Available online at www.sciencedirect.com
ScienceDirect
Computer-Aided Design 39 (2007) 1025–1036
COMPUTER-AIDED DESIGN
www.elsevier.com/locate/cad

Constraint-based beautification and dimensioning of 3D polyhedral models reconstructed from 2D sketches

H.L. Zou, Y.T. Lee*

School of Mechanical and Aerospace Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore

Received 18 October 2006; accepted 22 August 2007

Abstract

3D models reconstructed from 2D sketches are inaccurate because of the inherent inaccuracies in the input and the reconstruction method. It is therefore necessary to "beautify" them before use in CAD systems. We present a method that detects geometric constraints, such as parallel and orthogonal faces, present in the reconstructed model and then selects a subset that constrains the object sufficiently and consistently. The subset selection algorithm first prioritizes the constraints depending on their type and then uses a novel method, based on quasi-Newton optimization, to detect and eliminate redundant and inconsistent constraints. The remaining constraints then define the dimensions of the model fully and consistently. Results from our implementation show that the method can beautify and dimension recovered 3D models correctly at acceptable speed.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Sketching; 3D reconstruction; Beautification; Dimensioning; Constraints

1. Introduction

One active research area in computer-aided conceptual design concerns a designer sketching a design in 2D and a 3D model recovered automatically from the sketch [1–4]. Existing work is mainly restricted to polyhedral objects with planar faces only, which is also the case for this paper. Here all the edges of the object are present in the sketch and all the lines in the sketch represent edges. The sketch is first cleaned up to remove gaps at the vertices. A 3D model as perceived in human minds is then recovered, but it is inaccurate because the sketch, by its very nature, is inaccurate. The recovery method also affects the accuracy of the model. For example, the vertices in one face may not lie exactly on one plane and parallel faces may not be exactly parallel. Also, the size of the recovered model usually bears no resemblance to the actual size the designer has in mind, as a 2D sketch carries no dimensions. Improvement on the recovered model is therefore necessary to enforce proper geometric relationships and give it proper dimensions before the model can be used further. Fig. 1 illustrates the main stages.

It is neither practical nor desirable to require the user to provide the data for every entity – vertex, edge or face – present. One solution is to set up the constraints between the entities, and resolve the dimensions based on these constraints computationally. Many of the constraints can be established automatically, but there may be some that need to be given by the user. Clearly, it is desirable to minimize the user input on such a potentially tedious and error-prone task.

A similar problem arises in reverse engineering where a 3D model is recovered from a point cloud, although the inaccuracy here is less severe. Langbein [5] presented a method in which potential geometric constraints, such as parallelism, planarity and orthogonality, present in a 3D reverse-engineered model are found approximately at first, followed by a graph-based method to detect inconsistencies between these constraints. The result is a consistent subset of the potential constraints on the initial model. This subset, when resolved numerically, leads to an improvement in the geometry of the model. Langbein's study shows that the graph-based method works well and fast in detecting inconsistencies.

One fast and simple step in the beautification of an object recovered from a sketch is to enforce planarity on its faces. Chen [6] solves this problem using a least-square method to adjust vertices of a face on to the same plane. But he did not

* Corresponding author. Tel.: +65 6790 5493; fax: +65 6791 1859.
E-mail addresses: zouhongliu@pmail.ntu.edu.sg (H.L. Zou), mytlee@ntu.edu.sg (Y.T. Lee).

0010-4485/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2007.08.002

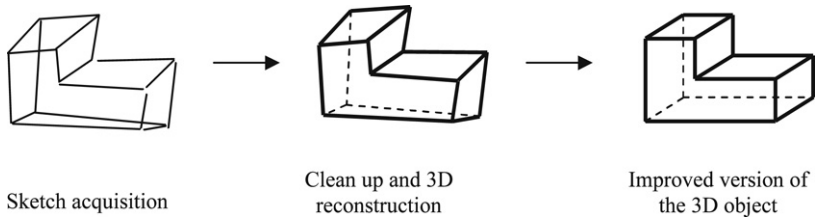

Fig. 1. Conceptual design by sketching.

take into account the effects of other constraints, such as faces being parallel.

Varley [4] beautified the 3D model reconstructed from a line drawing by determining the face normals and distances between faces separately using an optimization method.

Wilczkowiak et al. incorporated camera calibration and 3D reconstruction of 3D models built from images in computer vision [7]. The constraints, stored in a graph, are used to find a smaller set of input parameters and functions that yield all the parameters which are used in a subsequent refinement step. The approach is based on a dictionary of rules, called r-methods, that relate geometric entities through constraints, represented as a graph. Attempts are made iteratively to match the graph of an r-method to a subgraph of the constraint graph of the object, in which the entities are the nodes and the constraints are the edges. Then a reverse sequence of the r-methods identified is executed to solve the constraints. Implicit constraints are not considered because the system cannot contain redundancies.

The approaches described above beautify a 3D model, but the improved model carries no proper dimension. Martínez and Félez presented a constraint-based solver to provide a completely dimensioned 2D part [8]. Their method establishes the constraints from two or three sketches of different views of a model, and chooses a set of independent constraints of the system by determining if the system is over-constrained. The constraints form a system of equations, the Jacobian of which, when solved, reveals the redundant constraints. The work does not deal with dimensioning a 3D model.

A set of constraints needs to be sufficient to describe a model completely. There can be over-constraint or under-constraint, and there may also be redundancies, which include structural and numerical redundancies. A structural redundancy over-constrains the system. For instance, $f(x_1, x_2) = 0$, which constrains two variables x_1 and x_2 in a system, will lead to structural redundancy if two other constraints $g_1(x_1, x_2) = 0$ and $g_2(x_1, x_2) = 0$ are also present, since the values of x_1 and x_2 are implicitly determined already by g_1 and g_2. The problem can be rectified by discarding one of the constraints. A system can be numerically inconsistent or redundant. For example, two constraints expressed by $x_1 + x_2 = 1$ and $x_1 + x_2 = 0$ are inconsistent and one of $x_1 + x_2 = 1$ and $2x_1 + 2x_2 = 2$ is redundant.

Buchanan [9] determined whether a system of equations is inconsistent by using the Gröbner basis. Gao [10] gave a complete method for deciding whether the constraints in a set are independent and whether they are numerically inconsistent based on Wu–Ritt's [11] decomposition algorithm. Despite their advantages, both the Gröbner basis and the Wu–Ritt method require exponential time complexity. It is not uncommon for them to take tens of minutes or even hours, which is not acceptable in a real time interactive system.

Numerous researchers have addressed the problem of structural redundancies. In 2D, the triangle decomposition method was used by some researchers in geometric constraint solving [10,12–15]; their methods for identifying over-constrained subgraphs were based on triangle decomposition too. For example, Fudos's method [12] concludes that if two well-constrained clusters share more than one geometric element, then over-constraint exists. But the method can be used only within a limited domain and cannot be applied in 3D. Latham [16] proposed a method based on the maximum b-matching algorithm to decompose a constraint problem into a sequence of solvable subgraphs. The crux of this algorithm is the detection and correction of over-constraints and under-constraints. Maximum b-matching is a special case in generalized maximum matching (MM), of which Hoffmann stated [17], "the MM method may or may not detect over-constrained subgraphs, depending on the initial choice of vertices for reducing weights". He proposed a method MM1 to correct this drawback, but did not deal with 3D cases. Li's method [18] can detect over-constraint in 3D, but all the entities in the constraint graph must have six degrees of freedom. So his algorithm cannot be used in cases where the entities are points, lines and planes, which do not have six degrees of freedom. Recently, Langbein [5] proposed a method of identifying over-constraint based on Kramer's degree of freedom analysis [19] and Li's method of analyzing dependencies between geometric objects [18]. Jermann described a new concept of extensive structure rigidity [20,21], which is useful and is used in our algorithm.

There are limitations in redundant constraint detection using graph-based methods. Graphs have been used to detect structurally redundant constraints without solving the constraint system [5,16,18,22,23]. However, some constraints are implicit; they can be inferred from a set of constraints in a graph, but are not explicitly represented. Graph-based methods therefore cannot be used to detect all redundant constraints. We demonstrate it using two examples.

Fig. 2 illustrates Pappus's Theorem in 2D: If A, B, and C are three points on one line, D, E, and F are three points on another line, and AE meets BD at X, AF meets CD at Y, and BF meets CE at Z, then the three points X, Y, and Z are collinear [24]. The collinear constraint can be inferred from the given set of constraints. The constraint graph of the theorem, which includes all the points, lines, intersections and collinear constraints, is an under-constrained system, and there are no

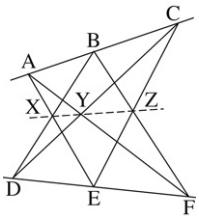

Fig. 2. Pappu's theorem.

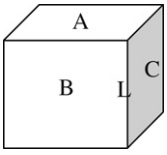

Fig. 3. An example of an implicit constraint in 3D.

over-rigid subgraphs or redundant constraints given the graph, though in fact the collinear constraint is redundant.

Fig. 3 illustrates another example in 3D: Let A, B and C be three faces of a block. B and C are perpendicular to A, and if Line L is the intersection of B and C, then L is also perpendicular to A.

L being perpendicular to A can be inferred from it being the intersection of B and C, and therefore would be redundant in a constraint system that includes both.

In a constraint system, many constraints can be inferred from the existing ones.

A constraint system can be expressed as a system of equations [25], which can be solved numerically. This solution accounts for a large proportion of the cost in geometric constraint solving, which is directly proportional to the size of the system. To minimize this cost, the constraint system is decomposed into smaller sub-systems [17], which are solved separately. The decomposition of constraints requires analyzing the relationships between the weights of entities and weights of constraints, without considering the properties of the entities and constraints [16]; implicit constraints are not considered. In the sequence of the partially ordered solvable subgraphs, a subgraph may include new constraints which can be inferred from previously solved subgraphs in an under-constrained system. New inferred constraints affect the solution to a subgraph, and hence the solution to the whole system. Therefore, before knowing if a system is well constrained, we cannot ignore the implicit constraints in redundancy detection or decomposition of constraints. When and which implicit constraints are selected for the decomposition is yet another problem.

It is necessary to obtain the inferred constraints in a constraint system. Redundant constraint detection requires the relationships between entities in the known constraint sets. Theorem proving [26–28] can be used to answer "true" or "false" to a specific geometry statement such as Pappus's Theorem. But it is not possible to use Gröbner basis techniques or Wu's method in redundant constraint detection because both have exponential time complexity. The extension of their methods to 3D cases is possible but expensive.

Because graph based methods can only detect redundancy from constraints explicit in a graph, a new method is needed for the detection in a graph with implicit constraints.

An increasing amount of literature is devoted to detecting redundancy by numerical methods. Li [29] and Langbein [30] determined the numerical redundancies of a 3D geometric constraint system by a disturbance method. If the system is unsolvable after adding the disturbance value to a constraint, then the constraint is redundant or inconsistent. If redundancy exists, then the Jacobian matrix of the system of equations is singular [25]. In the presence of redundancy, that is, when there is a set of mutually dependent constraints, Martínez [8] breaks the redundancy by removing a constraint from the set of dependent constraints. The methods of Light and Martínez work well on 2D cases, but extension to 3D cases is still an open issue.

The main objective of this paper is the "beautification" of 3D planar polyhedral models reconstructed from 2D sketches, to get the models close to the shapes the designers intend, and give the models proper dimensions. We first introduce the types and expressions for the geometric constraints and the method for finding them, followed by the procedure for selecting a consistent constraint subset from those found. Finally, this constraint subset is used to beautify and dimension the recovered model. Some experimental results are presented.

2. Constraint system

A geometric constraint system contains a set of entities and a set of constraints associated with the entities. The entities are defined by variables and the constraints specify a system that allows these variables to be determined.

2.1. Entities

The entities are faces, edges and vertices.

A vertex is represented by a three-component vector $\mathbf{p}(x, y, z)$. (We use bold to represent vectors and italics for scalars.)

An edge lies on a straight line which has four degrees of freedom (two translations and two rotations), represented by six variables and two equations:

$$a^2 + b^2 + c^2 = 1$$

and

$$ax + by + cz = 0$$

where the position vector $\mathbf{p}(x, y, z)$ is chosen to be the point on the edge nearest to the origin and $\mathbf{d}(a, b, c)$ is the unit vector of the edge direction.

A face lies on a plane which has three degrees of freedom (one translation and two rotations), represented by a system with six variables and three equations. The six variables are accounted for in the unit face normal $\mathbf{d}(a, b, c)$ and the position vector $\mathbf{p}(x, y, z)$, which is the nearest point on the face to the origin. $\mathbf{d}$ and $\mathbf{p}$ must be parallel vectors, which means $\mathbf{d} \times \mathbf{p} = \mathbf{0}$. This gives rise to three equations $ay - bx = 0, az - cx = 0$

Table 1
Representation of entities

Entity	Parameters	Equations to define the entity
Vertex	Location $\mathbf{p}(x, y, z)$	–
Edge	Edge direction $\mathbf{d}(a, b, c)$	$a^2 + b^2 + c^2 = 1$
	Edge position $\mathbf{p}(x, y, z)$, the point on the edge nearest to the origin	$ax + by + cz = 0$
Face	Face normal $\mathbf{d}(a, b, c)$	$ay - bx = 0$
	Face location $\mathbf{p}(x, y, z)$, the point on the face nearest to the origin	$az - cx = 0$
		$a^2 + b^2 + c^2 = 1$

and $bz - cy = 0$, only two of which, say the first two, are independent. The unit face normal requires that

$$a^2 + b^2 + c^2 = 1,$$

thus forming a system of three independent equations. Table 1 shows the equations that define the entities and their parameters.

2.2. Constraint equations

There are two forms of constraints: topological constraints, which define the connectivities between entities, and geometric constraints, such as distance or angle between entities in a model. Our system specifies only two topological constraints "vertex on edge" and "vertex on face" directly. Other constraints, such as edge on face and face/face adjacency, are not specified because they can be inferred. For instance, the two end points of an edge lying on a face are sufficient to ensure that the edge lies on the face.

A geometric constraint is represented by two positions and one or two directions as shown in Table 2, in which $\mathbf{p_1}$ is the point on Entity 1 closest to the origin and $\mathbf{d_1}$ is a unit direction vector, which is the normal for a plane and edge direction for an edge; $\mathbf{p_2}$ and $\mathbf{d_2}$ are the corresponding values for Entity 2. The weight of a constraint is the number of degrees of freedom (DOF) eliminated by the constraint, which is the number of equations required to define that constraint. For example, the distance between two vertices requires one equation that eliminates one degree of freedom, thus its weight is 1. The sign $\pm$ in the table indicates that a value can be either positive or negative, and the choice depends on the initial values of the parameters. For example, the distance between a vertex $\mathbf{p_1}$ and a face $(\mathbf{p_2}, \mathbf{d_2})$ is $|(\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2}|$, so the equation describing the constraint is $(\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2} = \pm h$, where h is a known value. The sign of this value depends on which half space of the face the vertex is in.

2.3. Detecting the constraints

Given a 2D sketch and its reconstructed 3D model, we need to automatically detect all the constraints that may be used to define the model. The types of constraint to be detected are those listed in Table 2. Some constraints may be introduced by the user too, if required.

A constraint is a geometric relationship between two entities, which can be expressed quantitatively. For example, how close two lines are to being parallel can be determined by the angle between the lines. The existence of a constraint is established by checking if the quantity is within a certain threshold. This threshold encompasses the tolerance which must be given to take account of the inaccurate nature of the input. For example, a parallelism constraint may be applied to lines within 7°, say, of each other. This tolerance value has a strong influence on the eventual outcome of the beautification, including the closeness of the beautified object to the original, because it dictates the existence of certain constraints. So long as a sufficient set of constraints that fully constrains the model can be found, a result

Table 2
Constraint equations

Geometric constraint		Constraint equations	Weight
Parallel	Edge–edge $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} = \pm\mathbf{d_2}$	2
	Face–face $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} = \pm\mathbf{d_2}$	2
	Edge–face $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = 0$	1
Perpendicular	Edge–edge $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = 0$	1
	Face–face $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = 0$	1
	Edge–face $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} = \pm\mathbf{d_2}$	2
Distance (h)	Vertex–vertex $(\mathbf{p_1} - \mathbf{p_2})$	$\lvert\mathbf{p_1} - \mathbf{p_2}\rvert = h$	1
	Vertex–edge $(\mathbf{p_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\lvert(\mathbf{p_1} - \mathbf{p_2}) - ((\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2})\mathbf{d_2}\rvert = h$	1
	Vertex–plane $(\mathbf{p_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$(\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2} = \pm h$	1
	Edge–edge $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} = \pm\mathbf{d_2}, \lvert\mathbf{p_1} - \mathbf{p_2}\rvert = h$	3
	Edge–plane $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = 0, (\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2} = \pm h$	2
	Plane–plane $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} = \pm\mathbf{d_2}, (\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2} = \pm h$	3
Angle (α)	Edge–edge $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = \pm\cos(\alpha)$	1
	Face–face $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = \pm\cos(\alpha)$	1
	Edge–face $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = \pm\sin(\alpha)$	1
Position (required by topology of model)	Vertex lie on plane $(\mathbf{p_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$(\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2} = 0$	1
	Vertex lie on edge $(\mathbf{p_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\lvert(\mathbf{p_1} - \mathbf{p_2}) - ((\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2})\mathbf{d_2}\rvert = 0$	2
	Edge lie on plane $(\mathbf{p_1}, \mathbf{d_1}$–$\mathbf{p_2}, \mathbf{d_2})$	$\mathbf{d_1} \cdot \mathbf{d_2} = 0, (\mathbf{p_1} - \mathbf{p_2}) \cdot \mathbf{d_2} = 0$	2

$\mathbf{p_1}$ is the point on Entity 1 closest to the origin and $\mathbf{d_1}$ is a direction vector of Entity 1. $\mathbf{p_2}$ and $\mathbf{d_2}$ are the corresponding values for Entity 2.

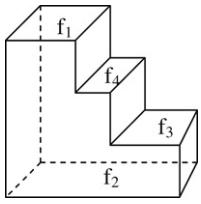

Fig. 4. A parallel face set.

Table 3
Priority values of constraints, a higher value denotes a higher priority

Entity 1	Entity 2	Constraint type	Priority
Vertex	Vertex	Distance between two vertices	1
Vertex	Edge	Distance between vertex and edge	1
		Vertex lies on edge	5
Vertex	Face	Distance between vertex and face	2
		Vertex lies on face	5
Edge	Edge	Angle between two intersecting edges	1
		Parallel & distance between two edges	2
Edge	Face	Parallel & distance between edge and face	3
		Angle between edge and face	1
		Edge and face perpendicular	2
Face	Face	Angle between two faces	3
		Two perpendicular faces	4
		Parallel & distance between two faces	4

can be obtained for the beautification. The establishment of sufficiency is dealt with in Section 3.

We use the hierarchical clustering algorithm presented in [31] to find groups of entities that are close to each other in some properties. For example, for parallel faces, the clustering into groups proceeds by a series of fusions of faces whose normal vectors are nearly parallel. The normal vectors in a group have a median, and the closeness between groups is measured by the difference between the group medians.

After establishing the parallel face groups, the faces in each group are sorted. In the example in Fig. 4, faces f_1, f_2, f_3 and f_4 are found to be parallel to each other. They are sorted in the order f_1, f_4, f_3 and f_2, according to their position. Then one of the end faces, f_1 or f_2, is set to be the base face for dimensioning, and only the parallel constraints between the faces in the group and the base face are added to the list of constraints for the object.

Rules such as "two faces are perpendicular if their normals are almost perpendicular" are defined for perpendicular constraints. The constraints of distance and angle are calculated by the equations listed in Table 2.

Choosing the correct type of constraint is essential to establishing the shape [8]. Certain types of constraint are more likely to occur than others. Hence we establish the priority of a constraint according to its type. Table 3 shows the priority values defined for the types in our system. A higher value denotes a higher priority. The choice of the values here is partly based on the values given in [5,8] and partly through our own experience.

3. Selecting and solving the constraints

In commercial CAD systems, the dimensions, which are constraints, of a 3D model need to be given by the user. It is up to the user to provide a complete set of consistent dimensions.

In our system, a large number of constraints are detected using the method described in Section 2.3. The user may also choose to introduce a constraint directly. Only a subset of these constraints is needed to define the model completely. The result should be a well-constrained system containing all the entities and the selected constraints.

Topological constraints – "vertex on face" and "vertex on edge" – are a part of the structure of a model and can be identified correctly and directly from the model.

Geometric constraints are selected subsequently, with constraints of higher priority selected first. A new constraint is accepted if it is not redundant or inconsistent with the existing ones. Otherwise, this new constraint or one of the existing ones is eliminated. Redundancy may be structural or numerical. This process repeats until the difference between the sum of the weights of the entities and the sum of the weights of the constraints is six, because a 3D model has six degrees of freedom, three rotations and three translations. Finally, the constraint equations are solved and the new model is obtained by updating the parameters of the entities.

3.1. Solving the constraints

Given a set of entities and constraints, we want to derive the values of the parameters that define the entities, which satisfy all the constraints. When a system is well constrained or even under-constrained, the parameters can be found. In this section, we focus on solving the constraints through numerical optimization.

Let $f_k(\mathbf{x}) = 0$ $(k = 1 \ldots m)$ be a set of m possibly non-linear equations defining the constraints described earlier, where the variable $\mathbf{x}$ is a vector of the parameters defining the entities of the model (see Table 1). Typically, $\mathbf{x}$ contains many elements, the number of which increases with the complexity of the model. This is therefore a multivariate optimization problem. Given the m equations, a solution for $\mathbf{x}$ can be found using a least-square method, by minimizing the objective function:

$$\sum_{k=1}^{m} (f_k(\mathbf{x}))^2 .$$

The initial value of $\mathbf{x}$ for the optimization can be obtained from the reconstructed model; the solution at each step of the optimization forms the input to the next.

When the objective function value approaches zero, the optimization approaches a configuration of the model that satisfies all the constraints. Failure to arrive at zero means that the constraint system is not solvable, which means there are conflicting constraints in the system.

Some global optimization techniques, such as genetic algorithms, simulated annealing, branch and bound [32], are effective even when the objective function is very complex, but

they are highly time consuming, and therefore are not suitable for an interactive real-time application, which we intend our system to be.

We use the BFGS (Boyden–Fletcher–Goldfarb–Shanno) method [33], which is mature, stable and effective. Derived from the method of steepest descent, BFGS is a quasi-Newton method based on the idea of reconstructing a quadratic approximation of a function from values of its gradients at a number of points, leading to a better approximation of the minimum value. The time complexity of BFGS is $O(n^2)$ per iteration, where n is the number of variables. But the number of iterations is indeterminate, as it varies with each execution.

We turn now to the details of checking if the last constraint selected is redundant or inconsistent.

3.2. Eliminating structurally redundant constraints

As had already been mentioned, it is difficult for graph-based methods to detect all the numerically redundant constraints, but they can detect the structurally redundant ones without solving the constraint equations. If a constraint is structurally redundant, then it must be numerically redundant also. Hence, it is useful to use a graph-based method to detect structural redundancy. We have published this method elsewhere [23]; a summary is given below.

A geometric constraint system can be represented by a constraint graph $G = (V, E)$, where V is the set of nodes representing the entities, including points, lines and planes, and E is the set of arcs representing constraints between the entities. Basically, it uses a flow-based algorithm to identify the existence of structural redundancy and over-constrained subgraphs in a 2D or 3D constraint system by distributing the weights of the constraints to the DOF (degree of freedom) of the entities, where the weight of a constraint is the number of DOFs it eliminates. It exploits the degree of freedom and degree of rigidity properties of subgraphs, and identifies the structurally over-constrained by adding the constraints one by one to the entities in the graph. If the weight of the detected constraint cannot be distributed to the constraint system, then it is inside an over-constrained subgraph, i.e. the detected constraint is redundant or inconsistent.

The complexity of our graph-based method to detect the over-constrained subgraph is $O(m^2(n + m/2))$, where n is the number of entities and m the number of constraints in the graph [23].

3.3. Detecting numerical redundancy

A set of constraints can be expressed by a set of equations, $f(\mathbf{x}) = 0$, where $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of n variables. More explicitly, the equation can be written as

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \quad\vdots \\ f_m(x_1, \ldots, x_n) = 0 \end{cases}$$

where m is the number of constraints. The Jacobian matrix is then

$$\mathbf{J}(x_1, \ldots x_n) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}} \end{bmatrix}.$$

If a constraint, with equation $g_k(\mathbf{x}) = 0$, is redundant, then it can be deduced from a subset of $f(\mathbf{x}) = 0, g_1(\mathbf{x}) = 0, \ldots, g_l(\mathbf{x}) = 0, k \notin 1 \ldots l$, say. We can express $g_k(\mathbf{x})$ as a function of $g_1(\mathbf{x}), \ldots, g_l(\mathbf{x})$:

$$g_k(\mathbf{x}) = F(g_1(\mathbf{x}), \ldots, g_l(\mathbf{x})). \tag{1}$$

When $g_1(\mathbf{x}) = 0, \ldots, g_l(\mathbf{x}) = 0$, then $g_k(\mathbf{x}) = 0$. By differentiating both sides of Eq. (1), we obtain

$$\frac{\partial g_k}{\partial \mathbf{x}} = \frac{\partial F}{\partial g_1}\frac{\partial g_1}{\partial \mathbf{x}} + \frac{\partial F}{\partial g_2}\frac{\partial g_2}{\partial \mathbf{x}} + \cdots + \frac{\partial F}{\partial g_l}\frac{\partial g_l}{\partial \mathbf{x}}. \tag{2}$$

When $f(\mathbf{x}) = 0$, that is, $g_1(\mathbf{x}) = 0, \ldots, g_l(\mathbf{x}) = 0$, $\frac{\partial F}{\partial g_1}, \frac{\partial F}{\partial g_2}, \ldots, \frac{\partial F}{\partial g_l}$ must be constants. To simplify the notation, we denote $\frac{\partial F}{\partial g_1}$ by c_1, $\frac{\partial F}{\partial g_2}$ by c_2, …, $\frac{\partial F}{\partial g_l}$ by c_l. So Eq. (2) becomes

$$\frac{\partial g_k}{\partial \mathbf{x}} = c_1\frac{\partial g_1}{\partial \mathbf{x}} + c_2\frac{\partial g_2}{\partial \mathbf{x}} + \cdots + c_l\frac{\partial g_l}{\partial \mathbf{x}}. \tag{3}$$

Thus, if an equation of a constraint $g_k(\mathbf{x}) = 0$ in the system is numerically redundant, its corresponding row vector in the Jacobian matrix can be expressed as a linear combination of the other row vectors, and the Jacobian matrix is singular as well.

We add the constraints with the highest priority to the system one by one. So the last row of the Jacobian matrix corresponds to the newly added constraint. The condition for a non-redundant constraint is that the Jacobian is non-singular. Let $f_m(\mathbf{x}) = 0$ be the last constraint added and $\boldsymbol{J}_1, \boldsymbol{J}_2 \ldots \boldsymbol{J}_m$ be the row vectors of the Jacobian matrix. If the last constraint is redundant, then $\boldsymbol{J}_1, \boldsymbol{J}_2 \ldots \boldsymbol{J}_{m-1}$ of the Jacobian matrix are linearly independent and $\boldsymbol{J}_1, \boldsymbol{J}_2 \ldots \boldsymbol{J}_m$ are linearly dependent. So we can write

$$\boldsymbol{J}_m = c_1\boldsymbol{J}_1 + c_2\boldsymbol{J}_2 + \cdots + c_{m-1}\boldsymbol{J}_{m-1.}$$

That is, $[\boldsymbol{J}_1^{\mathrm{T}}, \boldsymbol{J}_2^{\mathrm{T}} \ldots \boldsymbol{J}_{m-1}^{\mathrm{T}}, \boldsymbol{J}_m^{\mathrm{T}}][c_1, c_2 \ldots c_{m-1}, -1]^{\mathrm{T}} = 0$, where $c_1, c_2 \ldots c_{m-1}$ are scalars.

We detect the redundancy by solving the equation

$$\boldsymbol{J}^{\mathrm{T}}\mathbf{c} = 0, \tag{4}$$

where $\mathbf{c} = [c_1, c_2 \ldots c_{m-1}, -1]^{\mathrm{T}}$ is a vector of the unknowns and $\boldsymbol{J}^{\mathrm{T}}$ is an $n \times m$ matrix. If there is no $\mathbf{c}$ for which Eq. (4) holds, then the m row vectors are linearly independent. Otherwise, they are linearly dependent, because then we can express the last row vector of $\boldsymbol{J}$ as a linear combination of the other row vectors.

We need to solve Eq. (4), in which $\boldsymbol{J}^{\mathrm{T}}$ is n by m, with $n \geq m$ always. There are more equations than unknowns, so the system is over-determined. We will now briefly review the techniques related to solving a sparse over-determined system.

A direct method may often be preferable to an interactive method to solving an over-determined problem [34]. Normally, methods based on normal equations and QR factorization are used to solve an over-determined problem. Golub's investigation into solving over-determined systems showed that the flop (floating point operation) of the method via normal equations is $mn^2 + n^3/3$ and that via Householder orthogonalization is $2mn^2 - 2n^3/3$ when the matrix in the system is $m \times n$ [35]. For a full rank system, it is difficult to choose the right algorithm from the two. Note also that $\boldsymbol{J}^{\mathrm{T}}$ may be rank deficient when the last constraint is redundant, which raises another issue about selecting a numerical rank determination technique. QR factorization can be used to determine the rank of $\boldsymbol{J}^{\mathrm{T}}$; the method based on normal equations will fail in rank deficient cases.

In our system, $\boldsymbol{J}^{\mathrm{T}}$ may be rank deficient. There are a number of ways to compute the rank of a matrix. The singular value decomposition algorithm is very time consuming, so Bischof [36] developed an algorithm for rank revealing orthogonal factorization. His experimental results showed his approach performs up to three times faster than the less reliable QR factorization with column pivoting, and comes within 15% of the speed for computing a QR factorization without any column exchange.

The Jacobian matrix is a sparse matrix with about nine non-zeros in each row on average. The exact cost of solving the equations depends strongly on the structure of the problem, and efficient computation of sparse rank-revealing decompositions and sparse QR decomposition is an open area of research. Our experiments using Matlab [37], with a sparse 500×400 matrix with 4500 non-zero elements, show that the run time for sparse QR decomposition is almost the same as that of QR decomposition of a dense matrix. When the size of the matrix is smaller, QR decomposition of a dense matrix is much faster than sparse QR decomposition.

Therefore, the solution of the linear system is computed using QR factorization for a dense matrix in our system. The idea of factoring $\boldsymbol{J}^{\mathrm{T}} = QR$ is used, where Q is orthogonal and R is the upper triangular, and Q is n by n and R is n by m. Then $\boldsymbol{J}^{\mathrm{T}}\mathbf{c} = QR\mathbf{c} = 0$ is solved by

$$R\mathbf{c} = 0, \tag{5}$$

where $\mathbf{c} = [c_1, c_2 \ldots c_{m-1}, -1]^{\mathrm{T}}$, is a vector of the unknowns.

In contrast to general rank detection of a matrix, we know in advance that the last equation introduces the redundancy. So we detect the redundancy by simply using R_{mm}, which is the entry in the mth row and mth column of the upper triangular matrix R. When the magnitude of R_{mm} is not zero, it is impossible for Eq. (4) to have a solution, because the mth equation of the set in Eq. (5) is $R_{mm} * (-1) = 0$.

If R_{mm} is not zero, then Eqs. (4) and (5) have no solution; that means the last row vector of the Jacobian matrix is not a linear combination of the other row vectors and the last constraint is not redundant. When rank deficiency exists, we can detect it from the R of the QR factorization, without solving the equations. That is another reason for choosing QR factorization.

In our method, the redundancy or rank of the matrix $\boldsymbol{J}$ is determined by QR factorization of $\boldsymbol{J}$ only, without using another special technique for rank revealing such as singular-value decomposition or QR with column pivoting. In our system, Householder transformations are utilized for QR factorization.

The flow chart for determining if the last constraint is redundant or inconsistent is shown in Fig. 5.

If the last constraint is found to be structurally redundant by the graph-based method, it is rejected directly without invoking the numerical method described above. Otherwise, the action moves to the numerical stage. When a new constraint is added to the system, we need to solve the system of constraint equations to obtain the values of the entries in the Jacobian matrix. Because the cost for QR factorization of the Jacobian matrix is small compared to that for solving the equations, we try to avoid solving the equations. But this is possible only when the last constraint is a distance or angle between entities, because the values of such constraints can be obtained from the existing entities in the model.

The complexity of QR factorization is $O(2mn^2 - 2n^3/3)$, where m is number of variables and n is number of constraints in the Jacobian matrix [35].

3.4. A simple example for redundancy detection

A simple example is given below to demonstrate the general working principles in redundancy detection. The example object has three faces, nine edges and seven vertices (Fig. 6). Using Table 1, we see that these entities contain $3 \times 6 + 9 \times 6 + 7 \times 3 = 93$ variables and $9 \times 2 + 3 \times 3 = 27$ constraints within themselves. There are $4 \times 3 = 12$ "vertex on face" and $2 \times 9 = 18$ "vertex on edge" topological constraints; the weight of the former is 1 and that of the latter is 2 (see Table 2), thus giving a total of $12 + 2 \times 18 = 48$ topological constraints, which are identified automatically directly from the structure of the graph. Hence there are $27 + 48 = 75$ constraints in the structure. We demonstrate redundancy detection by adding three other constraints one by one (thus giving 78 constraints in all). Two of these constraints, Face f_0 perpendicular to Face f_1, f_0 perpendicular to Face f_2, are introduced first. There are no structural or numerical redundancies, so the two constraints are accepted.

The final constraint is Edge e_6 perpendicular to f_0. It is not structurally redundant; we thus focus on numerical redundancy. The variables and the constraints result in a 78×93 Jacobian matrix. Table 4 illustrates its structure, in which asterisks denote components which are not zero. To simplify the table, we use the entity name such as v_0 to represent all the parameters of each entity, instead of the parameters themselves. Each row indicates the row vector for the equation of a constraint. The inherent constraints of an edge or a face are the equations defining the entity given in Table 1, and their corresponding row vectors are in the top rows of Table 4. These are followed by the row vectors of the equations for the topological constraints. The last three rows are the vectors for the equations of the two vertical faces and the vertical edge between the two faces.

After the QR factorization of the Jacobian matrix, the element $R_{78,78}$, which is the entry in the 78th row and 78th

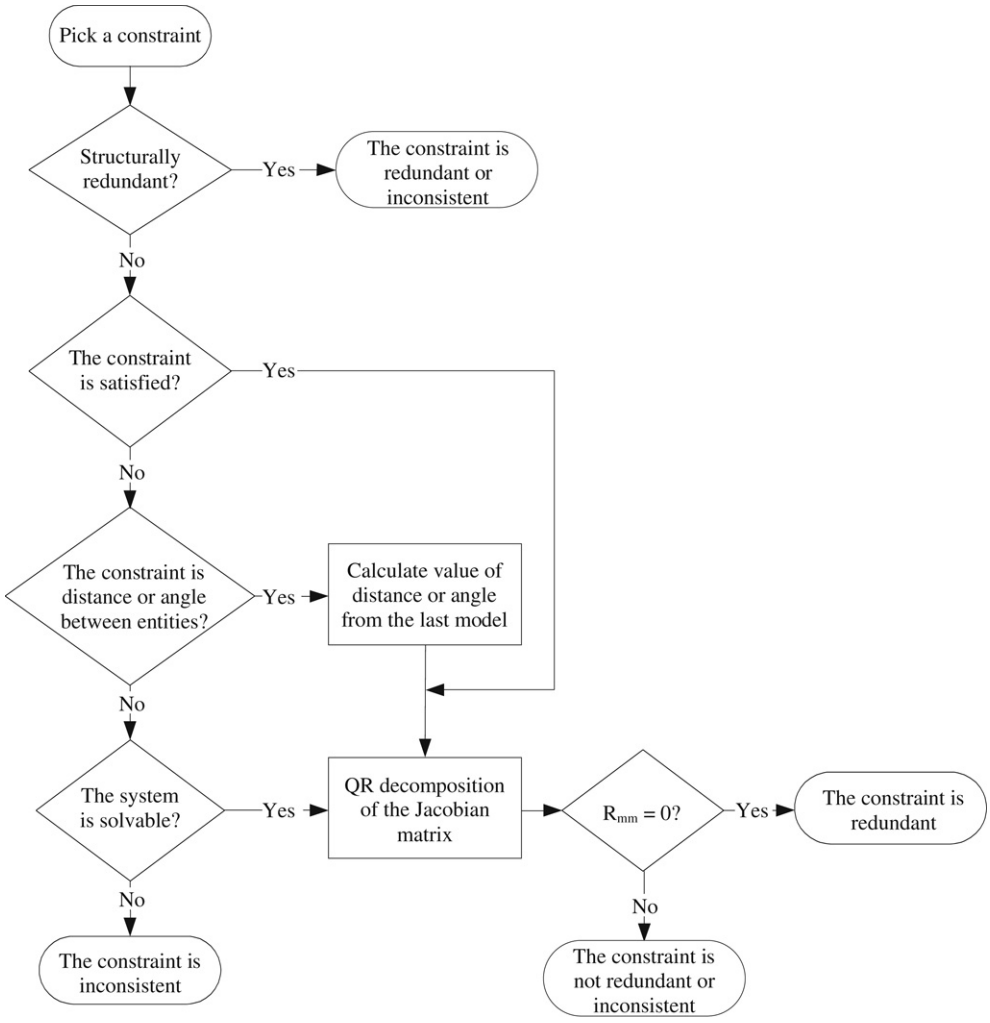

Fig. 5. Flow chart for determining if a constraint is redundant or inconsistent.

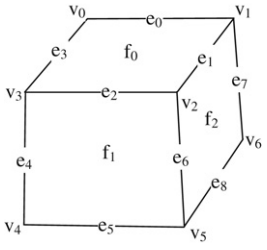

Fig. 6. An example with numerical redundant constraint.

column of the upper triangular matrix R, is zero. That means the last constraint is numerically redundant.

By solving Eq. (5), the vector of the linear combination can be obtained:

$$\begin{aligned}\mathbf{c} = \{&0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,-0.054\,0\,0\,0\,0\,0\,0\,0\\ &-0.054\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0.003\,0\,0\,-0.003\,0\\ &-0.008\,0.008\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\\ &-0.008\,0\,0.008\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0.389\,-0.951\,-1\}^{\mathrm{T}}.\end{aligned}$$

Not all the elements in $\mathbf{c}$ are zeros. In the Jacobian matrix, the row vector of the last constraint is a linear combination of the row vectors which correspond to the non-zero elements in $\mathbf{c}$. That is, if the ith element in $\mathbf{c}$ is non-zero, then the ith row vector in the Jacobian matrix is a member of the linear combination upon which the last row vector is dependent. Hence, in the example, the linear constraint of e_6 perpendicular to f_0, which is the last constraint added, is dependent on these eight constraints: v_5 on f_1, v_2 on f_1, v_5 on f_2, v_2 on f_2, v_5 on e_6, v_2 on e_6, f_0 perpendicular to f_1 and f_0 perpendicular to f_2. The system or user can select and delete one constraint from the group, except the topological constraints.

The cost of every procedure for checking for redundancy in the example is shown in Table 5. This result shows that the cost in solving the constraints is the largest among all the procedures. It is the same in other more complex examples. One observation we can make is that the system of equations is large. Given the very simple example here, the Jacobian matrix is already 78 × 93 in size. Hence an efficient solution is very important.

3.5. Comparison with Langbein's method

The method reported in the literature closest to ours is that of Langbein's [5,30], which uses a similar framework of selecting and then reducing the constraints to a sufficient and

Table 4
An example of a Jacobian matrix

		v_0	v_1	v_2	v_3	v_4	v_5	v_6	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	f_0	f_1	f_2
Constraints in edge (18)	Extra e_0								*											
	...																			
	Extra e_8																*			
Constraints in face (9)	Extra f_0																	*		
	Extra f_1																		*	
	Extra f_2																			*
Vertices on face (12)	v_3 lie on f_0				*													*		
	...																			
	v_1 lie on f_2		*																	*
Vertices on line (36)	v_0 lie on e_0	*							*											
	...																			
	v_5 lie on e_8						*										*			
Constraints detected (3)	$f_0 \perp f_1$																	*	*	
	$f_0 \perp f_2$																	*		*
	$l_6 \perp f_0$														*			*		

Table 5
Time (in seconds) for redundancy detection

Last constraint added	Checking structural redundancy	Solving the constraints	Checking numerical redundancy by QR factorization of the Jacobian matrix	Total
$f_0 \perp f_1$	0.015	0.375	0.031	0.421
$f_0 \perp f_2$	0.016	0.312	0.031	0.359
$e_6 \perp f_0$	0.015	0.265	0.031	0.311

consistent set. Langbein's work is applied to geometric models recovered from 3D point clouds, where the inaccuracy is much smaller than 3D objects recovered from sketches. Langbein's experiments show that his graph-based method is successful when applied to real problems. It saves time by avoiding computing for the solution of the constraint system. But his graph-based method does not investigate the property of rigid subgraphs, resulting in it missing some redundant constraints. It also cannot obtain the minimum over-constrained subgraph. Langbein has not provided a rigorous proof for the method.

Because only some of the redundancies and inconsistencies can be detected by a graph-based method, a numerical solvability test is required. Langbein's numerical solvability test checks for the redundancy of a constraint by perturbing the values involved in the constraint and solving the resulting system [38]. It requires performing BFGS n times when n constraints are needed to check for redundancy, a costly process. In our approach, QR decomposition of the Jacobian matrix is employed to detect redundancy without solving the constraint system by BFGS, which is cheaper. At the same time, if the last constraint added is found to be redundant, our method can return its dependent constraint group, which makes it possible for the user to remove the redundancy by selecting and deleting one constraint from the group.

4. Changing dimensions by the user

After setting up the constraints through the procedures described above, the object is well-constrained. At this point, the dimensions in the object are directly derived from the drawing, which do not normally correspond to the dimensions the designer has in mind.

The user can establish a relationship between the size of the reconstructed model and the size that the user requires by specifying the correct value to a known dimension. This then provides a scaling factor, which can be applied easily to scale the whole object to the required size. After that, the user can manually reset the value of any individual dimension if so required. Finally, the constraint system can be solved again to generate a model with the desired dimension.

The running time for constraint solving of a well constrained system can be reduced by decomposing the constraint problem into smaller ones. There are many methods that can be used to decompose the constraints of a well constrained system [16,39–41]. The method of Latham [16] and Gao [39] is employed in our system. By analyzing the connectivities between constraints and the geometric entities which they constrain, construction sequences can be generated. After the decomposition, the time needed to solve the constraint equations decreases greatly. So, when the user changes some dimension values, the system reacts much faster.

5. Application examples

The examples in Fig. 7 show some results of our system that implements our method of beautification and dimensioning of a reconstructed 3D model. The system was implemented in Visual C++ with OpenGL, running on a 2.8G Pentium 4 PC.

There are too many constraints detected for each example, so we do not present them in detail. Fig. 7 shows the original reconstructed 3D models, the models after beautification and the models with dimensions. In the examples, all the vertices of a model lie on faces, all the constraints selected are satisfied. Table 6 lists the time cost for detecting the constraints for beautification and dimensioning, the time for solving an equation set, the time for solving an equation set after constraint decomposition, the time for determining structural redundancy and then for numerical redundancy when a new constraint is added. It can be seen from the table that when the model is

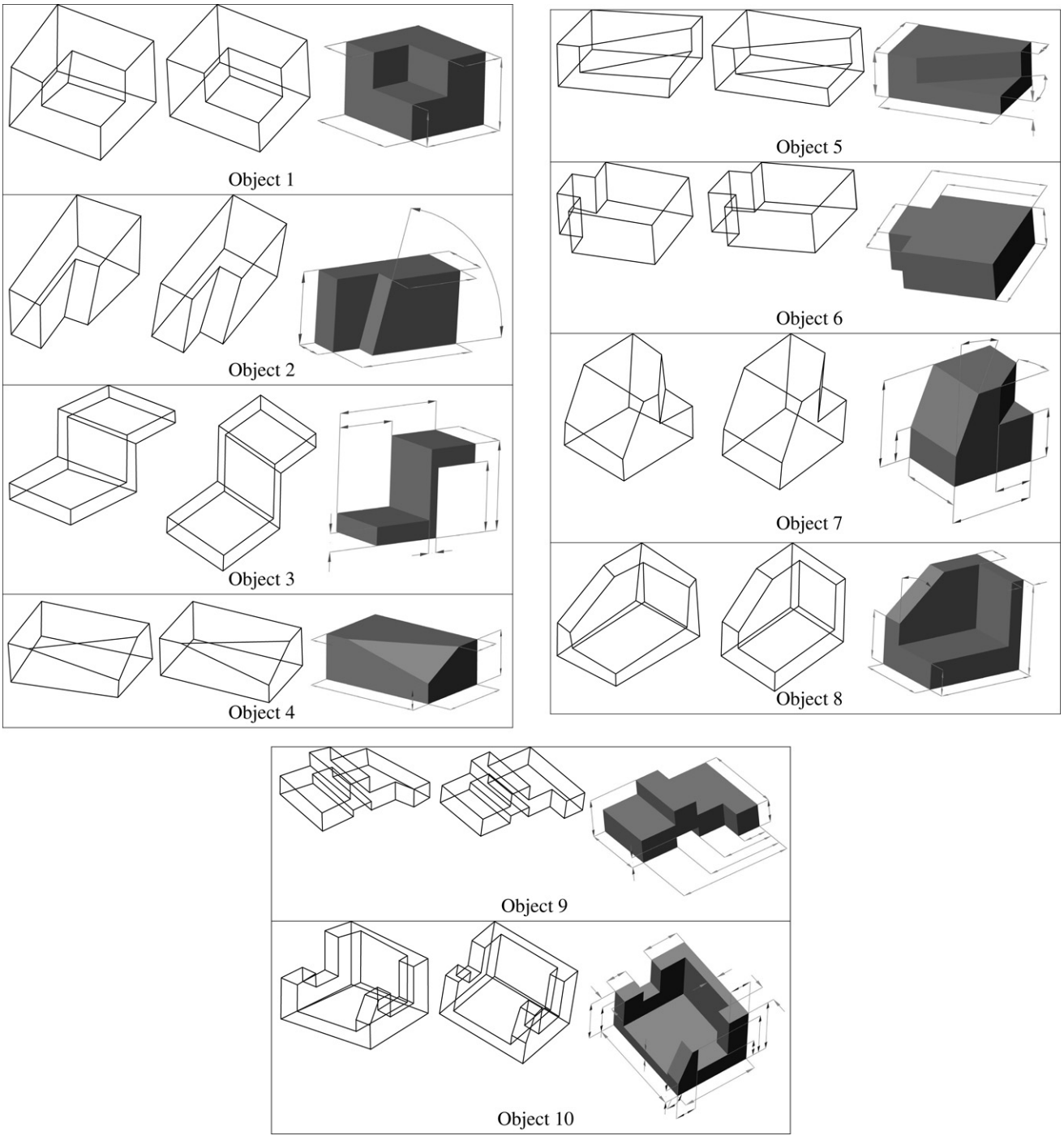

Fig. 7. Test examples. The left line drawing for each object is the input to our beautification system, and the centre drawing is the output. The right shaded figure is not generated by our system but created separately to show the dimensions (or constraints) that have been found and retained, as we have yet to include the posting of dimensions in our system.

more complex, the time for detecting a correct set of constraints is greater, as expected. The constraint solving process is the most time consuming. Fortunately, after the constraint decomposition, the processing time decreases dramatically.

6. Conclusions

This paper establishes a theoretical framework for beautification and automatic dimensioning of a 3D model recovered from a 2D sketch. The recovered model may be very "rough", given that a sketch is often inaccurate. Our experimental results show that our method beautifies the recovered model very well, to something more akin to a well-designed object. It also generates the dimensions correctly. The dimensions can be modified if they do not meet the design requirements.

It should be noted that the BFGS algorithm may fail if the initial input is far from the desired solution. That may happen when the initial model for the start point of the optimization

Table 6
Computing times (seconds) for objects in Fig. 7

Object	Number of faces	Number of constraints[a]	Constraint detection time	Solving time	Solving time after constraint decomposition	Checking structural redundancy of a new constraint	Checking numerical redundancy of a new constraint
1	9	9	19.93	2.56	0.14	0.03	0.25
2	8	10	24.18	1.96	0.60	0.03	0.17
3	10	10	30.00	2.96	0.20	0.03	0.35
4	7	9	8.51	1.09	0.23	0.01	0.10
5	8	10	18.90	1.65	0.37	0.01	0.17
6	10	9	35.82	3.15	0.18	0.03	0.37
7	9	13	34.35	2.39	1.11	0.03	0.26
8	10	12	40.78	3.43	0.56	0.03	0.37
9	16	16	112.15	6.36	0.13	0.04	0.70
10	19	21	300.00	12.54	0.11	0.04	0.97

[a] Exclude topological constraints.

is very different from the model defined by the constraints. Thus a good initial variable vector is important. In all our experiments, because the sketches are discernable, the initial variable vectors extracted from the recovered models are good enough for the beautification. But if the user interferes by drastically changing some dimensions, there is no guarantee that the BFGS algorithm will deliver the right solution. Though the method described in this paper is much more reliable for redundant constraint detection compared to graph-based methods, the running time, especially in constraint solving, needs to be improved still. A new speedy optimization method or a faster computer would help.

This paper has focused its attention on detecting redundant/inconsistent constraints and selecting constraints to form a well constrained system. However, it does not mean that using the simple priority scheme introduced in Section 2 will always lead to correct dimensioning of a model. It should be noted that the system, being well-constrained, is a necessary but not unique condition to a correct dimension set. The problem of how to define the priority of detected constraints is complex. Much work remains to be done here. We are considering a solution based on feature identification within the model and the constraints being applied within and between features.

External factors, like dimensioning and tolerancing standards, will also affect constraint selection.

Algorithms for locating dimensions, including position and angle of dimensions, also require further research.

Acknowledgements

We thank Dr. Christophe Jermann of University of Nantes for stimulating discussions on the degree of rigidity computation. Many thanks to SIAM for providing us with LAPACK and to Prof. Sven Hammarling from the Numerical Algorithms Group for the helpful discussion on using LAPACK. We also thank Prof. Robert Joan-Arinyo of Universitat Politècnica de Catalunya for valuable discussions and encouragement about our research on constraint solving during the CAD05 conference in Thailand.

References

[1] Marill T. Emulating the human interpretation of line drawings as three-dimensional objects. Computer Vision 1991;6(2):147–61.
[2] Lipson H, Shpitalni M. Optimization-based reconstruction of a 3D object from a single freehand line drawing. Computer-Aided Design 1996;28(8):651–63.
[3] Liu J, Lee YT. A Graph-based method for face identification from a single 2D line drawing. IEEE Transaction on Pattern Analysis and Machine Intelligence 2001;23(10):1089–106.
[4] Varley PAC. Automatic creation of boundary-representation models from single line drawings. PhD thesis. Cardiff University; 2002.
[5] Langbein FC, Marshall AD, Martin RR. Choosing consistent constraints for beautification of reverse engineered geometric models. Computer-Aided Design 2004;36(3):261–78.
[6] Chen XC. Planarity correction for object constructed from line drawings. Master thesis. Singapore: Nanyang Technological University; 2002.
[7] Wilczkowiak M, Trombettoni G, Jermann C, Sturm PF, Edmond B. Scene modeling based on constraint system decomposition techniques. In: International conference on computer vision. 2003. p. 1004–10.
[8] Martínez ML, Félez J. A constraint solver to define correctly dimensioned and over-dimensioned parts. Computer-Aided Design 2005;37(13):1353–69.
[9] Buchanan SA, de Pennington A. Constraint definition system: A computer-algebra based approach to solving geometric-constraint problems. Computer-Aided Design 1998;25(12):741–50.
[10] Gao XS, Chou SC. Solving geometric constraint systems. II, a symbolic approach and decision of rc-constructibility. Computer-Aided Design 1998;30(2):115–22.
[11] Wu W-T. Mechanical theorem proving in geometries. New York, USA: Springer-Verlag Inc.; 1994.
[12] Fudos I. Constraint solving for computer aided design. Ph.D. thesis. West Lafayette (IN, USA): Purdue University; 1998.
[13] Owen J. Algebraic solution for geometry from dimensional constraints. In: ACM symposium on foundations of solid modeling, 1991. p. 397–407.
[14] Bouma W, Fudos I, Hoffmann CM, Cai J, Paige R. A geometric constraint solver. Computer-Aided Design 1995;27(6):487–501.
[15] Joan-Arinyo R, Soto-Riera A, Vila-Marta S, Vilaplana J. Transforming an under-constrained geometric constraint problem into a well-constrained one. In: Proceedings of the eighth ACM symposium on solid modeling and application, 2003. p. 33–44.
[16] Latham RS, Middleditch AE. Connectivity analysis: A tool for processing geometric constraints. Computer-Aided Design 1996;28(11):917–28.
[17] Hoffmann CM, Lomonosov A, Sitharam M. Decomposition plans for geometric constraint systems, part I: Performance measures for CAD. Journal of Symbolic Computation 2001;31(4):367–408.
[18] Li YT, Hu SM, Sun JG. A constructive approach to solving 3-D geometric constraint systems using dependence analysis. Computer-Aided Design 2002;34(2):97–108.

[19] Kramer G. Solving geometric constraint systems. Cambridge (MA, USA): MIT Press; 1992.
[20] Jermann C, Neveu B, Trombettoni G. A new structural rigidity for geometric constraint systems. In: Proceedings of the 4th international workshop on automated deduction in geometry. 2002. p. 87–106.
[21] Jermann C, Neveu B, Trombettoni G. Algorithms for identifying rigid subsystems in geometric constraint systems. In: Proceedings of the eighteenth international joint conference on artificial intelligence. 2003. p. 233–8.
[22] Hoffmann CM, Lomonosov A, Sitharam M. Geometric constraint decomposition. In: Bruderlin B, Roller D, editors. Geometric constraint solving and applications. Berlin: Springer; 1998. p. 170–95.
[23] Zou HL, Lee YT. Detecting minimum over-constrained subgraphs in 2D and 3D based on degree of freedom analysis. Computer-Aided Design and Applications 2005;2:393–402.
[24] Pappas T. Pappus' theorem & the nine coin puzzle. In: The joy of mathematics. San Carlos (CA): Wide World Publishing; 1989.
[25] Light R, Gossard DC. Modification of geometric models through variational geometry. Computer-Aided Design 1982;14(4):209–14.
[26] Chou SC. Mechanical geometry theorem proving. Norwell (MA): Kluwer Academic Publishers; 1987.
[27] Buchberger B. Grobner bases: An algorithmic method in polynomial ideal theory. In: Bose NK, editor. Multidimensional systems theory. D. Reidel Publishing Co.; 1985.
[28] Buchberger B. Groebner bases: A short introduction for systems theorists. In: EUROCAST 2001 (8th international conference on computer aided systems theory — formal methods and tools for computer science). 2001.
[29] Li YT, Hu SM, Sun JG. On the numerical redundancies of geometric constraint systems. In: Proceedings of Pacific graphics 2001. Tokyo: IEEE Computer Society Press; 2001. p. 118–23.
[30] Langbein FC, Marshall AD, Martin RR. Numerical methods for beautification of reverse engineered geometric models. In: Proceedings of the geometric modeling and processing — theory and applications. 2002. p. 159–68.
[31] Everitt B. Cluster analysis. 2nd ed. London: Heinemann Educational Books Ltd.; 1986.
[32] Gray P, et al. A survey of global optimization methods. http://www.cs.sandia.gov/opt/survey/main.html; 1997.
[33] Deb K. Optimization for engineering design: Algorithms and examples. India: Prentice-Hall; 1995.
[34] Paige CC, Saunders MA. Algorithm 583: LSQR: Sparse linear equations and least squares problems. ACM Transactions on Mathematical Software 1982;8(2):195–209.
[35] Golub GH, Van Loan CF. In: Baltimore MD, editor. Matrix computations. 2nd ed. Johns Hopkins University Press; 1996.
[36] Bischof CH, Quintana-Orti G. Computing rank-revealing QR factorizations of dense matrices. ACM Transactions on Mathematical Software 1998;24(2):226–53.
[37] Mathworks. Matlab — The language of technical computing. 2006 [cited 17 Oct.]; Available from: http://www.mathworks.co.uk/products/matlab/.
[38] Langbein FC. Beautification of reverse engineered geometric models. Ph.D. thesis. UK: Cardiff University; 2003.
[39] Gao XS, Lin Q, Zhang G. A C-tree decomposition algorithm for 2D and 3D geometric constraint solving. Computer-Aided Design 2006;38(1): 1–13.
[40] Hoffmann CM, Lomonosov A, Sitharam MJ. Decomposition plans for geometric constraint problems, part II: New algorithms. Symbolic Computation 2001;31(4):409–27.
[41] Sitharam M. Combinatorial approaches to geometric constraint solving: Problems, progress, directions. In: Janardan R, Smid M, Dutta D, editors. Geometric and algorithmic aspects of computer-aided design and manufacturing. AMS-DIMACS series in discrete mathematics and computer science, vol. 67. 2005. p. 117–64.