Detecting Design Intent in Approximate CAD Models Using Symmetry

Ming Li[a,b], Frank C. Langbein[a], Ralph R. Martin[a]

[a] *School of Computer Science, Cardiff University, Cardiff, UK*
[b] *State Key Lab of CAD & CG, Zhejiang University, Hangzhou, P. R. China*

Abstract

Finding design intent embodied as high-level geometric relations between a CAD model's sub-parts facilitates various tasks such as model editing and analysis. This is especially important for boundary-representation models arising from, e.g., reverse engineering or CAD data transfer. These lack explicit information about design intent, and often the intended geometric relations are only approximately present. The novel solution to this problem presented is based on detecting *approximate local incomplete symmetries*, in a hierarchical decomposition of the model into simpler, more symmetric sub-parts. Design intent is detected as congruencies, symmetries and symmetric arrangements of the leaf-parts in this decomposition. All elementary 3D symmetry types and common symmetric arrangements are considered. They may be present only *locally* in subsets of the leaf-parts, and may also be *incomplete*, i.e. not all elements required for a symmetry need be present. Adaptive tolerance intervals are detected automatically for matching inter-point distances, enabling efficient, robust and consistent detection of approximate symmetries. Doing so avoids finding many spurious relations, reliably resolves ambiguities between relations, and reduces inconsistencies. Experiments show that detected relations reveal significant design intent.

Key words: Design intent, approximate symmetry, feature recognition, beautification, reverse engineering, CAD data transfer.

Email addresses: eMing.Li@gmail.com (Ming Li),
F.C.Langbein@cs.cardiff.ac.uk (Frank C. Langbein),
Ralph.Martin@cs.cardiff.ac.uk (Ralph R. Martin)

1. Introduction

Design intent concerning the *shape* of a CAD model can be expressed via geometric properties of, and relations between, its vertices, edges, faces and sub-parts. As shape is often essential to function, such relations must be enforced on the model to fulfil its purpose. Many intentional geometric relations form *geometric regularities*. However, information about a model's design intent is not always explicitly available. E.g., reverse engineering [42] captures the shape of a model but does not explicitly detect intended regularities. Such models are approximate due to measurement errors, and approximation and numerical errors occurring during reconstruction. Similarly, models constructed from inexact user input, e.g. sketches [26, 43], are also approximate, and lack explicit design intent. Exchanging models between different CAD systems [32] may break intended, exact regularities due to incompatible tolerance systems and representations; design intent is often not explicitly transferred. Detecting design intent in such approximate models can reveal high-level information that is necessary for the model's function or purpose. Such information may be used to constrain and guide editing operations. It may also allow us to improve an approximate model by enforcing intended regularities. It may enable faster analysis and more compact representation, if the model has symmetric sub-parts. It may also allow models to be more meaningfully indexed for shape search, etc. Thus, this paper considers algorithmic detection of geometric design intent in approximate boundary-representation (B-rep) models of engineering objects, such as the one in Fig. 1.

Symmetry is a key concept in design. Engineering objects often exhibit symmetries for functional, aesthetic, and manufacturing reasons [2, 40]. Many regularities can be represented via symmetries [19]. A symmetry is an isometry that maps a set exactly onto itself. However, symmetry may be present *approximately*—the set is almost invariant under an isometry, *locally*—only part of the set is invariant, *incompletely*—not all elements building a symmetry are present, and *compatibly*—multiple subsets share the same symmetry. We thus later define a precise concept of *approximate incomplete symmetry* which includes exact and global symmetries as special cases, generalising the ideas in [29]. For brevity, henceforth, we refer to *approximate* symmetry or congruency as symmetry or congruency, unless stated otherwise. An alternative approach [21] considers *asymmetries* in a model to describe design intent as a sequence of symmetry breaking operations.

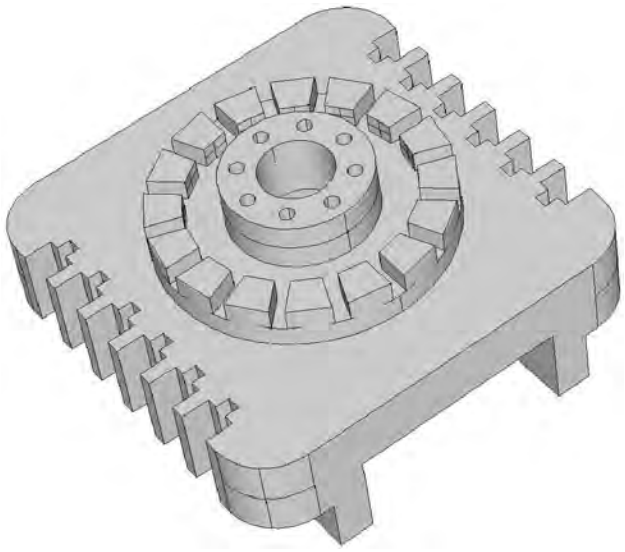


Figure 1: An example of an approximate CAD model: Monster

Complex models often exhibit far too many alternative plausible approximate regularities for *exhaustive* methods to be able to determine which regularities represent the original design intent of the whole model [20]. As a simple example, consider a rectangular block with many prisms attached to its faces. Analysing the *whole* model without finding the prisms creates many candidate angles and distances forming plausible regularities between the model's planes. By first identifying the individual prisms as sub-parts, we can detect their approximate prismatic symmetries, and separately determine symmetric arrangements of the prisms on the block. Analysing *sub-parts* of the model separately increases the speed of regularity detection *and* provides more reliable results. Hence, our design intent detection algorithm performs model decomposition before detecting regularities in the resulting sub-parts.

The decomposition phase builds a *regularity feature tree* (RFT) forming a hierarchy of *regularity features*: simple, closed volumes which in combination describe the original shape. The regularity features at the leaves of the RFT describe the complete shape of the object; the tree indicates how to build the complete model from the leaf-parts. Unlike a CSG tree, the RFT does not contain standard primitives, nor does it give a Boolean decomposition [22]. Instead, the emphasis is on the fact that the leaf-parts are simpler and more symmetric than other parts in the tree, and not on

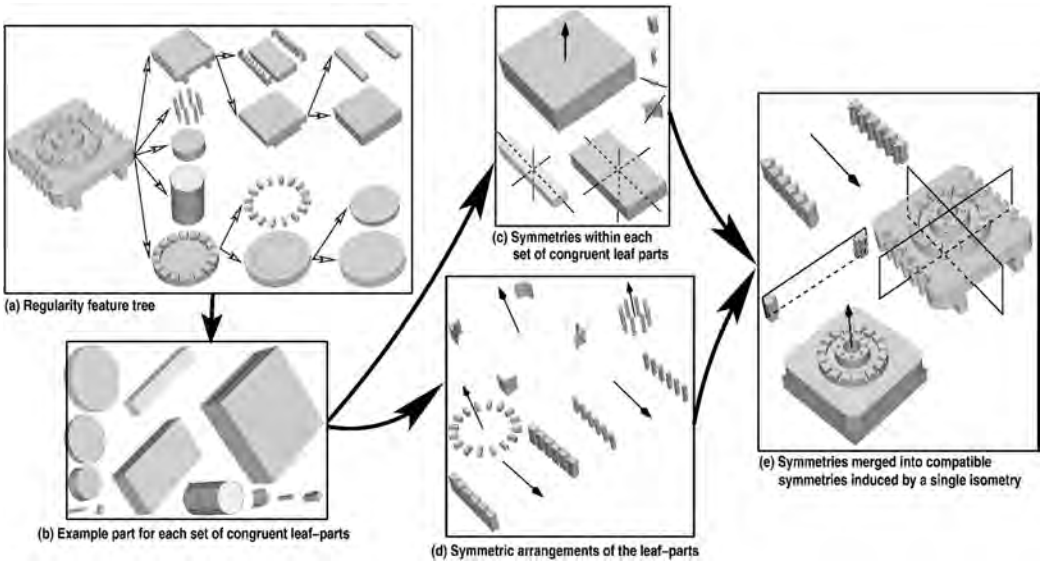


Figure 2: Overview of algorithmic steps for detecting design intent of the Monster model in Fig. 1

how the object was or might have been constructed. The second phase of the algorithm seeks regularities within the model in terms of congruencies, incomplete symmetries and symmetric arrangements of these leaf-parts. It first detects congruencies to partition the leaf-parts into *congruence sets*, each containing one or more congruent leaf-parts. Next, for each congruence set, it seeks subsets forming incomplete symmetries and incomplete symmetric arrangements. Compatible symmetries shared by leaf-parts, and symmetric arrangements, are further combined before we output all detected regularities as transformations matching sub-parts of the model. The process is illustrated in Fig. 2 for the model in Fig. 1. Fig. 2(a) shows the computed RFT, Fig. 2(b) shows the congruent leaf-parts found, and the detected symmetries are given in Figs. 2(c)–(e). The output may, e.g., be used to describe a model by geometric constraints [36], or be processed by regularity selection techniques [20, 45].

As the models are approximate, the method has to consider tolerances carefully. We compute suitable (tolerance) *validity intervals* directly from distances present in the model to ensure that model entities match unambiguously (i.e. in a one-to-one manner) at any tolerance in the interval. During decomposition, each different validity interval yields a different, well-

defined RFT. We let the user select a suitable RFT, which is often straightforward as appropriate tolerances are often known. Regularity detection is then restricted to that particular validity interval. For a particular decomposition, regularities may also exist at different tolerance levels. To avoid missing any important regularities, we seek all of these. We ensure that the regularities are unambiguously present to avoid inconsistencies between regularities and to reduce the number of spurious regularities found. Regularities are detected in a certain sequence for efficiency, and to ensure that relations between regularities are preserved (e.g. congruent sub-parts must have the same symmetries). Tolerance information is used to ensure that these inter-regularity relations are preserved at the tolerance intervals at which the regularities are present.

Throughout this paper, we assume that the 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 [30]. The only reason for this restriction is the difficulty of extending the geometry of free-form surfaces involved in the RFT construction; see Section 6. We assume that blends have been identified and suppressed using existing blend-removal methods [35, 46] (or have not been added during a reverse engineering process).

This paper uses our previous results on constructing RFTs [22] and detecting incomplete symmetries of discrete point sets [24, 23]. Here we combine and extend these results to efficiently and robustly detect design intent in B-rep models for a wide range of symmetry-based regularities. We extend our earlier work to include *all* elementary symmetry types in 3D (mirror, inversion, translation, rotation, rotation-mirror, glide, screw), not just rotations and rotation-mirrors, with a single, consistent algorithm. We also give further previously unpublished details of our incomplete symmetry detection algorithm, and explain how to adapt it to detecting symmetries of B-rep models (not just point sets).

Detecting symmetries in a B-rep model is achieved by using *characteristic points*. They are the model's vertices and some other special points which characterise curved edges and faces (e.g. a circular arc is uniquely determined by its two end points and the arc's mid-point). These, together with topological and face type information, uniquely characterise the model [10, 29]. Hence, although a wireframe model is insufficient to define a volumetric object uniquely, only regularities of the solid model are detected via *consistent* mappings between characteristic point sets; see Section 3.2.

In summary, this paper presents a novel solution to efficiently detect geometric design intent as geometric regularities of and between model sub-parts using symmetry. It can handle approximate models robustly, yielding well-defined, unambiguous regularities and providing a higher-level description for downstream processing. Detecting local symmetries using the RFT, instead of working directly on the model as a whole, avoids considering many spurious, almost certainly unintentional geometric relations. It greatly reduces the computation time required and leads to a better understanding of regularities of the model. Moreover, we consider all possible elementary symmetry types with a single method based on inter-point distances. The results do *not* depend on arbitrary user-chosen tolerances, but most tolerances are inferred automatically from the model. This improves robustness and reliability, as well as efficiency. As some approximate models have multiple consistent interpretations in terms of regularities at different tolerance ranges, the user must make a simple choice of a suitable tolerance interval which gives the desired interpretation. Our experiments show that we can *efficiently* detect regularities which describe *significant design intent*.

The rest of the paper is organised as follows. Section 2 discusses related previous work and Section 3 gives our definition of incomplete symmetry for discrete point sets and B-rep models. Section 4 outlines our design intent detection algorithm. Details follow in Section 5 on a clustering algorithm to handle tolerance ranges, Section 6 outlines the RFT construction, and Sections 7 to 11 discuss in detail the stages of our regularity detection approach. Section 12 presents experimental results, and Section 13 concludes the paper.

2. Related Work

We now overview relevant work on representing and detecting geometric design intent, and on symmetry detection.

2.1. Design Intent

Previous consideration of design intent has been based on geometric constraints, features and construction history. Efforts have been made to use these (including symmetries) to extend the STEP ISO standard for design intent description [32, 18]. Such approaches mostly focus on *exact* geometric relations and representation, while our approach is aimed at approximate models and design intent detection.

Geometric constraints annotate relations between geometric entities [14, 12]. However, they usually describe design intent at a low level, prescribing many simple relations and parameters, instead of higher-level relations such as a symmetric arrangement of identical complex sub-parts. *Solving* constraint systems provided by a user is the main topic considered, rather than *detecting* constraints. Our algorithm finds high-level relations using symmetries, from which low-level constraints can be derived if required.

Feature recognition is often aimed at determining machining operations needed to manufacture a model [11, 13, 37]. Recording a complete *model construction history*, e.g. [7], views design as akin to writing a program to construct the model. However, such histories contain artificial construction steps as well as the intended regularities. Furthermore, histories are not unique—many can produce the same final object. Bidarra [4] has, however, proposed a semantic feature modelling method in which such design recording is independent of the model design history. Sitharam et al [36] present a geometric constraint solver which can handle features. In our view theirs is the most promising approach for representing design intent; their system may well be suited for downstream processing of the output of our algorithm. Leyton [21] views design as a sequence of symmetry breaking operations yielding a *generative history*. As with construction histories, however, the generative history of most complex models is ambiguous. In contrast, we are only concerned with detecting regularities and not inferring a history. He utilises symmetry and symmetry breaks, but for the *representation* of design intent, not its *detection*.

Reverse engineering solid models from measured data, using prescribed features or geometric constraints, has also been considered [3, 27, 41], but few authors discuss how to extract these from point data [17]. However, detecting *candidate* geometric constraints *after* reverse engineering models has been previously examined [10, 19, 20, 29]. This approach can handle fairly simple approximate models, but suffers from ambiguities and inconsistencies between regularities in more complex models. Our approach resolves this issue for complex models by first decomposing them; it also detects more general incomplete symmetries and symmetric arrangements of sub-parts. Beautification of 3D polyhedral models reconstructed from 2D sketches was also considered recently [45].

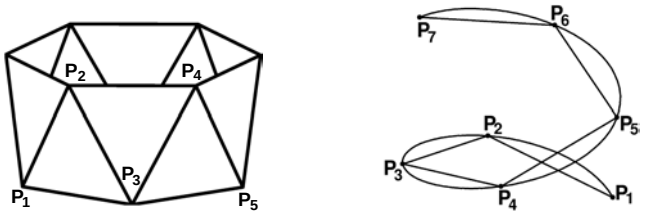
2.2. Symmetry Detection

Previous work on symmetry detection has mainly considered rotation and mirror symmetries, with different approaches used for different symmetry types. Here, we consider all seven elementary 3D symmetry types using a uniform approach.

Exact symmetries can be found in $O(n \log n)$ time: see [24] and the references therein. However, differing definitions of (global) *approximate symmetry* cause its detection to be NP-hard [15] or to take $O(n^8)$ time [1]. Our approach extends the global approximate symmetry detection algorithm in [29], which takes only $O(n^{3.5} \log^4 n)$ time.

Little work exists on detecting approximately symmetric *subsets* of data. Robins [33] considers subsets of points regularly arranged along a line. While for *global* approximate symmetries a *single* tolerance applies, for *subset* symmetries, a tolerance only applies to a particular subset: different subsets may have *different* tolerances. This complicates the automatic detection of tolerances. Another approach to finding subset symmetries in solid models is presented by Tate and Jared [39]. It is based on matching every pair of edge loops and finding the isometries that relate them. The isometries are then grouped according to similarity. Their implementation only finds axes of rotation and mirror planes. The approach may be used to detect approximate symmetries by 'relaxing' matching tolerance conditions, but this means the resulting notion of approximate symmetry is ambiguous and does not easily enable automated tolerance detection. We aim to detect what is present unambiguously in the data without predetermined tolerances. Limiting the search to edge loops, mainly due to efficiency reasons, is also too restrictive for finding all symmetries. Instead we employ a volumetric decomposition of the model for efficiency reasons and to detect predominantly intended symmetries. [24] describes an approach for detecting point subsets which form approximate, *complete* rotation or rotation-mirror symmetries. We extend this work here to *all incomplete* symmetries of *sub-parts*.

We seek discrete *point* symmetries to find symmetries of B-rep models as explained later, but we use quite different point sets compared to symmetry approaches used in image processing, e.g. [38], and mesh processing, e.g. [31]. Generally, the aim there is to detect *one* or a few *dominant* approximate symmetries by partial matching of images or meshes described by *dense* point data, under user-selected tolerances. In contrast, we wish to generate *all* unambiguous subset symmetries in a carefully chosen *characteristic* point set derived from the B-rep model, sufficient to uniquely characterise its geometry



(a) Rotation-mirror $\mathbf{S_n}$ (b) Screw $\mathbf{W}$

Figure 3: Some elementary symmetries in 3D

given its face types and topology. Our algorithm thus processes far fewer points than a mesh symmetry algorithm, but the position and existence of each and every point is significant.

Our method carries out hierarchical model decomposition before applying symmetry detection. A similar idea is found in [34], which uses a symmetry based decomposition of 2D B-rep models to obtain an axial shape description. However, only three symmetry types are considered there: skew symmetries, parallel symmetries and smooth local symmetries. Moreover, only one dominant symmetry is detected for each decomposed part, and symmetry relations between parts are not addressed. Detecting global rotational symmetries of 3D objects using invariant feature indexing has also been considered [8].

3. Definition of Approximate Symmetry

We first introduce approximate incomplete subset symmetries for point sets in Section 3.1, and use them to define such symmetries for B-rep models in Section 3.2. This definition covers all elementary isometries in 3D [28]: mirror $\mathbf{M}$, inversion $\mathbf{I}$, translation $\mathbf{T}$, n-fold rotation $\mathbf{C_n}$, n-fold rotation-mirror $\mathbf{S_n}$ (mirror followed by n-fold rotation about an axis orthogonal to the mirror plane, Fig 3(a)), glide $\mathbf{Z}$ (mirror in a line followed by translation parallel to the line), and screw $\mathbf{W}$ (rotation about an axis followed by translation along the axis, Fig. 3(b)). Throughout the paper we denote the set of all distances $\{\|P - Q\| : P, Q \in \mathcal{P}\}$ between members of a point set $\mathcal{P}$ as $\mathcal{D}(\mathcal{P})$. We also denote approximate equality of real numbers a, b within tolerance ϵ, where $|a - b| \leq \epsilon$, by $a =_\epsilon b$.

3.1. Approximate Symmetries of Point Sets

We recap global approximate symmetries as defined in [29], and then give an extended definition of (incomplete) symmetry cycles based on our previous

work [24]. Merging such cycles leads to incomplete symmetries [23]. These definitions cover symmetries in a wide mathematical sense: approximate, subset, incomplete, compatible symmetries, and include all the seven types of elementary symmetries. Further details on these definitions are given in [24, 23].

Approximate symmetry of a point set is defined in terms of a permutation of the points which maps distances between the points approximately onto each other. This definition allows an algorithm to be devised based on expanding local matches without backtracking, enabling us to keep the efficiency of the approach used in [29].

Specifically, let $\mu : \mathcal{P}_1 \rightarrow \mathcal{P}_2$ be a bijection between two point sets $\mathcal{P}_1$, $\mathcal{P}_2$, and let $\epsilon \geq 0$ be a tolerance. We say $\mathrm{DEC}(\mathcal{P}_1, \mathcal{P}_2, \mu, \epsilon)$ is satisfied if $\|P - Q\| =_\epsilon \|\mu(P) - \mu(Q)\|$ for all $P, Q \in \mathcal{P}_1$, and if $=_\epsilon$ is an *equivalence relation* on $\mathcal{D}(\mathcal{P}_1) \cup \mathcal{D}(\mathcal{P}_2)$ (see below; this idea was introduced in [29] to define global approximate symmetry). We define two point sets $\mathcal{P}_1$, $\mathcal{P}_2$ to be *approximately congruent* at tolerance ϵ if, for at least one of all possible bijections $\mu : \mathcal{P}_1 \rightarrow \mathcal{P}_2$, $\mathrm{DEC}(\mathcal{P}_1, \mathcal{P}_2, \mu, \epsilon)$ is satisfied. We say that a point set $\mathcal{P}$ has an *approximate symmetry* (μ, ϵ) for bijection $\mu : \mathcal{P} \rightarrow \mathcal{P}$ and tolerance ϵ if $\mathrm{DEC}(\mathcal{P}, \mathcal{P}, \mu, \epsilon)$ is satisfied.

Assuming that some ϵ can be found for which a set of points exhibits an approximate symmetry, in general, a tolerance *validity interval* $E_\mathcal{P} = [E_{\min}(\mathcal{P}), E_{\max}(\mathcal{P}))$ around ϵ exists at which $\mathcal{P}$ is approximately symmetric. For tolerances ϵ smaller than some *minimal tolerance* $E_{\min}(\mathcal{P})$, some distances in the same distance class would no longer be considered equal, and so the approximate symmetry would not exist. Conversely, for tolerances ϵ greater than some *maximal tolerance* $E_{\max}(\mathcal{P})$, the points would no longer map onto each other in a one-to-one fashion, as more than one point would be considered to be at the 'same' (approximate) position. The validity interval plays a key role in our previous work on detecting (incomplete) point symmetries [24, 23], and will also be used throughout this paper.

A *complete* symmetry consists of *cycles*: orbits arising by repeatedly applying the symmetry mapping to a single point. We define *incomplete cycles* as subsets of a point set $\mathcal{P}$ comprising *consecutive* points from a full cycle; otherwise, multiple and ambiguous symmetries may appear, e.g. a six-fold rotational cycle can be seen as incomplete twelve-fold cycle by allowing gaps. Further conditions are necessary to *uniquely* define an incomplete cycle: (C1) its points must potentially belong to *some* symmetric set of points, (C2) its points must be sufficiently far apart from other points in $\mathcal{P}$

to avoid ambiguity, and (C3) it should contain as many points as possible from $\mathcal{P}$ while still satisfying (C1) and (C2). Formally, let $\mathcal{C} = (P_1, \ldots, P_c)$ be a sequence of $c \geq 2$ points from point set $\mathcal{P}$, which induces a bijection μ mapping P_k to P_{k+1} for $k = 1, \ldots, c-1$. We say that $\mathcal{C}$ is a *(maximal approximate) incomplete cycle* at tolerance $\epsilon \geq 0$ if

(C1) $\mathrm{DEC}(\mathcal{C}, \mathcal{C}, \mu, \epsilon)$ is satisfied (P_c maps to P_1 in a complete cycle, and to no point otherwise—for simplicity we skip over the fact that μ is not defined on the last point of an incomplete cycle);

(C2) no point in $\mathcal{P} \setminus \mathcal{C}$ can replace a point in $\mathcal{C}$ while (C1) still holds under the same μ; and

(C3) no *single* point in $\mathcal{P} \setminus \mathcal{C}$ can be added to $\mathcal{C}$ for any tolerance ϵ while still satisfying (C1) and (C2).

Incomplete symmetries arise by merging compatible cycles under similar conditions to ensure unambiguity as detailed in [23]. A point subset $\mathcal{S} \subset \mathcal{P}$ has an *(approximate) incomplete symmetry* (μ, ϵ) of symmetry type t if

(I1) $\mathcal{S}$ is the union of a set of non-intersecting cycles of type t, each having at least $N(t)$ points to determine a symmetry of a given type t: 2 points for $\mathbf{M}$, $\mathbf{I}$, $\mathbf{C_2}$; 3 for $\mathbf{T}$, $\mathbf{C_3}$; 4 for $\mathbf{C_{n\geq 4}}$, $\mathbf{Z}$ and a regular tetrahedron; 5 for $\mathbf{S_n}$, $\mathbf{W}$.

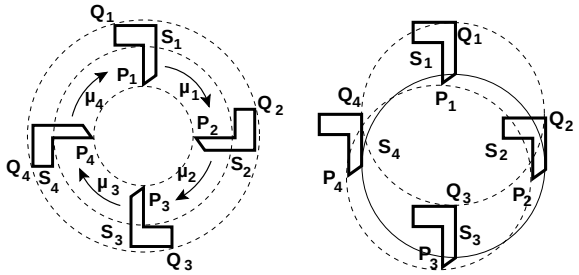
(I2) $\mathrm{DEC}(\mathcal{S}, \mathcal{S}, \mu^*, \epsilon)$ is satisfied, where μ^* is the concatenation of the individual μs of the cycles from (I1); and

(I3) no cycle $\mathcal{C}^*$ of type t in $\mathcal{P} \setminus \mathcal{S}$ exists such that (I2) is true for $\mathcal{C}^* \cup \mathcal{C}$ at tolerance less than ϵ for any cycle $\mathcal{C}$ of $\mathcal{S}$.

3.2. Approximate Symmetries of B-rep Models

The above ideas can be used to detect congruencies, incomplete symmetries and symmetric arrangements of sub-parts of a B-rep model, using *characteristic points* of a B-rep model [10, 29]; see Section 1.

For sub-parts to be congruent or symmetric, any mappings have to satisfy the *consistency condition* that entities are matched to others of the same *geometric type*, e.g. circular arcs to circular arcs. Let $\mu : \mathcal{S}_1 \to \mathcal{S}_2$ be a bijection between the characteristic point sets $\mathcal{S}_1$, $\mathcal{S}_2$ of two sub-parts S_1, S_2. We say μ is *consistent* if whenever a subset $\mathcal{S}^* \subset \mathcal{S}_1$ defines an entity



(a) Type I (b) Type II

Figure 4: Two types of symmetric arrangement of sub-parts

of S_1, $\mu(\mathcal{S}^*)$ are corresponding characteristic points of an entity of the same type. Two sub-parts thus are *approximately congruent* if their characteristic point sets are congruent at tolerance ϵ under a consistent bijection μ. A sub-part is *approximately symmetric* at tolerance ϵ if its characteristic points are symmetric under a consistent bijection μ.

A symmetric arrangement of congruent sub-parts forms a pattern given by a symmetry group. Even if the relations between them are limited to three independent translations in 3D, 230 *space groups* exist [9]. However, most of the cases are not of interest to mechanical engineering, and we only consider two particularly important types: I and II in Fig. 4. Type I is formed by sub-parts which exhibit global symmetry. Thus, their characteristic points form an (incomplete) symmetry, each cycle of which consists of one point from each model at a specific location, called *location cycles*. E.g., in Fig. 4(a), objects S_1, S_2, S_3, S_4 have such an arrangement: cycles (P_1, P_2, P_3, P_4) and (Q_1, Q_2, Q_3, Q_4) are two of its location cycles forming a symmetry. Alternatively, placing a sub-part at symmetric locations while keeping its orientation unchanged gives Type II. In this way, the location cycles are not compatible, but are related by a translation. E.g., in Fig. 4(b), models S_1, S_2, S_3, S_4 have such an arrangement: cycle (Q_1, Q_2, Q_3, Q_4) is a translation of cycle (P_1, P_2, P_3, P_4) by $Q_1 - P_1$.

Thus, a set of congruent sub-parts $S_1, \ldots, S_s$ with characteristic point sets $\mathcal{S}_1, \ldots, \mathcal{S}_s$ has a *symmetric arrangement* at tolerance ϵ if consistent bijections $\mu_k : \mathcal{S}_1 \to \mathcal{S}_k$, $k = 2, \ldots, s$ exist such that for all $P_1 \in \mathcal{S}_1$ with location cycle $\mathcal{C}(P_1) = (P_1, \mu_2(P_1), \ldots, \mu_s(P_1))$, and, for Type I or Type II respectively,

I: all location cycles $\mathcal{C}(P_1)$, $P_1 \in \mathcal{S}_1$ form an incomplete symmetry at tolerance ϵ; or

II: for any other point $Q_1 \in \mathcal{S}_1$, translating cycle $\mathcal{C}(P_1)$ by vector $Q_1 - P_1$ gives cycle $\mathcal{C}(Q_1)$ at tolerance ϵ.

4. Main Design Intent Detection Algorithm

Given an approximate B-rep model as input, our algorithm finds congruencies, incomplete symmetries and symmetric arrangements of sub-parts, with corresponding tolerance levels, which describe the model's geometric design intent. Here we describe the top-level algorithm (Algorithm 1). It takes as input a B-rep model M. It outputs a hierarchy of congruence sets at different tolerance levels with associated symmetries and symmetric arrangements. In general, different regularities may be detected for each congruence set in the hierarchy. We must detect *all* regularities at *all* tolerance levels so as not to miss any intended regularities—a problem noted in [16] with the symmetry detection approach of Zabrodsky et al [44]. We do not consider the problem of *choosing* between alternative, mutually inconsistent regularities (which share entities) here.

As sub-parts are congruent at different tolerance levels, the congruence sets form a tree or forest rather than a simple partition of the sub-parts. Each congruency is described by a set C of sub-parts, a set of pairwise mappings Γ_C giving the congruency matchings, and a validity interval E_C for the congruence set. For each C, incomplete symmetries are detected as symmetries of the congruent shape: $S[C]$ gives its symmetries as mappings μ matching the faces of the congruent shape of C within a validity interval. The congruencies and the symmetries indicate how the elements of a congruence set match and are hence necessary to find symmetric arrangements $A[C]$ of C: we store symmetric arrangements in terms of the mappings between the involved sub-parts, and a validity interval.

Initially the algorithm decomposes the model into a regularity feature tree (Line 01), as explained in Section 6. Congruencies between leaf-parts are then found (Lines 02–03). All sets of congruent leaf-parts (Lines 04–13) are next analysed for incomplete symmetries (Line 09) and symmetric arrangements of Type I and II (Lines 10–13). Finally compatible symmetries are merged (Line 14). We now explain these steps further.

We first construct an RFT T for M, hierarchically decomposing it into simpler sub-parts (Line 01). More than one possible RFT may exist, each with a different validity interval; we discuss later how one is selected. A validity interval E_T is also reported which indicates the range of tolerances

Algorithm $(\mathcal{C}, \mathcal{R}) \leftarrow$DesignIntent$(M)$

Input: M: B-rep model
Output: $\mathcal{C}$: congruency hierarchy
$\mathcal{R}$: symmetries and symmetric arrangements

```
01  (T, E_T) ← RFT(M)
02  L ← leaf_nodes(T)
03  C = {(C_k, Γ_{C_k}, E_{C_k})} ← Congruencies(L, E_T)
04  S ← empty,   A ← empty
05  Q ← roots(C)
06  while not empty(Q)
07    (C, Γ_C, E_C) ← pop(Q)
08    Q ← append(Q,children(C))
09    S[C] ← BRepSymmetries(C, Γ_C, E_C)
10    if S[C] ≠ S[parent(C)]
11      I ← IncompleteCycles(Centroids(C), E_C)
12      G ← FilterGlobalSymmetries(S[C], C)
13      A[C] ← TypeI(C, Γ_C, E_C, G, I) ∪ TypeII(C, Γ_C, E_C, G, I)
14  R ← MergeCompatibleCycles(A ∪ S, E_T)
```

Algorithm 1: Main design intent detection algorithm

for which this tree is valid. Subsequent regularity detection is limited to this range. The set $\mathcal{L}$ of leaf-parts of this tree is extracted (Line 02) for regularity detection.

Congruencies are detected first (Line 03) as these facilitate other regularity detection: all members of a congruence set must exhibit the same incomplete symmetries; symmetrically arranged leaf-parts must be congruent. Congruencies are detected as a hierarchy of congruent leaf-part sets C_k at different tolerance levels. For each such congruence set, the bijections Γ_{C_k} indicate how the leaf-parts are matched, and the tolerance validity interval E_{C_k} is also computed. Congruence detection is detailed in Section 7. The congruence set hierarchy is examined top-down for incomplete symmetries S and symmetric arrangements A using a FIFO queue $\mathcal{Q}$ (Lines 04–08): all congruencies at different tolerance levels in the hierarchy have to be considered as they may exhibit different regularities.

Detecting incomplete symmetries of a congruence set is achieved by first detecting the incomplete symmetries of an *exemplar* leaf-part in the set and then retaining those shared by *all* leaf-parts in the set (Line 09). The in-

complete symmetries of the example leaf-part may have been detected previously when examining the parent congruence set. Hence, for efficiency, these symmetries are cached. Details of the symmetry detection algorithm are described in Section 8.

Next, symmetric arrangements of the elements of a congruence set are detected (Lines 10-13). We first detect incomplete cycles formed by the centroids of the leaf-parts (Line 11) and then select those which also match the leaf-parts (Line 13); see Section 10. In order to do this, the global symmetries of the congruent shape of the congruence set are necessary: combining these symmetries with the congruencies yields all possible ways of matching the leaf-parts. The global symmetries are easily found by filtering the set of all detected symmetries (Line 12). Note that unless a congruence set has different symmetries from its parent in the congruency hierarchy, its symmetric arrangements are also present in its parent. Hence, we only seek symmetric arrangements (Line 10) if new symmetries have been detected. We only need to store and detect symmetric arrangements in that first congruence set in the hierarchy in which they occur.

Finally all incomplete symmetries and symmetric arrangements are stored in $\mathcal{R}$ for output (Line 14). As several cycles may be induced by the same isometry, we combine such compatible cycles; see Section 11.

5. Consistent Clustering

Hierarchical clustering is used in several places in our algorithm to deal with different tolerance levels, to merge mappings, etc. Our clustering algorithm is adopted from [29]. Initially each entity starts in its own cluster, and we compute pairwise similarities between all entities. Clusters are considered for merging in order of increasing entity similarity, leading to a cluster hierarchy. However, clusters must fulfil two conditions. The first is a *transitivity* condition which requires clusters to be sufficiently separate from each other: the distances between all entities within a cluster must all be smaller than the distance between any entity inside the cluster and any other entity outside the cluster. Secondly, the cluster must exist at a tolerance which allows the entities to be appropriately merged, e.g. for clustering symmetries, the existence of the corresponding validity interval, as defined in Section 3.1, for the merged symmetries must also be ensured. Thus, each cluster must have a suitable validity interval which is within an overall validity interval bound to ensure consistency with other tolerances, e.g. from the RFT construction.

Algorithm $\mathcal{H} \leftarrow$ ConsistentClustering(Q, E^*)

Input: $Q = \{(I_k, E_{I_k})\}$: entities I_k with validity intervals E_{I_k}
E^*: validity interval bound

Output: $\mathcal{H}$: consistent cluster hierarchy

```
01  D = {(l,k)} ← sort(Similarity(Q))
02  H ← empty
03  for each (I_k, E_{I_k}) ∈ Q
04    C ← {(I_k, E_{I_k})}, v[C] ← 1, e[C] ← 0, E[C] ← E_{I_k}
05    H ← add_root(H, C)
06    if not empty(E[C] ∩ E*),    Flag(C)
07  for each (l,k) ∈ D // in increasing order of similarity
08    if I_l, I_k are in the same cluster C of H
09      e[C] ← e[C] + 1
10    else I_l, I_k are in distinct clusters C_1, C_2 of H
11      C ← Merge(C_1, C_2, H)
12      v[C] ← v[C_1] + v[C_2], e[C] ← e[C_1] + e[C_2] + 1
13      E[C] ← ValidityInterval(C_1, C_2)
14    if Complete(v[C], e[C]) and not empty(E[C] ∩ E*)
15      Flag(C)
```

Algorithm 2: Consistent hierarchical clustering algorithm

If we construct a graph with nodes representing the entities to be merged, we can detect transitive clusters as *complete* sub-graphs, i.e. sub-graphs of v vertices having e edges with acceptable similarities where $e = v(v+1)/2$.

The clustering algorithm in Algorithm 2 takes as input a set of distinct entities $\{I_k\}$ with corresponding tolerance intervals E_{I_k}, plus an overall tolerance range E^*. It outputs a consistent cluster hierarchy. Each distinct pair of entities (l, k) from $\{I_k\}$ is stored in a list D sorted by increasing order of entity similarity (Line 01). The similarity measure depends on the type of entities being clustered; details are given later. For each entity I_k, a cluster C is created with $v[C] = 1$ vertices, $e[C] = 0$ edges, and a validity interval $E[C] = E_{I_k}$; this is added to the cluster hierarchy $\mathcal{H}$ (Lines 03–06). We only wish to keep clusters fulfilling the two consistency conditions, so we flag such clusters. The initially created clusters are consistent if the intersection of the clusters' validity interval $E[C]$ with the tolerance range bound E^* is not empty (Line 06).

In increasing order of similarity, clusters corresponding to the pairs from

D are merged: an edge is inserted into the graph of entities (Lines 07–15). During this process, the cluster hierarchy $\mathcal{H}$ is updated, and complete components of the graph are detected by tracking the number of edges $e[C]$ and nodes $v[C]$ in each cluster C. Firstly, the clusters linked by the current entity pair (l, k) from D are merged. If the current pair links nodes inside the same cluster, we increase the edge count for the cluster (Lines 08–09). Otherwise, we merge the two distinct clusters and create new vertex and edge counts accordingly (Lines 10–13). We next check whether the resulting cluster is consistent. Clusters which are complete and fulfil the validity condition within the overall tolerance range E^* are flagged (Lines 14–15). The earlier merging of two clusters (Line 11) only preserves sub-clusters if these are flagged. Clusters are ignored if they do not fulfil the validity condition and hence are not a valid merged entity, or if they violate the overall tolerance bound.

Line 13 involves computation of the validity interval $E_C = [E_{\min}(C), E_{\max}(C))$ for a set of entities $C = C_1 \cup C_2$ to decide if it is valid. Assuming C comprises the entities $I_1, \ldots, I_c$ with given minimal and maximal tolerances for each I_k, we set $E_{\min}(C) = \max_{1 \le l \ne k \le c}(E_{\min}(I_l \cup I_k))$, $E_{\max}(C) = \min_{1 \le k \le c}(E_{\max}(I_l \cup I_k))$.

This clustering method takes $O(n^2 \log n)$ time for n entities to be clustered: n^2 pairs have to be sorted and all $n(n-1)/2$ pairs must be merged, taking $O(\log n)$ time for each pair [29].

6. Regularity Feature Tree Construction

Next we outline the RFT construction used to initially hierarchically decompose the input model (Line 01, Algorithm 1). Our algorithm is detailed in [22], so here we give a brief overview and concentrate on the associated tolerance levels in the context of our design intent detection algorithm.

Many regularities can be expressed in terms of symmetries [19]. Our RFT construction is based on recovering symmetries that were broken during construction of the (ideal, rather than approximate) original model by using modelling operations on geometric primitives. The *regularity feature* is used to represent the newly generated sub-parts for this purpose. 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. They hence are not a class of pre-defined simple geometric primitives, but are determined by the model's geometry.

The RFT is used here to record the hierarchal process of producing these regularity features. For example, the RFT for the Monster model in Fig. 1 is displayed in Fig. 2, and the RFT for the Angle in Fig. 11 is displayed in Fig. 10.

Specifically, our approach groups model entities into sub-parts, i.e. regularity features, with the aid of newly constructed edges, faces and sub-parts derived from the initial geometry. No Boolean operations are involved. By analysing the model's faces, missing edges are constructed which give rise to new faces, and consequently new sub-parts when combined with existing geometry. These sub-parts have to be combined with each other to build the model. By recursively decomposing the arising sub-parts, the model is decomposed hierarchically giving the RFT. For details see [22]. Sub-parts in this hierarchy increasingly become simpler and more symmetric until no further decomposition can be found. When combined, the leaf-parts of the RFT describe the whole model.

As the input model is approximate, RFT construction requires careful matching and intersection of edges and faces. Handling of the tolerances necessary for design intent detection was not addressed in [22]. We do so here robustly by clustering model faces which share the same underlying surface within tolerance—different edges between face pairs which share the same underlying surface must belong to the same underlying curve. Where two surfaces intersect in multiple intersection curves, we consider each in turn, and allocate edges to each intersection curve based on its minimum distance. To decide which underlying surfaces are approximately the same we use our ConsistentClustering algorithm. The input entities I_k are the faces F_k of the input model. We set no predefined tolerance bound ($E^* = [0, \infty)$) and set $E_{I_k} = [0, d^*)$, where d^* is the minimal distance between any two characteristic points of F_k (we need these points to be distinctly identifiable). The symmetric Hausdorff distance between two faces is a good measure of whether they share the same underlying surface, and we use it to compute the minimal tolerances and similarity measure.

As the face clusters form a hierarchy, different RFTs may be constructed at different tolerance levels. Instead of expensively examining all resulting RFTs for regularities, we let the user interactively choose a tolerance level by selecting suitable clusters and with them an RFT T. This results in a minimum distance tolerance $E_{\min}(T)$ required to match the faces in the clusters and a maximum distance tolerance $E_{\max}(T)$ at which the selected clusters would need to be merged with other clusters higher up in the hierarchy. The

validity interval $E_T = [E_{\min}(T), E_{\max}(T))$ is then used to determine which regularities are found by the rest of the design intent algorithm.

7. Detecting Congruencies

After RFT construction, congruencies between leaf-parts are detected (Line 03, Algorithm 1). The congruency detection algorithm (Algorithm 3) takes as input a set $\mathcal{L}$ of leaf-parts (each is a closed B-rep model), and the validity interval E_T from the RFT construction. It hierarchically groups $\mathcal{L}$ into congruence sets, each of which contains (i) a set of leaf-parts C_k, (ii) the point mappings Γ_{C_k} giving all pairwise congruency matchings between the leaf-parts, and (iii) the validity interval E_{C_k} for each set. If the congruent parts are symmetric, multiple matchings between them exist, but we only find the match with the minimum error.

7.1. Idea

Congruencies can be detected by adapting the clustering algorithm (Algorithm 2) to cluster the leaf-parts into congruence sets based on a similarity measure describing their congruency. This measure has to clearly describe the congruency difference between every leaf-part pair, and we compute it as the minimal matching error given by a consistent congruency mapping between the leaf-parts' characteristic points. The algorithm's efficiency can be further improved by noticing that all the leaf-parts in a congruence set have the same geometric structure, e.g., the same number of faces of each geometric type, etc. This observation helps to group the initial set into sub-sets which may potentially be congruent. The clustering algorithm is only run on these sub-sets separately to reduce the number of leaf-part pairs considered.

7.2. Algorithm Description

The algorithm first computes the similarity (based on congruency) between every leaf-part pair, and then uses the similarities to cluster the leaf-parts into congruence sets. For leaf-parts to be congruent, their geometric entities must be matched consistently by a congruency mapping. For efficiency, we first group the leaf-parts $\mathcal{L}$ into subsets according to whether a congruency can exist at all (Line 02): e.g., parts in each group must have the same number of vertices, number of faces of each geometric type, etc; see [10]. Within each group $\mathcal{G}$, the best congruency match for each pair of leaf-parts $L_1, L_2 \in \mathcal{G}$ is computed by finding the mapping which maps the

Algorithm $C \leftarrow$ CONGRUENCIES$(\mathcal{L}, E_T)$

Input:	$\mathcal{L}$: leaf-part set
	E_T: validity interval bound
Output:	$C = \{(C_k, \Gamma_{C_k}, E_{C_k})\}$: set of congruencies between elements of $\mathcal{L}$: each C_k is a set of leaf-parts, matched using the point mappings Γ_{C_k} with validity interval E_{C_k}

```
01  C ← empty
02  L* ← CongruenceGrouping(L)
03  for each G ∈ L*
04    for each (L1, L2) ∈ distinct_pairs(G)
05      Γ[L1, L2] ← BestConsistentMatch(L1, L2)
06      σ[L1, L2] ← MatchError(L1, L2, Γ[L1, L2], E_T)
07    C ← C ∪ ConsistentClustering((G, Γ, σ), E_T)
```

Algorithm 3: Detecting congruent leaf-parts

characteristic points of the leaf-parts entities consistently onto each other with minimum matching error. From this match, their similarity $\sigma[L_1, L_2]$ is computed as described shortly (Lines 03-06). This is used to cluster the leaf-parts into congruent sets (Line 07). Here, our CONSISTENTCLUSTERING algorithm is called with validity interval bound $E^* = E_T$. It is adapted to cluster the leaf-parts L_k in each group $\mathcal{G}$ linked with the validity intervals $E_{L_k} = [0, d_k^*)$ where d_k^* is the minimal inter-point distance between the characteristic points of L_k. Γ stores the pairwise point matchings for each resulting cluster and the similarity is given by σ.

To compute the similarity $\sigma[L_1, L_2]$ between leaf-parts L_1, L_2 with characteristic point sets $\mathcal{P}_1$, $\mathcal{P}_2$ we need their best match. Among all bijective mappings $\gamma : \mathcal{P}_1 \rightarrow P_2$, mapping L_1 to L_2 consistently, we seek the one with minimum matching error $\sigma_\gamma = \max_{P,Q \in \mathcal{P}_1} |\|P - Q\| - \|\gamma(P) - \gamma(Q)\||$ To find it, we follow [10]: we select a 'large' tetrahedron T_1 from $\mathcal{P}_1$ and map it to all possible tetrahedra in $\mathcal{P}_2$. Each mapping between these tetrahedra determines an isometry, and hence a unique mapping from $\mathcal{P}_1$ to $\mathcal{P}_2$. The mapping between the characteristic points of L_1 and L_2 thus determines correspondence between their entities, from which we can check their consistency. Clearly, the minimal σ_γ computed in this way must also be the minimal tolerance $E_{\min}(L_1, L_2)$ for the congruency between L_1, L_2. For 2D objects, e.g. planar faces, a triangle must be used instead of a tetrahedron.

Generally, for symmetric leaf-parts, more than one mapping between two

congruent leaf-parts exists. We do not need to consider these here, as such symmetries are detected as incomplete symmetries later (Section 8), and are then considered for symmetric arrangements (Section 10).

Given L leaf-parts with at most p characteristic points per leaf-part, the CONGRUENCIES algorithm is expected to take $O(L^2 p^{2.5} \log^4 p)$ time: $L(L-1)/2$ distinct leaf-part pairs are checked for congruency, each requiring the same time as global symmetry detection: $O(p^{2.5} \log^4 p)$ [29].

8. Detecting Incomplete B-Rep Symmetries

We now discuss the detection of common incomplete symmetries of a set of congruent leaf-parts (Line 09, Algorithm 1). This algorithm is called once for each congruence set in the congruency hierarchy.

8.1. Idea

We first describe the idea behind Algorithm 4. As the output incomplete B-rep symmetries have to be present in all the leaf-parts in a congruence set C, an efficient strategy is applied: we first detect symmetries of an exemplar leaf-part, and then verify which of these are also present in all the other leaf-parts in the set. Detecting symmetries within an exemplar leaf-part is very similar to the overall design intent detection algorithm (Algorithm 1) with the main difference that we are processing faces instead of leaf-parts; it is further described in Section 8.3. The verification process is efficiently achieved using the point mapping between the leaf-parts in the congruence set. This mapping prescribes how the exemplars points must be mapped onto the other leaf-parts. Therefore, they can be used to transfer the symmetry mappings from the exemplar part to the other parts. The transferred symmetry is only valid if there is a non-empty validity interval for it.

For example, suppose in Fig. 5 that $L_1, \ldots, L_3$ are congruent leaf-parts given by mappings $\gamma_k : L_1 \rightarrow L_k$ which map O_1 to O_k, P_1 to P_k, Q_1 to Q_k and R_1 to R_k for $k = 2, 3$, and L_1 is the picked exemplar model. Let $\sigma = (O_1, P_1, Q_1, R_1)$ denote a cycle of L_1 that builds a four-fold rotational symmetry of L_1 at a proper validity interval. To verify that the rotational symmetry σ is also shared by other leaf-parts L_k, $k = 2, 3$ in the congruence set, we only need to check that each point set $\gamma_k(\sigma(L_1))$ (i.e. the characteristic points) also builds a valid four-fold rotational cycle of L_k at a proper validity interval, and all these intervals together share a common range with that of L_1.

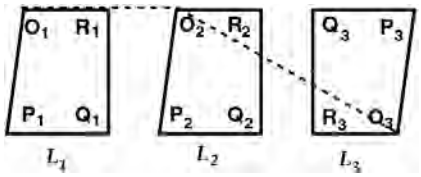


Figure 5: Finding common incomplete symmetries of a set of congruent leaf-parts

Algorithm $\mathcal{I} \leftarrow$BRepSymmetries(C, Γ_C, E_C)

Input: C: set of congruent leaf-parts
Γ_C: pairwise congruency matchings for C
E_C: validity interval for congruencies of C

Output: $\mathcal{I} = \{(\mathcal{F}_k, \mu_{\mathcal{F}_k}, E_{\mathcal{F}_k})\}$: incomplete symmetries of C

01 $(L^*, I^*) \leftarrow$ FindCachedSymmetry(C)
02 **if empty**(I^*)
03 $L^* \leftarrow$ **first_element**(C)
04 $I^* = \{(F_k, \mu_{F_k}, E_{F_k})\} \leftarrow$FaceSymmetries$(L^*, E_C)$
05 $L^* \leftarrow$ SymmetryCache(L^*, I^*)
06 **for each** $(F_k, \mu_{F_k}, E_{F_k}) \in \mathcal{I}^*$
07 $E^* \leftarrow E_{F_k}$
08 **for each** $L \in C$
09 $E^* \leftarrow E^* \cap$ ValidityInterval$(L, L^*, \mu_{F_k}, \Gamma_C)$
10 **if not empty**(E^*), $\mathcal{I} \leftarrow$**append**$(\mathcal{I}, (F_k, \mu_{F_k}, E^*))$

Algorithm 4: Detecting incomplete symmetries of a congruence set

8.2. Algorithm Description

Our algorithm (Algorithm 4) takes as input a congruence set given by C, Γ_C, E_C, and outputs the incomplete symmetries shared by all leaf-parts in C. Each incomplete symmetry is represented by a set of faces $\mathcal{F}_k$, a bijection μ_{F_k} indicating how these faces are matched, and a corresponding validity interval $E_{\mathcal{F}_k}$.

The algorithm first detects the incomplete symmetries of an exemplar leaf-part L^* selected from the congruence set. *Any* part may be chosen as the exemplar, as we are only interested in symmetries shown by *all* leaf-parts in the set. For efficiency, to avoid re-finding the symmetries of exemplars while processing the congruency hierarchy, we cache all symmetries of exemplars. Lines 01–05 detect incomplete symmetries in the exemplar, while Lines 06–10 find which of these are also present in all other leaf-parts in the set.

To find incomplete symmetries we first check whether the current congruence set contains a leaf-part with previously cached symmetries (Line 01). If so, we use this leaf-part L^* and its symmetries I^* to check the other leaf-parts. Otherwise, we arbitrarily select the first leaf-part in the congruence set as L^*, find its symmetries I^* as explained below, and cache these with the leaf-part (Lines 02-05).

We only keep those symmetries of the exemplar L^* verifiably present in all other leaf-parts (Lines 07–10), to within tolerance. Given the congruency mappings from Γ_C, we can convert the symmetry mapping μ for the exemplar to a potential symmetry mapping μ_L of any other leaf-part $L \in C$; we compute its validity interval E_{μ_L} as described in Section 9. If the intersection of all the validity intervals involved is not empty, a validity interval exists at which the exemplar's symmetry is present in *all* leaf-parts.

8.3. Detecting Incomplete Leaf-part Symmetries

We now describe the algorithm for finding incomplete leaf-part symmetries (Line 04, Algorithm 4). Our definition of incomplete symmetries of B-rep models (Section 3.2) requires detection of incomplete symmetries formed by a part's faces. This is the same problem as finding Type I symmetric arrangements of a leaf-part's faces. Hence, our face symmetry algorithm (Algorithm 5) finds such arrangements of faces. It is very similar to the overall design intent detection algorithm (Algorithm 1) with the difference that we are processing faces instead of leaf-parts, and only output Type I symmetric arrangements.

We first find a hierarchy of congruent faces (Line 01). To detect congruent faces the CONGRUENCIES algorithm in Algorithm 3 is used, except triangles replace tetrahedra for planar faces, when the plane containing the faces is considered to determine the complete mapping in 3D. As in the main design intent detection algorithm, we then process the congruency hierarchy top-down using a FIFO queue Q to detect the global symmetries $S[C]$ and Type I symmetric arrangements $A[C]$ for each cluster C of congruent faces (Line 02–08). We only need global symmetries of faces, instead of incomplete symmetries, in this case, so we use a global symmetry detection algorithm [29] (Line 05). We then look for Type I symmetric arrangements if the symmetries of the current cluster are not the same as those of its parent cluster (Lines 06–08). Firstly, incomplete cycles of the centroids of the faces in the cluster are found (Line 07, Section 9) and then these cycles are used to

Algorithm $A \leftarrow$FaceSymmetries(L, E)

Input: L: leaf-part B-rep model
E: validity interval bound

Output: $A = \{(F_k, \mu_{F_k}, E_{F_k})\}$: face symmetries of L given by a set of faces F_k with the symmetry mapping μ_{F_k} and its validity interval E_{F_k}

```
01  C ← Congruencies(Faces(L), E)
02  S ← empty, A ← empty, Q ← roots(C)
03  while not empty(Q)
04    (C, Γ_C, E_C) ← pop(Q), Q ← append(Q,children(C))
05    S[C] ← GlobalFaceSymmetries(C, Γ_C, E_C)
06    if S[C] ≠ S[parent(C)]
07      I ← IncompleteCycles(Centroids(C), E_C)
08      A[C] ← TypeI(C, Γ_C, E_C, S[C], I)
09  A ← MergeCompatibleCycles(A)
```

Algorithm 5: Detecting incomplete symmetries of faces in a leaf-part

find Type I symmetric arrangements (Section 10). The symmetric arrangements are represented as face cycles $\{(\mathcal{F}_k, \mu_{\mathcal{F}_k}, E_{\mathcal{F}_k})\}$ formed by the faces $\mathcal{F}_k$ of L. Finally, these are merged into compatible symmetry transformations (Line 09, Section 11). Merging has to be done here in order to identify global symmetries as preparation for detecting symmetric arrangements of leaf-parts.

9. Detecting Incomplete Symmetry Cycles

We now describe the detection of incomplete symmetry cycles in point sets using the ideas from Section 3; this is needed by the symmetry and symmetric arrangement detection algorithms. The algorithm is a core element of out design intent detection approach, and essentially decides which regularities we can detect. An overview was originally reported in [23]; here we give the complete algorithm with previously unpublished details.

9.1. Idea

The underlying idea is to expand an (approximate) isosceles triangle point by point to build up a symmetry cycle. A similar process was proposed by Brass [5] for detecting exact, complete, rotational symmetries by merging isosceles triangles. We previously extended it to approximate, complete,

rotational and rotation-mirror cycles [24]. By careful analysis of all possible expansion point locations, and reasoning about the potential symmetry structure, incomplete cycles of all seven basic symmetry types in 3D can be detected, as presented here.

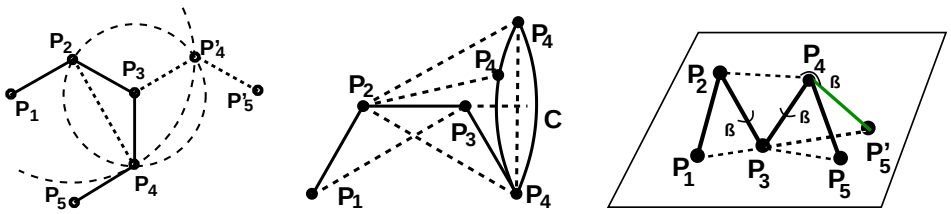
Point Expansion for Exact Cycles. The idea behind the cycle detection algorithm is now introduced. We first describe the exact 2D case. Let $P_1, P_2, P_3 \in \mathbb{R}^2$ be an isosceles triangle with $\|P_1 - P_2\| = \|P_2 - P_3\|$ and $\|P_1 - P_3\| \geq \|P_1 - P_2\|$ (Fig. 6(a)). These three points partially define an isometry, under which P_1 moves to P_2, and P_2 to P_3. To completely define an isometry, we have to find an image for P_3. This image exists if there is a point P_4 such that

$$\|P_4 - P_3\| = \|P_2 - P_1\|, \quad \|P_4 - P_2\| = \|P_3 - P_1\|. \tag{1}$$

Two possible locations exist: P_4 and P_4' in Fig. 6(a). Let us choose P_4 and consider a further point P_5. Replacing P_1, P_2, P_3, P_4 by P_2, P_3, P_4, P_5 in Eq. (1) under the additional condition $\|P_5 - P_2\| = \|P_4 - P_1\|$, we can *only* get a $\mathbf{C_n}$ symmetry. Similarly taking P_4' instead of P_4 can only lead to a $\mathbf{Z}$ symmetry. The special case of translational symmetry arises if P_1, P_2, P_3 are collinear.

In 3D, the fourth point P_4 satisfying Eq. (1) can lie *anywhere* on a circle C (Fig. 6(b)). Different locations of P_4 induce symmetries of different types. If P_1, P_2, P_3, and P_4 are all coplanar, all further expansion points in this cycle must also be coplanar (see [24]), and a cycle of type $\mathbf{C_n}$, $\mathbf{Z}$ or $\mathbf{T}$ can exist. Other locations of P_4 on the circle C may give a regular tetrahedron, if all distances involved are equal, or $\mathbf{S_n}$ or $\mathbf{W}$ symmetries, depending on the next expansion point P_5. As illustrated in Fig. 6(c), by replacing P_1, P_2, P_3, P_4 by P_2, P_3, P_4, P_5 in Eq. (1) with the additional condition $\|P_5 - P_2\| = \|P_4 - P_1\|$, P_5 can still lie on either side of the plane defined by P_2, P_3, P_4: the angle between faces P_3, P_4, P_5 and P_2, P_3, P_4 is equal to that between P_2, P_3, P_4 and P_1, P_2, P_3. If P_5 lies on the same side as P_1, the cycle and its further expansion produce an $\mathbf{S_n}$ symmetry, otherwise it gives $\mathbf{W}$ symmetry.

Thus, all seven elementary symmetries can be identified during the expansion process via equality of certain distances, and differing locations of certain subsequent points given the first three points. Having found the first few expansion points, the locations of all further expansion points are then fixed by certain distance equalities.



(a) Expansion from three points in 2D

(b) Expansion from three points in 3D

(c) Expansion from four points in 3D

Figure 6: Different expansion points of cycles yield different symmetry types

Extension to Approximate Cycles. Unfortunately, the exact expansion process cannot be applied directly to the approximate case, due to the difficulty of choosing a tolerance to determine equality of distances, and possible accumulation of errors as a cycle is built up. To avoid these problems, we add further constraints when determining each successive expansion point. We require approximate equality of *all* point-pair distances that should be equal in the exact case, such that this approximate equality forms an equivalence relation on all these distances (Section 3.1). E.g., to determine the expansion point P_5 from the seed set (P_1, P_2, P_3, P_4) in Fig. 6(a), we require $\|P_5 - P_4\| =_\epsilon \|P_{k+1} - P_k\|$, $k = 1, 2, 3$, $\|P_5 - P_3\| =_\epsilon \|P_{l+2} - P_l\|$, $l = 1, 2$ and $\|P_5 - P_2\| =_\epsilon \|P_4 - P_1\|$. This determines the elements of each equivalence class $\mathcal{G}_r(\mathcal{C})$, $1 \leq r \leq R$ of distances, for a seed set $\mathcal{C} = \{P_1, \ldots, P_c\}$ of c points. From the definition of minimal and maximal tolerances (Section 3.1) we get

$$\begin{array}{rcl} E_{\min}(\mathcal{C}) & = & \max_{1 \leq r \leq R}(D^r_{\max}(\mathcal{C}) - D^r_{\min}(\mathcal{C})), \\ E_{\max}(\mathcal{C}) & = & \min_{1 \leq r \leq R-1}(D^{r+1}_{\min}(\mathcal{C}) - D^r_{\max}(\mathcal{C})), \end{array} \tag{2}$$

where $D^r_{\min}(\mathcal{C})$ is the minimum and $D^r_{\max}(\mathcal{C})$ the maximum distance in class $\mathcal{G}_r(\mathcal{C})$. Combining Eq. (2) with the existence of the validity interval (see Section 3.1) yields a condition on the next expansion point P of $\mathcal{C}$: $E_{\min}(\mathcal{C} \cup \{P\}) < E_{\max}(\mathcal{C} \cup \{P\})$.

The main issue for the computation of $E_{\min}(\mathcal{C} \cup \{P\})$, $E_{\max}(\mathcal{C} \cup \{P\})$ is to determine which class $\mathcal{G}_r(\mathcal{C} \cup \{P\})$ the distances $L_k = \|P - P_{c-k+1}\|, k = 1, \ldots, c$ belong to. A method for doing this is given in [24] for $\mathbf{C_n}$ and $\mathbf{S_n}$ symmetries. Cycles for $\mathbf{W}, \mathbf{T}, \mathbf{Z}$ symmetries can be handled as special cases: we must have $L_k \in \mathcal{G}_k$, as all involve translation.

More than one acceptable candidate expansion point may exist in the input point set, so we must consider all possibilities. We choose the one

minimising $E_{\min}(\mathcal{C} \cup \{P\})$ to avoid adding any point that would violate the cycle separation condition (C2) (Section 3.1). If the exact case allows more than one location for the next expansion point (for different symmetry types as explained above), we must consider *every* possibility separately. Thus, all potential fourth expansion points, and the best fifth expansion point for each location in 3D have to be considered.

9.2. Algorithm Description

Based on the ideas described above, the algorithm on detecting incomplete cycles in a point set is devised in Algorithm 6, and now further explained. This algorithm takes as input a set of distinct 3D points $\mathcal{P}$ and a validity interval bound E^*. We assume no two input points have the same position. It outputs all cycles $(\mathcal{C}_k, E_{\mathcal{C}_k})$ where $\mathcal{C}_k$ are the ordered points of the cycle, with validity interval $E_{\mathcal{C}_k}$. Their symmetry types are determined later when merging the cycles into symmetries (Section 11).

To find the set $\mathcal{I}$ of all cycles in $\mathcal{P}$, every distinct point triple $\mathcal{C}$ in $\mathcal{P}$ is considered for cycle generation (Line 02). A FIFO queue $\mathcal{Q}$ (Line 03) is used to store all expansions of the current triple with their validity intervals, so as to consider all possible expansion points. For each triple $\mathcal{C}$ of points P_1, P_2, P_3, we first compute its validity interval $E_{\mathcal{C}}$ (Line 04). Assuming without loss of generality that $\|P_1 - P_3\|$ is the largest distance, the validity interval is the intersection of the overall validity interval bound E^* and the interval $[0, |\|P_2 - P_1\| - \|P_3 - P_2\||)$ given by the distance matching error. If $\mathcal{C}$ satisfies the validity condition and also does not appear consecutively in a previously detected cycle, checked by calling CONTAINED (Line 05), it is added to $\mathcal{Q}$ for further expansion (Line 06). For the initial triple the validity condition is simply that the intersection of its validity interval $E_{\mathcal{C}}$ and the validity interval bound E^* is not empty. CONTAINED can be efficiently implemented by an $O(|\mathcal{P}|^3)$ Boolean lookup table representing all triples of points as $|\mathcal{P}|$ is typically small. As cycles are found, entries corresponding to all consecutive triples in a cycle are set to true.

Each cycle $\mathcal{C}$ with validity interval $E_{\mathcal{C}}$ in $\mathcal{Q}$ is expanded one point at a time (Lines 07–29). We first collect in M all valid potential expansion points P^* (Line 09), i.e. all points P which can be added to $\mathcal{C}$ such that the validity interval of the expanded cycle is not empty: $E_{\min}(\mathcal{C} \cup \{P\}) < E_{\max}(\mathcal{C} \cup \{P\})$. If no such point exists, we have found an incomplete cycle $\mathcal{C}$, which is appended to the output and processed no further (Lines 10–12). Otherwise, depending on the number of points in $\mathcal{C}$, alternative locations for

Algorithm $\mathcal{I} \leftarrow$ IncompleteCycles$(\mathcal{P}, E^*)$

Input: $\mathcal{P}$: 3D point set
E^*: validity interval bound
Output: $\mathcal{I} = \{(\mathcal{C}_k, E_{\mathcal{C}_k})\}$: incomplete cycles of $\mathcal{P}$

```
01  I ← empty
02  for each C = (P1, P2, P3) ∈ distinct_triples(P)
03    Q ← empty  // FIFO queue of cycles
04    E_C ← MatchingError(P1, P2, P3) ∩ E*
05    if not empty(E_C) and not Contained(C, I)
06      Q ← append(Q, (C, E_C))
07    while not empty(Q)
08      (C, E_C) ← pop(Q)
09      M ← {P : E_min(C ∪ {P}) < E_max(C ∪ {P}), P ∈ P \ C}
10      if empty(M)
11        I ← append(I, (C, E_C))
12        continue  // go to Line 07
13      if |C| = 3
14        if RegularTriangle(C)
15          I ← append(I, (C, E_C))
16        A ← M
17      else if |C| = 4
18        if RegularTetrahedron(C)
19          I ← append(I, (C, E_C))
20        (M1, M2) ← SplitAtPlane(M, C)
21        A ← {arg min{E_min(C ∪ {P}) : P ∈ M1},
                    arg min{E_min(C ∪ {P}) : P ∈ M2}}
22      else
23        A ← {arg min{E_min(C ∪ {P}) : P ∈ M}}
24      for each P* ∈ A
25        C* ← C ∪ {P*},    E_C* ← E_C* ∩ E*
26        if not Ambiguous(C*, E_C*)
27          if Complete(C*),   I ← I ∪ {(C*, E_C*)}
28          else   Q ← append(Q, (C*, E_C*))
29  I ← append(I, TwoCycles(distinct_pairs(P)))
```

Algorithm 6: Detecting incomplete cycles in a point set

expansion points are considered to find *all* possible expanded cycles from $\mathcal{C}$ (Lines 13–23). In each case we determine the actual expansion points for $\mathcal{C}$ from the list M of potential expansion points, and store them in a list A to construct the expanded cycles later. If $|\mathcal{C}| = 3$ (Lines 14–16), we first check for the special case where $\mathcal{C}$ is a regular triangle and if found, append it to $\mathcal{I}$ as it is a complete symmetry cycle. To test whether three points form a regular triangle the incomplete cycle conditions (C1) to (C3) are verified directly based on the point distances. Independently, all points in M are added to A as in the $|\mathcal{C}| = 3$ case all potential expansion points give an expanded cycle. If $|\mathcal{C}| = 4$ (Lines 17–21), points in M are put into two subsets M_1, M_2 according to which side of the plane P_2, P_3, P_4 they lie in. The point minimising $E_{\min}(\mathcal{C} \cup \{P\})$ for each of these two sets is chosen as an expansion point and added to A (Line 21). Also for $|\mathcal{C}| = 4$ there is a special case giving a complete symmetry: the points may form a regular tetrahedron (Lines 18–19). As for the regular triangle, the incomplete cycle conditions can be verified directly via the distances. For all other cases (Lines 22–23), i.e. $|\mathcal{C}| \geq 5$, a unique expansion point minimising the matching error is selected.

Next, each expansion point P^* in A is used in turn for expanding $\mathcal{C}$ (Lines 24–28). If the cycle is complete, and unambiguous, i.e. fulfils condition (C3) (Section 3.1), it is added to the output. Otherwise, only if the new cycle is unambiguous, it is added to the queue of cycles being considered further. A cycle $\mathcal{C}$ with c points describes a complete cycle if $\|P_c - P_1\| =_\epsilon \|P_l - P_{l+1}\|$ for $1 \leq l \leq c-1$. An efficient method to verify unambiguity of a complete cycle, which can also be applied to incomplete cycles, is given in [24].

Lastly, each input point pair forms a trivial cycle, corresponding to a mirror, an inversion and a $\mathbf{C_2}$ symmetry, so we add them all (Line 29). These pairs are considered later when detecting symmetry types and merging compatible cycles. The validity interval for each pair is simply $[0, d)$, where d is the distance between the point pair.

This algorithm takes $O(Cn^4)$ time, where C is the maximal number of elements in a cycle, and n is the number of points in $\mathcal{P}$: all triples of points are taken as initial seeds, and then each remaining point is considered as an expansion point in an iterative expansion process.

10. Detecting Symmetric Arrangements

We now discuss detecting Type I and II symmetric arrangements (defined in Section 3.2) of congruent leaf-parts (Line 12, Algorithm 1).

10.1. Detecting Type I Symmetric Arrangements

The algorithm for detecting Type I symmetric arrangements of congruent leaf-parts (Algorithm 7) takes as input a congruence set C with congruency mappings Γ_C and validity interval E_C, global symmetries $\mathcal{G}$ of the congruent shape of congruence cluster C and incomplete cycles $\mathcal{I}$ of the centroids of the leaf-parts in C. It outputs $\mathcal{A} = \{(A_k, \Xi_{A_k}, E_{A_k})\}$: the symmetric arrangement of leaf-parts A_k with mappings Ξ_{A_k} between them, such that the leaf-parts form an (incomplete) cycle with corresponding validity interval E_{A_k}.

10.1.1. Idea

If a set of leaf-parts has a symmetric arrangement, so do their centroids. Thus we first seek cycles of the leaf-part centroids. These induce location cycles for corresponding characteristic points, one from each leaf-part, related by the congruency mappings; see, e.g., Fig. 4(a). However, although we know the congruency correspondence between the leaf-part points from Γ_C, these cannot be directly applied to produce location cycles: these mappings only represent the best match between leaf-parts. Multiple matchings arising from global symmetries of the leaf-parts are not represented.

For example, suppose in Fig. 7 that $L_1, \ldots, L_5$ are congruent leaf-parts given by mappings $\gamma_k : L_1 \rightarrow L_k$ which map O_1 to O_k, P_1 to P_k, Q_1 to Q_k and R_1 to R_k for $k = 2, \ldots, 5$, and that their centroids $B_1, \ldots, B_5$ make a valid cycle. Directly employing γ_k produces the point sequences: $(O_1, \ldots, O_5)$, $(P_1, \ldots, P_5)$, $(Q_1, \ldots, Q_5)$ and $(R_1, \ldots, R_5)$. Clearly, none of these represent a cycle (as indicated by the dashed line). Global symmetries of the leaf-parts also have to be considered: we need to swap O_3 and Q_3, P_3 and R_3, which is permissible using the (approximate) $\mathbf{C}_2$ symmetry of L_3: $(O_3, Q_3), (P_3, R_3)$. Valid location cycles can then be constructed by further ruling out inconsistent points from L_5 (as explained later): (O_1, O_2, Q_3, O_4), (P_1, P_2, R_3, P_4), (Q_1, Q_2, O_3, Q_4), (R_1, R_2, P_3, R_4). These are compatible with (B_1, B_2, B_3, B_4), and give the symmetric arrangement of the leaf-parts $L_1, \ldots, L_4$.

Thus, to successfully detect symmetric arrangements of leaf-parts from location cycles induced by centroid cycles, we must take into account global

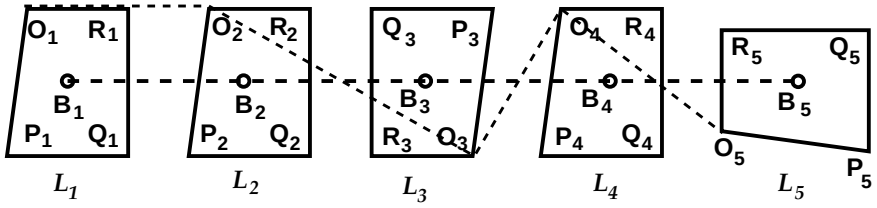


Figure 7: Detecting symmetric arrangement of congruent leaf-parts must consider location cycles as well as its centroid arrangements

symmetries of leaf-parts. Specifically, given a generic point $P_l^1 \in L_1$, we must consider putting it in correspondence with all points of leaf-part L_k which come from $\gamma_k(g(P_l^1))$ for all global symmetries $g \in \mathcal{G}$. We also add the identity to $\mathcal{G}$, as clearly $\gamma_k(P_1^1)$ has to be considered. Then, if a compatible set of location cycles exists, we have a Type I symmetric arrangement of the involved leaf-parts.

Furthermore, we must consider cases in which only a subset of all leaf-parts involved in a centroid cycle is symmetrically arranged, as for some only the centroids and not the actual leaf-parts may be symmetrically arranged. E.g., in Fig. 6, only $L_1, \ldots, L_4$ form a symmetric arrangement with L_5 being ruled out. This means we seek *sub-cycles* of the centroid cycle which yield symmetric arrangements of a leaf-part subset, which can be verified by the compatibility of their location cycles. We need not consider leaf-part subsets that correspond to points of the centroid cycle with fixed index differences: such sub-cycles are actually also detected as another incomplete centroid cycle in $\mathcal{I}$, as the incomplete cycle detection algorithm also finds cycles completely contained in another cycle. Potential symmetric arrangements formed by the corresponding leaf-parts are detected when considering these cycles instead.

10.1.2. Algorithm Description

Using these ideas, we get the algorithm in Algorithm 7 for detecting Type I symmetric arrangements. It is carefully designed to avoid combinatorial explosion of cases when considering symmetric leaf-parts, and to detect leaf-parts that may have symmetric arrangements corresponding to sub-cycles of a centroid cycle. We consider each centroid cycle $(\mathcal{B}, E_\mathcal{B}) \in \mathcal{I}$ in turn (Line 01). First the points of the leaf-parts involved in $\mathcal{B}$ are extracted as P_l^k: the l-th points of the k-th leaf-part in cycle $\mathcal{B}$ such that P_l^k is mapped onto P_l^j by the congruency mapping in Γ_C for leaf-part j (Line 02); see Fig. 8. Then we

Algorithm $\mathcal{A} \leftarrow \textsc{TypeI}(C, \Gamma_C, E_C, \mathcal{G}, \mathcal{I})$

Input: C: congruent leaf-parts with N points each
Γ_c: congruency mappings
E_C: congruency validity interval
$\mathcal{G}$: global symmetries
$\mathcal{I}$: incomplete cycles of the centroids of C

Output: $\mathcal{A} = \{(A_k, \Xi_{A_k}, E_{A_k})\}$: Type I symmetric arrangements of leaf-part sets A_k given by mappings Ξ_{A_k} at validity interval E_{A_k}

01 **for each** $(\mathcal{B}, E_B) \in \mathcal{I}$
02 $\{P_l^k\} \leftarrow \textsc{Points}(\mathcal{B}, C)$ // *P_l^k: l-th point of k-th part*
03 $W \leftarrow$**empty**, $\mathcal{A} \leftarrow$**empty**
04 $\mathcal{Q} \leftarrow ((1,1), \ldots, (1,N))$ // *P_1^k indices of 1st leaf-part*
05 **while** $(k,l) \leftarrow$ **pop**(Q)
06 $\mathcal{D}_l^k \leftarrow \textsc{LocationCycles}(P_l^k, \mathcal{B}, C, \Gamma_C, E_C, \mathcal{G})$
07 $W \leftarrow$ **append**$(W, \mathcal{D}_l^k)$
08 $Q \leftarrow$ **append**$(\mathcal{Q}, (k + |\mathcal{D}_l^k| - 1, l))$
09 $\mathcal{I} = \{R^m\} \leftarrow \textsc{Intervals}\big(\bigcap_l^* \big(\bigcup_k^* \big([k, k + |\mathcal{D}_l^k| - 1], l\big)\big)\big)$
10 **for each** $R^m = [a,b] \in \mathcal{I}$
11 **for each** $l \in \{1, \ldots, N\}$
12 $F_l^m \leftarrow (P_l^a, \ldots, P_l^b)$
13 $F_0^m \leftarrow \textsc{CentroidSubCycle}(\mathcal{B}, R^m)$
14 $E_m \leftarrow E\,(\bigcup_l F_l^m) \cap E_C$
15 **if not empty**(E_m)
16 $\mathcal{A} \leftarrow$ **append**$(\mathcal{A}, (\textsc{LeafParts}(C, \mathcal{B}, R^m), \textsc{Map}(F_l^m), E_m))$

Algorithm 7: Detecting Type I symmetric arrangements of congruent leaf-parts

detect the cycle D_l^k starting from some P_l^k which matches a sub-cycle of the centroid cycle $\mathcal{B}$ (Lines 03–08). As a Type I symmetric arrangement must have compatible location cycles that have the same number of points, we find the consecutive leaf-parts that are mapped onto each other by combining the detected cycles D_l^k (Line 09). For each set of these sub-indices, we check the compatibility of their corresponding location cycles to detect symmetric arrangements (Lines 10–16).

We now explain how to find the set W of the D_l^k location cycles from the centroid cycle $\mathcal{B}$ (Lines 03–08). To detect potential maximal location cycles corresponding to sub-cycles of $\mathcal{B}$, we aim to expand each point of

the leaf-parts to a location cycle without checking sub-cycles of already detected cycles. A queue $\mathcal{Q}$ is used to store the points for expansion. It is initialised with the indices of the characteristic points $P_1^1, \ldots, P_l^1$ of the first leaf-part L_1 in $\mathcal{B}$ (Line 04). A maximal location cycle $\mathcal{D}_l^k$ for each point P_l^k in Q is found (Line 06) using a LocationCycles algorithm which is similar to IncompleteCycles (Algorithm 6). However, it involves fewer computations as the point sequence is already known from $\mathcal{B}$. Hence, for a starting point P_l^k, the r-th point only comes from $\gamma_{k+r-1}(g(P_l^1))$ for all global symmetries $g \in \mathcal{G}$. Thus, only a few characteristic points from leaf-part L_{k+r-1} have to be considered for expansion, based on the centroid cycle and the leaf-part's global symmetries: Lines 02, 09 of Algorithm 6 must be modified for LocationCycles. Specifically, in Line 02, we only consider $(P_1, P_2, P_3) = (P_l^k, \gamma_{k+1}(g(P_l^1))), \gamma_{k+2}(g(P_l^1))$, and in Line 09, we replace $\mathcal{P} \backslash \mathcal{C}$ by $\{\gamma_{k+r-1}(g(P_l^1))\}$ for the r-th expansion point. Unlike IncompleteCycles, only one cycle can be found in this expansion step for each P_l^k by further requiring their consistency with $\mathcal{B}$. The detected cycle $\mathcal{D}_l^k$ is stored in W (Line 07). If there is a point at which the detected location cycle cannot be expanded further (i.e. $\mathcal{D}_l^k$ is a sub-cycle not expanded to the end of the $\mathcal{B}$ cycle), this point may be a starting point for another location cycle, so its index is added to Q for further processing (Line 08).

We find the location cycles by expanding characteristic points of the leaf-parts in sequence of the centroid cycle, to find sub-cycles of consecutive indices. Note that the D_l^k cycles are essentially intervals $[k, k + |D_l^k| - 1]$ of leaf-part indices k in $\mathcal{B}$ for each leaf-part point index l. Each location cycle must have the same number of points for a symmetric arrangement, so we find the common maximal consecutive index intervals R^m of the leaf-part indices k for different locations l in the location cycles D_l^k (Line 09). See also Fig. 8: the centroid cycle induces an order on the leaf-parts L_k, associated with their characteristic points P_l^k. Location cycles, indicated in the figure by solid lines between the P_l^k, describe the matching between the P_l^k induced by centroid cycle. In order for the leaf-pars $L_a, \ldots, L_{a+c}$ (for some leaf-part-index a and integer c) to form a symmetric arrangement, there has to be a location cycle for each characteristic point P_a^k of the first leaf-part that spans up to at least leaf-part L_{a+c}. In Fig. 8, only the leaf-parts L_a to L_{a+c} with characteristic points inside the dashed box form a symmetric arrangement. Such leaf-part index intervals R^m can be found by taking the intersection of the union of the location cycle intervals. However, note that only one location cycle D_l^k per starting point index l may be used to find a single R^m

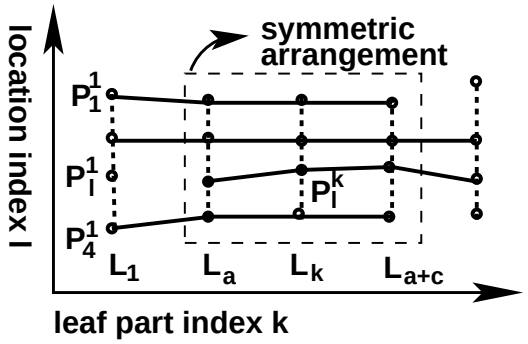


Figure 8: Finding symmetric arrangements by intersecting location cycles

interval, otherwise incompatible location cycles would be combined by the union operation. Hence, in Line 09 the intersection of the union of the D_l^k intervals only intersects intervals with different l indices (Fig. 8).

For each such leaf-part index interval $R^m = [a, b]$ (Line 10) we then find the corresponding location cycles F_l^m for characteristic points with index l from leaf-parts $a, \ldots, b$ (Lines 11–12). We also add the centroid sub-cycle corresponding to R^m (Line 13) to check the compatibility of the cycles F_l^m and the corresponding centroid sub-cycle, over all characteristic point indices l, by computing their validity interval (Line 14). If the validity interval is not empty, a symmetric arrangement of the corresponding leaf-part set exists, which, with associated tolerance and point mappings from the cycles $\mathcal{F}_l^m$, is output in $\mathcal{A}$ (Line 16).

This algorithm takes $O(LN^2)$ time for each cycle $\mathcal{B} \in \mathcal{I}$, where L is the number of leaf-parts and N is the number of characteristic points per leaf-part. We detect all potential location cycles $\{\mathcal{D}_l^k\}$ (Lines 05–08). During expansion, in the worst case, each point of the leaf-parts in $\mathcal{B}$ has to be considered (there are at most LN). For each such point we have $|\mathcal{G}|$ possibilities, where $|\mathcal{G}|$ is the number of global symmetries of the congruence set. Thus we have $LN|\mathcal{G}|$, and as $|\mathcal{G}| \leq \max(2N, 120)$ [28], the time taken is $O(LN^2)$.

10.2. Detecting Type II Symmetric Arrangements

Determining symmetric arrangements of Type II is performed in a similar way to detecting those of Type I. The difference is that the location cycles and $\mathcal{B}$ are compatible for Type I while they are linked by a translation for Type II. This requires different computations for the validity intervals of the location cycles in Line 14, Algorithm 7. Determining location cycle relations

for Type II is equivalent to requiring corresponding inter-point distances of different location cycles to be in the same distance equivalence class: e.g., in Fig. 4(b), $\|P_k - P_{(k+1) \bmod 4}\|$, $\|Q_k - Q_{(k+1) \bmod 4}\|$, $k = 1, 2, 3, 4$, should all be in the same distance classes.

Validity intervals for Type II are computed as follows. Let $\{F_l^m\}$ be the compatible location cycles detected by the algorithm in Algorithm 7. As in Eq. (2), let $\mathcal{H}_l^m$ be the distance classes for F_l^m, and let $D_{\min,l}^m = \min(\mathcal{H}_l^m)$, $D_{\max,l}^m = \max(\mathcal{H}_l^m)$ be the minimum and maximum distances for each cycle. For Type II, for the same index m (giving a set of compatible location cycles), the distances in $\mathcal{H}_l^m$ for different location cycles (i.e. different l) lie in the same distance class. Hence, for the union $F^m = \cup_l F_l^m$ of compatible cycles, we have $D_{\min}^m(F^m) = \max_l(D_{\min,l}^m)$ and $D_{\max}^m(F^m) = \min_l(D_{\max,l}^m)$ as the largest smallest and the smallest largest distance. From these, using Eq. (2), we can then find the validity interval $[E_{\min}(F^m), E_{\max}(F^m))$.

11. Merging Compatible Cycles

The symmetry and symmetric arrangement detection algorithms consider symmetry cycles only. The last step of the DesignIntent algorithm (Algorithm 1) merges compatible cycles which can be represented by a single isometry. To do so, we first must detect the possible symmetry types for each cycle (it may have more than one symmetry type as it is approximate and may be incomplete). In the second step we cluster cycles of the same symmetry type to find the incomplete symmetries.

Two-cycles are trivially assigned to the groups for mirror, inversion and $\mathbf{C_2}$ symmetry. Limited space precludes an exhaustive list of all other cases, and we just outline the basic ideas. Instead of trying to find the optimal isometry for a cycle [6], we determine symmetry types by considering the relative locations of the centres of the circles formed by each consecutive triple of points in $\mathcal{C}$. In 2D, for $\mathbf{C_n}$ these centres lie on the same side of all edges joining consecutive points, whereas they lie alternately on opposite sides for successive triples for $\mathbf{Z}$. If they do not consistently fit either pattern, we assign $\mathbf{C_n}$ and $\mathbf{Z}$ as possible symmetry types. For a given symmetry type, e.g. $\mathbf{C_n}$, determining the valid values of $\mathbf{n}$ for an incomplete cycle is done by first computing the least squares fitting circle to $\mathcal{C}$, and then computing the angles between successive points. Each angle is divided by 2π and the nearest integer above and below are included in a list. The overall minimum and maximum values in the list give the permissible range for $\mathbf{n}$. If $\mathbf{n}$ is

larger than a user-specified maximal rotation order believed to be present in the object, we also select translational symmetry. For a *complete* cycle with $\mathbf{C_n}$ symmetry, $\mathbf{n}$ is uniquely determined. Allowing the same points to have more than one possible symmetry type is necessary, as it is important not to rule out any possible symmetry type too soon, losing information. Cycle clustering helps to rule out inappropriate symmetries.

Incomplete symmetries are found by clustering cycles, sharing the same symmetry type, induced by the same isometry at a compatible tolerance level. This is done using the consistent clustering algorithm (Algorithm 2) where now the I_k are cycles with associated validity intervals and E^* is the overall bound from RFT selection. Within the algorithm, we need to compute $E_{\min}(\mathcal{C}_1 \cup \mathcal{C}_2)$ for two cycles $\mathcal{C}_1, \mathcal{C}_2$. Let $\mathcal{C}_1 = \{P_1, \ldots, P_{|c_1|}\}$, $\mathcal{C}_2 = \{Q_1, \ldots, Q_{|c_2|}\}$ be two cycles of the same symmetry type and let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Taking a concrete case, suppose that the symmetry type is $\mathbf{C_n}$ in 2D and both cycles have the same orientation; other symmetry types can be handled similarly. By definition, $E_{\min}(\mathcal{C})$ is the minimal tolerance such that $\|P - Q\| =_\epsilon \|\mu(P) - \mu(Q)\|$ for all P, Q in $\mathcal{C}$. This is already satisfied for $P, Q \in \mathcal{C}_1$ at $E_{\min}(\mathcal{C}_1)$ and $P, Q \in \mathcal{C}_2$ at $E_{\min}(\mathcal{C}_2)$. We consider all pairs P, Q where $P \in \mathcal{C}_1$ and $Q \in \mathcal{C}_2$, and then allocate distances between such pairs to distance equivalence classes as in Section 9. We then find the maximal difference between two such distances in each class, and finally take the maximum e^* of these differences over all classes. Consequently, $E_{\min}(\mathcal{C}) = \max(e^*, E_{\min}(\mathcal{C}_1), E_{\min}(\mathcal{C}_2))$. $E_{\max}(\mathcal{C})$ is computed similarly.

12. Experiments

We have tested our algorithm on various B-rep models. Approximate models were generated from them by perturbing each face (see below). Regularities detected by our algorithm in these approximate models were compared with regularities present in the exact model indicated by a human. We ran our algorithm on a 3.4GHz Pentium 4 processor with 1GB RAM. RFT construction and congruence detection were implemented in C++ using OpenCASCADE; other parts were implemented in Matlab.

To generate an approximate model from a given model, we perturbed the unit normal direction of each planar face and the axis direction of each cylinder by adding a random vector in $[-L, L]^3$, and then translating it by a random vector in $[-sL, sL]^3$, where L indicates the noise level (by default set to 0.1) and s is the area of the face or the radius of the cylinder. Radii

Model	V	F	L	T_T	T_C	T_R	R_D	R_S
Angle-Bracket: Fig. 9	124	67	27	0.73s	0.04s	3.55s	66	13
Monster: Fig. 1	438	198	60	81.62s	17.98s	28.34s	64	5
ANC: Fig. 15(a)	236	124	46	13.34s	0.35s	6.82	31	7

Table 1: Results for example models; V: number of vertices; F: number of faces; L: number of detected leaf-parts; T_D: RFT decomposition time; T_C: congruence detection time; T_R: regularity detection time; R_D: number of detected regularities; R_S: number of spurious regularities.

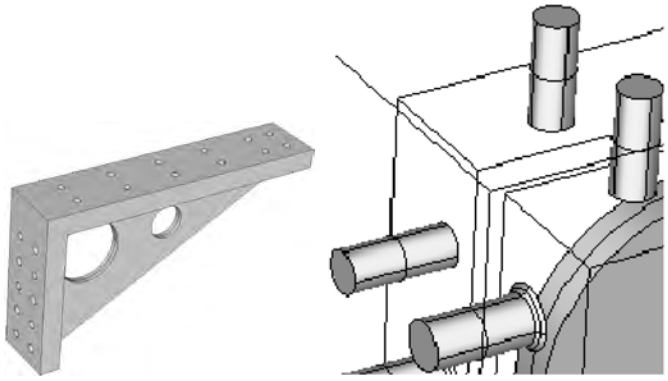
of cylinders were also perturbed by a random value in $[-sL, sL]$. The resulting perturbed faces were intersected to produce new approximate edges and vertices based on the original model's topology. We refer to such models in the section simply as approximate models at noise level $L = 0.1$; other noise levels were also considered as described below. Perturbing models like this gives us exact control of the level of noise present and with that enable comparison with other results. As our algorithms work on relatively high level CAD models and not measured point clouds, using real scanner data is less important. Also note that typically in real noise levels are lower than those considered here.

E.g., for the Monster in Fig. 1, where minimal and maximal distances between points of the original model were 7 and 385 units respectively, the maximum distance between an original and perturbed vertex was 2 units using this method.

Our algorithm outputs design intent as a congruency hierarchy with associated symmetries and symmetric arrangements at each level, at various tolerance levels: certain output regularities may be complete subsets of other regularities. It is hard to display such structures, and often in our experimental results the intended regularities are described by the top-level regularities. Hence, we only show top-level regularities here. Output and timing results are summarised in Table 1 for each model.

12.1. Angle-Bracket Example

We first consider the Angle-Bracket model in Fig. 9. The exact model has a dominant mirror symmetry which is broken on the left side (see the close-up in Fig. 9(b)): the 10 cylindrical holes come in two groups of 5 identical cylinders. Cylinders in one group are blended while the others are not, and the blended cylinders have larger radii than the unblended ones. Moreover,



(a) Complete model (b) Close-up view of holes

Figure 9: Angle-Bracket model

a $\mathbf{C_4}$ and a $\mathbf{T}$ symmetric arrangement are exhibited by cylindrical holes in the top. On the left, two $\mathbf{Z}$ symmetric arrangements are formed by the five cylindrical holes with and without blends respectively. Minimal and maximal inter-point distances between the model vertices are 2.8 and 520.7.

The RFT at validity interval $[0.0, 2.8)$ units shown in Fig. 10 consists of 27 leaf-parts, clustered into seven maximal congruence sets: B_1, B_2, B_3, D_1, D_2, $\{H_1, \ldots, H_{12}, V_1, \ldots, V_5\}$, $\{T_1, \ldots, T_5\}$. The regularities found are listed in Table 2: 72 symmetries were found, 13 of which are spurious (i.e. not exactly present in the exact model), while all the regularities determined as present in the exact model were detected. In detail, for each symmetry type we give the number of regularities found, and how many of these are spurious. We also list the detected regularities as merged cycles of leaf-parts, e.g. the dominant mirror symmetry is listed as $\{(B_1), (B_2), (B_3), (D_1), (D_2), (H_1, H_2), \ldots, (H_{11}, H_{12})\}$, indicating that its cycles consist of pairs of cylinders (H_1, H_2), etc. and the mirror is also present in individual leaf-parts such as (B_1). Similarly for the $\mathbf{Z}$ regularity on the left side we have $\{(V_1, V_2, \ldots, V_5), (T_1, T_2, \ldots, T_5)\}$ as the two leaf-part cycles; this regularity and the location cycles for $\{(V_1, V_2, \ldots, V_5), (T_1, T_2, \ldots, T_5)\}$ are shown in Fig. 11. Note that here for clarity, we ignore $\mathbf{M}$ and $\mathbf{C_2}$ regularities consisting of only a single cycle.

All intended regularities have been found, as well as a few unexpected and spurious symmetries, some of which are illustrated in Fig. 12; the others are similar. The former are mainly $\mathbf{C_2}$ symmetries between various leaf-

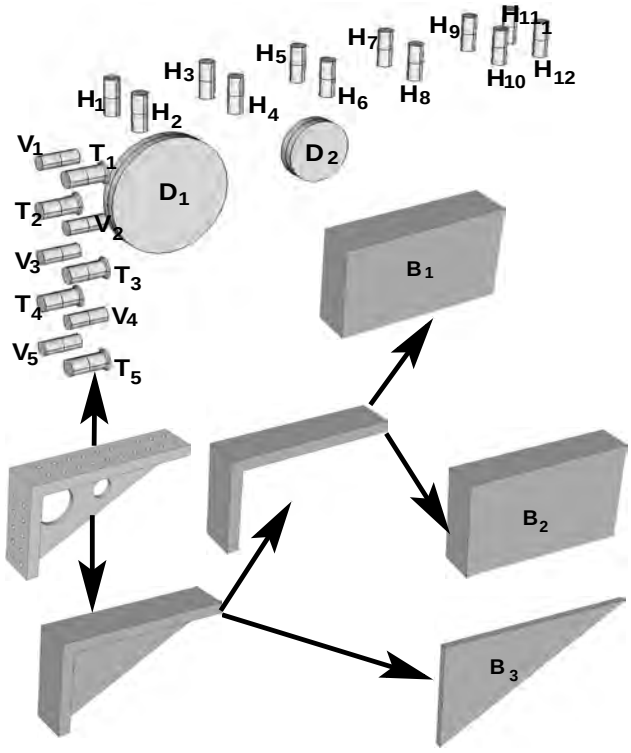


Figure 10: RFT decomposition of the Angle Bracket model in Fig. 9

parts $H_1, \ldots, H_{12}$ and $V_1, \ldots, V_5$. Such symmetries may be present in the exact model, but their presence may well be unintentional. The spurious symmetries are generally $\mathbf{C_n}, \mathbf{n} > 2$ symmetries (see the last three examples in the second row of Fig. 12). These clearly do not form an exact symmetry in the exact model and are inconsistent with the translational arrangement. They are detected because when considered as incomplete symmetries, they are within a valid tolerance.

To test the algorithm's robustness to increasing noise levels, we also tested it with the Angle-Bracket model with noise levels of $0.0, 0.1, \ldots, 0.3$ units. The number of detected symmetries may increase or decrease as the noise increases. However, only on reaching 0.3 units of noise were certain intended symmetries not found; the particular symmetries lost varied on repeating the experiment. However, this is quite a high noise level, and we assert that our algorithm is thus robust in presence of moderate noise, as hoped. In particular, expected symmetries are found at the levels of noise beyond

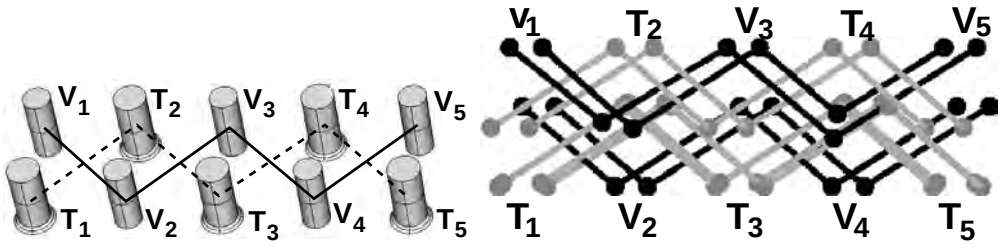


Figure 11: **Z** regularity and location cycles for $\{(V_1, V_2, \ldots, V_5), (T_1, T_2, \ldots, T_5)\}$ detected in the Angle Bracket after RFT construction

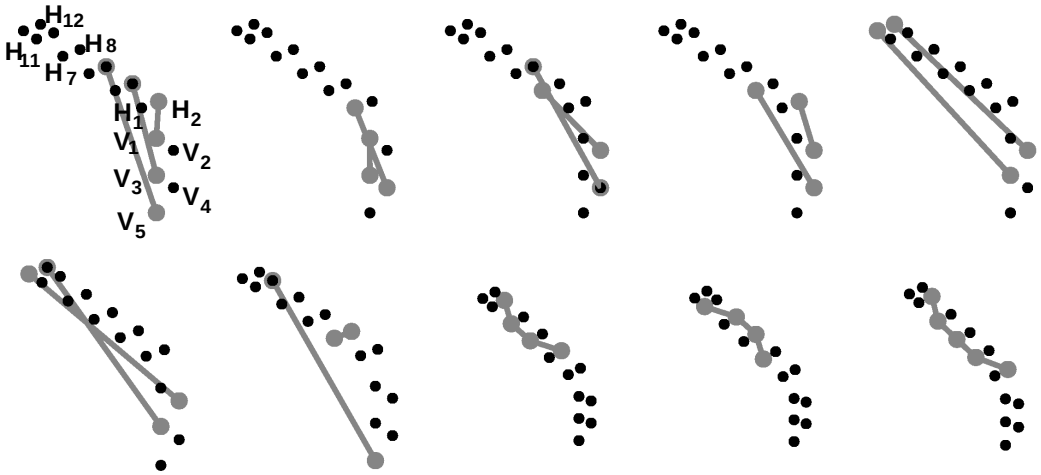


Figure 12: Unexpected and spurious symmetries in the Angle Bracket model. Each black point represents one cylinder in $H_1, \ldots, H_{12}$ or $V_1, \ldots, V_5$

those which might be expected in CAD data exchange and reverse engineering applications (less than 0.1 units).

From this example one disadvantage of our algorithm becomes apparent: as the cylinders at the left side do not all have the same radius and some have blends, their regular translational arrangement is not detected. Even if the decomposition method could identify and remove the blends, the cylinders would still have different radii and hence not be part of the same congruence set for symmetric arrangement detection. Strictly, we do not look for such regularities, but it is likely to be important for this model to detect this. It would require a change in the model decomposition approach, but also more importantly our notion of symmetric arrangements would have to be generalised to non-congruent shapes. One could consider all point combinations

Type	Found	Spurious	Detected Regularities
M	21	0	Dominant mirror (1): $\{(B_1),(B_2),(B_3),(D_1),(D_2),(H_1,H_2),\ldots,(H_{11},H_{12})\}$. Mirrors between top holes (12): $\{(H_1,H_3),\ (H_2,H_4)\}$, $\{(H_1,H_5),\ (H_3),\ (H_2,H_6),\ (H_4)\}$, $\{(H_1,H_7),\ (H_3,\ H_5),\ (H_2,H_8),\ (H_4,H_6)\}$, $\{(H_1,H_9),\ (H_3,H_7),\ (H_5),\ (H_2,H_{10}),\ (H_4,H_8),\ (H_6)\}$, $\{(H_3,H_9),\ (H_5,H_7),\ (H_4,H_{10}),\ (H_6,H_8)\}$, $\{(H_5,H_9),\ (H_7),\ (H_6,H_{10}),\ (H_8)\}$, $\{(H_7,H_{11}),\ (H_8,H_{12})\}$, $\{(H_5,H_{11}),\ (H_6,H_{12})\}$, $\{(H_3,H_{11}),\ (H_4,H_{12})\}$, $\{(H_1,H_{11}),\ (H_2,H_{12})\}$, $\{(H_9,H_{10}),\ (H_{11},H_{12})\}$, $\{(H_9,H_{11}),\ (H_{10},H_{12})\}$. Mirrors on left side (4): $\{(V_1,V_5),\ (V_3),\ (V_2,V_4),\ (T_1,T_5),\ (T_3),\ (T_2,T_4),\ (B_2)\}$, $\{(V_1,V_3),\ (V_2),\ (T_1,T_3),\ (T_2)\}$, $\{(V_2,V_4),\ (T_2,T_4),\ (V_3),\ (T_3)\}$, $\{(V_3,V_5),\ (V_4),\ (T_3,T_5),\ (T_4)\}$. Mirror between left and top sides (1): $\{(V_1,H_1),\ (V_3,H_3),\ (V_5,H_5)\}$. Additional mirrors in rectangular blocks (3): two in $\{(B_1)\}$, $\{(B_2)\}$.
$\mathbf{C_2}$	35	7	All mirrors on top with rotation centre their centroid (12). Additional $\mathbf{C_2}$ on top (6): $\{(H_2,H_8),\ (H_4,H_6)\}$, $\{(H_1,H_7),\ (H_3,H_5)\}$, $\{(H_4,H_{10}),\ (H_6,H_8)\}$, $\{(H_3,H_9),\ (H_5,H_7)\}$, $\{(H_2,H_{10}),\ (H_4,H_8),\ (H_6)\}$, $\{(H_1,H_9),\ (H_3,H_7),\ (H_5)\}$. $\mathbf{C_2}$ in blocks: three in $\{(B_1)\}$, three in $\{(B_2)\}$. Left side (10): $\{(V_1,V_4),\ (V_2,V_3)\}$, $\{(V_2,V_5),\ (V_3,V_4)\}$, $\{(T_1,T_4),\ (T_2,T_3)\}$, $\{(T_2,T_5),\ (T_3,T_4)\}$, $\{(V_1,V_2),\ (T_1,T_2)\}$, $\{(V_2,V_3),\ (T_2,T_3)\}$, $\{(V_3,V_4),\ (T_3,T_4)\}$, $\{(V_4,V_5),\ (T_4,T_5)\}$, $\{(V_1,V_5),\ (V_3),\ (T_2,T_4)\}$, $\{(T_1,T_5),\ (T_3),(V_2,V_4)\}$. Spurious (7): $\{(V_1,H_2),\ (V_3,H_4),\ (V_5,H_6)\}$, $\{(V_1,V_3),\ (V_4,H_1)\}$, $\{(V_2,H_3),\ (V_4,H_5)\}$, $\{(V_2,H_2),\ (V_4,H_3)\}$, $\{(V_2,H_{12}),\ (V_3,H_{11})\}$, $\{(V_2,H_{11}),\ (V_3,H_{12})\}$, $\{(V_5,H_{11}),\ (H_3,H_4)\}$.
T	1	0	Top holes: $\{(H_1,H_3,H_5,H_7,H_9),\ (H_2,H_4,H_6,H_8,H_{10})\}$.
Z	2	0	Side holes: $\{(H_1,H_4,H_5,H_8,H_9),\ (H_2,H_3,H_6,H_7,H_{10})\}$, $\{(V_1,V_2,\ldots,V_5),(T_1,T_2,\ldots,T_5)\}$.
$\mathbf{C_4}$	1	0	Right-most top holes: $\{(H_9,H_{10},H_{12},H_{11})\}$.
$\mathbf{C_8}$	4	4	Symmetric arrangements of top holes: $\{(H_1,H_4,H_6,H_7)\}$, $\{(H_2,H_3,H_5,H_8)\}$, $\{(H_3,H_6,H_8,H_9)\}$, $\{(H_4,H_5,\ H_7,H_{10})\}$.
$\mathbf{C_{12}}$	2	2	Symmetric arrangements of top holes: $\{(H_1,H_4,H_6,H_8,H_9)\}$, $\{(H_2,H_3,H_5,H_7,H_{10})\}$.

Table 2: Regularities detected in the approximate Angle-Bracket model

or even all decompositions, but this would clearly be too computationally expensive.

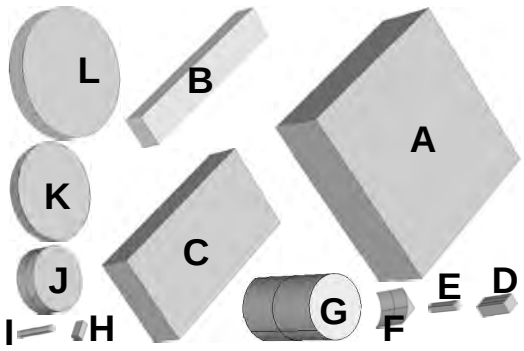


Figure 13: An exemplar leaf-part for each congruence set of the Monster model

12.2. Monster Example

Intermediate results of running our algorithm on the Monster model (Fig. 1) are shown in Fig. 2. The exact model has a major $\mathbf{C_4}$ symmetry broken by sets of congruent blocks on two sides; these together form a translational arrangement. $\mathbf{C_8}$ and $\mathbf{C_{16}}$ symmetries are present in the centre of the model formed by the small cylinders and the cylindrical slots respectively. Detailed results of running our algorithm on an approximate Monster model at noise level 0.1 are given in Table 3; we use $[X]$ to refer to the whole set of congruent leaf-parts represented by some exemplar X in the model. An exemplar from each congruence set is shown in Fig. 13. All together, 59 symmetries were found, only 5 of which are spurious. All the symmetries expected from the exact model were detected.

We do not list all 12 $\mathbf{M}$ and 8 $\mathbf{C_2}$ regularities formed by translational symmetric arrangements of side blocks E, D; they are similar in nature to those listed for the Angle-Bracket example. Various regularities are formed by combinations of different kinds of leaf-parts arranged with the same symmetry: Fig. 14 shows one example for a $\mathbf{C_4}$ regularity. The 5 spurious symmetries are of two types. The first type comprises alternative interpretations. E.g. E, D were interpreted both as $\mathbf{M}, \mathbf{C_2}$ and as $\mathbf{C_4}$, but only the former two are intended, and the other interpretation is spurious. Other spurious symmetries do not exist: the $\mathbf{M}$ and $\mathbf{C_2}$ symmetry found for H are invalid as the faces of H come from the cylinders K, L with different radii, but they were found to be approximately equal.

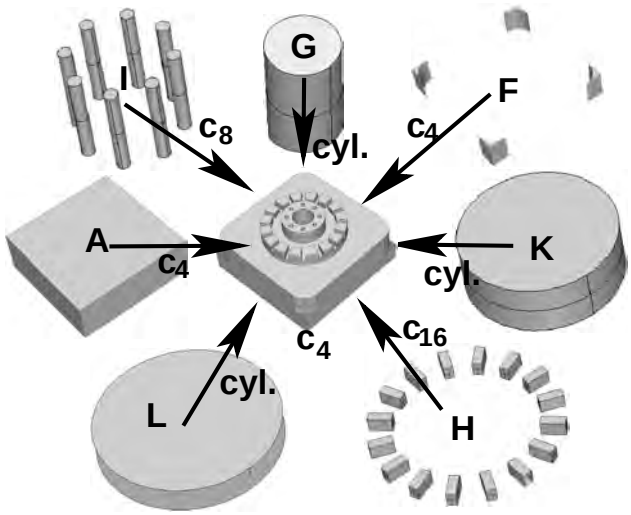


Figure 14: $\mathbf{C_4}$ regularity linking rotational regularities in the Monster model

12.3. ANC Example

We also tested our algorithm on the ANC model shown in Fig. 15. The regularities are simpler to detect in this example due to the simplicity and small number of leaf-parts: we only describe the results briefly. Fig. 15(c) shows detected cylindrical leaf-parts, and Fig. 15(d)-(f) give the major symmetries found out of the 31 regularities found. 6 regularities were spurious, coming from the mirror and $\mathbf{C_2}$ symmetries of regularities shown in Fig. 15(f). All symmetries present in the exact model were detected. Note that the part in Fig. 15(d) is not the original model, but comprises those sub-parts sharing the dominant mirror symmetry. The $\mathbf{C_{12}}$ symmetry of the cylindrical holes in the top face, with three cylinders missing, was also detected.

To demonstrate the utility of hierarchically detecting symmetries, we show in Fig. 16 the result of detecting symmetric arrangements of the nine congruent cylinders in the top face, where for simplicity we only show the centroids. From left to right these leaf-parts were generated with noise levels of $1/10, 1/5, 1/2$ units to indicate the effect of increasing noise on the cycles detected. On the left, besides the maximal symmetric arrangement $\mathbf{C_{12}}$, two $\mathbf{C_6}$ and one $\mathbf{C_3}$ symmetries were found. The results in the middle are similar, and all intended symmetries were detected. The results on the right are different. Two $\mathbf{C_{12}}$ symmetries and one $\mathbf{C_3}$ symmetry were detected. The larger noise in this case has obscured the design intent. Generally it is useful

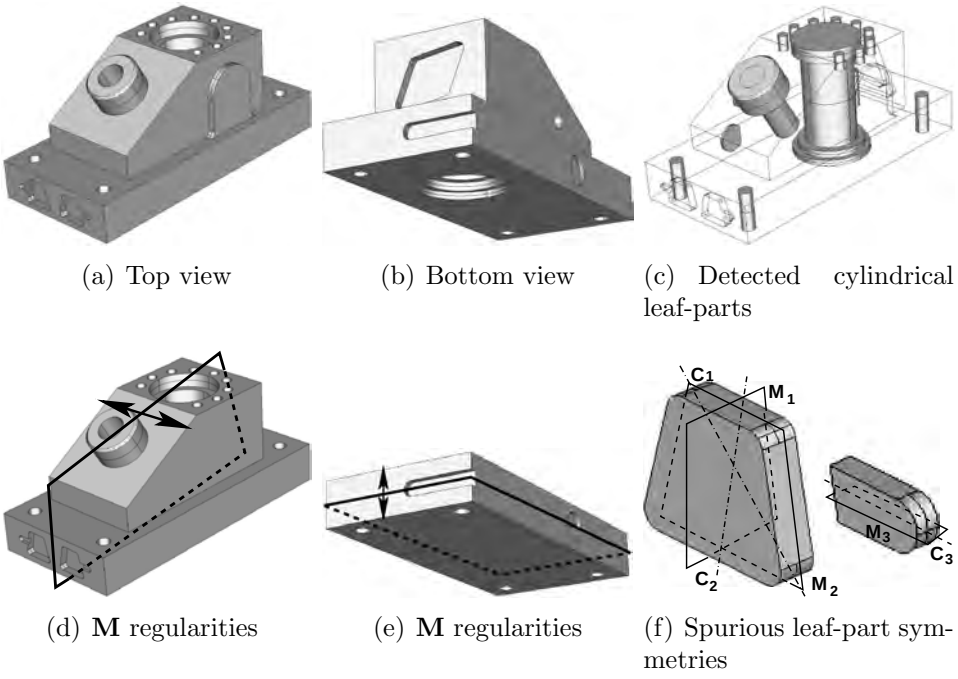
Type	Detected	Spurious	Detected Regularities
M	34	1	Mirror parallel to two edges (2): two involving $\{A, [B], C, [D], [E], [F], [H], [I], G, J, K, L\}$. Biangular direction mirror (2): two involving $\{A, [F], [H], [I], G, J, K, L\}$. Others related induced by $\mathbf{C_8}$ rotation (4): four involving $\{[I], [H], G, J, K, L\}$. Others related induced by $\mathbf{C_{16}}$ rotation (8): eight involving $\{[H], G, J, K, L\}$. Mirror between top and bottom (1): one involving $\{[D], [E], [F]\}$. Various arrangements between side slots (9): nine involving subsets of D_l and E_l. Additional mirrors in congruence sets (8): C, A, two of H, two of B, two of F, of which one of H is spurious.
$\mathbf{C_2}$	23	2	Various arrangements of side slots (8): eight involving subsets of D_l and E_l. Rotation of complete model (1): one of $\{A, [B], C, [D], [E], [F], G, [H], [I], J, K, L\}$. Additional rotations in congruence sets (14): two in A, three in B, two in C, three in D, three in E, one in F, two in H, of which two in H are spurious.
$\mathbf{C_4}$	3	2	Symmetry induced by the bottom block A: $\{A, [F], [I], [H], G, J, K, L\}$; see also Fig. 14. Rotations in congruence sets: D, E, both are spurious.
$\mathbf{C_8}$	1	0	Symmetry induced by the holes $[I]$: $\{[I], [H], G, J, K, L\}$.
$\mathbf{C_{16}}$	1	0	Symmetry induced by small blocks $[H]$: $\{[H], G, J, K, L\}$.
T	1	0	Symmetric arrangements of side blocks on both sides: $\{[D], [E]\}$.
Z	1	0	Symmetric arrangements of side blocks induced by their translational symmetry: $\{[D], [E]\}$.

Table 3: Regularities detected in the approximate Monster model

for such symmetries to be detected hierarchically, to assist a post-processing step to decide which one was indeed intended by the designer.

12.4. Discussion

Our experiments show that overall most regularities detected by our algorithm are those actually intended in the models. We attribute this to our robust concept of approximate regularity coupled with the use of RFT decomposition. Spurious regularities arising from various sources are few, at realistic noise levels. In all examples, no regularities indicated by a human



(a) Top view (b) Bottom view (c) Detected cylindrical leaf-parts

(d) **M** regularities (e) **M** regularities (f) Spurious leaf-part symmetries

Figure 15: Some regularities detected in the ANC model

as significant were missed. We thus claim that our approach provides a good balance between (i) detecting all regularities, potentially including many spurious regularities, and minor, exact but irrelevant regularities implied by the major regularities, and (ii) detecting only a few top level regularities, potentially missing many desired regularities.

Our regularities are based on incomplete cycles determined by unambiguously matching point distances. However, due to this unambiguity condition, the presence of noisy points or cycles may disrupt a generally well-defined regularity if no unambiguous match is possible. Fig. 17 gives a simple 2D example, where the twelve points in the dashed box form a translational symmetry T and the other three points P_1, P_2, P_3 disrupt this symmetry. Using our point expansion process, three cycles C_1, C_2, C_3 of five points will be found, respectively containing P_1, P_2, P_3. However, these three cycles are not compatible with each other due to the big difference between distances d_1 and d_3, even though a smaller tolerance still allows three cycles to be found. Consequently, symmetry including all three cycles will not be detected. However, we have not seen such cases in practice: noise levels do not typically

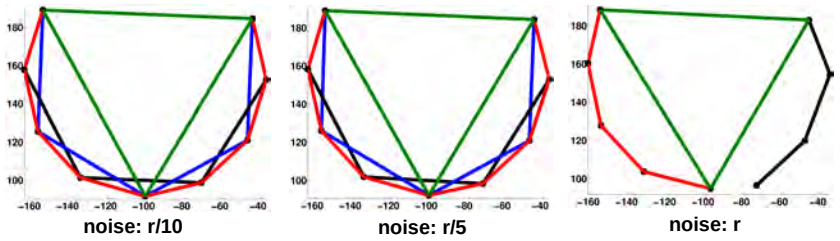


Figure 16: Incomplete symmetry hierarchies detected at different noise levels

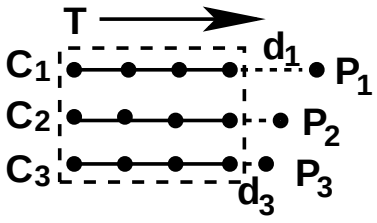
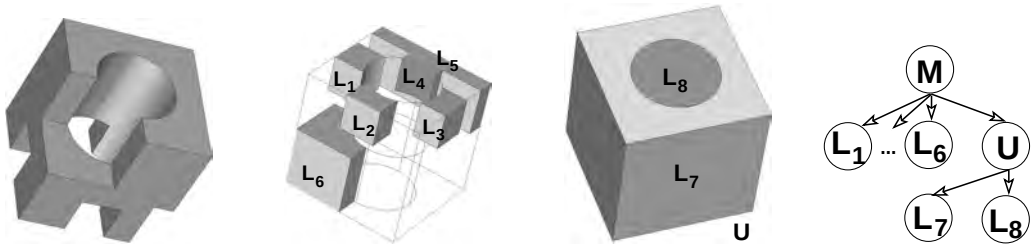


Figure 17: Potential symmetry may be missed by cycle merging

vary significantly over a single model.

Some spurious regularities are also caused by the fact that we seek incomplete cycles. An incomplete cycle is less reliable than a complete cycle due to the missing positions (and with them the inter-point distances) and its symmetry type is not always uniquely determined. Our symmetry definition actually limits the incomplete cases to be considered such that the overall number of incomplete cycles remains small, but includes those common in engineering objects.

A significant aspect of our algorithm is that using RFT decomposition allows regularity detection to work without checking *all* possible relations between *all* model entities. An algorithm considering and comparing all possible pieces of the object would certainly find considerably more spurious regularities (e.g. [19]). However, such decomposition already interprets the model in a certain way which might prevent the algorithm from finding certain regularities, if the decomposition is inappropriate. E.g., consider the four leaf-parts L_1 to L_4 created by the crossed slots of the model in Fig. 18. L_4 is not congruent to $L_1, \ldots, L_3$ which means an incomplete $\mathbf{C_4}$ symmetry containing three leaf-parts is found instead of four. A different decomposition would be more suitable in this case. Unfortunately, for every model there is in principle an infinite number of possible decompositions. The particular RFT



(a) Example model (b) Leaf-parts (c) Leaf-parts (d) RFT

Figure 18: A four-fold rotational symmetric arrangements was not detected due to the interpretation of RFT construction

decomposition approach taken here seems to be very successful in practice, but if desired, it could be replaced with some other feature-detection-based decomposition method.

Compared with the previous beautification approach in [19] for detecting *global* symmetries (a comparison with the local regularities in [19] is beyond the scope of this paper), our method can handle much more complex models, and results in many fewer spurious symmetries. Examples given here include models with up to 438 vertices and 198 faces, and result in at most 66 detected *incomplete, local* symmetries while the previous work only included highly symmetric models with up to 60 faces and 250 vertices and found 180 global symmetries, but no local symmetries. The initial decomposition used in the current method finds a suitable high-level sub-part structure. The potential cost may be loss of regularities made of different entities from different leaf-parts.

In summary, the experiments show that our algorithm for detecting design intent in approximate models extracts most intended regularities from the model and few spurious regularities. The method is very robust towards realistic levels of noise. Initial model decomposition requires a simple user choice of a suitable decomposition, which restricts which regularities can be detected. In principle this means that certain regularities may not be found, yet experiments show that this approach generally yields the intended symmetry structure as indicated by a human. As *approximate* symmetries are detected, unavoidably some spurious regularities are also found, but their number is small. The output is clearly suitable for modelling and analysis tasks, and may be useful for shape search and indexing.

13. Conclusions

This paper gives a novel solution to detecting design intent of approximate B-rep models using symmetry. It considers symmetries in a wide mathematical sense, including all the seven elementary types, which may be present approximately, locally, incompletely or compatibly. To achieve this goal, models are first hierarchically decomposed into simpler, more symmetric sub-parts. Design intent is then detected as congruencies, symmetries and symmetric arrangements of the leaf-parts in this decomposition. Doing so avoids finding many spurious regularities, reliably resolves ambiguities between regularities, and reduces inconsistencies. Yet the decomposition also limits what can be found in terms of which model types can be processed and how these are decomposed into simpler sub-parts. The detection algorithms are based on a single principle using distances between points for adaptive tolerance interval selection, and hence produce consistent, well-defined approximate regularities. The strict conditions may mean that certain regularities are not detected even if we have not observed this in general under typical noise levels. Experiments show that detected relations reveal significant design intent.

Despite such good results there is scope for future work to increase the range of regularities detected and the types of models that can be handled. We consider the following as particularly interesting directions:

(1) Our current methods are limited to CAD models bounded by planar, spherical, cylindrical, conical and toroidal surfaces and do not consider more general curved geometries. Extending our approach is currently hampered by the difficulty of extending the geometry of free-form surfaces required for constructing RFTs of more general models. Work on identifying DP-features on free-form solids [25] may help to extend this.

(2) Symmetry detection has attracted researchers from various fields, and much good work has been proposed, covering different topics, e.g. [8, 29, 39, 31]. However, this work is focused on the detection of "single" point symmetry, e.g. reflection, rotation or translation. This paper steps further considering their combinations as glide symmetry or rotation-reflection symmetry. More work is required to detect more general notions of symmetry, and symmetry groups formed by combining symmetry transformations, e.g. wallpaper groups and tetrahedral symmetry. Detecting such regularities would simplify the output regularities by representing them in terms of combined transformations instead of elementary symmetries. This could lead to

more efficient detection, and also help to better represent a model's intended structure.

(3) The design intent detection work covered in this paper as well as other previous work, e.g. [41, 19], aims to detect potential geometric regularities in the rebuilt approximate model for downstream processing. However, reproducing the original model for CAD/CAM systems and rebuilding a likely construction history should also be considered.

Acknowledgements

This work was supported by EPSRC UK Grant GR/S69085/01. The example models used in this paper were obtained from the National Design Repository, `http://www.designrepository.org/`.

References

[1] H. Alt, K. Mehlhorn, H. Wagener, E. Welzl. Congruence, similarity and symmetries of geometric objects. Disc. Comp. Geometry, 3(3):237–256, 1988.

[2] K. Barratt. Logic and Design in Art, Science and Mathematics, Herbert Press, London, 1989.

[3] P. Benko, G. Kos, T. Varady, L. Andor, R. Martin. Constraint fitting in reverse engineering. Computer-Aided Geometric Design, 19 (3):173–205, 2002.

[4] R. Bidarra, W. Bronsvoort. Semantic feature modelling. Computer-Aided Design, 32 (3):201–225, 2000.

[5] P. Brass. On finding maximum-cardinality symmetric subsets. Comp. Geometry, 24(1):19–25, 2003.

[6] D. Eggert, A. Lorusso, R. Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithm. Machine Vision and App., 9(5–6):272–290, 1997.

[7] C. Feng, A. Kusiak. Constraint-based design of parts. Computer-Aided Design, 27(5):343–352, 1995.

[8] P. Flynn. 3-D object recognition with symmetric models: symmetry extraction and encoding. IEEE Trans. Pattern Analysis and Machine Intelligence, 16(8):814–818, 1994.

[9] J. Gallian. Contemporary abstract algebra, D.C. Heath and Company, Lexington, Mass, 1986.

[10] C. Gao, F. Langbein, A. Marshall, R. Martin. Approximate congruence detection of model features for reverse engineering. In: Proc. Int. Conf. Shape Modeling and Applications, pp. 69–77, 2003.

[11] S. Gao, J. Shah. Automatic recognition of interacting machining features based on minimal condition subgraph. Computer-Aided Design, 30(9):727-739, 1998.

[12] X. Gao, Q. Lin, G. Zhang. A C-tree decomposition algorithm for 2D and 3D geometric constraint solving. Computer-Aided Design, 38 (1):1–13, 2006.

[13] 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.

[14] C. Hoffman, A. Lomonosov, M. Sitharam. Decomposition plans for geometric constraint problems, part II: new algorithms. J. Symb. Comp., 31(4):409–427, 2001.

[15] S. Iwanowski. Testing approximate symmetry in the plane is NP-hard. Theo. Comp. Sc., 80:227–262, 1991.

[16] K. Kanatani. Comments on 'symmetry as a continuous feature'. IEEE Trans. Pattern Analysis and Machine Intelligence, 19(2):246–247, 1997.

[17] Y. Ke, S. Fan, W. Zhu, A. Li, F. Liu, X. Shi. Feature-based reverse modeling strategies. Computer-Aided Design, 38(5):485–506, 2006.

[18] J. Kim, M. Pratt, R. Iyer, R. Sriram. Standardized data exchange of CAD models with design intent. Computer-Aided Design, 40(7):760–777, 2008.

[19] F. Langbein. Beautification of reverse engineered geometric models, Ph.D. thesis, Cardiff University, UK, 2003.

[20] F. Langbein, A. Marshall, R. Martin. Choosing consistent constraints for beautification of reverse engineered geometric models. Computer-Aided Design, 36(3):261–278, 2004.

[21] M. Leyton. A generative theory of shape, LNCS 2145, Springer, 2001.

[22] M. Li, F. Langbein, R. Martin. Constructing regularity feature trees for solid models. In: Proc. Geometric Modeling and Processing, LNCS, Springer, pp. 267–286, 2006.

[23] M. Li, F. Langbein, R. Martin. Detecting approximate incomplete symmetries in discrete point sets. In: Proc. ACM Symp. Solid and Physical Modeling, pp. 335-340, 2007.

[24] M. Li, F. Langbein, R. Martin. Detecting approximate symmetries of discrete point subsets. Computer-Aided Design, 40 (1):76–93, 2008.

[25] T. Lim, H. Medellin, C. Torres-Sanchez, J. Corney, J. Ritchie, J. Davies. Edge-based identification of DP-features on free-form solids. IEEE Trans. Pattern Analysis and Machine Intelligence, 27(6):851–860, 2005.

[26] S. Liu, S. Hu, Y. Chen, J. Sun. Reconstruction of curved solids from engineering drawings. Computer-Aided Design, 33(14):1059–1072, 2001.

[27] Y. Liu, H. Pottmann, W. Wang. Constrained 3D shape reconstruction using a combination of surface fitting and registration. Computer-Aided Design, 38(6):572–583, 2006.

[28] E. Lockwood, R. Macmillan. Geometric Symmetry, Cambridge University Press, 1978.

[29] B. Mills, F. Langbein, A. Marshall, R. Martin. Approximate symmetry detection for reverse engineering. In: Proc. 6th ACM Symp. Solid and Physical Modeling, pp. 241–248, 2001.

[30] 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.

[31] N. Mitra, L. Guibas, M. Pauly. Partial and approximate symmetry detection for 3D geometry. ACM Trans. Graph., 25(3):560–568, 2006.

[32] M. Pratt, B. Anderson, T. Ranger. Towards the standardized exchange of parameterized feature-based CAD models. Computer-Aided Design, 37(12):1251-1265, 2005.

[33] G. Robins, B. Robinson, B. Sethi. On detecting spatial regularity in noisy images. Inf. Proc. Letters, 69:189–195, 1999.

[34] H. Rom, G. Medioni. Hierarchical decomposition and axial shape description. IEEE Trans. Pattern Analysis and Machine Intelligence, 15(10):973–981, 1993.

[35] 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.

[36] M. Sitharam, J. Oung, Y. Zhou, A. Arbree. Geometric constraints within feature hierarchies. Computer-Aided Design, 38(1):22–38, 2006.

[37] P. Stefano, F. Bianconi, L. Angelo. An approach for feature semantics recognition in geometric models. Computer-Aided Design, 36(10):993–1009, 2004.

[38] C. Sun, J. Sherrah. 3D symmetry detection using the extended Gaussian image. IEEE Trans. Pattern Analysis and Machine Intelligence, 19(2):164–168, 1997.

[39] S. Tate, G. Jared. Recognising symmetry in solid models. Computer-Aided Design, 35(7):673–692, 2003.

[40] D. Thompson. On growth and form, Cambridge University Press, 1917.

[41] W. Thompson, J. Owen, H. Germain, S. Stark, T. Henderson. Feature-based reverse engineering of mechanical parts, IEEE Trans. Robotics and Automation, 15(1):57–66, 1999.

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

[43] P. Varley, R. Martin, H. Suzuki. Frontal geometry from sketches of engineering objects: is line labelling necessary. Computer-Aided Design, 37(12):1285–1307, 2005.

[44] H. Zabrodsky, S. Pelog, D. Avnir. Symmetry as a continuous feature. IEEE Trans. Pattern Analysis and Machine Intelligence, 17(12):1154–1166, 1995.

[45] H. Zou, Y. Lee. Constraint-based beautification and dimensioning of 3D polyhedral models reconstructed from 2D sketches. Computer-Aided Design, 39(11):1025–1036, 2007.

[46] 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.

Ming Li received a PhD in applied mathematics from the Chinese Academy Sciences in 2004. He then worked as a postdoctoral research fellow at Cardiff University in the UK for three years and Drexel University in the US for one year. He is now an associate professor at Zhejiang University in P.R. China. His research interests include solid and geometric modelling, CAD and CAGD. His current work involves building high-level model description and virtual prototyping.

Frank C. Langbein received a diploma in mathematics from Stuttgart University in 1998 and a PhD from Cardiff University in 2003, where he is now a lecturer in computer science. His research interests include geometric, solid and physical modelling and related topics in computer graphics and vision. He has been working on reverse engineering geometric models, focused on beautification and design intent, approximate symmetries and pattern recognition, geometric constraints, mesh processing, and point-based modelling. He is also interested in the simulation, identification, and control of quantum systems.

Ralph R. Martin obtained a PhD in 1983 from Cambridge University. He has been a Professor at Cardiff University since 2000. His publications include over 170 papers and 10 books covering such topics as solid and surface modelling, reverse engineering, intelligent sketch input, and mesh processing. He is a Fellow of the IMA, and a Member of the BCS. He is on the editorial boards of 'Computer Aided Design', the 'International Journal of Shape Modelling', 'CAD and Applications', and the 'International Journal of CAD-CAM'.