Graphical Models 82 (2015) 44–57

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Applying geometric constraints for perfecting CAD models in reverse engineering

István Kovács*, Tamás Várady, Péter Salvi

Budapest University of Technology and Economics, Hungary

ARTICLE INFO

Article history:
Received 12 November 2014
Revised 15 April 2015
Accepted 10 June 2015
Available online 16 June 2015

Keywords:
Reverse engineering
Constrained fitting
Approximate and partial symmetry

ABSTRACT

An important area of reverse engineering is to produce digital models of mechanical parts from measured data points. In this process inaccuracies may occur due to noise and the numerical nature of the algorithms, such as, aligning point clouds, mesh processing, segmentation and surface fitting. As a consequence, faces will not be precisely parallel or orthogonal, smooth connections may be of poor quality, axes of concentric cylinders may be slightly tilted, and so on. In this paper we present algorithms to eliminate these inaccuracies and create "perfected" B-rep models suitable for downstream CAD/CAM applications.

Using a segmented and classified set of smooth surface regions we enforce various constraints for automatically selected groups of surfaces. We extend a formerly published technology of Benkő et al. (2002). It is an essential element of our approach, however, that we do *not* know in advance the set of surfaces that will actually get involved in the final constrained fitting. We propose *local* methods to select and synchronize "likely" geometric constraints, detected between pairs of entities. We also propose *global* methods to determine constraints related to the whole object, although the best-fit coordinate systems, reference grids and symmetry planes will be determined only by surface entities qualified as relevant. Lots of examples illustrate how these constrained fitting algorithms improve the quality of reconstructed objects.

1. Introduction

Reverse engineering (digital shape reconstruction) is an expanding and challenging area of Computer Aided Geometric Design [2]. This technology is utilized in various applications where a given physical object is scanned in 3D, and a computer representation is needed in order to perform various computations. A wide range of applications emerges in engineering, medical sciences, and to preserve the cultural heritage of mankind [3].

For CAD models the process can be split into the following phases:

(1) 3D data acquisition (scanning), (2) filtering and merging point clouds, (3) creating triangular meshes, (4) simplifying and repairing meshes, (5) segmentation (partitioning into disjoint regions), (6) region classification, (7) fitting primary (functional) surfaces, (8) fitting connecting surfaces (e.g. fillets), (9) *perfecting surfaces* (including constrained fitting and surface fairing), (10) creating a B-rep model (i.e., stitching surfaces and building up a topological structure), (11) exporting to CAD/CAM systems.

In the majority of engineering applications, it is crucial that the reconstructed models satisfy various geometric constraints. The primary surfaces—and their associated directions and axes, if any—must obey various rules, such as being orthogonal, parallel, tangential, symmetric, concentric, and so on. If we approximate the segmented regions *separately*, one by one, we may obtain inaccurate surfaces and poor CAD models. This is due to the noise and incompleteness of

* Corresponding author.
E-mail address: kovika91@gmail.com (I. Kovács).

http://dx.doi.org/10.1016/j.gmod.2015.06.002
1524-0703/

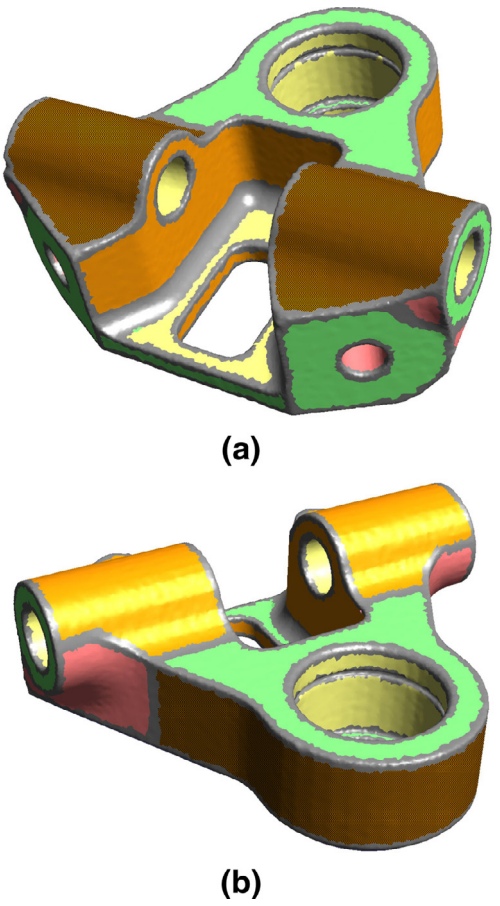

Fig. 1. Segmented object: (a) view 1; (b) view 2. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article)..

measured data, and the numerical nature of the subsequent algorithmic phases, such as merging multiple data sets, decimating and repairing triangular meshes, segmentation, and least-squares fitting without constraints.

The goal of our research is to perfect CAD models created from measured data. We introduce techniques to automatically detect likely engineering constraints, and enforce these by performing *constrained fitting*. We work with the following assumptions.

(i) Our input is a segmented mesh containing a set of regions. In most cases, this means that the highly curved triangle-strips corresponding to fillets or sharp edges get removed, and thus a numerically stable set of *disjoint regions* is obtained that corresponds to the face structure of the final B-rep model—an example is shown in Fig. 1.

(ii) By means of classification, a surface type has already been assigned to each region providing the best local approximation to the underlying data points.

(iii) We deal with the most frequent engineering surfaces: planes, quadric surfaces (cylinders, cones, spheres), extruded and drafted surfaces (defined by a direction vector and a profile), rotational surfaces (defined by an axis and a profile), and the remaining surfaces considered as free-form. (Constraints for free-form surfaces and general sweeps are going to be subject of future research.)

Let us take the above simple CAD model and look at a few problems that motivate our work. This object has been measured in a general 3D workspace, so its base surfaces are not aligned to an optimal coordinate system. The planar surfaces at the bottom and top (green) are not perfectly parallel, neither perpendicular to the planar faces at the left and right sides. The horizontal cylindrical holes (yellow) are unlikely to share the same axis and radius. The two extruded surfaces (beige) on the top may have two slightly different profiles and directions of extrusion. The two vertical cylinders (yellow) in Fig. 1(b) will not be exactly concentric, their axis will not necessarily be contained in the plane of symmetry, and will not be perpendicular to the top planar face (green), and so on. Nothing guarantees that the reconstructed object will be symmetric.

Our proposed techniques are based on recognizing the most likely constraints using the initial parameters of individually fitted surfaces. We select groups of relevant surfaces that are likely to comprise a set of parallel/orthogonal entities, share common axes and directions, force profile curves to be identical or symmetric, and so on; then refit while maintaining these constraints. We also search for likely best-fit coordinate systems that align the majority of surface elements and refit accordingly. We locate relevant surface groups that are found fully or partially symmetric and enforce symmetry accordingly. We attempt to compute a global reference grid, where the dimensions match a well-defined grid-size, that likely corresponds to the original design intent, and snap the related elements. All these problems are converted into solving a large system of non-linear algebraic equations.

Handling constraints may seem relatively easy for a human engineer, when the constraint types and the set of relevant surfaces are explicitly specified. However, this is fairly difficult for a program, where even the group of surfaces is unknown for which the constraints needs to be set up. While the majority of commercial systems offers a wide range of operations to tweak surface parameters directly, here we search for *automatic* techniques, that can save a large amount of manual work and reduce errors. In this context users remain involved at two levels: (i) prescribe "likelihoods" by setting various tolerances (depending on the quality of measured data and the dimensions of the object), and (ii) check and approve perfected surface geometries, as constrained fitting is moving forward.

The paper is structured as follows. In Section 2 previous work is reviewed. In Section 3, the method of Benkő et al. [1] is revisited, as this constitutes the basis of the current work. Section 4 deals with *local* constraint detection and satisfaction. In Sections 5–7 algorithms are presented to compute *global* constraints that relate to the whole, or at least large parts of the object including best-fit coordinate systems, reference grids and symmetry planes. Test examples at the end of each section illustrate the results.

2. Related work

Discovering design intent and inherent structural properties is an important topic in many areas. Related publications

span a diverse range of research, from computer graphics and medical imaging to computer aided design and reverse engineering. This variety is based on algorithms that use different types of input data: bitmaps, point clouds, meshes, voxels, or even complete boundary representation models. Naturally, the choice of data largely influences the techniques we may apply and the output we may produce including various properties, geometric relationships, shape features, segmented data sets or fully-evaluated CAD models.

One essential intrinsic characteristic of a model is symmetry. A good state-of-the-art review was published recently by Mitra et al. [4], discussing the most essential issues and algorithms, including symmetry groups, global and local, exact and approximate symmetries, and other topics. A long list of applications is also given including model acquisition, representation and synthesis, though relatively small attention is directed to the specific problems of perfecting mechanical engineering objects, which is the main subject of our paper.

Here we do not go into the details of the most important symmetry algorithms, since this report contains a long list of related publications and analyzes the "pros and cons" of the different methods. The first approaches (e.g. [5]) handled only exact global symmetries. This was, however, soon amended by the definition of symmetry distances [6,7], eventually leading to efficient detection of partial symmetries (both local and global) on voxel data.

Discovering structural regularities, in particular partial or full symmetries, is a well-studied area also for B-rep models. Tate and Jared [8] present a detailed review on related topics, and discuss an algorithm to compute symmetries based on matching various topological entities, such as edge-loops. In a recent paper, Li et al. [9] explore the problem of extracting exact symmetries from B-rep models using a divide-and-conquer approach. The above publications, however, are not well-suited for reverse engineering applications and in the rest of this section we will review some research articles dealing with this specific problem.

The reconstruction of CAD models is a very complex computational process with various phases involving mesh processing, segmentation and surface fitting [2]. Here we deal only with one step, which comprises a separate research area: perfecting models. In general, for detecting various engineering constraints and high-level structures, a segmentation and some preliminary surface approximations need to be created. The majority of engineering objects obey well-defined construction rules, and combine a set of surfaces ranging from planes and natural quadric surfaces, through extruded, rotational and lofted surfaces, to free-form geometries.

Reconstruction methods differ in what sort of rules can be detected and enforced including (i) a set of usual local constraints, such as parallelism, tangency, orthogonality, concentricity, and (ii) high-level structural dependencies, e.g. alignments or different sorts of symmetries. Methods also differ in the range of surface types they are capable to handle. State-of-the-art solutions follow different strategies.

In the first group of approaches information is directly extracted from point cloud data. RANSAC-based methods [10,11] iteratively segment the point cloud through a sequence of trial-and-error steps discovering primitive surfaces, such as planes, cylinders, cones and spheres, then enforce well-defined angular and alignment constraints amongst these elements. This is an impressive approach: it can automatically locate disjoint regions of the same surface, and seems to be computationally efficient. Difficulties may arise when more complex, profile-based surfaces need to be reconstructed over random point samples, and more general constraints—e.g., tangencies—need to be satisfied. For conflict-free constraints a graph-reduction method is used. The lack of an underlying mesh may produce difficulties at model creation.

Important point cloud-based methods for detecting profile curves and related constraints of extruded and rotational surfaces are discussed in Ke et al. [12]. Automatically recognized constraints include orthogonality, tangency, equal distances, etc. They also use a modification of the Iterative Closest Point algorithm to find symmetry planes. Another interesting point cloud-based method was presented by Liu et al. [13], that combines surface fitting with multiple-view registration.

In the second group of approaches we assume that there exists some *a priori* knowledge about the entities that need to be linked by different constraints. The important publication of Benkő et al. [1] thoroughly investigates this problem; it deals with a very wide variety of possible constraints over a given set of entities, presents an efficient numerical method to solve large non-linear constraint systems. A very useful feature of this method is that redundant and contradictory constraints get automatically discarded. Our work is considered to be an extension of this approach.

There are two crucial issues here worth mentioning. (i) To compute Euclidean distances from the entities involved would typically require formulae with square-roots. These are generally replaced by so-called *faithful* representations, that are algebraically much simpler and still closely approximate Euclidean distances in the vicinity of data points, see related work of [14,15]. (ii) Another important issue is applying *efficient representations*, that make it possible that terms containing data points do not need to be recomputed in each iteration step, thus computations can be significantly speeded up [1].

A couple of interesting papers related to symmetry analysis and beautification of reverse engineered models were published at Cardiff University [16–18]. These papers focus on detecting full and partial, approximate symmetries; they apply algebraic methods and fundamentally process feature points. One potential difficulty is that these algorithms assume that these feature points always exist after creating and evaluating a preliminary model. The authors also deal with perfecting distances, and recognizing divisions by regular units. This problem is also investigated in our paper in a more general manner. In [16] the solution of the constraint system is improved by eliminating the inconsistencies of the constraint set by graph operations.

Although we have adapted some techniques from previously published papers, our approach can be distinguished by the following aspects. We focus on reverse engineering CAD models using a *pre-segmented mesh* of smooth disjoint regions and initial surface approximations. This is needed to build up a *hypothetical constraint set* and select a group of *relevant surfaces* that will actually take part in the final constrained fitting. Besides planes and natural quadrics we

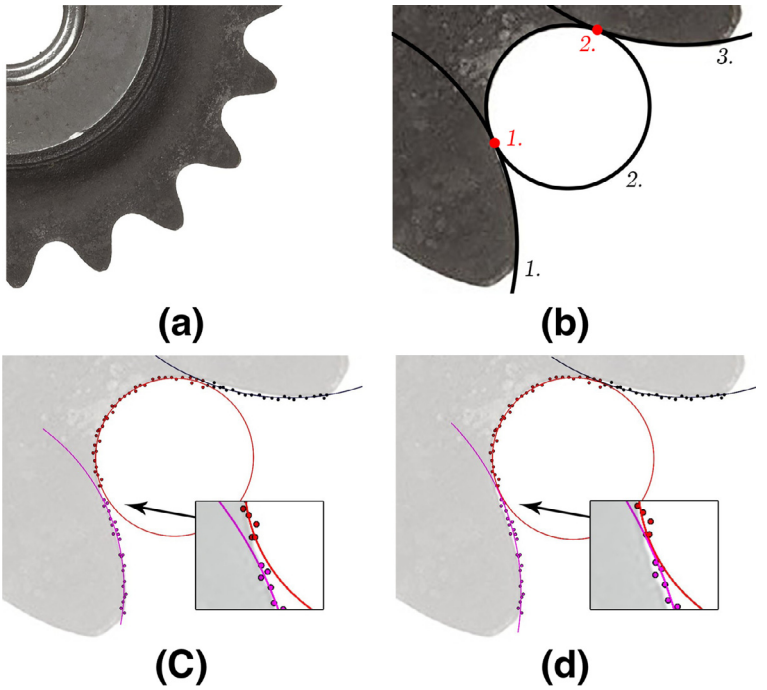

Fig. 2. Constrained fitting: (a) profile of a gear wheel; (b) three circles with prescribed tangency; (c) independent fitting—discontinuity; (d) fitting with constraints—smooth connections.

work with extruded and rotational surfaces, as well. We attempt to perfect the primary surfaces of a B-rep model, and do not deal with its final topological structures, such as edges and face loops, as our view is that a good B-rep can only be built after having the surfaces perfected. We handle a fairly wide set of local and global constraints, and focus on semi-automatic techniques.

3. Basic concept of constrained fitting

The segmented mesh in itself does not carry information about the constraints associated with the surfaces of the object; these need to be prescribed either by the user, or detected and set by some "intelligent" algorithm.

Local surface fitting can be formulated as follows. Let us denote the surfaces by $\{s_i\}$, and the corresponding data points by $\{p_{ij}\}$. All surface approximation methods minimize the sum of squared distances between the surface and the point cloud in some way. Let $\mathbf{x_i}$ contain the parameters of the ith surface, then we wish to minimize

$$f_i(\mathbf{x_i}) = \sum_j d(p_{ij}, s_i(\mathbf{x_i}))^2. \tag{1}$$

In this paper we cannot go into the details of how different types of surfaces (planar, quadric, extruded, rotational, free-form surfaces) can be approximated; some related algorithms can be found in [2].

In order to perfect our models we apply *constrained fitting* that always involves approximating multiple point clouds contained in different surface regions. The parameter vector $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_n})$ stores all the parameters of the corresponding surfaces. A constraint is typically given as a non-linear algebraic equation in the form $c_k(\mathbf{x}) = 0$, where each constraint connects certain parameters of some surfaces. We formulate the global constraint system using the full parameter vector as

$$\mathbf{c}(\mathbf{x}) = 0. \tag{2}$$

We need to minimize a single expression of squared distances while the constraints are satisfied

$$f(\mathbf{x}) = \sum_i \alpha_i \sum_j d(p_{ij}, s_i)^2, \tag{3}$$

where α_i-s denote positive weights, that can be associated with the individual surfaces, for example, by surface area. To sum it up: constraints have high priority and get satisfied with a very tight tolerance, while approximating the data is "secondary" and accuracy can be ensured only in best-fit (least-squares) sense.

Before going into the details of solving large constraint systems, we show a simple example of how constraints can be formulated.

3.1. Constraints for a smooth profile curve

Consider a planar profile curve with three already segmented data sets, as shown in Fig. 2(b). If we fit these circles independently, the tangential constraints will not be satisfied, but with constrained fitting a good solution will be obtained. In this example, c_i denotes the circles to be constrained with parameters (A_i, B_i, C_i, D_i), defined by the equations $f_i(x, y) = A_i(x^2 + y^2) + B_i x + C_i y + D_i = 0$. c_i will approximate n_i points. The average squared distance to be

minimized is as follows:

$$f(\mathbf{x}) = \sum_{i,j} d_{i,j}^2 = \sum_i \frac{1}{n_i} \sum_j (A_i(x_{ij}^2 + y_{ij}^2) + B_i x_{ij} + C_i y_{ij} + D_i)^2.$$

The constraint system includes

- three normalization constraints $(B_i^2 + C_i^2 - 4A_iD_i = 1)$, and
- two tangential constraints $(2A_jD_i + 2A_iD_j - B_iB_j - C_iC_j \pm 1 = 0)$.

The normalization constraints come from applying a faithful distance representation, i.e. $|\nabla(f_i(x,y))| = 1$. The tangential constraints come from the equations $f_i(x_0,y_0) = f_j(x_0,y_0)$ (circular arcs have a common endpoint) and $\nabla(f_i(x_0,y_0)) = \nabla(f_j(x_0,y_0))$ (circular arcs share a common tangent), where x_0 and y_0 will be eliminated. Finally, seven independent variables remain to compute the best-fit circles for the given data set. Related numerical values of this test example will be given at the end of the section.

3.2. The numerical method

Let us continue to briefly present the method of Benkő et al. [1], as this is necessary to understand the rest of the paper. Using the previous notations, let us order the constraints by priority as $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), ..., c_k(\mathbf{x}))$, and assume that $f(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$ are smooth enough (at least C^2). Here we have a highly non-linear system of equations to be solved using a special Newton iteration. We approximate $\mathbf{c}$ in first order, and f in second order. The Taylor approximations of $\mathbf{c}$ and f around $\mathbf{x}_0$ are the following:

$$\mathbf{c}(\mathbf{x}_0 + \mathbf{d}) \approx \mathbf{c}(\mathbf{x}_0) + \mathbf{c}'(\mathbf{x}_0)\mathbf{d}, \tag{4}$$

$$f(\mathbf{x}_0 + \mathbf{d}) \approx f(\mathbf{x}_0) + f'(\mathbf{x}_0)\mathbf{d} + \frac{1}{2}\mathbf{d}^T f''(\mathbf{x}_0)\mathbf{d}. \tag{5}$$

In each step of the iteration we need to determine a small difference vector $\mathbf{d}$. Using the above equations, these can be written locally in the form

$$C\tilde{\mathbf{d}} = 0, \tag{6}$$

$$\tilde{\mathbf{d}}^T A \tilde{\mathbf{d}} \to \min, \tag{7}$$

where $\tilde{\mathbf{d}} = (d_1, \ldots, d_n, 1)$, $C = [\mathbf{c}'(\mathbf{x}_0)|\mathbf{c}(\mathbf{x}_0)]$ and A is an $(n+1) \times (n+1)$ size matrix, as follows:

$$A = \left[\begin{array}{c|c} f''(\mathbf{x}_0) & f'(\mathbf{x}_0) \\ \hline f'(\mathbf{x}_0)^T & 0 \end{array}\right]. \tag{8}$$

In order to calculate $\tilde{\mathbf{d}}$ we have to reduce it to a lower dimensional vector $\mathbf{d}^*$ by (6), such that $\mathbf{d}^*$ has only *independent* coordinates. We calculate a matrix M, such that $\mathbf{d} = M\mathbf{d}^*$, and $CM = 0$. Now the dimension of $\mathbf{d}^*$ tells us how many independent variables exist in the system. Finally, we can solve $\mathbf{d}^{*T}A^*\mathbf{d}^* \to \min$ without constraints, where $A^* = M^T A M$, and this can be solved as a simple system of linear equations (see details in [1]). We note that this minimization is always solvable. The proof is based on the fact, that $f'(\mathbf{x}_0)$ is symmetric positive definite in this case.

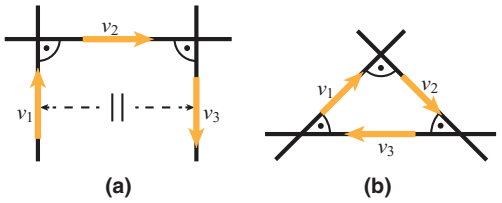

Fig. 3. Three lines with (a) redundant, or (b) contradictory constraints.

The way of calculating M is very similar to Gauss elimination as discussed in [1]. During elimination, we check if the system is over-determined or if some of the constraints contradict each other. This is illustrated by two very simple examples, see Fig. 3. The first is a redundant case, the direction vectors of three lines must satisfy the following three constraints: $v_1 \perp v_2$, $v_2 \perp v_3$ and $v_1 \| v_3$. The second is a contradictory case, where the three lines should be pairwise perpendicular, i.e., $v_1 \perp v_2$, $v_2 \perp v_3$ and $v_1 \perp v_3$. Then we obtain the following two plus one constraint equations, respectively:

$$\begin{aligned} c_1 &: (v_{1i} + d_1, v_{2i} + d_2) = 0, \\ c_2 &: (v_{2i} + d_2, v_{3i} + d_3) = 0, \\ c_{3a} &: (v_{3i} + d_3, v_{1i} + d_1) = -1, \\ c_{3b} &: (v_{3i} + d_3, v_{1i} + d_1) = 0, \end{aligned}$$

where v_{1i}, v_{2i}, v_{3i} denote the direction vectors at the current iteration, and $d = (d_1, d_2, d_3)$ are the small unknown difference vectors to solve the system. In the first "redundant" case, take the third constraint c_{3a}; here $v_{1i} \simeq -v_{3i}$, thus $d_1 \simeq -d_3$, consequently the second equation becomes identical to the first one within a very small tolerance, and can be disregarded. In the second "contradictory" case with constraint c_{3b}, the vector d_2 can be expressed in two different ways from c_1 and c_2, thus one of these constraints must be omitted.

3.3. Auxiliary objects

The use of the so-called *auxiliary objects* is an important idea in constrained fitting. We illustrate this through a simple example. Take the three lines of Fig. 5(a) that are supposed to meet in a common point. We can formulate the related constraints by taking lines 1 and 2, compute their intersection point and constrain this to lie on line 3; then we take lines 2 and 3, and lines 3 and 1 with similar constraints. This set of equations defines a relatively simple problem in a very complicated way.

An alternative solution is to introduce an auxiliary point p. This is an unknown entity, but now we can define our constraints by three simple equations, i.e., all three lines must pass through p. Clearly, we have *increased* the number of unknowns in the parameter vector $\mathbf{x}$, but the system of equations—and all related Taylor approximants—have become much simpler. Note, that the unknown surface parameters are generally associated with corresponding point sets, but for auxiliary objects such a data point has no meaning. Typical auxiliary entities include a point, a point and a normal, a distance, etc.; their exclusive role is to simplify the system of equations and thus our computations. We will use

Table 1
Numerical analysis of unconstrained vs. constrained fitting for the three-arc profile curve.

	Independent fit	Tangential constraints	Tangential and "radii are multiple of 10" constraints
Radius 1 (r_1)	125.762	155.810	160
Radius 2 (r_2)	50.590	51.683	50
Radius 3 (r_3)	145.962	156.278	160
Average deviation from data points	0.996	1.046	1.132
Error of 1st tangential constraint	2.142	0	0
Error of 2nd tangential constraint	1.010	0	0
Error of 'r_1 multiple of 10' constraint	4. 238	4.190	0
Error of 'r_2 multiple of 10' constraint	0.590	1.683	0
Error of 'r_3 multiple of 10' constraint	4. 038	3.722	0

auxiliary objects for computing the best-fit coordinate systems, reference grids and symmetry planes in Sections 5–7, respectively.

3.4. Test example: smooth profile curve

In this example, the three-arc profile case is analyzed; the effect of constrained fitting is demonstrated by values in Table 1. If we fit the three circles independently (Fig. 2(c)), there will be small gaps and no tangential continuity between the adjacent arcs yielding poor quality (column 2). When we apply constrained fitting (Fig. 2(d)), although the average approximation error increases to a negligible extent, the prescribed constraints will be satisfied (rows 5 and 6 in column 3). We will return to this example in Section 6, when further constraints to snap the radii to multiples of unit 10 will be added (rows 7–9 in column 4).

4. Automatic detection of local constraints

Let us start with a simple example. We wonder whether pairs of lines are perpendicular or not, and we wish to incorporate additional constraints into our system, if the likelihood of being perpendicular is high. This can clearly be controlled by a user-defined angular tolerance, i.e., extra constraints will be added automatically, if two lines span an angle between $90 \pm \varepsilon$.

Formally: let $c(\mathbf{x}) = 0$ a simple constraint between two objects, and ε the tolerance level. The c constraint is within tolerance if and only if $|c(\mathbf{x})| < \varepsilon$, and we want to validate whether the constraint holds. For this, we introduce the following function:

$$s_\varepsilon(x) := \begin{cases} x & \text{if } |x| < \varepsilon \\ 0 & \text{otherwise.} \end{cases}$$

We observe that if $c(\mathbf{x})$ is out of tolerance, then $s_\varepsilon(c(\mathbf{x}))$ vanishes, and the constant zero constraint will not modify our system, otherwise $s_\varepsilon(c(\mathbf{x}))$ reproduces $c(\mathbf{x})$.

Here again we assume that $c(\mathbf{x})$ represents a faithful (exact or close to Euclidean) distance representation. For example, for the *line meets point* constraint, we must use a normalized line-point distance function

$$c_{pl}(\mathbf{x}) = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}.$$

Also note that s_ε is not continuous, but a piecewise continuous function, so if we calculate the derivative, we need to make it piecewise, as well.

To detect constraints automatically, we take the modified constraints for all object pairs. The numerical method will enable only those constraints that are within the related tolerance level. The user typically defines different tolerances for different (parallel, perpendicular, tangential, concentric, etc.) constraints. Let us denote the set of objects by $S = \{s_i\}$, and the constraint types by $\{c_j\}$, such that $c_j(s_1, s_2)$ denotes an actual constraint between s_1 and s_2. Then for all c_j, consider the following constraint set:

$$C_j = \{s_{\varepsilon_j}(c_j(s_1, s_2)) : s_1, s_2 \text{ suitable for } c_j\}.$$

Thus the global constraint system includes the explicitly defined constraints, and the "likely" constraint set of C_j-s.

A somewhat artificial example with three circles and three lines can be seen in Fig. 4 that shows different configurations created by different tolerances. Compare cases (b) and (c). The angular tolerance of the "orthogonal lines" constraint is relatively tight in both cases, involving two lines out of the three. The tolerance of "tangential line and circle" is tight in case (b) snapping only the blue circle, and loose in case (c) forcing two circles to be tangential to the almost horizontal line.

We can also handle more complex local (i.e., not pairwise) constraints. For example, take the previously mentioned "three lines meet in a single point" constraint in Fig. 5. We may create auxiliary intersection points for all three pairs of lines, and by means of corresponding "line close to point" constraints the algorithm can detect whether the three intersection points are *likely to be* coincident or not. In the former case the three lines will be fitted simultaneously, enforcing a common point of intersection.

4.1. Test example: a selected constraint set

The simple L-shaped object in Fig. 6 has nine surface entities representing eight planar faces and a cylindrical hole. We may request to set the pre-fitted surfaces to be accurately parallel and perpendicular. At this point we do not know whether all nine entities will be pairwise constrained, which otherwise would lead to a set of 36 equations. If we select a tight tolerance, it turns out that two vertical surfaces (shown darker in the figure) do not belong to this constraint set and must preferably be kept slightly tilted. Thus only seven relevant surfaces will be linked and refitted over data points in seven regions, and the constraint set will be reduced to 21 constraints. During the elimination steps further 15 redundant constraints fall out, leaving only six independent constraints. An example for such a minimal set can be the

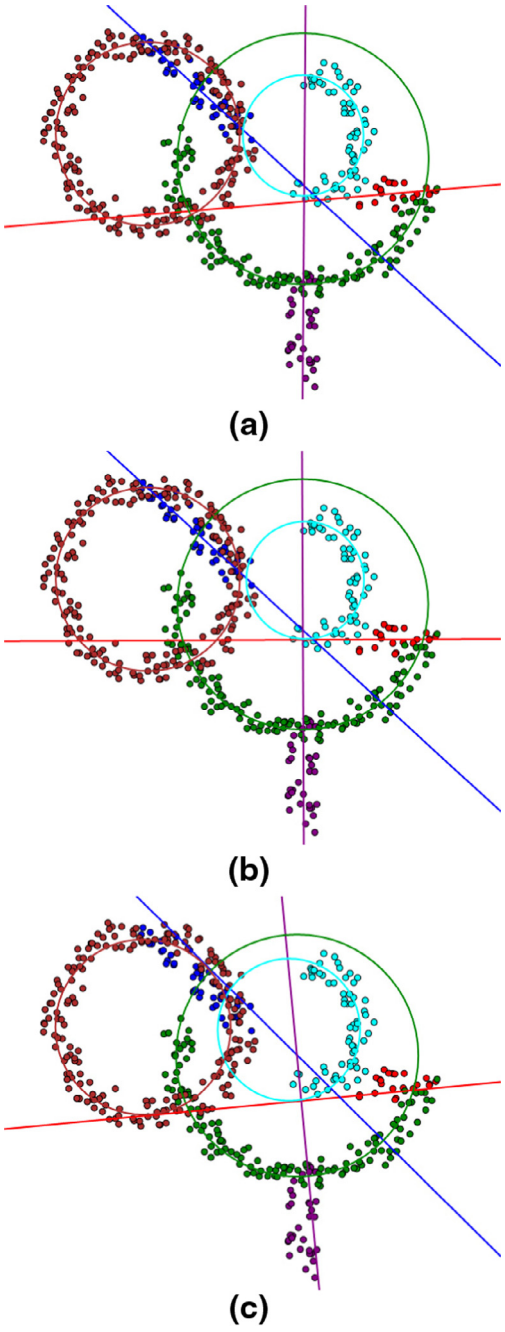

Fig. 4. Automatically detected constraints: (a) initial state; (b and c) different configurations created by different tolerance levels. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

following: start with the bottom face, add three "parallel" and one "perpendicular" constraints for the side and top faces, and one more for the axis.

5. Best aligned coordinate systems

The majority of mechanical engineering parts are defined in a well-aligned coordinate system, which means that the majority of surface geometries are aligned to the principle coordinate axes. Some other surfaces may be aligned to different, dependent coordinate systems, or not at all. In the reverse engineering context, scanning may take place in a workspace, where the object is positioned in a general orientation. We wish to create a CAD model that is optimally aligned, thus after a transformation we avoid tilted orthogonal views, and conveniently enforce constraints by the best alignment(s). We take into consideration the most frequent surface geometries, but will not deal with general sweeps and free-form surfaces.

For each segmented region we associate not only the surface type, but a direction vector d_i and the surface area A_i. The direction for planes is the best-fit normal vector. The direction for extrusions and drafted extrusions is an orthogonal vector to the plane of their profile curves. For rotational surfaces, including cylinders and cones, the associated direction is defined by the axis of rotation. The surface area is estimated as the sum of triangles in the given region. The total estimated area of the object is $A = \sum_i A_i$.

Assume we have a coordinate system by mutually orthogonal unit vectors (n_x, n_y, n_z). We can characterize the current alignment by an angular tolerance ε in the following way. First we collect surface entities that are being aligned

$$I(n_x, n_y, n_z) = \{i : |d_i \times n_x| < \varepsilon \vee |d_i \times n_y| < \varepsilon \vee |d_i \times n_z| < \varepsilon\},$$

then compute the *Coverage* of the coordinate system by

$$\text{Coverage}(n_x, n_y, n_z, \varepsilon) = \frac{\sum_{i \in I(n_x, n_y, n_z)} A_i}{A}.$$

For a poor alignment this number will be small, for a good alignment it will be greater, but not necessarily close to 1, as there may be free-form surfaces, or other surfaces that belong to different coordinate systems. Another measure is the magnitude of the global *approximation error* for the aligned surfaces, i.e.,

$$\text{ApprErr}(n_x, n_y, n_z) = \frac{\sum_{i \in I(n_x, n_y, n_z)} A_i \min\left(|d_i \times n_x|, |d_i \times n_y|, |d_i \times n_z|\right)}{A}.$$

Our goal is to search for a coordinate system with the best Coverage, while ApprErr remains below a reasonable threshold. Accepting a group of surfaces as "well-aligned" also means that we have identified a complementary group of "non-aligned" surfaces, so a new round of search for the second best coordinate system can be started.

5.1. The algorithm

Each d_i vector represents a vector on the Gaussian sphere weighted by A_i, and the surfaces will create a set of clusters. Our goal is to find the best vector triplet to match this set. The algorithm is organized by the principle of cluster growing. A cluster C is characterized by a vector p, as follows:

$$p' = \sum_{d_i \in C} A_i d_i, \quad p = \frac{p'}{||p'||}.$$

Step 1. Take the direction vector d_i from the ith surface, then check whether there already exists a cluster where p

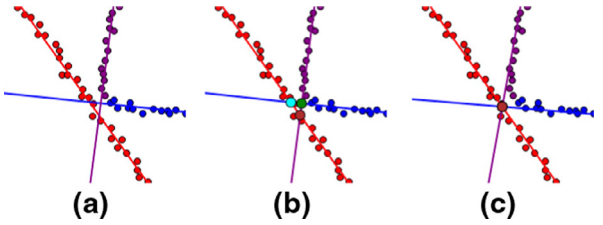

Fig. 5. Three lines meet at a common point: (a) initial state; (b) pairwise intersection (auxiliary points); and (c) "three points are equal" constraint enforced.

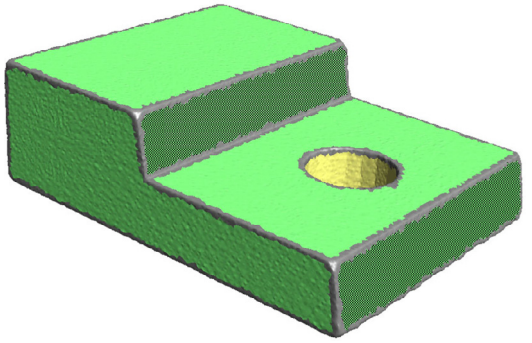

Fig. 6. Test object with parallel and perpendicular constraints.

(and at the same time $-p$) and d_i (or $-d_i$) span an angle less than ε; if yes, include this surface, otherwise open a new one. Perform this for all surfaces.

Step 2. Compute a primary weight for each cluster, as

$$G(p) = \{i : |d_i \times p| < \varepsilon\}, \quad W_1(p) = \sum_{i \in G(p)} A_i.$$

Step 3. Compute a secondary weight for each cluster, that measures those clusters p_k that are orthogonal to p, i.e.,

$$H(p) = \{k : |\langle p_k, p\rangle| < \varepsilon\}, \quad W_2(p) = \sum_{k \in H(p)} A_k.$$

Step 4. Rank the clusters by $W_1(p) + W_2(p)$, and select the strongest p to be axis n_z. In this way we have also defined an orthogonal cluster set in a principle plane, which will contain n_x and n_y.

Step 5. We compute a tertiary weight for p_k from the orthogonal clusters of n_z (i.e. $k \in H(n_z)$) as

$$L(p_k) = \{l : |\langle p_l, p_k\rangle| < \varepsilon, l \in H(n_z)\}, \quad W_3(p_k) = \sum_{l \in L(p_k)} A_l,$$

and pick n_x by the largest value of $W_1(p_k) + W_3(p_k)$.

Step 6. Then $n_y = -n_x \times n_z$.

The heuristics of the algorithm can easily be understood by means of Fig. 7. Here the red cluster is the strongest, having good support from the blue clusters which are closely orthogonal to the red. The other clusters (grey) are weaker and get discarded. Once n_z is fixed, we compute the strongest cluster in the principle plane (blue) that determines the other two axes.

This algorithm defines the direction of the main coordinate axes to be associated with our model, however, further steps are needed to determine the position of its origin. This will be done in the next sections where symmetry planes and best-fit reference grids are computed.

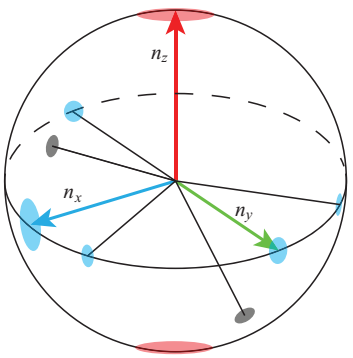

Fig. 7. Dominant clusters on the Gaussian sphere. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

5.2. Constrained fitting for the best alignment

Based on the above computations we select the relevant surfaces that are going to take part in the final constrained fitting to obtain the best alignment; these surfaces belong to the current $I(n_x, n_y, n_z)$. We introduce an auxiliary object of three orthogonal unit vectors (n_x, n_y, n_z), and refit by minimizing the squared angular deviations related to the direction vectors d_i, i.e.,

$$\sum_{i \in I(n_x, n_y, n_z)} A_i \min(|d_i \times n_x|^2, |d_i \times n_y|^2, |d_i \times n_z|^2).$$

5.3. Test example: enhanced coordinate system

In this example we return to our previous test object (Fig. 1). Compare the original coordinate system coming from the measurement (a) with the computed best-fit alignment (b); the whole data set has been transformed accordingly, as shown in Fig. 8. The surfaces that match the new coordinate system are colored dark. The number of aligned surfaces grow from 6 to 18 (see planes, cylinders and extruded surfaces); the coverage indicator grows from 29.82% to 86.11%, and the relative average approximation error changes from 1.864 to 0.078.

5.4. Test example: approximation error reduced

Our second example is about an object, which visually seems to be well-aligned, but analysis shows that the approximation error for the matching surfaces is relatively high. A closer visual inspection also discloses some tilting of the face-set in Fig. 9(a). We need to determine the best alignment and reposition the object accordingly. As expected, this does not change the coverage measure, which is 96.34% before and after, but significantly reduces the approximation error from 0.419 to 0.041. There remained a few faces that do not belong to the main coordinate system. For these the process has been repeated, and a second, local coordinate system was obtained, see Fig. 10.

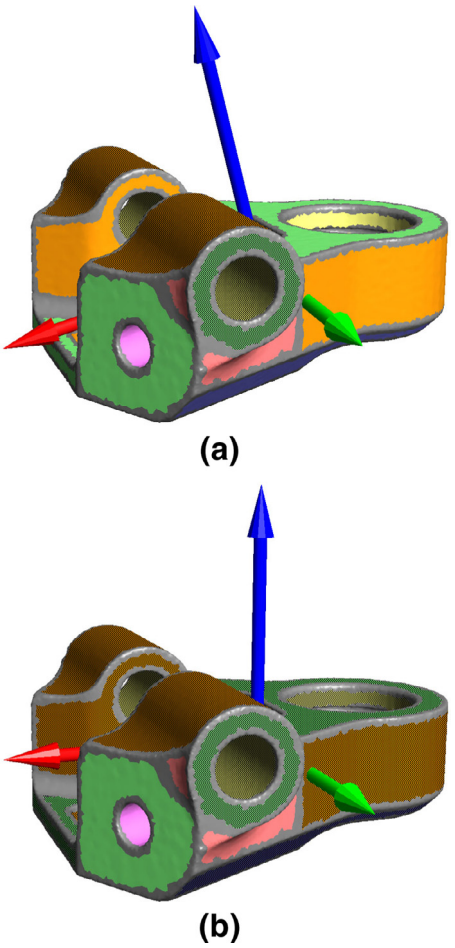

Fig. 8. Enhanced alignment: (a) original, (b) optimized.

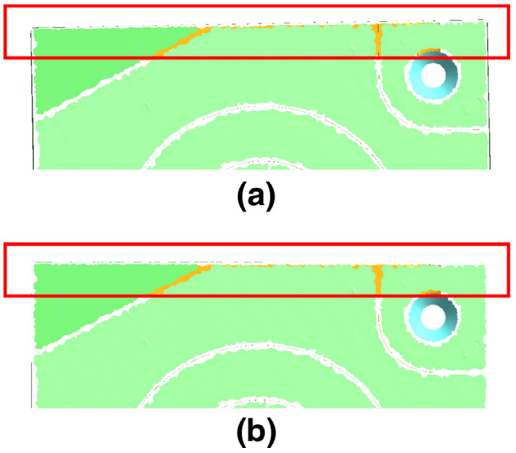

Fig. 9. (a) Minor misalignment, (b) correct alignment.

5.5. Test example: almost full alignment

Here we investigate the classical Darmstadt benchmark object (Fig. 11). After segmentation it has four planar (green) faces (S_1: bottom, S_2: top, S_3: front, S_4: back); two rotational faces (pink) (S_5 and S_6: aligned to z- and y-axis, respectively; one cone (cyan), (S_7 with z-axis), one drafted extrusion (blue), and a hidden free-form face. All the planar faces, the rotational axes and the direction of extrusion match the best fit alignment (coverage: 91.53%, approximation error 0.179).

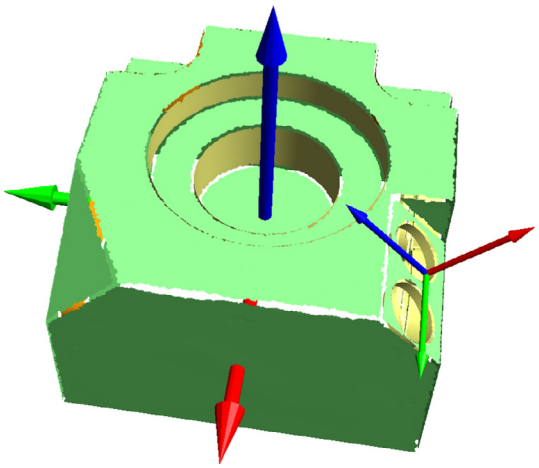

Fig. 10. Main and a local coordinate systems.

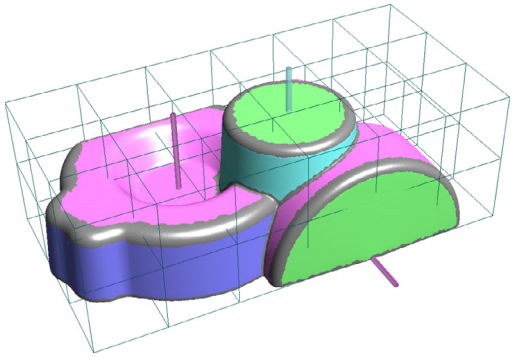

Fig. 11. Darmstadt test object (aligned, with grid). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

6. Snapping and the best fit reference grid

Snapping values to integers or quantifying dimensions to some multiples of base units is essential for good CAD models.

(i) Explicit specification of parameters occurs frequently in practice. For example, let an estimated radius of a cylinder be 10.2, then the user may prescribe a radius 10, and refit such a cylinder using the same data set.

(ii) A more complicated case is when only the quantification number is prescribed (hereinafter denoted by h), and the parameter values to be snapped are loosely estimated. As an example, take the previous three-arc profile. We can add constraints to set the radii to be multiples of prescribed values, i.e.,

$$r_i/h - round(r_i/h) = 0,$$

using the standard rounding rule. In this example, h was set to 10 producing perfect radii, as shown in column 4 of Table 1. It is nice to observe, that these constraints have set radius 1 and 3 to the same value.

(iii) The next level is when h is unknown. For example, we wish to quantify the heights of the three horizontal planar faces of the L-shaped object in Fig. 6. Assume we have initial height estimations as $(z_1 = 2.12, z_2 = 11.97, z_3 = 22.06)$, and want to minimize the remainders when dividing the height differences with the unknown h, i.e., we minimize the expression

$$\delta(h) = \sum_{i,j} \min\left(\left\{\frac{z_i - z_j}{h}\right\}, 1 - \left\{\frac{z_i - z_j}{h}\right\}\right), \tag{9}$$

where $\{x\}$ denotes the fractional part of x. As a result we expect to obtain $(z_1 = 2, z_2 = 12, z_3 = 22)$.

(iv) The most general problem is, when an optimal *reference grid* needs to be determined with an unknown origin (x_0, y_0, z_0) and a cell size h. Taking our example, we wish to minimize

$$\delta(h, z_0) = \sum_{i} \min\left(\left\{\frac{z_i - z_0}{h}\right\}, 1 - \left\{\frac{z_i - z_0}{h}\right\}\right), \tag{10}$$

which is going to shift the new origin to $z_0 = 2$ yielding heights $(z_1 = 0, z_2 = 10, z_3 = 20)$ in the new coordinate system.

The search for a reference grid corresponds to the usual engineering design practice, where objects are placed into a coordinate system with an imaginary grid in the background, and the majority of the surface parameters are set to quantified values. In our context, we assume that the coordinate axes have already been aligned, and we deal only with planes and rotational surfaces (including cylinders and cones) that have already been aligned parallel or orthogonal to the principal planes. We search for a reference grid that relocates surfaces onto the vertices, lines and planes of the reference grid, where this is possible. The basic difficulty of computing a best-fit reference grid is that the objective function to be minimized is not smooth, this is why we need to apply relatively expensive algorithms. We describe two algorithms corresponding to Eqs. 9 and 10. In the first *two-step algorithm* we work with orthogonal distances between pairs of Plane–Plane, Axis–Axis and Plane–Axis. First we determine the best cell size, then we optimize the position of the origin. In the second *single-step* algorithm the unknowns are simultaneously optimized. We will use weights for the minimization by the surface areas A_i. Examples will be given at the end of the section.

6.1. Computing a best-fit grid in two-steps

This algorithm first computes an optimal cell size, and then sets the optimal origin. We compute aligned orthogonal distances between the pairs of surface entities denoted by $\{d_i\}$, $i = 1, \ldots, N$, and assign a weight to each distance as $w_i = min(A_k, A_l)$. As before, we search for the best common divisor of the distances and minimize the sum of remainders as

$$\delta(h) = \sum_{i=1}^{N} \frac{w_i}{h} \min\left(\left\{\frac{d_i}{h}\right\}, 1 - \left\{\frac{d_i}{h}\right\}\right).$$

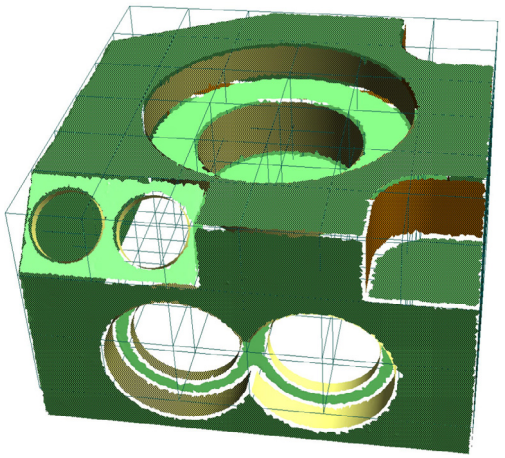

Fig. 12. Grid matches prismatic object.

Our goal is to find the minimum of $\delta(h)$ in the $[h_{min}, h_{max}]$ interval. It is easy to see that this function is piecewise monotone, with breakpoints at numbers of the form of d_i/k, for certain d_i-s, where k is a positive integer. Therefore, we need to search for the minimum only at these points, and we can find this in $O(N^2 d_{\max}/h_{min})$ steps, where $d_{\max} = \max_i n_i$.

Once we have an optimal cell size h, the origin of the grid needs to be located. A given CAD object may fully or partially be symmetric, in other words there may exist "dominant" symmetry planes that fully or partially restrict the position of the origin (see next section). The most general case is when the origin is not constrained at all, see Fig. 11. If there is one dominant symmetry plane, we restrict the origin to slide on that. If there are two such planes, the origin may slide along their line of intersection. Finally, if there are three dominant planes, as in Fig. 12, the best origin is uniquely defined. This means that we may want to determine the position of the origin by computing its 3, 2, 1 or 0 number of coordinates. Let u denote any of x, y or z, then the expressions below are independently minimized

$$\delta(u_0) = \sum_{i} \frac{w_i}{h} \min\left(\left\{\frac{u_i - u_0}{h}\right\}, 1 - \left\{\frac{u_i - u_0}{h}\right\}\right). \tag{11}$$

6.2. Computing a best-fit grid in a single step

Optimizing the cell size and the origin simultaneously may yield better results than the two-step method, though from a computational point of view this is much more demanding, in particular when the origin is free to set. For simplicity's sake, let us look at only the one-dimensional case by coordinates x_i; and assign weights to each entity as $w_i = A_i$. The unknown value of the best origin is x_0, thus we minimize the following expression:

$$\delta(x_0, h) = \sum_{i} \frac{w_i}{h} \min\left(\left\{\frac{x_i - x_0}{h}\right\}, 1 - \left\{\frac{x_i - x_0}{h}\right\}\right).$$

It is easy to see, that when x_0 is fixed, monotonicity breaks only at $h = (x_i - x_0)/l$ values. Similarly, when h is fixed, monotonicity breaks at numbers of the form $x_i - kh$, where l and k are integer numbers. Consequently, when we determine the minimum at fixed x_0 or h, it is sufficient to

search for numbers by these formulae, i.e.,

$$x_0 = \frac{lx_i - kx_j}{l-k},$$

$$h = \frac{x_j - x_i}{l-k}.$$

This means

$$O\left(N^2 \frac{\max_i(x_i)}{h_{\min}}\right)$$

choices altogether, and computing the function at a given point requires $O(N)$ steps, so the computation time will be $O(N^3)$.

For higher dimensional cases, when 1–3 coordinates of the origin must be determined, the solution is similar, as with fixed h minimization can be performed for each coordinate. Since for setting h there are $O(N^2 \max_{ij}(x_{ij})/h_{\min})$ choices, for three coordinates maximum

$$O\left(N^4 \left(\frac{\max_{ij}(x_{ij})}{h_{\min}}\right)^2\right)$$

steps are needed. Thus in the worst case when we determine the full set of x_0, y_0, z_0, h the algorithm is $O(N^5)$, which indicates that the single step algorithm may be very expensive for a high number of surfaces.

6.3. Constrained fitting for grids

At this point, we have a preliminary reference grid. We select those surfaces that actually match this "hypothetical" grid within a tolerance ε, as only these surfaces will take part in the final constrained fitting. The normal vectors of planes and the direction vectors of rotational surfaces have already been aligned, so we check the reference points of the planes and the axes whether they match the grid. For example, for a plane with x, y or z-aligned normal we request

$$(u_i - (u_0 + k * h))^2 < \varepsilon,$$

for a rotational surface with x, y or z-aligned axis we request

$$(u_i - (u_0 + k * h))^2 + (v_i - (v_0 + l * h))^2 < \varepsilon,$$

where u and v denotes any of the x, y, z coordinates.

The final constrained fitting will be performed using an auxiliary object with four unknowns x_0, y_0, z_0, h in the general case, and with constraints of the type above for each relevant surface. Based on the above equations we can define *coverage* and *approximation error* to measure how good is the best-fit grid. This computation also helps identifying the complementary group of surfaces, which were likely "off" the grid in the original design.

6.4. Test example: partial reference grid

We computed a best-fit reference grid for the Darmstadt object (Fig. 11). The following "meaningful" distances have been taken into consideration (here we refer to the coordinates of the reference points associated with surface S_i, as was indexed earlier):

- Plane–Plane distances: $z_1 - z_2, y_3 - y_4$;
- Axis–Axis distances: $x_5 - x_7, y_5 - y_7, x_5 - x_6, x_6 - x_7$;
- Plane–Axis distances: $y_3 - y_5, y_3 - y_7, y_4 - y_5, y_4 - y_7, z_1 - z_6, z_2 - z_6$.

The coverage value of this grid is relatively low (66.82%), as the top face, the cone, the drafted extrusion and the free-form face do not match the grid, the approximation error for the matching surfaces relative to the cell size is small (0.66%). Here the origin is not constrained by any symmetry plane, and we have freedom to shift it by any multiple of h.

6.5. Test example: reference grid with high coverage

In the second example (Fig. 12) the algorithm produced a good grid, which is 1/6th of the full width of this prismatic object. Looking at the faces colored dark, one can identify the faces matching the reference grid. The coverage value is 77.45%, the average approximation error relative to the bounding box is 0.25%. Here the origin gets well-defined by the three dominant symmetry planes, which drastically reduces the complexity of the computations.

7. Reflective symmetry planes

A very important issue in perfecting CAD models is how to detect and enforce constraints by symmetry planes. In the majority of cases, symmetries are only *partial*, however, these must be detected and enforced, as well. Let us denote a surface by s_i and its counterpart through a plane of symmetry P by $P(s_i)$. The measure of partial symmetry for two accurate surface portions s_1 and s_2 can be computed by the overlapping area, as

$$\frac{2Area(s_1 \cap P(s_2))}{Area(s_1) + Area(s_2)}.$$

In the case of full symmetry this yields 1.

In the reverse engineering context, we mostly deal with *approximate* symmetries due to measured data. The counterparts s_1 and $P(s_2)$ will not exactly fit onto each other, only within a certain tolerance ε, and we will need a measure of overlap by mapping the points of s_1 to $P(s_2)$ and vice versa, as will be described later.

We present the basic idea of our algorithm, but space limitation prevents us to discuss every detail. The algorithm is split into the following steps:

- Step 1: compute candidate planes between pairs of surfaces having acceptable measure of symmetry.
- Step 2: apply clustering to collect candidate planes, and rank these by special weights.
- Step 3: select a best symmetry plane and its matching surfaces, and refit by constrained fitting.
- Step 4: repeat the procedure for the next clusters to obtain further symmetry planes with coverage above a given threshold.

7.1. Candidate planes

In order to define candidates of symmetry planes, we pick two surfaces of the *same* type, including pairs of planes, cylinders, cones, and rotational surfaces. For a pair of planes p_i and p_j we define a candidate plane $P(p_i, p_j)$ by selecting a suitable bisecting plane. For a pair of rotational, cylindrical

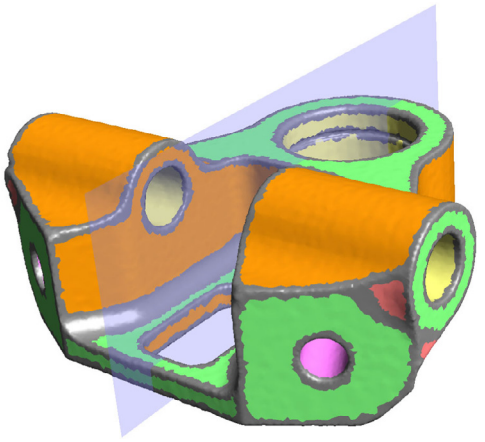

Fig. 13. Full symmetry.

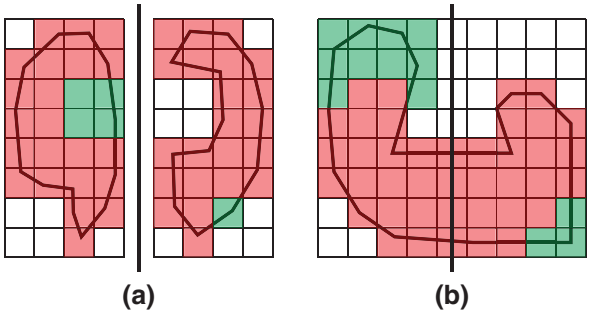

Fig. 14. Bitmatrix: (a) symmetry of two objects; (b) self symmetry. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

or conical surfaces r_i and r_j, we take their axis lines l_i and l_j. If these are in the same plane $\pi_{i,j}$, we compute a candidate plane $P(l_i, l_j)$ as a halving plane between the axis lines, which is also perpendicular to $\pi_{i,j}$. In the current algorithm, pairs of extruded or drafted extruded surfaces are not used to produce candidate planes, as it would require a more complex algorithm to determine an optimal map between two point clouds. One possible solution would be to apply ICP techniques, as recommended in [12].

7.2. Clustering

By definition: two planes are considered identical by tolerances (ε, δ), if their normal vectors span an angle less than ε, and the difference between their distances from the origin is less than δ. We build clusters by collecting candidate planes, which are pairwise identical by (ε, δ). In each cluster we sum up the *primary weights* that measure the extent of overlap for each associated surface pair. Here we also follow a cluster growing approach and process candidate planes one by one. If one is found close to an existing cluster we insert it, and increase the overall weight of the cluster; otherwise we open a new one. Eventually, for each cluster we assign a weighted sum of normals vectors, and a weighted sum of distances from the origin, and then rank the clusters by the sum of their primary weights.

In the next phase, we assign *secondary weights* to each clustered candidate plane P, that measures the "self-symmetry" of individual surfaces that do not belong to the generating surface pairs. These may include planar surfaces that are (i) orthogonal to P, rotational surfaces whose axis is (ii) contained or (iii) orthogonal to P, and extruded surfaces whose direction vector is (iv) contained or (v) orthogonal to P. As an example, take the object in Fig. 13 and the symmetry plane P, that was generated by two circular planar faces on the sides and lies in the x-z plane. This cluster will have relatively small primary weights, since the area of the corresponding face pairs is small. However, there are several large "supporting" faces that produce significant secondary weights. For example, the top and bottom planar faces are orthogonal to P, the axes of the vertical cylinders are contained in P, the direction of the side extrusions in the front and the back are contained in P, and so on.

In fact, we can add further *tertiary weights* to support a candidate plane by pairs of surface regions that have been discarded earlier. These include pairs of rotational surfaces with identical axes, or pairs of extruded surfaces with identical directions, that are perpendicular to the candidate plane. Such examples can also be found in Fig. 13: take the two symmetric horizontal cylindrical holes or the two symmetric extruded regions on the top.

The final ranking of the clusters will be computed by the sum of all associated weights. Each weight represents an overlap measure between a pair of faces or a mapping of a face onto itself. We need to emphasize that only surface pairs with large overlaps are considered; non-symmetric or weakly symmetric surface pairs are discarded. The same holds for self-symmetry. In the next section we describe how an approximate overlap measure can be computed using bitmatrices.

7.3. Computing an approximate measure of symmetry

Given a candidate symmetry plane and two surface regions, we need a computationally efficient method to determine the measure of approximate symmetry, and decide whether the given pair of surfaces should contribute to the current symmetry plane or not. All surfaces we are dealing with can be parameterized by a regular grid. For extruded and rotational surfaces we can take the grid of the curvature lines, for free-form surfaces the constant parameter lines provide such a grid. Having a given resolution, we can create a *discretized matrix representation* of each surface region by taking those cells, whose center points lie inside the boundaries of a given surface region. This provides an approximate area for the region and also facilitates overlap computation, which is based on mapping the center points of the cells from the first surface region to the second. If the mapped point is within tolerance to the second surface, this cell will be marked as contribution to the symmetry, and similarly we take the center points from the second surface and measure their approximate distance from the first one. This simple approach is shown in Fig. 14(a), where correspondence between two regions is shown. The cells that are found mutually symmetric are colored red, while the cells that fall out due to lack of symmetry are colored green.

The same algorithm can be applied for computing the self-symmetry measure of a single surface region related to a given candidate plane. Here the center points of the cells are mapped onto the surface itself and the approximate distances are computed accordingly. Such an example can be seen in Fig. 14(b) using the same coloring of the cells as before.

This approximation, of course, is very rough and depends on the resolution of the matrix, but it is computationally efficient, and does select those relatively large surface pairs that must be considered when calculating the symmetry weights while ranking the clusters.

7.4. Constrained fitting for symmetry planes

In the previous phases of the algorithm we ranked the candidate planes. Let us take the strongest and collect those surface elements that were found partially and approximately symmetric. Now we are ready for constrained fitting. The best symmetry plane is represented as an auxiliary object with a reference point p_0 and a normal vector n_0. Without enumerating all possible constraints for the different types of surface types, we remark that these are formulated related to the above auxiliary element. For planes we use reference points p_i, normal vectors n_i, and optionally distances from the symmetry plane d_i. For rotational surfaces reference points on the axes o_i, directions v_i and optionally distances d_i are needed. For extruded and drafted extruded surfaces we will need direction vectors v_i. All important constraints for determining the best symmetry plane can be easily formulated using the above entities. The most important constraint are the following:

(i) planar surface pair is symmetric,
(ii) single planar surface is orthogonal to symmetry plane,
(iii) cylindrical (rotational) surface pair is symmetric,
(iv) axis of a single cylindrical (rotational) surface is contained in the symmetry plane,
(v) axis of a single cylindrical (rotational) surface is orthogonal to the symmetry plane,
(vi) (drafted) extruded surface pair is symmetric,
(vii) direction of (drafted) extruded surface pair is orthogonal to the symmetry plane, etc.

Once constrained fitting is performed, an approximate *measure of symmetry* for the relevant surface set and a related *approximation error* for the corresponding data points can be determined.

7.5. Test example: single symmetry plane

We have already analyzed the object in Fig. 13 in the previous paragraphs. This plane is characterized by a relatively high value of symmetry. This object also demonstrates that in order to guarantee perfect symmetry for individual surfaces with profile curves special operations are needed. Symmetry can be guaranteed, if we mirror the points on the left to the right side, and vice versa, then the duplicated point set will be symmetric, and constrained fitting also produces perfectly symmetric profiles. For example, such a profile curve is generated for the extruded surfaces in the front and the back parts. Similar operations are needed when a common profile for a pair of symmetric surfaces needed to be created; for example, the final profile curve of the two extruded regions on the top must be generated based on a united point set. Symmetric free-form surfaces can also be generated by this principle, i.e., mirroring all the related points through the given symmetry plane and uniting these with the original points will yield a symmetric data set and thus a symmetric surface approximation.

Fig. 15. Approximate symmetry test. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

7.6. Test example: multiple symmetry planes

This object is a good example of strong partial symmetries, see Fig. 16. There are two such symmetry planes, the green one is symmetric by 86.74%, the blue one by 79.52%. The large planar faces are strongly symmetric, but (i) there is an extruded through-hole in the back side, facing two pairs of concentric cylinders in the front, and (ii) there are extruded

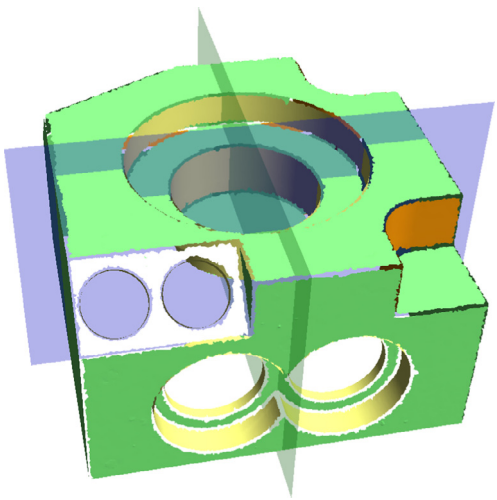

Fig. 16. Two partial planes of symmetry. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

surfaces at the right corner of the top face, facing chamfered faces with two small cylindrical holes on the left side. These explain the values of partial symmetry. The numerical overlap computation for the top face is illustrated in Fig. 15. It can be observed that the chamfered faces on the left and the corner extrusions on the right violate symmetry, but all the remaining parts (colored in red) are found symmetric.

8. Conclusion

A crucial issue in reverse engineering is to perfect objects that need to be reconstructed from measured data. We have presented various techniques that reset the parameters of independently fitted primary surfaces by applying constrained fitting, now for collections of surfaces. This approach is based on detecting "hypothetical"—local and global—constraints, that numerically qualify as relevant surfaces to be kept for computing best-fit structures by related constraints. Formerly developed algorithms have been extended for (i) setting automatically appropriate local constraint-sets; (ii) computing optimal coordinate systems for the whole object and for not-yet aligned subparts; (iii) computing reference grids in order to optimally snap dimensions, and (iv) determining symmetry planes for linking fully, or partially symmetric surfaces in a united system.

The algorithms have been implemented in a test program, which produced the pictures and the numerical results in the paper. For creating 3D test objects, we have used simulated and scanned data sets. These algorithms are fast (apart from the single-step grid algorithm), and produce results in a few seconds.

Future work is going to be directed towards generating common profile curves for constrained entities, handling general sweeps and lofted surfaces and adding constraints related to free-form geometry. Computational efficiency for large objects is also an important area of interest; we have already started to investigate an enhanced approach, where at the beginning only large faces are processed, and then smaller faces are added in a hierarchical manner.

Acknowledgment

Authors would like to thank György Karikó for discussing important algorithmic issues and participating in the development of the constrained fitting test program. Thanks are due to the anonymous reviewers for constructive comments. This research is supported by the Hungarian Scientific Research Fund (OTKA No. 101845).

References

[1] P. Benkő, G. Kós, T. Várady, L. Andor, R. Martin, Constrained fitting in reverse engineering, Comput. Aided Geometric Des. 19 (3) (2002) 173–205.
[2] T. Várady, R. Martin, Reverse engineering, Chapter 26, in: G. Farin, J. Hoschek, M.-S. Kim (Eds.), Handbook of Computer Aided Geometric Design, 2002, pp. 651–681.
[3] P. Marks, Capturing a competitive edge through digital shape sampling & processing (DSSP), SME Blue Book Ser., Society of Manufacturing Engineers (2005) (http://www.ewp.rpi.edu/hartford/users/papers/engr/ernesto/mannan/EP/References/DSSPBlueBook.pdf).
[4] N.J. Mitra, M. Pauly, M. Wand, D. Ceylan, Symmetry in 3d geometry: Extraction and applications, Comput. Graph. Forum 32 (6) (2013) 1–23.
[5] J.D. Wolter, T.C. Woo, R.A. Volz, Optimal algorithms for symmetry detection in two and three dimensions, Vis. Comput. 1 (1) (1985) 37–48.
[6] H. Zabrodsky, S. Peleg, D. Avnir, Symmetry as a continuous feature, Pattern Anal. Mach. Intell. IEEE Trans. 17 (12) (1995) 1154–1166.
[7] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, T. Funkhouser, A planar-reflective symmetry transform for 3d shapes, ACM Trans. Graph. (TOG) 25 (3) (2006) 549–559.
[8] S.J. Tate, G.E. Jared, Recognising symmetry in solid models, Comput. Aided Des. 35 (7) (2003) 673–692.
[9] K. Li, G. Foucault, J.-C. Léon, M. Trlin, Fast global and partial reflective symmetry analyses using boundary surfaces of mechanical components, Comput. Aided Des. 53 (2014) 70–89.
[10] R. Schnabel, R. Wahl, R. Klein, Efficient ransac for point-cloud shape detection, Comput. Graph. Forum 26 (2) (2007) 214–226.
[11] Y. Li, X. Wu, Y. Chrysathou, A. Sharf, D. Cohen-Or, N.J. Mitra, Globfit: Consistently fitting primitives by discovering global relations, ACM Trans. Graph. (TOG) 30 (4) (2011) 52.
[12] Y. Ke, S. Fan, W. Zhu, A. Li, F. Liu, X. Shi, Feature-based reverse modeling strategies, Comput. Aided Des. 38 (5) (2006) 485–506.
[13] Y. Liu, H. Pottmann, W. Wang, Constrained 3d shape reconstruction using a combination of surface fitting and registration, Comput. Aided Des. 38 (6) (2006) 572–583.
[14] V. Pratt, Direct least-squares fitting of algebraic surfaces, ACM SIGGRAPH Comput. Graph. 21 (4) (1987) 145–152.
[15] G. Lukács, R. Martin, D. Marshall, Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation, in: Computer Vision ECCV'98, Springer, 1998, pp. 671–686.
[16] F.C. Langbein, A.D. Marshall, R.R. Martin, Choosing consistent constraints for beautification of reverse engineered geometric models, Comput. Aided Des. 36 (3) (2004) 261–278.
[17] M. Li, F.C. Langbein, R.R. Martin, Detecting approximate symmetries of discrete point subsets, Comput. Aided Des. 40 (1) (2008) 76–93.
[18] M. Li, F.C. Langbein, R.R. Martin, Detecting design intent in approximate cad models using symmetry, Comput. Aided Des. 42 (3) (2010) 183–201.