Journal for Geometry and Graphics
Volume 15 (2011), No. 1, 79–91.

The Reconstruction Problem: Integrating Different Approaches into a Systematic Procedure for Pseudo Wireframe Retrieval

Rocco Furferi, Lapo Governi, Matteo Palai, Yary Volpe

Dept. of Mechanics and Industrial Technologies, Università degli Studi di Firenze
Via di Santa Marta 3, 50139 Firenze, Italy
emails: {rocco.furferi, lapo.governi, matteo.palai, yary.volpe}@unifi.it

Abstract. Nowadays three-dimensional Computer Aided modeling is of outstanding importance in the mechanical design process since it impacts on several issues like visualization, simulation, machining, etc. Anyway, multi orthographic view engineering drawings have been widely used up to latest decade and still are, so they play an essential role in traditional engineering. The conversion from 2D drawings to 3D CAD models is still a key task in a wide range of applications. In order to cope with this issue a number of works have been proposed in the last decades, providing a series of methodologies for solving the reconstruction problem. On the basis of such methodologies the main aim of the present paper is to suggest a comprehensive, orderly, unambiguous and automatic procedure meant to help researchers and practitioners who want to deal with the reconstruction problem. The procedure, by using an appropriate formal mathematic language, systematize and integrates some of the methods proposed so far.

Key Words: Pseudo-wireframe, 3D Reconstruction, Engineering Drawings, Orthographic Projections, Computer Aided Design, Computational Geometry

MSC 2010: 51N05, 68U05

1. Introduction

Nowadays, 3D CAD modelers are commonly used by designers. Both solid and surface CAD models have become crucial for a large number of CAE techniques (e.g. visualization, simulation, CNC machining, ...). Nevertheless, multi orthographic view engineering drawings have been widely used up to latest decade and still are, so they play an essential role in traditional engineering. As a matter of fact, many products are still designed by means of orthographic views. Moreover, many engineering tasks involve modification of existing design, thus an automatic tool for reconstructing a 3D CAD model, starting from multi orthographic view

ISSN 1433-8157/$ 2.50

engineering drawings, would prove to be particularly useful in many applications. In addition to its research significance, this kind of tool would ease a number of practical issues, mostly in the field of automatic conversion of digitized engineering drawings into 3D CAD models.

In order to develop such a procedure, the interpretation of engineering drawings by computers is a key prerequisite. In the past three decades many scientific studies have been carried out confronting this issue, that has come to be known as *geometrical reconstruction* or simply *reconstruction* problem [2, 4, 7, 9]. A dramatic boost to the research on *geometrical reconstruction* was provided by Wesley and Markowsky in their studies [14, 15] which are probably the best known work among the researchers working on this subject. Wesley and Markowsky provided a comprehensive set of guide lines for reconstructing a 3D CAD model starting from orthographic projections. Their procedure is, still today, a milestone for almost every B-rep based reconstruction study. Since then, many works have been presented dealing with the reconstruction problem using B-rep approaches [12, 6, 1, 10, 13, 8, 17, 5].

In spite of the huge literature on the *reconstruction* problem, almost all methods, proposed by several different authors, are mainly described by a conceptual point of view, so that deriving an orderly procedure which covers the necessary steps from 2D data to a *pseudo-wireframe* model requires always a great effort and a considerable amount of work. The most challenging tasks that are to be faced when trying to derive such a procedure are related to the presence of ambiguities in the methodology description and to the lack in enumerating all possible cases that can be found when having to deal with real-life drawings. For instance, Yan et al. [16] describe a conceptually flawless method to detect 3D edges on the basis of a table of possible configurations. Nevertheless such a method has to be improved by adding more new configurations in order to derive a comprehensive and unambiguous operative procedure.

The aim of the present work is to provide researchers and practitioners with an orderly and automatic procedure enabling a straightforward implementation of the *pseudo-wireframe* model reconstruction. The works proposed by the scientific literature, which confront the reconstruction problem for curvilinear objects usually present, quite "tricky" approaches, generally involving heavy user interaction [5, 17, 8, 10]. At the moment, though very interesting and promising, such approaches are not well established in the scientific community. Since existing approaches are certainly better recognized in the case of polyhedrical objects, the authors decided to confront the reconstruction problem for this kind of geometric entities. The proposed method integrates the approaches presented in a number of studies [14, 15, 16, 6], and — making use of an appropriate formal language — presents a comprehensive implementation-oriented procedure.

2. Method

According to [11], a representation of a drawing can be generated in some neutral file formats such as DXF (Drawing Exchange Format) developed by Autodesk®.

Given three projections of an object in the DXF format, it is immediate to detect each entity (line, circle, etc.) composing it. Unfortunately such a file format does not provide topological information, i.e., the entities in a DXF file are in no logical order so that no explicit information regarding their connectivity is accessible. For such a reason, a method for separating the drawing into three views is needed before further processing can take place. If $\mathbf{\Pi}^i$ is defined as the i^{th} projection ($i = 1, 2, 3$) in the orthographic view system, several approaches for separating the three views can be used (e.g., [11]). By using one of these

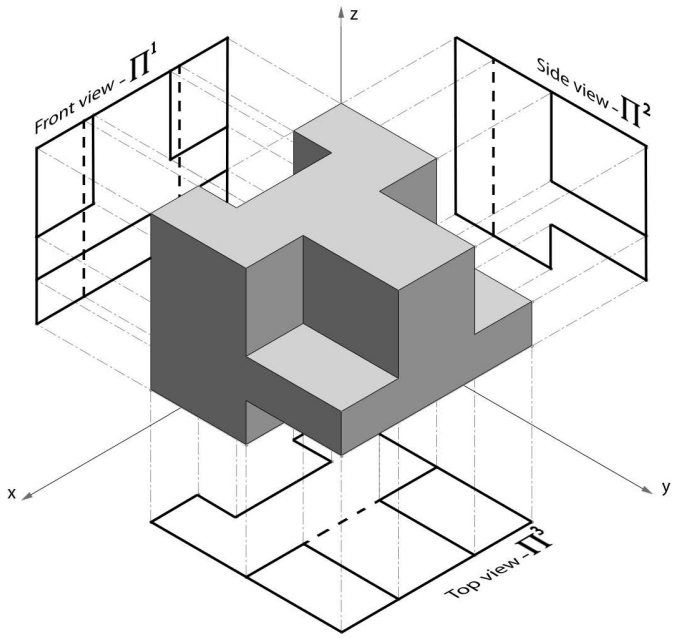


Figure 1: 3D projected object

approaches, the result is the creation of two families of sets:

1. $\boldsymbol{V}^i$, with $i = 1, 2, 3$, represent the three sets of vertices, each one corresponding to a projection in the orthographic view system (i.e., $\boldsymbol{V}^i$ is the set of vertices of $\boldsymbol{\Pi}^i$ — *metrical data*).
2. $\boldsymbol{E}^i$, with $i = 1, 2, 3$, represent the three sets of edges, each one corresponding to a projection in the orthographic view system (i.e., $\boldsymbol{E}^i$ is the set of edges of $\boldsymbol{\Pi}^i$ — *geometrical data*).

On the basis of the six sets described above, $\boldsymbol{V}^i(j) = \boldsymbol{v}^i_j = [x^i_j, y^i_j, z^i_j]$ represents the j^{th} vertex in the i^{th} orthographic view. By definition, each vertex lies on one of the coordinate planes. Consequently, one of the elements of $\boldsymbol{v}^i_j$ is always equal to zero (see Fig. 1).

The j^{th} edge in the i^{th} orthographic view is identified by its two vertices $\boldsymbol{v}^i_h$ and $\boldsymbol{v}^i_k$ as follows:

$$\boldsymbol{e}^i_j = \begin{bmatrix} \boldsymbol{v}^i_h \\ \boldsymbol{v}^i_k \end{bmatrix}_{2\times 3} \tag{1}$$

The three standard views (top, front and side) can be expressed as follows:

$$\begin{aligned} \boldsymbol{\Pi}^1 &= [\boldsymbol{e}^1_1, \boldsymbol{e}^1_2, \ldots, \boldsymbol{e}^1_a]^T_a \\ \boldsymbol{\Pi}^2 &= [\boldsymbol{e}^2_1, \boldsymbol{e}^2_2, \ldots, \boldsymbol{e}^2_b]^T_b \\ \boldsymbol{\Pi}^3 &= [\boldsymbol{e}^3_1, \boldsymbol{e}^3_2, \ldots, \boldsymbol{e}^3_c]^T_c \end{aligned} \tag{2}$$

where a, b and c are, respectively, the number of edges in $\boldsymbol{\Pi}^1$, $\boldsymbol{\Pi}^2$ and $\boldsymbol{\Pi}^3$. Finally the set of orthographic projections (standard views) is defined as:

$$\mathbf{OBJ} = [\boldsymbol{\Pi}^1, \boldsymbol{\Pi}^2, \boldsymbol{\Pi}^3]^T \tag{3}$$

where the size of matrix **OBJ** is $(2a + 2b + 2c) \times 3$.

For each projection, the coordinates of the projection vertices are:

$$\begin{aligned} \boldsymbol{V}^1 &= [\boldsymbol{v}_1^1, \boldsymbol{v}_2^1, \ldots, \boldsymbol{v}_\beta^1]_{\beta\times 3}^T \\ \boldsymbol{V}^2 &= [\boldsymbol{v}_1^2, \boldsymbol{v}_2^2, \ldots, \boldsymbol{v}_\gamma^1]_{\gamma\times 3}^T \\ \boldsymbol{V}^3 &= [\boldsymbol{v}_1^3, \boldsymbol{v}_2^3, \ldots, \boldsymbol{v}_\varphi^3]_{\varphi\times 3}^T \end{aligned} \tag{4}$$

where β, γ and φ are respectively the numbers of vertices in $\boldsymbol{\Pi}^1$, $\boldsymbol{\Pi}^2$ and $\boldsymbol{\Pi}^3$.

2.1. 3D reconstruction of edges and vertices

As described in the section above pre-processing data may be manipulated in order to generate a mathematical 3D *pseudo-wireframe* model. With the aim of obtaining such a model, the following tasks have to be carried out.

2.1.1. Labeling of vertices

Once the set of orthographic projections (standard views) is known, it is possible to perform a reconstruction of vertices and edges in the 3D space, i.e., it is possible to create a topological data structure starting by labeling each vertex of each projection with a progressive number. As a result we obtain:

$$\begin{aligned} \boldsymbol{v}_1^i = [x_1^i, y_1^i, z_1^i] &\Rightarrow \boldsymbol{v}_1^i = 1 \\ \boldsymbol{v}_2^i = [x_2^i, y_2^i, z_2^i] &\Rightarrow \boldsymbol{v}_2^i = 2 \\ &\vdots \\ \boldsymbol{v}_n^i = [x_n^i, y_n^i, z_n^i] &\Rightarrow \boldsymbol{v}_n^i = n \end{aligned} \tag{5}$$

2.1.2. Labeling of edges

As depicted in Fig. 2, each edge may be defined by means of the set of labels of its vertices. As a consequence each edge $\boldsymbol{e}_j^i$ can be rewritten as follows:

$$\boldsymbol{e}_j^i = \begin{bmatrix} x_h^i, y_h^i, z_h^i \\ x_k^i, y_k^i, z_k^i \end{bmatrix}_{2\times 3} \Longrightarrow \boldsymbol{e}_j^i = [h^i, k^i]_{1\times 2} \tag{6}$$

Accordingly, only 3 parameters (h, k, i) are now used to identify each edge properly instead of 7 parameters previously used $(x_h^i, y_h^i, z_h^i, x_k^i, y_k^i, z_k^i, i)$.

2.1.3. Check for intermediate vertices and collinear edges

Though an object is represented by a univocal set of projections, these can be drawn by using different combination of geometric entities: the segment highlighted in Fig. 3b (as shown, for instance, in a printed copy of the drawing) in the DXF file can be made up of a number of straight vectors (from 1 to 8 as shown in Fig. 4a). Note that such combination of vectors may be different from the one which would be generated by the projection of the object's 3D edges lying on the plane orthogonal to the view and whose trace contains the original segment (Fig. 3a).

Therefore, an approach for the processing of different DXF files of the same drawn object is provided, in order to obtain a univocally defined vectorial representation, comprising all the possible configurations.

First, for each edge, an iterative procedure checks for the possible existence of intermediate vertices. If any intermediate vertex is found, the procedure stops. Otherwise the found

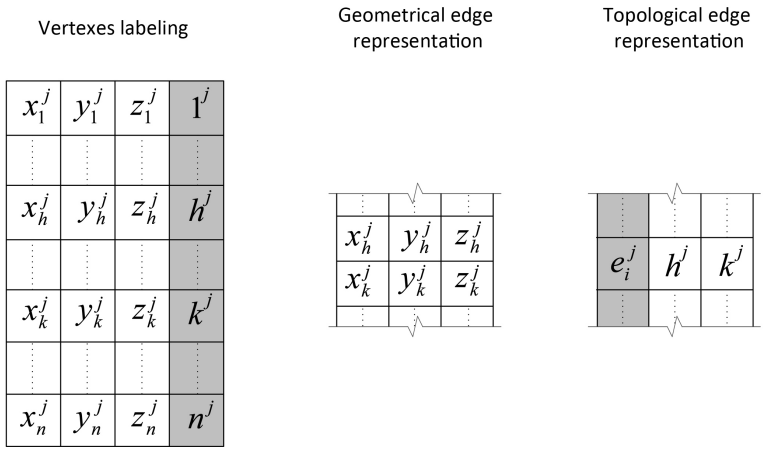
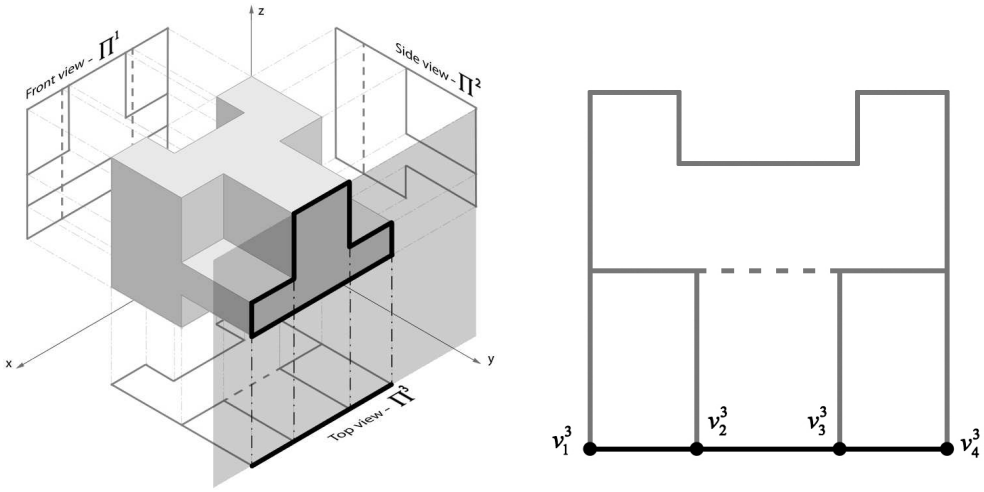


Figure 2: Labeling of edges



(a) Plane of projection (b) Projected view

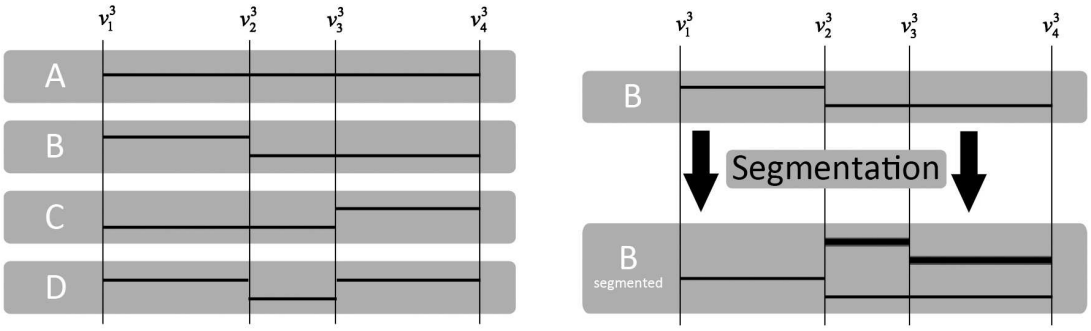
Figure 3: Projection on an orthogonal view

intermediate vertex causes the creation of two new edges (unless one of them already exists). This task, called *"segmentation"*, is performed for each set of edges belonging to a projection, thus adding new edges to the original set. Referring to Fig. 3b, supposing that the portion of projection highlighted is represented by the configuration "B" of Fig. 4a, the "segmentation" process produces the results shown in Fig. 4b. Particularly, it is important to note that two new edges, highlighted in Fig. 4b, have been generated in the projection.

From the mathematical point of view, the segmentation task can be accomplished as follows. According to the notation described in (5) and (6), the following definitions can be provided referring, for instance, to $\mathbf{\Pi}^3$ (where processing $\mathbf{\Pi}^1$ the variable y has to be replaced by the variable z, in the case of $\mathbf{\Pi}^2$ the variable x has to be replaced by the variable z):

1.
$$f_j^i(x, y) = 0 \tag{7}$$
equation of the curve on which the edge $\boldsymbol{e}_j^i$ lies;

2.
$$\boldsymbol{B}_j^i = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}_{2\times 2} \tag{8}$$



(a) Possible combinations of edges (b) "Segmentation" procedure

Figure 4: "Segmentation"

bounding box of the edge $\boldsymbol{e}_j^i$ whose components are

$$\begin{array}{ll} B_{1,1} = \max\{x_h^i, x_k^i\} & B_{1,2} = \max\{y_h^i, y_k^i\} \\ B_{2,1} = \min\{x_h^i, x_k^i\} & B_{2,2} = \min\{y_h^i, y_k^i\} \end{array} \tag{9}$$

For each projection $\boldsymbol{\Pi}^i$, for each edge $\boldsymbol{e}_j^i$, for each vertex $\boldsymbol{v}_d^i$, the performed procedure check the existence of the two edges defined as follows:

$$\begin{array}{l} \boldsymbol{e}_s^i = [\boldsymbol{v}_h^i, \boldsymbol{v}_d^i] \\ \boldsymbol{e}_t^i = [\boldsymbol{v}_d^i, \boldsymbol{v}_k^i] \end{array} \quad \text{with} \quad \begin{array}{l} 0 < s \leq a, b, c \\ 0 < t \leq a, b, c \end{array} \tag{10}$$

where the choice among a or b or c is determined by the index i (a if $i = 1$, b if $i = 2$, c if $i = 3$). Let's define:

$$\begin{array}{lll} l_1 = 1 & \text{if} & f_j^i(x_d^i, y_d^i) = 0 \text{ and } l_1 = 0 \text{ else;} \\ l_2 = 1 & \text{if} & B_{1,1} \leq x_d^i \leq B_{2,1} \text{ and } l_2 = 0 \text{ else;} \\ l_3 = 1 & \text{if} & B_{1,2} \leq y_d^i \leq B_{2,2} \text{ and } l_3 = 0 \text{ else;} \\ l_4 = 1 & \text{if} & x_h^i = x_d^i \text{ and } l_4 = 0 \text{ else;} \\ l_5 = 1 & \text{if} & y_h^i = y_d^i \text{ and } l_5 = 0 \text{ else;} \\ l_6 = 1 & \text{if} & x_k^i = x_d^i \text{ and } l_6 = 0 \text{ else;} \\ l_7 = 1 & \text{if} & y_k^i = y_d^i \text{ and } l_7 = 0 \text{ else;} \end{array} \tag{11}$$

When one or both of the edges described in (10) do not exist, the following logical equation is evaluated:

$$l_1 \wedge l_2 \wedge l_3 \wedge [(l_4 \oplus l_5) + \overline{l_4 \wedge l_5}] \wedge [(l_6 \oplus l_7) + \overline{l_6 \wedge l_7}] = 1 \tag{12}$$

If the equation (12) is verified, then:

1. If both edges do not exist, both, $\boldsymbol{e}_s^i$ and $\boldsymbol{e}_t^i$, have to be added to $\boldsymbol{\Pi}^i$.
2. If one of the edges does not exist, e.g., $\boldsymbol{e}_s^i$, only the missing one, $\boldsymbol{e}_s^i$, have to be added to $\boldsymbol{\Pi}^i$.
3. If both edges exist, the procedure progresses with the following vertex.

When no more vertices have to be checked, the procedure progresses with the following edge. When all the edges lying on a projection have been processed, the method switches to another projection. The procedure stops when all the three projections have been analyzed.

The results of the segmentation task applied to the highlighted part of the projection in Fig. 3b, are shown in Fig. 4b. Thus, the two highlighted edges in Fig. 4b have been added to the set of edge.

After the "segmentation" task, a check of *collinearity of edges* is performed. As shown in Fig. 5, this phase is crucial when two non contiguous vertices are linked by two or more collinear edges. If this check is neglected or inaccurate it could happen that two visually identical projections are described by two different datasets. When a collinearity of edges is detected, a new set of edges is added to the original one as described below. The collinearity can be detected as the logical product of concatenation (two edges sharing the same vertex) and parallelism.

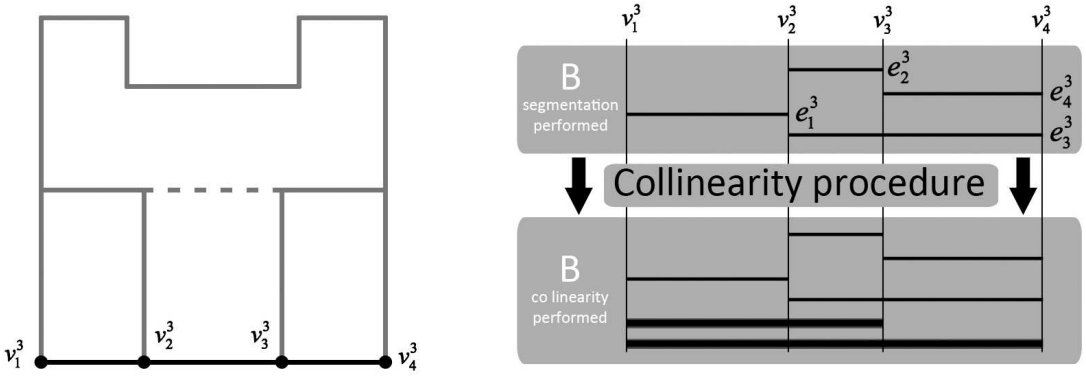


Figure 5: "Collinearity" procedure

In order to describe the concatenation and parallelism relationship among the edges, the following two matrices has been defined:

1. the concatenation matrix $\mathbf{CM}^i$ of the projection $\mathbf{\Pi}^i$;
2. the parallelism matrix $\mathbf{PM}^i$ of the projection $\mathbf{\Pi}^i$.

Since a single projection can be considered as a planar graph [3], matrices $\mathbf{CM}^i$ are defined similarly to the adjacency matrix in graph theory.

Accordingly, each matrix $\mathbf{CM}^i$ is a logical matrix whose elements $\mathrm{CM}^i_{s,t}$ are equal to 1 when the s^{th} and the t^{th} edges of $\mathbf{\Pi}^i$ are concatenated and equal to 0 elsewhere. Matrices $\mathbf{PM}^i$ are also logical and their elements $\mathrm{PM}^i_{s,t}$ are equal to 1 when parallelism subsists between the s^{th} and the t^{th} edges of $\mathbf{\Pi}^i$ and 0 elsewhere.

In order to further clarify the structure of these matrices an example is provided in Fig. 6a.

The collinearity between edges may be defined by the matrices $\mathbf{C}^i$ (Fig. 6d) obtained as the element by element product between matrices $\mathbf{CM}^i$ and $\mathbf{PM}^i$, as illustrated in the example of Fig. 6. Obviously, the collinearity between the s^{th} and the t^{th} edges in the $\mathbf{\Pi}^i$ projection subsists only when the entry of the matrix $\mathbf{C}^i$ (i.e., $C^i_{s,t}$) is equal to 1.

It is important to remark that the elements of the matrices $\mathbf{C}^i$ represent the collinearity relationship only for a couple of adjacent edges. If matrices $\mathbf{C}^i$ are all *zero* matrices, no collinear edges exists in the three projections. Otherwise each *non zero* matrix $\mathbf{C}^i$ has to be checked column by column, starting from the top of the matrix, in order to detect all the collinear set of edges. This task is performed according to the following procedure for each projection $\mathbf{\Pi}^i$:

Step 1: set to zero all diagonal element of $\mathbf{C}^i$;

Step 2: find the first column $\mathbf{C}^i(k)$ containing at least a *nonzero* value;

Step 3: find the position of all the *nonzero* values in column $\mathbf{C}^i(k)$;

Step 4: store these positions in a column vector $\boldsymbol{\gamma}$;

Step 5: for each value $\gamma \in \boldsymbol{\gamma}$ find the position of all the *nonzero* values in the γ^{th} row, except for the k^{th} position of the row;

Step 6: store these position in a row vector $\boldsymbol{\delta}$;

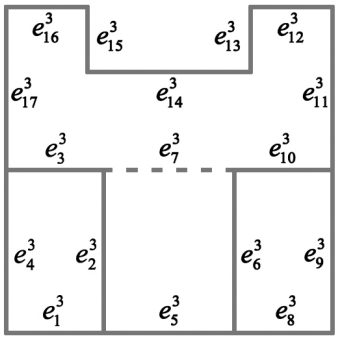
Step 7: for each value $\delta \in \boldsymbol{\delta}$ replace the k^{th} column of $\mathbf{C}^i$ with the logical sum (OR) of the δ^{th} and the k^{th} columns of $\mathbf{C}^i$;

Step 8: replace the δ^{th} column of $\mathbf{C}^i$ with a *zero* column vector;

Step 9: find the next column k containing at least a *nonzero* value; then repeat from *Step 2* until no more column has to be processed.

The final result of this procedure is to redefine the matrices $\mathbf{C}^i$ so that the position of *nonzero* elements in each column represent the collinear edges. All the permutations of the collinear edges that are not already stored in the matrices $\mathbf{\Pi}^i$ are then appended as new edges.

"Segmentation" and "collinearity" tasks, performed on the highlighted part of the pro-



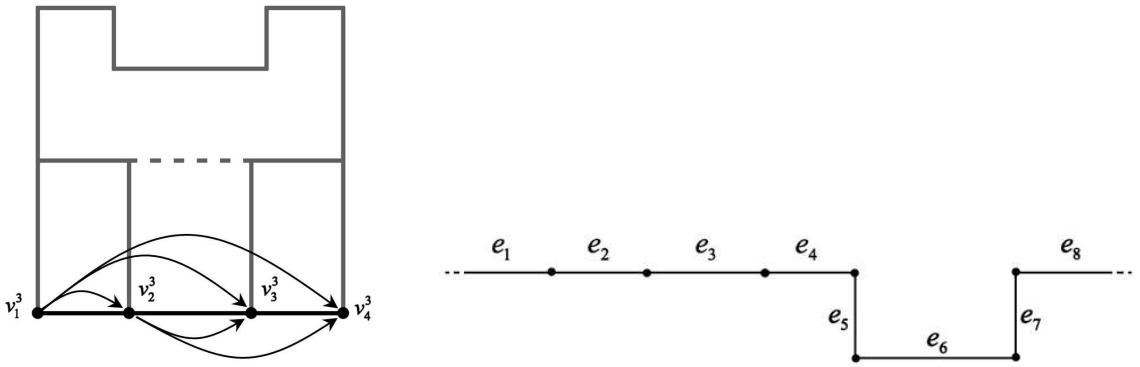
(a) Input data

(b) **CM** matrix

(c) **CP** matrix

(d) **C** matrix

Figure 6: Collinearity datasets



(a) "Segmentation" and "collinearity" results

(b) "Collinearity" sample

Figure 7: "Segmentation" and "collinearity" tasks

jection in Fig. 3b, allow to obtain the finally set of edges as shown in Fig. 7a.

The described procedure allows to correctly extend the edge set in each projection. For example, referring to Fig. 7b, the collinearity task produces, as a result, the set of edges $\{e_1, e_2, e_3, e_4\}$; by excluding edge e_8 from the set, the collinearity task prevents the generation of new pathological features in the projections (i.e., edges between e_4 and e_8).

2.1.4. Construction of vertices and edges in the 3D space

Once a database of edges and vertices for each view is obtained, it is possible to build a pseudo vertex skeleton.

Let $\boldsymbol{\omega} = [\emptyset]$ and $\boldsymbol{\sigma} = [\emptyset]$ be two auxiliary vectors and $\boldsymbol{\Lambda} = [\emptyset]$ the matrix of 3D vertices:

Step 1: for each vertex $\boldsymbol{v}_n^1 = [x_n^1, 0, z_n^1]$ store in the vector $\boldsymbol{\omega}$ all the vertex labels ω so that $v_{\omega,1}^3 = v_{n,1}^1$;

Step 2: for each vertex $\boldsymbol{v}_n^1 = [x_n^1, 0, z_n^1]$ store in the vector $\boldsymbol{\sigma}$ all the vertex labels σ so that $v_{\sigma,3}^2 = v_{n,3}^1$;

Step 3: for each element $\omega \in \boldsymbol{\omega}$, for each element $\sigma \in \boldsymbol{\sigma}$, if $v_{\sigma,2}^2 = v_{\omega,2}^3$ then append to the 3D matrix of vertices $\boldsymbol{\Lambda}$ the row vector $\boldsymbol{\lambda} = [x_n^1, y_\sigma^2, z_n^1, n, \sigma, \omega]$ and reset to the initial values vectors $\boldsymbol{\omega}$ and $\boldsymbol{\sigma}$.

Step 4: for each vertex $\boldsymbol{v}_n^2 = [0, y_n^2, z_n^2]$ store in the vector $\boldsymbol{\omega}$ all the vertex labels ω so that $v_{\omega,2}^3 = v_{n,2}^3$;

Step 5: for each vertex $\boldsymbol{v}_n^2 = [0, y_n^2, z_n^2]$ store in the vector $\boldsymbol{\sigma}$ all the vertex labels σ so that $v_{\sigma,3}^1 = v_{n,3}^2$;

Step 6: for each element $\omega \in \boldsymbol{\omega}$, for each element $\sigma \in \boldsymbol{\sigma}$, if $v_{\sigma,1}^1 = v_{\omega,1}^3$ then append to the 3D matrix of vertices $\boldsymbol{\Lambda}$ the row vector $\boldsymbol{\lambda} = [x_n^1, y_\sigma^2, z_n^1, \sigma, n, \omega]$ and reset to the initial values vectors $\boldsymbol{\omega}$ and $\boldsymbol{\sigma}$.

Step 7: for each vertex $\boldsymbol{v}_n^3 = [x_n^3, y_n^3, 0]$ store in the vector $\boldsymbol{\omega}$ all the vertex labels ω so that $v_{\omega,1}^1 = v_{n,1}^3$;

Step 8: for each vertex $\boldsymbol{v}_n^3 = [x_n^3, y_n^3, 0]$ store in the vector $\boldsymbol{\sigma}$ all the vertex labels σ so that $v_{\sigma,2}^2 = v_{n,2}^3$;

Step 9: for each element $\omega \in \boldsymbol{\omega}$, for each element $\sigma \in \boldsymbol{\sigma}$, if $v^2_{\sigma,3} = v^1_{\omega,3}$ then append to the 3D matrix of vertices $\boldsymbol{\Lambda}$ the row vector $\boldsymbol{\lambda} = [x^1_n, y^2_\sigma, z^1_n, \omega, \sigma, n]$ and reset to the initial values vectors $\boldsymbol{\omega}$ and $\boldsymbol{\sigma}$.

At the end of the above procedure, an additional check verifies the possible presence of multiple identical rows in $\boldsymbol{\Lambda}$; these possible row groups are so simplified and only one of them is preserved inside the matrix $\boldsymbol{\Lambda}$. The result of this task, called pseudo-vertex skeleton, is a new dataset structured as follows:

$$\boldsymbol{\Lambda} = \begin{bmatrix} x_1 & y_1 & z_1 & v_1 & v_2 & v_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_\epsilon & y_\epsilon & z_\epsilon & v_\epsilon & v_\epsilon & v_\epsilon \end{bmatrix}_{\epsilon \times 6} \tag{13}$$

Once the matrix $\boldsymbol{\Lambda}$ has been compiled, four additional phases are required to accomplish the 3D edges construction task:

1. Construction of 3D edges that are not orthogonal to any projection (Steps 1 to 4 described below);
2. Construction of 3D edges that are orthogonal to $\boldsymbol{\Pi}^1$(Steps 6 to 9);
3. Construction of 3D edges that are orthogonal to $\boldsymbol{\Pi}^2$ (according to Steps 6 to 9);
4. Construction of 3D edges that are orthogonal to $\boldsymbol{\Pi}^3$ (according to Steps 6 to 9).

If we define:

$\boldsymbol{\omega} = [\emptyset]$ and $\boldsymbol{\sigma} = [\emptyset]$ two auxiliary vectors;

$\rho = 0$ an auxiliary variable;

$\boldsymbol{\Theta} = [\emptyset]$ the matrix of the 3D edges,

the above mentioned steps can be detailed with the following procedure:

Step 1: for each row vector $\boldsymbol{\lambda}_s \in \boldsymbol{\Lambda}$, set $\rho = s$ and store in the vector $\boldsymbol{\omega}$ all the labels ω of the edges e^1_ω lying on $\boldsymbol{\Pi}^1$ that share the 2D vertex $v^1_n = \lambda_4$.

Step 2: for each element $\omega \in \boldsymbol{\omega}$ store in the vector $\boldsymbol{\sigma}$ the label σ of the 2D vertex v^1_σ so that $\boldsymbol{e}^1_\omega = [\boldsymbol{v}^1_{\boldsymbol{\sigma}}, \boldsymbol{v}^1_{\boldsymbol{\omega}}]$ or $\boldsymbol{e}^1_\omega = [\boldsymbol{v}^1_{\boldsymbol{\omega}}, \boldsymbol{v}^1_{\boldsymbol{\sigma}}]$

Step 3: for each element $\sigma \in \boldsymbol{\sigma}$ extract all the rows of $\boldsymbol{\Lambda}$ such as $\Lambda_{s,4} = \sigma$ and store them in the $\boldsymbol{\Xi}$ matrix as follows:

$$\boldsymbol{\Xi} = \begin{bmatrix} s_1 & x_1 & y_1 & z_1 & \sigma & \mu_1 & \zeta_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s_\eta & x_\eta & y_\eta & z_\eta & \sigma & \mu_\eta & \zeta_\eta \end{bmatrix}_{\eta \times 7} \tag{14}$$

Step 4: for each row $\boldsymbol{\xi}_m \in \boldsymbol{\Xi}$, if:

$$\left\{\exists \boldsymbol{e}^2_h | \boldsymbol{e}^2_h = [\xi_{m,6}, \lambda_{s,5}] \vee \boldsymbol{e}^2_h = [\lambda_{s,5}, \xi_{m,6}]\right\} \wedge \left\{\exists \boldsymbol{e}^3_k | \boldsymbol{e}^3_k = [\xi_{m,7}, \lambda_{s,5}] \vee \boldsymbol{e}^3_k = [\lambda_{s,5}, \xi_{m,7}]\right\} = 1 \tag{15}$$

then append the 3D edge $\boldsymbol{\theta} = [\rho, \xi_{m,1}]$ to the 3D edges list $\boldsymbol{\Theta}$.

Step 5: repeat steps from 1 to 5 until no row $\boldsymbol{\lambda}_s \in \boldsymbol{\Lambda}$ remains.

Step 6: reset to the initial values the vectors $\boldsymbol{\omega}$, $\boldsymbol{\sigma}$ and the matrix $\boldsymbol{\Xi}$.

Step 7: for each row vector $\boldsymbol{\lambda}_s \in \boldsymbol{\Lambda}$ search all the rows $\boldsymbol{\lambda}_t \in \boldsymbol{\Lambda}$ (including $\boldsymbol{\lambda}_s$) so that $\lambda_{t,4} = \lambda_{s,4}$ and store them in the $\boldsymbol{\Xi}$ matrix as follows:

$$\boldsymbol{\Xi} = \begin{bmatrix} t_1 & x_1 & y_1 & z_1 & \lambda_{t,4} & \mu_1 & \zeta_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_\eta & x_\eta & y_\eta & z_\eta & \lambda_{t,4} & \mu_\eta & \zeta_\eta \end{bmatrix}_{\eta \times 7} \tag{16}$$

Step 8: let the vector $\boldsymbol{\tau}$ be the 1^{st} column of $\boldsymbol{\Xi}$; for each combination of two elements of $\boldsymbol{\tau}$, $\{\tau_h, \tau_k\}$, with $h \neq k$, if:

$$\left\{\exists \boldsymbol{e}_i^2 = [\xi_{h,6}, \xi_{k,6}] \vee \boldsymbol{e}_i^2 = [\xi_{k,6}, \xi_{h,6}]\right\} \wedge \left\{\exists \boldsymbol{e}_i^3 = [\xi_{h,7}, \xi_{k,7}] \vee \boldsymbol{e}_i^3 = [\xi_{k,7}, \xi_{h,7}]\right\} = 1 \quad (17)$$

then append $\boldsymbol{\theta} = [h, k]$ to the list $\boldsymbol{\Theta}$ of 3D edges.

The result of these phases consists of two sets. One of them, $\boldsymbol{\Lambda}$ represents the list of 3D vertices, while the second one, $\boldsymbol{\Theta}$, represent the list of 3D edges. Such sets mathematically represent, eventually, the *pseudo-wireframe* model (or the set of *pseudo-wireframe* models if the three views are not sufficient for defining a spatial object uniquely) of the object.

3. Results

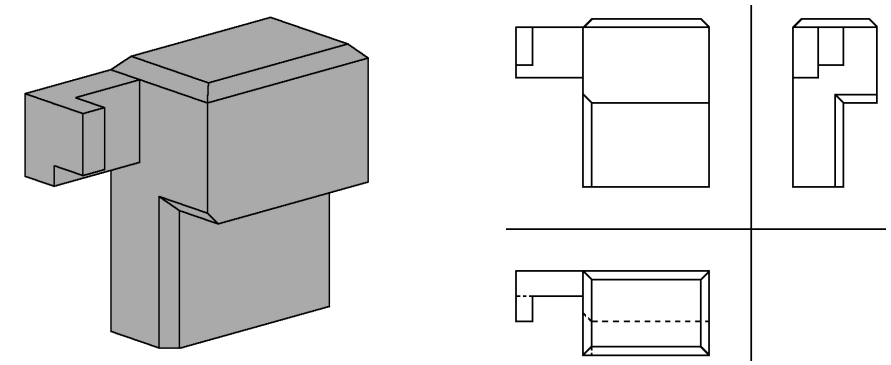
The mathematical procedure provided in Section 2 has been implemented in the MatLab® environment. The resulting software has been thoroughly tested with a large number of case studies. The test process has been carried out as follows:

Step 1: development of a 3D model for each object selected for the test;

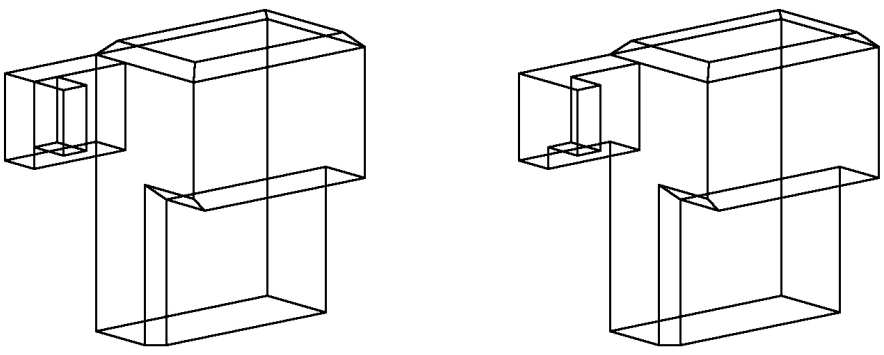
Step 2: extraction of the orthographic projections from the 3D model (in the form of a DXF file);

Step 3: processing of the DXF file by means of the presented reconstruction procedure thereby obtaining the test object's pseudo-wireframe;

Step 4: comparison between the test object's actual wireframe model and its pseudo-wireframe one.

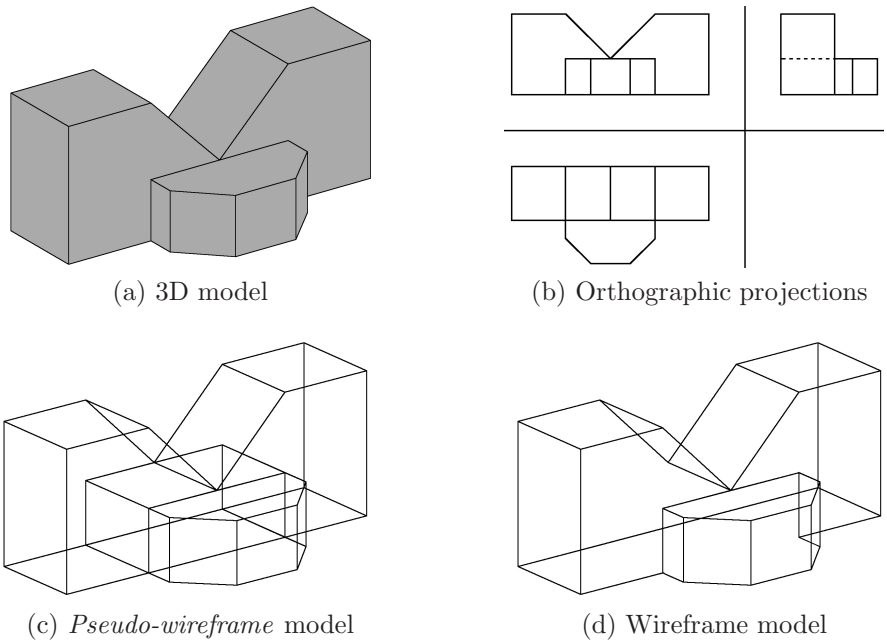


(a) 3D model (b) Orthographic projections



(c) *Pseudo-wireframe* model (d) Wireframe model

Figure 8: *Reconstruction* example — case A



(a) 3D model (b) Orthographic projections

(c) *Pseudo-wireframe* model (d) Wireframe model

Figure 9: *Reconstruction* example — case B

More in detail, in Step 4 it is necessary to prove that

- the actual wireframe model is a subset of the obtained pseudo-wireframe one;
- the exceeding edges (which can be found in the second model, but not in the first) can be projected on segments actually existing in the orthographic projections obtained in Step 2.

In Figures 8 and 9, the results of *pseudo-wireframe* model reconstruction for a small selection of examples are presented.

In particular, in Fig. 8d segments which are not parallel to at least one coordinate plane are properly reconstructed. As described above, if the three views are not sufficient for defining a spatial object uniquely, the proposed algorithm allows the reconstruction of all the possible wireframes coherent with the provided orthographic views. In this case the result consists of a set of wireframe models rather than a single wireframe.

4. Conclusions

In this work an orderly, unambiguous and automatic procedure to cope with the reconstruction problem from the implementation point of view is provided. Particularly, the proposed method allows the reconstruction of the pseudo-wireframe starting from 2D vectorial data. The presented procedure has been designed like a *support tool* for researchers who want to deal with the *reconstruction* problem in order to facilitate their work.

In order to assess its effectiveness, the procedure has been implemented using Matlab® programming language and tested on a number of case studies. The presented results demonstrate the functionality and the reliability of the provided method. Future work will be addressed to the second phase of the *reconstruction* problem; accordingly it will deal with the reconstruction of 3D solid (or surface) model(s) starting from the *pseudo-wireframe* ones, obtained by means of the presented procedure.

References

[1] Z. Chen, D. Perng, C. Chen, C. Wu: *Fast reconstruction of 3D mechanical parts from 2D orthographic views with rules*. Internat. Journal of Computer Integrated Manufacturing **5** (1), 2–9 (1992).

[2] P. Company, A. Piquer, M. Contero, F. Mnaya: *A survey on geometrical reconstruction as a core technology to sketch-based modeling*. Computers & Graphics **29** (6), 892–904 (2005).

[3] R. Diestel: *Graph theory*. Graduate texts in mathematics, Springer, Berlin 2006.

[4] M.A. Fahiem, S.a. Haq, F. Saleemi: *A Review of 3D Reconstruction Techniques from 2D Orthographic Line Drawings*. Geometric Modeling and Imaging (GMAI '07), 60–66, (2007).

[5] J. Gong, G. Zhang, H. Zhang, J. Sun: *Reconstruction of 3D curvilinear wire-frame from three orthographic views*. Computers & Graphics **30** (2), 213–224 (2006).

[6] U.G. Gujar, I. Nagendra: *Construction of 3D solid objects from orthographic views*. Computers & Graphics **13** (4), 505–521 (1989).

[7] M. Idesawa: *A System to Generate a Solid Figure from Three View*. Bulletin of JSME **16** (92), 216–225 (1973).

[8] K. Inoue, K. Shimada, K. Chilaka: *Solid Model Reconstruction of Wireframe CAD Models Based on Topological Embeddings of Planar Graphs*. Journal of Mechanical Design **125** (3), 434–442 (2003).

[9] G. Lafue: *Recognition of Three-Dimensional Objects from Orthographic Views*. Proc. 3rd Annual Conference on Computer Graphics, Interactive Techniques, and Image Processing, ACWSIGGRAPH 1976, pp. 103–108.

[10] S. Liu: *Reconstruction of curved solids from engineering drawings*. Computer-Aided Design **33** (14), 1059–1072 (2001).

[11] S. Meeran, M. Pratt: *Automated feature recognition from 2D drawings*. Computer-Aided Design **25** (1), 7–17 (1993).

[12] H. Sakurai, D. Gossard: *Solid model input through orthographic views*. ACM SIGGRAPH Computer Graphics **17** (3) (1983).

[13] L. Shixia, H. Shimin, S. Jiaguang: *Two accelerating techniques for 3D reconstruction*. Journal of Computer Science and Technology **17** (3), 2002.

[14] M. Wesley, G. Markowsky: *Fleshing out wire frames*. IBM Journal of Research and Development **24** (5), 582–597 (1980).

[15] M. Wesley, G. Markowsky: *Fleshing out projections*. IBM Journal of Research and Development **25** (6), 934–954 (1981).

[16] Q. Yan, C. Chen, Z. Tang: *Efficient algorithm for the reconstruction of 3D objects from orthographic projections*. Computer-Aided Design **26** (9), 699–717 (1994).

[17] A. Zhang, Y. Xue, X. Sun, Y. Hu, Y. Luo, Y. Wang, S. Zhong, J. Wang, J. Tang, G. Cai: *Reconstruction of 3D Curvilinear Wireframe Model from 2D Orthographic Views*. Lecture Notes in Computer Science, In: 4th International Conference on Computational Science – ICCS, June 2004, pp. 404–412.

Received September 30, 2010; final form April 22, 2011