall shape. Furthermore, we will analyse the relations *between* regularity features to detect, e.g., congruences and symmetries. As a simple example, consider a rectangular block with many prisms attached to its faces. Analysing the *whole* model without finding the prismatic features creates many candidate plausible angles to enforce between planes in the model. By first identifying the individual prisms as features, we can detect their approximate prismatic symmetries, and separately determine potential regular arrangements of the prisms on the block. Only considering *parts* of the model at any one time will increase the speed of regularity analysis *and* provide more reliable results.

By *regularity features*, we mean simple volumes *derived from the model* which expose hidden approximate regularities in the model. Just like machining features [4],

they are not a class of pre-defined simple geometric primitives, but are determined by the model's geometry. However, in this case, they are constructed to expose intended regularities in the model, rather than describing how to manufacture it. For example, in Fig. 1, the entities of the model S are grouped into volumes S_0 to S_3 , whose union gives an updated model U . U is further grouped into S_4 and S_5 . As explained in detail below this groups S into regularity features S_2 and U , and U further into S_4 , S_5 as shown in Fig. 1(d).

As many regularities can be expressed in terms of symmetries [8], our method for finding regularity features relies on recovering broken symmetries. While this means that some regularity features may become more symmetric, others may just represent the *symmetry break* in the model, i.e. some part which reduces the model's symmetry. To find regularity features, we look for *recoverable edges* and *recoverable faces*. A recoverable face is a newly generated face, with the same underlying geometry as an existing face, but different boundaries, and which, when added to or removed from the existing face, allows us to recover the broken symmetry of the face. We recover the symmetry of a face by modifying its boundary within its underlying geometry.

In order to find recoverable faces, we first detect recoverable edges: newly generated edges based on symmetry expectations derived from faces of the approximate model. The recoverable faces are bounded by a combination of original and recoverable edges. E.g. in Fig. 2(a), using the broken symmetry hints, the two recoverable edges V_1V and V_2V are first constructed. From these we can then determine the recoverable face D to produce the rectangular face B_0 . The asymmetry of F with respect to B_0 is represented by D .

Once detected, the recoverable faces are used together with appropriate original faces of the model to find regularity features as *cells*, closed volumes which do not contain any (new or original) faces in their interior. This leads to *negative* and *positive* regularity features: a regularity feature is negative if the orientation of its faces is inverse to any original faces with the same geometry in the model. E.g. for model S in Fig. 1(a), recoverable edges and faces illustrated by dot-dashed lines in Fig. 1(b) are detected and four regularity features are constructed: blocks S_0 , S_1 , S_2 and S_3 ; S_2 is negative and the others positive.

Using these regularity features, a more symmetric updated model is constructed which exposes partly recovered symmetries of the original model. Volumes corresponding to negative regularity features are added to the original model, while (certain) positive features are cut off. Note that only positive solids *not* adjacent to negative solids can be cut off from S . Other positive regularity features are ignored, because adding the negative solid may change the local connectivity and yield a simpler structure overall. E.g. the negative regularity feature S_2 in Fig. 1(b) is first added to the model S , resulting in an updated model U (Fig. 1(c)). Since all positive regularity features S_0 , S_1 and S_3 have faces in common with S_2 , no further updating is required. The original model is decomposed into the negative regularity feature S_2 and the updated regularity feature U , which become the children of S . In this way, all geometric entities of S are transmitted to its children, which are processed recursively to construct

the tree. Recursion stops when no further regularity features are found. For U this creates two additional regularity features S_4 and S_5 (Fig. 1(c)), while S_2 is not decomposed any further. The corresponding RFT is shown in Fig. 1(d).

For simplicity, throughout this paper, we assume that the approximate input model is a manifold 3D solid represented by a valid, watertight B-rep data structure, and is bounded by planar, spherical, cylindrical, conical and toroidal surfaces, which covers a wide range of mechanical components [15]. The only reason for this restriction is the difficulty of extending the geometry of free-form surfaces. We assume that blends have been identified and suppressed using existing blend-removal methods [21,30]. Finally, we assume that all geometries are represented parametrically and we denote the complete underlying parametric curve of an edge E as \tilde{E} .

The next section discusses how our novel ideas are related to earlier work. In Sect. 3 we describe recoverable edges and faces in detail. We then outline our algorithm in Sect. 4, and give further algorithmic details in Sect. 5. Section 6 presents some experimental results.

2 Related Work

Our proposed algorithm is clearly closely related to previous work on feature recognition, conversion of B-rep models to CSG models, and conversion of wire-frame models to solid models. We discuss relevant results in these areas and compare them to our method.

The current work, and *feature recognition*, both detect local shape information in solid models. However, feature recognition mainly focuses on detecting information needed to manufacture a solid model, such as holes, slots and pockets [4,17,24]. More closely related to our method are feature recognition techniques which try to construct feature volumes for such applications as tool accessibility analysis and process planning [2,6,18,19,20,26,27,28,29].

Convex hull decomposition, also called *Alternating Sum of Volumes (ASV)*, and its variations, produce a hierarchical volumetric representation of solids from boundary information. It then recognises feature volumes in this decomposition. This approach subtracts an object from its convex hull to produce a new object recursively until a convex object is produced [6,16,26]. Such methods are not directly applicable for producing RFTs, for two reasons. Firstly, while such approaches work for polyhedra, they are difficult to generalise to objects with curved surfaces. Secondly, and more importantly, in general the convex hull does not recover broken symmetries such as cut off corners of cubes—this approach does not explicitly consider regularities.

In our method of finding regularity features, new recoverable edges and faces are generated to find feature volumes. Similar ideas have been applied to create feature volumes from *feature face-sets* [2,19,20,29]. The latter have to be detected or are sometimes assumed to be provided in advance, reducing the complexity of the problem. Our approach is applied directly to a solid model: all the needed geometric information is obtained from the model.

The idea of face intersections has also been applied in *cell-based* methods: the model is decomposed into a set of minimal cells by intersecting all faces of the model, having extended them as necessary using their underlying geometry. These cells are then merged into subsets to form feature volumes [18,27]. However, whereas features are *local* parts of a model, this approach makes *global* use of local geometry, leading to a combinatorial explosion in the number of cells generated and merged; nevertheless, recent work has limited this problem by using localised face extensions, and cell collection using seed cells [28]. In contrast, our algorithm detects regularity features as minimal volumes, defined by recoverable faces generated via *local* extensions of existing geometry. In combination with recursive processing of regularity features, this greatly reduces the number of volumes to be considered.

In feature recognition, it is important to carefully choose the newly generated geometric entities used to form features. Regularity features are supposed to reveal symmetries of the model, an issue not been addressed in previous work. Our approach systematically constructs regularity features using carefully chosen newly generated entities based on the idea of recovering broken symmetries. To obtain all the possible regularity features and avoid the need for heuristics, all possible local entities are constructed, and selection amongst them occurs naturally during the entity construction process. Specifically, *only* those recoverable edges, faces or regularity features making a contribution towards generating a recoverable face, regularity feature or updated model respectively, are kept.

Our approach is also closely related to methods for converting B-rep models to CSG models [22], which try to determine how to construct a model from primitives using Boolean operations. Our method shifts the emphasis from the construction process to finding suitable primitives, the regularity features. In CSG models, the primitives are simple regular solids, whereas in our method, the regularity features are solids which, when added to or removed from other solids, produce a more regular solid. For example, the 2D face F in Fig. 2(a) is not difficult to express using Boolean operations between simple CSG primitives, e.g. rectangle B_0 minus rectangle B_1 plus circle C as shown in Fig. 2(b); original geometric entities in F are drawn with solid lines and added geometric entities are drawn with dashed lines. Such expressions are not directly required for regularity detection, and can involve extra unnecessary geometric primitives, such as the rectangle B_1 completely composed of dashed lines. Building up such expressions consequently can require additional unnecessary computation. In our method, we simply express F as regularity features B_0 , the intended model symmetry, and D , the symmetry break. As a further difference, note that D could in principle intersect with another part of the model due to some unwanted *global*, rather than *local*, interaction. This must be taken into consideration when constructing a CSG model, but for regularity processing can simply be ignored.

Construction of regularity features from recoverable edges requires similar steps to *converting wire-frame models* into solid models [1,5,11,13]. Given a wire-frame model linking vertices and edges, these methods produce a corresponding volumetric description by deciding which edge loops are covered by faces. The

main issues here are algorithmic efficiency, and how to enumerate all multiple interpretations—or to choose the most appropriate one from amongst them. A general approach for polyhedra is based on ordering edges around each vertex and faces around each edge to create all possible solutions [13]. More general work to reconstruct curved solids from engineering drawings uses a maximal turning angle method to detect all possible faces in a wire-frame with straight and conic edges [11]. Graph theory has been applied to resolve the conversion problem for 0-, 2- and 3-connected wire-frames [1,5]. In our method, unlike in the wire-frame case, the face normals of the model are available, as well as those of any generated recoverable faces (determined uniquely by the original edge orientations), greatly simplifying the construction of regularity features.

In summary, like previous work on feature recognition, B-rep to CSG conversion, ASV and related approaches, our method hierarchically decomposes a model into parts. These approaches are closely related and even sometimes produce similar results. However, our approach has the goal of recovering symmetries in the model, while previous work is based on other geometric aspects such as the convex hull, specific primitives, or machining features. For our application—detecting geometric design intent—symmetry is the important geometric aspect.

3 Recoverable Edges and Faces

Our algorithm is based on recovering symmetries that were broken during construction of the (ideal, rather than approximate) original model. Constructing geometric models by using modelling operations on geometric primitives usually destroys regularity in structures which were originally simple. One can understand this process as a sequence of symmetry breaking operations—see e.g. [10]. These operations often leave hints in the model from which they may be recovered. We recover broken symmetries by first analysing symmetry breaks in faces, from which symmetry breaks of the model are constructed. Specifically, the symmetries of a primitive face might be broken in its interior, across one or more edges, or surrounding one or more vertices—or some combination of these. By adding entities we may recover the original symmetries.

To simplify the processing required, we reduce complex cases of broken symmetry to combinations of elementary cases which we call *Missing Face Segment (MFS)*, *Missing Edge Segment (MES)*, and *Missing Vertex Segment (MVS)*, examples of which are shown in Fig. 3. For MFS cases, to regain the symmetry we must generate a new face bounded by an inner edge loop of a face of the input model. MES cases require a new edge, and MVS cases require a new vertex; other associated geometry is also required in these cases.

The new geometry required to recover the symmetry is composed of recoverable vertices, recoverable edges and recoverable faces. A *recoverable vertex* is a newly generated intersection vertex between certain edges, e.g. vertex V in Fig. 3(c). A *recoverable edge* is a newly generated edge that is not already represented in the model and connects two vertices (original or newly generated) that lie in the underlying geometry \tilde{E} of some edge E of the model, e.g. edge

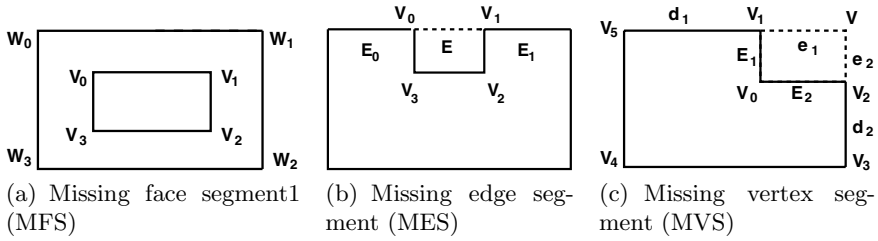


Fig. 3. Three elementary cases of breaking face symmetries

E in Fig. 3(b) or e_1 and e_2 in 3(c). A *recoverable face* is a newly generated face with an underlying surface derived from a face in the model, bounded by recoverable or original edges, e.g. the face bounded by the loop V_0, V_1, V_2, V_3 or W_0, W_1, W_2, W_3 in Fig. 3(a), the face bounded by the loop V_0, V_1, V_2, V_3 in Fig. 3(b) or the face bounded by the loop V_1, V, V_2, V_0 in Fig. 3(c). We distinguish between two recoverable edge types depending on whether they relate to the MES or MVS case as explained next.

3.1 MES Recoverable Edges

MES recoverable edges are straight-forward to define. Let V, V' be two vertices of the input model M such that

- (1) these vertices lie on the underlying geometry of some edge E of the model M , i.e. $V, V' \in \tilde{E}$;
- (2) the segment S of the underlying curve \tilde{E} bounded by V, V' contains no other vertex of M in its interior;
- (3) there is no edge E' of M with underlying geometry \tilde{E} bounded by V, V' .

Then the edge E^* with underlying geometry \tilde{E} , bounded by V and V' , is called an *MES recoverable edge*. E.g., in Fig. 3(b), the newly generated edge E bounded by V_0 and V_1 is an MES recoverable edge with underlying geometry \tilde{E}_0 .

3.2 MVS Recoverable Edges

MVS recoverable edges are more complex than MES ones, as we have to decide which two vertices, original or newly generated, to use to construct the edges needed when recovering the face’s symmetry. We first show how this can be done for a vertex on a planar face at which straight edges meet, and then how to extend the idea to more general cases.

We first introduce some terminology. A vertex on a planar face bounded by two straight edges at which the interior angle is greater than π is called *concave*, e.g. V_0 in Fig. 3(c). We call the two incident edges at a concave vertex V_0 on a planar face F *boundary edges*, e.g. E_1, E_2 in Fig. 3(c). The end-points of boundary edges, other than V_0 , are called *boundary vertices*, e.g. V_1, V_2 . An edge of F that is not a boundary edge but contains a boundary vertex is called an *external edge*, e.g. d_1, d_2 . A line normal to an external edge at the corresponding boundary vertex is called an *associated external line*, e.g. n_1, n_2 in Fig. 4 at the concave vertex V_0 .

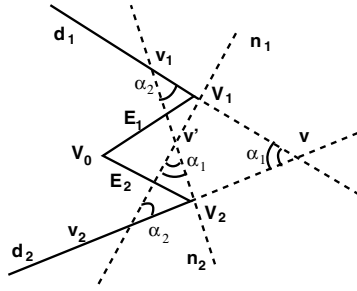


Fig. 4. MVS recoverable vertex and edge generation

A face of the model sharing the external edge with F is called an *external face*, e.g. F_1, F_2 in Fig. 6(a). As the input model M is a manifold solid, the number of boundary vertices, boundary edges, external edges, associated external lines and external faces is always two. Any edge containing a boundary vertex but not lying on F is called an *associated edge*, e.g. E_3 and E_4 in Fig. 6(a). An *associated external plane* is a new plane containing the associated external line and an associated edge originating from a boundary vertex. More than three edges can originate from one vertex, e.g. at the apex of a pyramid, so there may be more than one associated edge or associated external plane originating from a boundary vertex.

The external edges and faces usually give some indication of the local symmetry at a concave vertex, and are helpful in recovering the broken symmetry of the face. Furthermore, in engineering, rectangles play a particularly important role and are often present. In the following definitions of MVS recoverable vertex and edge, we seek the ‘most plausible’ way of completing a rectangle using the external edges and associated external lines. Two principles are used. Firstly, the two generated recoverable edges should be those that form the closest possible angle to a right angle. This avoids having to use a tolerance value to determine whether or not two lines are orthogonal—choosing such a tolerance is non-trivial given an approximate input model. Secondly, the constructed recoverable vertex should lie on the opposite side of the line V_1V_2 determined by the two boundary vertices V_1, V_2 from the concave vertex V_0 . This ensures that the constructed recoverable edges do not intersect the two boundary edges, which must lie on the same side of line V_1V_2 as V_0 due to the concavity at V_0 . See also Fig. 4.

With these considerations, a unique line pair (l_1, l_2) is selected from amongst the external edges and associated external lines around one concave vertex; their intersection point is the MVS recoverable vertex. For any two lines l_1, l_2 , we denote their intersection vertex as $V_I(l_1, l_2)$ and their intersection angle ($\leq \pi/2$) as $A_I(l_1, l_2)$. Suppose V_0 is a concave vertex with boundary vertices V_1, V_2 , external edges d_1, d_2 and associated external lines n_1, n_2 (see Fig. 4). Since n_i is orthogonal to $d_i, i = 1, 2$, it can be seen that $A_I(d_1, \tilde{d}_2) = A_I(n_1, n_2), A_I(\tilde{d}_1, n_2) = A_I(n_1, \tilde{d}_2)$, and $A_I(\tilde{d}_1, \tilde{d}_2) + A_I(\tilde{d}_1, n_2) = \pi/2$. Taking the four line pairs, we select (l_1, l_2) using the following method:

1. If $A_I(\tilde{d}_1, \tilde{d}_2) < \pi/4$:
 - If $V_I(\tilde{d}_1, n_2)$ lies on the opposite side of line V_1V_2 from V_0 and $V_I(n_1, \tilde{d}_2)$ does not, set $l_1 = \tilde{d}_1$ and $l_2 = n_2$.
 - If $V_I(n_1, \tilde{d}_2)$ lies on the opposite side of line V_1V_2 from V_0 and $V_I(\tilde{d}_1, n_2)$ does not, set $l_1 = n_1$ and $l_2 = \tilde{d}_2$.
2. If l_1 and l_2 do not yet have values:
 - If $V_I(\tilde{d}_1, \tilde{d}_2)$ lies on the opposite of line V_1V_2 from V_0 , set $l_i = \tilde{d}_i, i = 1, 2$;
 - otherwise set $l_i = n_i, i = 1, 2$.

A unique pair (l_1, l_2) is always obtained using the above definition: the second step always provides l_1 and l_2 whenever \tilde{d}_1 is not parallel to \tilde{d}_2 , because $V_I(\tilde{d}_1, \tilde{d}_2)$ and $V_I(n_1, n_2)$ must lie on different sides of line V_1V_2 . Whenever \tilde{d}_1 is parallel to \tilde{d}_2 , there always exists a unique point $V_I(\tilde{d}_1, n_2)$ or $V_I(n_1, \tilde{d}_2)$ lying on the opposite of line V_1V_2 from V_0 , and hence the first rule applies in this case.

The vertex $V_I(l_1, l_2)$ is the desired *MVS recoverable vertex*, e.g. vertex V in Fig. 3(c). Edges bounded by the recoverable vertex and the boundary vertices, with underlying curves l_1 or l_2 , are called the *MVS recoverable edges*, e.g. edges e_1, e_2 in Fig. 3(c). Fig. 5 shows some examples of recoverable vertices V with recoverable edges e_1, e_2 obtained by this selection method.

More generally, for a face with at least two straight edges, we try to convert the involved curved edges into straight edges and then analyse them using the above approach, following these principles:

1. If one or more connected curved edges lie between two straight edges on the face, we treat them in the same way as we earlier treated a concave vertex lying between two straight edges; the two straight edges are considered to be the corresponding external edges. This is reasonable as the straight edges in the face may be hints for a broken symmetry in the input model. E.g. in Fig. 2(a) the curved edge E_0 is adjacent to two straight edges E_1 and E_2 . Thus, E_1, E_2 are treated as the corresponding external edges, resulting in recoverable vertex V and recoverable edges e_1, e_2 , which ultimately give a rectangle. In Fig. 6(b), the external edges for the curved edge C_0 are E_1 and E_2 , resulting in a recoverable vertex V and recoverable edges e_1 and e_2 .
2. If a concave vertex has straight boundary edges but a curved external edge, the external edge is replaced by its tangent line at the boundary vertex during generation of MVS recoverable edges. Generally this is a more plausible

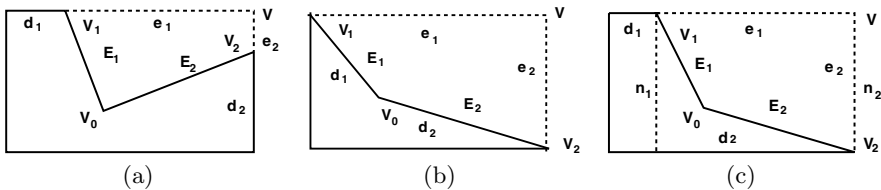


Fig. 5. Different intersection cases of external edges on a face and corresponding MVS recoverable vertex V and edges e_1 and e_2

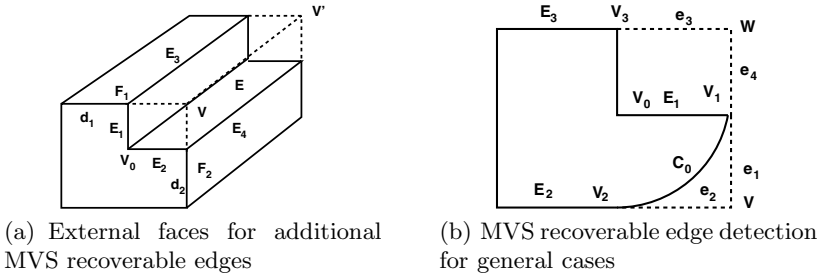


Fig. 6. More general MVS cases

way of recovering model symmetry—leading to a rectangle—than using the curved edge directly. E.g. for the concave vertex V_0 in Fig. 6(b), the tangent line of C_0 at V_1 is used instead of C_0 in the generation of MVS recoverable edges, resulting in a recoverable vertex W and recoverable edges e_3 and e_4 .

The model building operations used to construct the original model may have removed certain edges completely, e.g. edge E bounded by vertices V and V' in Fig. 6(a). External faces or associated external planes help to find the solution in such cases. The vertices V, V' which bound an MES recoverable edge may be MVS recoverable. If the MVS recoverable vertices are constructed from an external edge, the underlying curve of the MES recoverable edge is the intersection of the external faces. If the MVS recoverable vertices are constructed from an external line (instead of an external edge), the curve underlying the MES edge is the intersection of the associated external planes.

In the case of faces having at most one straight edge, it is difficult to design a universally useful method for finding MVS recoverable edges. We do not process them in our current approach, and leave a more sophisticated approach for future work. Our current implementation detects all MVS recoverable edges on planar faces. However, our concept of MVS recoverable edges is not limited to such cases in principle. For curved surfaces, we could consider the geometry of the boundary loop inside the underlying surface, and follow geodesics instead of straight lines.

4 Algorithm Overview

We now give an overview of our algorithm for constructing an RFT from an input manifold B-rep model M . Further details are presented in the next section. The RFT for M is built up by first constructing regularity features for M and then processing the resulting features recursively. In the following we refer to the solid currently being processed in this recursion as S .

We decompose S , if possible, into *edge-connected* solids, each of which is determined by a *separate component* of the edge graph of S . Each edge loop representing a boundary of an edge-connected solid is closed with a face as appropriate. If S consists of several edge-connected solids, they become children of S in the RFT, and each is processed by the following steps in turn as the current solid.

Algorithm. RFTCONSTRUCTION (M)*Input:* M —a manifold B-rep solid (with approximate geometry)*Output:* T —a Regularity Feature Tree for M

1. $T \leftarrow$ tree with root M to store regularity feature tree
2. $Q \leftarrow$ FIFO queue of solids, initially containing M
3. $C \leftarrow$ GROUPFACES (M)
4. **while** $Q \neq$ empty **do**
5. $S \leftarrow$ **pop** (Q)
6. $R \leftarrow$ EDGECONNECTEDSOLIDS (S)
7. **if** $|R| = 1$ **then**
8. $\mathcal{E}_E \leftarrow$ MESRECOVERABLEEDGES (S, C)
9. $\mathcal{E}_V \leftarrow$ MVSRECOVERABLEEDGES (S, \mathcal{E}_E, C)
10. $\mathcal{F}_R \leftarrow$ RECOVERABLEFACES ($S, \mathcal{E}_E \cup \mathcal{E}_V$)
11. **if** $\mathcal{F}_R =$ empty **then break** (continue at line 4)
12. $D \leftarrow$ REGULARITYFEATURES (S, \mathcal{F}_R)
13. **if** $D =$ empty **then break** (continue at line 4)
14. $R \leftarrow (U, D_0, \dots, D_d) \leftarrow$ UPDATE SOLID (S, D)
15. **for** $N \in R$ **do**
16. Add N as child of S to T
17. **push** (Q, N)
18. **return** (T)

Fig. 7. RFT construction algorithm

Next, MES and MVS recoverable edges of S are detected. Existing edges, together with newly generated recoverable edges, create new edge loops from which subsequent recoverable faces are constructed. The orientations of the faces constructed using these loops are determined from orientations of the edges. After extending the solid S using these newly constructed recoverable faces, the regularity features are identified as closed cells, which are used to recover broken symmetries of S and form the basis of the construction of regularity features of S . The negativity or positivity of each regularity feature comes naturally from the orientations of the faces bounding them.

Once the regularity features of S have been identified, we update S to create an updated model U by first adding all negative regularity features to S and then removing all positive regularity features from S except for those adjacent to negative regularity features (see Sect. 1). The added negative and removed positive regularity features (*not all* positive ones), together with U , become the children of S in the RFT, and are recursively processed. Recursion stops when all solids in the RFT have been processed.

Pseudo-code for the algorithm is given in Fig. 7. The tree T contains the RFT as it is built up; its root is set to M . Recursion is implemented via a queue Q storing solids still to be processed; it initially contains M (Lines 1–2). In order to determine recoverable edges lying in the same surface geometry, we first group faces of M sharing the same underlying surface to within tolerance (Line 3). For this we cluster the faces according to a similarity measure based on the symmetric Hausdorff distance. Using the face clusters we can also determine whether other geometric entities in the model are the same. E.g. we check whether two edges

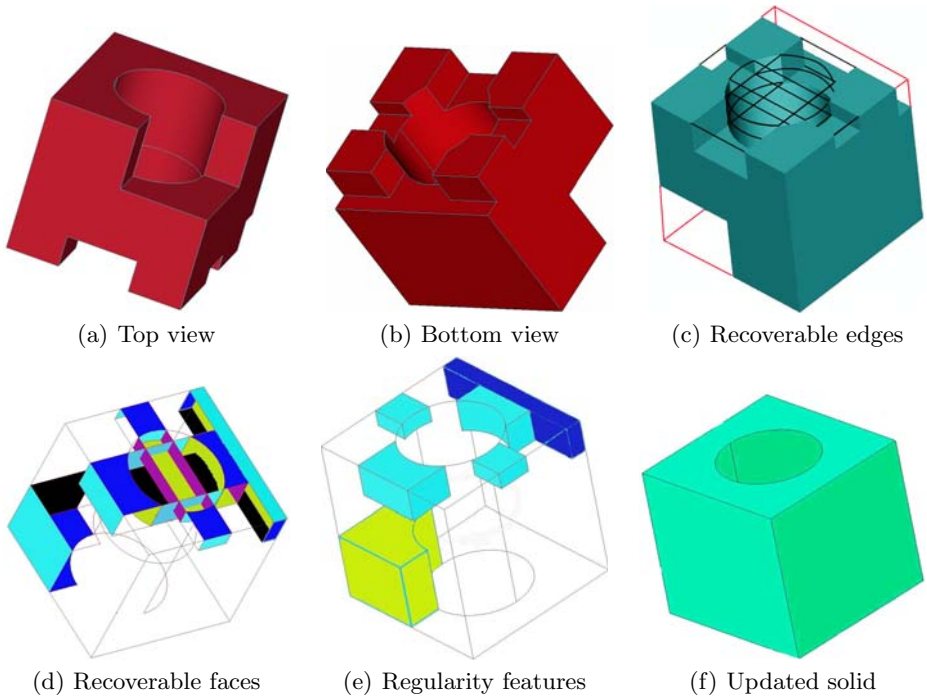


Fig. 8. Example model

share the same underlying curve by checking whether they are the intersection of two face pairs which come from the same two underlying surfaces.

Repeatedly, until the queue is empty, the first solid is removed from the queue and its regularity features are constructed (Lines 4–17). First, the solid S is analysed to determine whether its edge graph consists of a single connected component (Line 6). If not, it is decomposed into edge-connected solids, which are added as children of S to the tree and queued for further processing (Lines 15–17). For efficiency, we tag edge-connected solids detected to avoid re-checking them when they are processed recursively.

If S consists only of one edge-connected solid, we determine its regularity features (Lines 7–14). First all the MES and MVS recoverable edges in S are constructed using the ideas in Sect. 3 (Lines 8–9). From these we then detect the recoverable faces as described in Sect. 5.2 (Line 10). If no recoverable faces are detected, we stop processing S and proceed to the next element of the queue (Line 11). Otherwise regularity features are found by detecting cells bounded by recoverable faces and the original faces of S (Line 12). If no regularity features are detected, we stop processing S and proceed to the next element of the queue (Line 13). Otherwise, an updated solid U for S is computed based on the detected regularity features, together with the added negative and removed positive regularity features (Line 14). They are added as children to S and scheduled for further processing (Line 15–17).

At each step we simplify the solid being processed by filling in or cutting off material as determined by the recoverable faces. Each new solid is simpler, having less recoverable edges. As the overall number of recoverable edges in a model is finite, recursion eventually stops.

As an example, consider the model M shown from above and below in Figs. 8(a) and (b). It becomes the root of the output RFT T . M is edge-connected, so we construct its MES (red) and MVS (black) recoverable edges as shown in Fig. 8(c). The recoverable faces and the regularity features are then generated in turn as shown in Figs. 8(d) and (e). As all the generated regularity features are negative, they are added to M , producing the updated model shown in Fig. 8(f). Both the constructed regularity features and the updated model become the children of M in T . Except for the updated solid in Fig. 8(f), no further regularity features are obtained when the initial regularity features are reconsidered. The updated model is further processed to give two edge-connected solids: a cube and a cylinder, which become the children of the updated model. No further regularity features are found on considering them.

5 Algorithm Details

This section discusses further important details of our algorithm. As Lines 8–14 only process edge-connected solids, we assume in the following that we have an edge-connected solid S with edge set \mathcal{E} and face set \mathcal{F} .

5.1 Recoverable Edge Construction

The recoverable edge set $\mathcal{E}_R = \mathcal{E}_E \cup \mathcal{E}_V$ consisting of MES and MVS recoverable edges is constructed in sequence using the ideas in Sect. 3 (Lines 8–9). This sequence is required to resolve conflicts between MES and MVS cases (see below). For constructing MVS recoverable edges, two issues must be considered carefully.

Firstly, if several concave vertices are consecutively linked to each other by edges in \mathcal{E} , we have to decide which of these vertices to process. We choose the shortest edge having one or two concave vertices as end-points. If this edge has two concave vertices as end-points, we choose the vertex for which the adjacent edge is shorter. Otherwise, we simply choose the concave vertex. This way we expect to find the ‘smallest’ broken symmetry first. We process this vertex as a standard MVS vertex and leave the other vertices for later processing in the resulting regularity features. For instance, vertices V_1 , V_2 and V_3 in Fig. 9(a) are all concave vertices. Only V_3 is selected for construction of MVS recoverable edges.

Secondly, if a concave vertex or the boundary vertex of an MVS case also serves as an end-point of an MES recoverable edge, we have to decide which to select such that similar sub-parts create similar decomposition results. This is particularly important for models where all geometric relations between model entities are only approximately satisfied. In Fig. 9(b), for example, if we construct edge V_7V_8 as an MES recoverable edge for the right-hand feature, while edges

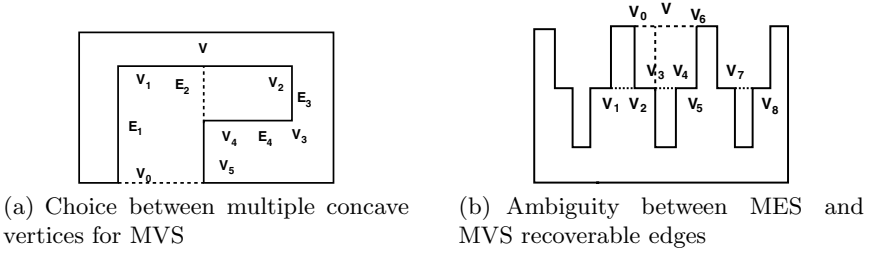


Fig. 9. Ambiguous recoverable edge cases

V_0V , V_3V are found as MVS recoverable edges for the central feature, the bottom of the right slot will be filled in, while for the central feature V , V_0 , V_2 , V_3 is filled. Such inconsistency in the construction will hamper application of our ideas to design intent detection. To avoid this problem, we always give preference to MES recoverable edges, since they are constructed from clear hints for broken symmetries, and usually produce fewer regularity features. Thus, if a concave vertex or the boundary vertices of an MVS case serve as an end-point of an MES recoverable edge, we do not construct MVS recoverable vertices and edges. For the model in Fig. 9(b), recoverable edges V_3V_4 and V_7V_8 are constructed as MES recoverable edges while VV_0 and VV_3 are not taken into consideration.

Finally, note that an MVS recoverable vertex might be the same as another MVS recoverable vertex, or an original vertex, within tolerance. We combine such vertices using the face grouping information gathered in Line 3 of our algorithm.

5.2 Recoverable Face Construction

After the recoverable edge set \mathcal{E}_R of the solid S has been obtained as above, the recoverable face set \mathcal{F}_R is constructed from the union of the edges in \mathcal{E} and \mathcal{E}_R based on taking minimal turning angles between these edges (Line 10).

Using an edge ordering algorithm, for each recoverable edge E in \mathcal{E}_R , we find all the minimal edge loops lying on the same surface and containing E in the edge graph given by the union of \mathcal{E} and \mathcal{E}_R . Each such non-self-intersecting loop bounds a new recoverable face. Such loops must be detected on *both* surfaces on which E lies, proceeding in *both* clockwise and counter-clockwise directions. If a loop does not contain any edge in \mathcal{E} , it makes no contribution to recovering symmetries of existing faces and hence is ignored.

5.3 Regularity Feature Construction

Having obtained the recoverable face set \mathcal{F}_R , we generate all possible regularity features of the model S (Line 12). These are the cells determined by original faces \mathcal{F} together with the recoverable faces \mathcal{F}_R . As the input solid S already has proper loops and face normals, generating these cells is much simpler than the general problem of converting wire-frame models to solid models [1,5,11,13].

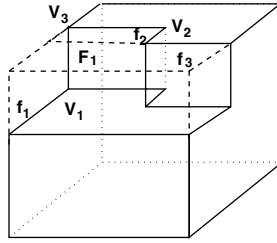


Fig. 10. Two recoverable faces to be considered in sequence

To ensure that the resulting solids are manifold and closed, the Möbius rule is applied: each edge belongs to *two* faces and the orientation of the edge is *opposite* on each face [9]. The negativity or positivity of each resulting regularity feature is determined from the face orientations.

Note that each original edge can only lie on the intersection of two faces as S is manifold. Moreover, our algorithm guarantees that at most two recoverable faces are generated around a given edge on one surface. Hence, the number of faces around one edge, both in \mathcal{F}_R and \mathcal{F} , is at most six, two of which are original faces while others are recoverable faces. All faces having the same underlying surface also have the same or opposite normals. If the normals of two faces point in the same direction, we cannot uniquely order them around their common edge. To resolve this issue, we note that: (i) if the two faces are lying on different sides of an edge on the surface, the unique one contributing to the Möbius rule is considered; (ii) if the two faces lie on the same side of an edge on the surface, each of them is selected in sequence to generate all possible volumes. E.g. in Fig 10 we start with recoverable face f_1 at the left which is expanded to F_1 via edge V_1V_3 . If we expand F_1 via V_2V_3 , both recoverable faces f_2 and f_3 give the same turning angle. Considering f_2 first, no volume is detected. Then considering f_3 , a volume cut off from the cube is produced.

5.4 Model Update

Having found all the regularity features, negative and positive, we compute an updated, more symmetric model U incorporating geometric information from the original solid S and the regularity features (Line 14). Firstly, if the intersection of the interior of two regularity features is not empty, the one with smaller volume is kept while the other one is discarded. Secondly, volumes determined by negative regularity features are added to S , while volumes corresponding to the positive solids are cut from S , except for those adjacent to negative features. Both rules are aimed at constructing simple updated models and avoid the inclusion of unnecessary regularity features in the RFT.

We create the updated model using *face combination operations* (rather than potentially non-robust Boolean operations) by analysing the geometric and topological relations between the regularity features and their parent. This is feasible as the faces of a regularity feature come from its parent's faces or underlying surfaces. This approach fills in a set of negative regularity features or splits off

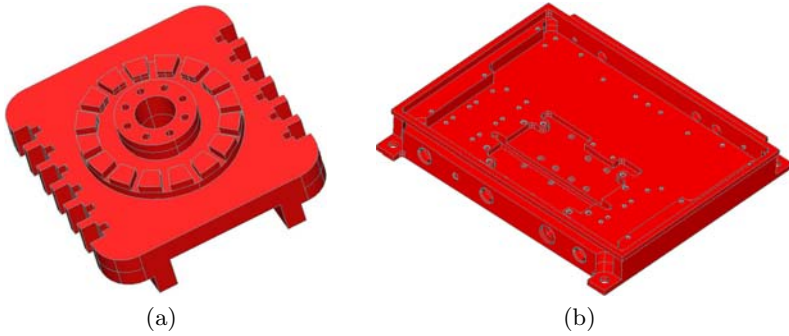


Fig. 11. Two example models

a set of regularity features by changing the topology of the model; both can be done in the same way. We simply combine the regularity features and the parent solid along their shared faces; similarly, adjacent faces on the same underlying surface are further combined along their common edges. Finally, all connected edges lying on the same curve are combined into a single edge.

6 Examples

Our algorithm has been implemented under Linux using OpenCASCADE and experiments were run on a 3.4GHz Pentium 4E with 1GB RAM. This section shows the RFTs constructed for two complicated example models shown in Fig. 11, and considers the algorithm’s performance.

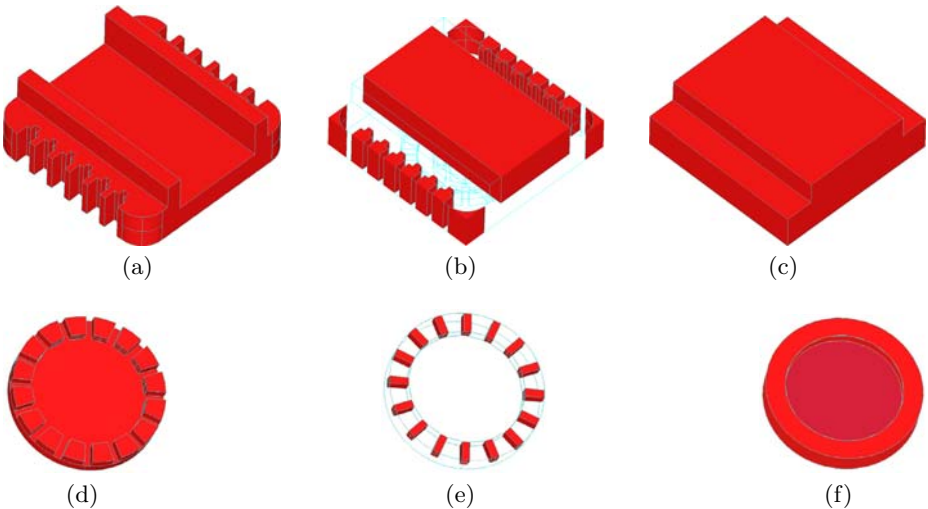


Fig. 12. Decomposition of the model in Fig. 11(a)

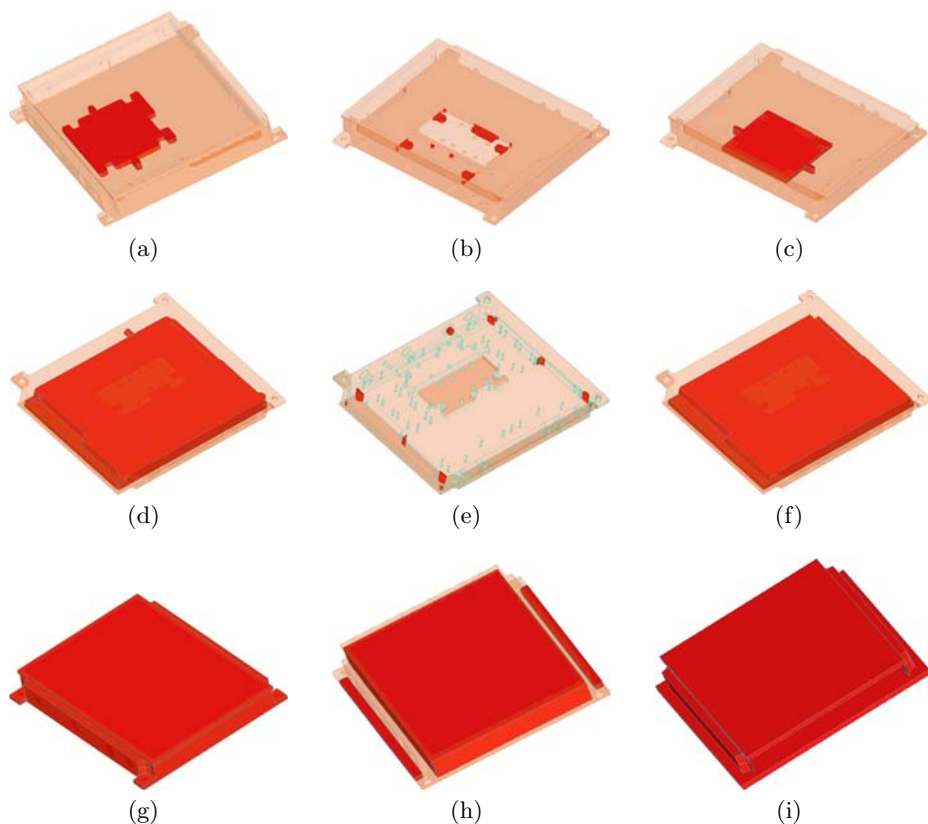


Fig. 13. Decomposition of the model in Fig. 11(b)

The decomposition for the model in Fig. 11(a) with 281 faces is shown in Fig. 12. On the first decomposition level 10 edge-connected solids are produced, two of which are (a) and (d). The other edge-connected solids are simple cylinders which do not decompose further. Sub-part (a) leads to 29 negative regularity features at the next level in the tree as shown in (b), and results in the updated model (c), which is further decomposed into two rectangular blocks. Note that (b) properly identifies the congruent negative regularity features in a regular translational arrangement. Similarly, (d) leads to 16 negative solids as shown in (e). The resulting updated model is shown in (f), which is further decomposed into a positive and a negative cylinder. In (e) the symmetric arrangement of the negative regularity features has been extracted explicitly. It takes 27.50 seconds to produce the final RFT of depth 4 consisting of 59 regularity features. Overall the decomposition reveals the rotational symmetries of the main cylinder as well as the regular translational arrangement of congruent regularity features at the sides. The regularity features at the leaves of the tree are simple and regular, which would allow easy subsequent regularity analysis.

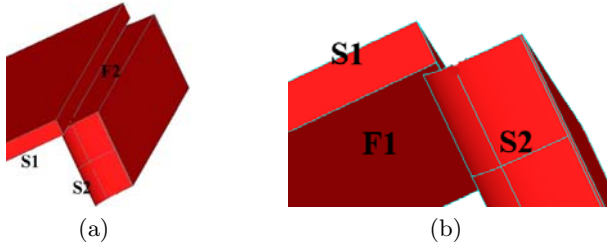


Fig. 14. Close-up of cause for incomplete decomposition of Fig. 13(i)

Fig. 11(b) shows another complicated example with 355 faces; its decomposition is shown in Fig. 13. On the first decomposition level 93 edge-connected solids are produced, three of which are (a), (d) and (g). The other edge-connected solids are simple, non-decomposable, cylinders. Sub-part (a) is further decomposed into 11 regularity features at the next level in the tree as shown in (b), resulting in the updated model (c). Sub-part (d) leads to 11 regularity features at the next level, as shown in (e); the updated model is shown in (f). The resulting regularity features are further decomposed but are straight-forward and not shown due to space limitations. It takes 18.96 seconds to produce the final RFT of depth 5 consisting of 155 regularity features. However, the updated model in Fig. 13(i) was only decomposed into a bottom block and a second part. We expected the latter to be decomposed further into three blocks. Fig. 14 shows a closeup of the regularity feature revealing the cause of the incomplete decomposition: blocks S_1 , S_2 touch each other at faces F_1 , F_2 , but their interiors do not intersect. Both faces would have to be extended to recover the original blocks. However, in such contact cases, our current algorithm cannot construct recoverable faces to separate the two blocks due to the lack of suitable recoverable edges. In order to address this problem, original edges which form loops around ‘holes’ would have to be determined efficiently so they can be filled in by the geometry of neighbouring faces. Several further examples of realistic industrial parts are provided at <http://www.langbein.org/research/DID/rftexamples>. In each case the computation took less than 30 seconds for models with a maximum of 355 faces and 164 regularity features, demonstrating the algorithm’s practical utility. In these tests, any rotationally symmetric arrangements and regular translational arrangements are clearly exposed by the decomposition; often the regularity features are congruent or similar to each other. Furthermore, the regularity features at deep levels in the tree show a high level of regularity as they are often the basic rectangular blocks, cylinders, etc. present in the model.

7 Conclusions

We have presented an algorithm for constructing *regularity feature trees* for B-rep solid models, for detecting geometric design intent. Our experiments indicate that it produces suitable RFTs for detecting regularities of a model’s sub-parts and intended geometric relations between the sub-parts. The algorithm takes a

relatively short time, suitable for practical applications. We note that the ideas presented are applicable to more general curved models, although the decomposition method may need further refinement and generalisation. In future work we intend to detect potential geometric regularities more efficiently using the RFT and ultimately select a suitable subset of these regularities to describe the geometric design intent of approximate models.

Acknowledgements

This project was supported by EPSRC grant GR/S69085/01. The example models in Sect. 6 were obtained from the National Design Repository at Drexel University, <http://www.designrepository.com/>.

References

1. S. Bagali, J. Waggenspack. A shortest path approach to wireframe to solid model conversion. In: *Proc. 3rd ACM Symp. Solid Modeling and Appl.*, pp. 339–349, 1995.
2. X. Dong, M. Wozny. A method for generating volumetric features from surface features. In: *Proc. 1st ACM Symp. Solid Modeling and Appl.*, pp. 185–194, 1991.
3. C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Local topological beautification of reverse engineered models. *Computer-Aided Design*, 36(13):1337–1355, 2004.
4. J. Han, M. Pratt, W. Regli. Manufacturing feature recognition from solid models: a status report. *IEEE Trans. Robotics and Automation*, 6(6):782–796, 2000.
5. K. Inoue, K. Shimada, K. Chilaka. Solid model reconstruction of wireframe CAD models based on topological embeddings of planar graphs. *J. Mechanical Design*, 125(3):434–442, 2003.
6. Y. Kim. *Convex decomposition and solid geometric modeling*. PhD thesis, Stanford University, USA, 1990.
7. F. C. Langbein, B. I. Mills, A. D. Marshall, R. R. Martin. Approximate geometric regularities. *Int. J. Shape Modeling*, 7(2):129–162, 2001.
8. F. C. Langbein. *Beautification of reverse engineered geometric models*. PhD thesis, Cardiff University, UK, 2003.
9. R. Lequette. Automatic construction of curvilinear solids from wireframe views. *Computer-Aided Design*, 20(4):171–179, 1988.
10. M. Leyton. *A generative theory of shape*. Lecture Notes in Computer Science 2145, Springer, Berlin, 2001.
11. S. Liu, S. Hu, Y. Chen, J. Sun. Reconstruction of curved solids from engineering drawings. *Computer-Aided Design*, 33(14):1059–1072, 2001.
12. E. Lockwood, R. Macmillan. Geometric symmetry. *Mathematical Intelligence*, 6(3):63–67, 1984.
13. G. Markowsky, M. Wesley. Fleshing out wire frames. *IBM J. Research and Development*, 24(5):582–597, 1980.
14. B. Mills, F. Langbein, A. Marshall, R. Martin. Approximate symmetry detection for reverse engineering. In: *Proc. 6th ACM Symp. Solid Modeling and Appl.*, pp. 241–248, 2001.

15. B. Mills, F. Langbein, A. Marshall, R. Martin. Estimate of frequencies of geometric regularities for use in reverse engineering of simple mechanical components. Tech. report GVG 2001-1, Dept. Computer Science, Cardiff University, 2001.
16. A. Rappoport. The extended convex differences tree (ECDT) representation for n-dimensional polyhedra. *Intl. J. Comp. Geometry and Appl.*, 1(3):227-241, 1991.
17. W. Regli. *Geometric algorithms for recognition of features from solid models*. PhD thesis, University of Maryland, USA, 1995.
18. H. Sakurai, P. Dave. Volume decomposition and feature recognition, Part II: curved objects. *Computer-Aided Design*, 28(6-7):519-537, 1996.
19. D. Sandiford, S. Hinduja. Construction of feature volumes using intersection of adjacent surfaces. *Computer-Aided Design*, 33(6):455-473, 2001.
20. V. Sashikumar, S. Milind. Reconstruction of feature volumes and feature suppression. In: *Proc. 7th ACM Symp. Solid Modeling and Appl.*, pp. 60-71, 2002.
21. V. Sashikumar, S. Milind, R. Rahul. Removal of blends from boundary representation models. In: *Proc. 7th ACM Symp. Solid Modeling and Appl.*, pp. 83-94, 2002.
22. V. Shapiro, D. Vossler. Separation for boundary to CSG conversion. *ACM Trans. Graphics*, 12(1):35-55, 1993.
23. M. Sitharam, J.-J. Oung, Y. Zhou, A. Arbree. Geometric constraints within feature hierarchies. *Computer-Aided Design*, 38(1):22-38, 2006.
24. J. Vandenbrande. *Automatic recognition of Machinable Features in Solid Models*. PhD thesis, University of Rochester, USA, 1990.
25. T. Varady, R. Martin, J. Cox. Reverse engineering of geometric models - an introduction. *Computer-Aided Design*, 29(4):255-268, 1997.
26. D. Waco, Y. Kim. Geometric reasoning for machining features using convex decomposition. *Computer-Aided Design*, 26(6), 477-489, 1994.
27. Y. Woo, H. Sakurai. Recognition of maximal features by volume decomposition. *Computer-Aided Design*, 34(3):195-207, 2002.
28. Y. Woo. Fast cell-based decomposition and applications to solid modeling. *Computer-Aided Design*, 35(11):969-977, 2003.
29. X. Xu, S. Hinduja. Recognition of rough machining features in $2\frac{1}{2}$ components. *Computer-Aided Design*, 30(7):503-516, 1998.
30. H. Zhu, C. Menq. B-rep model simplification by automatic fillet/round suppressing for efficient automatic feature recognition. *Computer-Aided Design*, 34(2):109-123, 2002.