Computer-Aided Design 44 (2012) 367–378

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

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

Semiautomatic detection of floor topology from CAD architectural drawings

B. Domínguez *, Á.L. García, F.R. Feito

Departamento de Informática, Universidad de Jaén - Campus de Las Lagunillas s/n. - 23071 Jaén, Spain

ARTICLE INFO

Article history:
Received 15 August 2010
Accepted 29 December 2011

Keywords:
Building information models
CAD drawings
Computational geometry
Graph theory

ABSTRACT

A method for the semiautomatic detection of the topology of building floors represented as CAD drawings stored in vector file format is presented in this paper. This method involves the detection of walls and joint points amid walls and openings, and the search of intersection points amid walls.

To give support to the wall detection process, this paper introduces the wall adjacency graph (WAG), a data structure created to detect walls from sets of planar segments contained in architectural floor plans. Wall adjacency graphs allow us to obtain a consistent and exhaustive set of walls very quickly (less than one second for real floor plans). A generalized version of the wall adjacency graph is also presented to deal with some of the limitations of the initial WAG. Algorithms for the detection of joint points and wall intersection points are presented as well, based on the analysis of the geometry from the input CAD drawings. Moreover, all this process works appropriately with both straight and circular segments.

The obtained floor topology can later be used as input to generate 3D models of buildings, which are widely used on virtual cities, BIM systems and GIS.

1. Introduction

There is little doubt about the evolution of the way humans interact with computers. Year after year, new advances in hardware and software are leading us to new and amazing sensory experiences in which virtual environments are combined with other kinds of data in order to fulfill all the informational needs of the users. Moreover, the computer performance/price ratio is continuously growing, and nowadays the users can afford computers able to handle most of these features easily.

The popularity of navigation through virtual 3D environments is rapidly growing, thanks to applications and development platforms like Google Earth© or Microsoft Virtual Earth©, which allow users to not only explore aerial and satellite photographs all over the world, but also examine three-dimensional reconstructions of buildings and monuments (when available). While most of these tools are intended for outdoor navigation, there also exist websites which offer interactive indoor 3D navigation in order to let the users enjoy virtual tours through the facilities of a company, a museum, a university, etc. These interactive virtual tours usually rely on a third party plug-in, and use VRML, X3D [1], COLLADA [2], and other technologies related to Web3D [3] to describe the geometry and appearance of the scenes as well as the behavior of their elements.

Architectural 3D indoor scenes are typically developed from scratch, or commercial applications like Autodesk© 3ds Max© are used to perform some kind of 2D-to-3D conversion using the floor plan of the scene as input, but this process is not straightforward, and the user must modify the preliminary result in order to obtain the final geometry. It is therefore desirable to have automatic tools which can take a 2D floor plan as input, and generate a 3D model in a format general enough to be used for different purposes.

As a first step of this automatic process, this paper is focused on extracting topological information from 2D vector floor plans composed of low-level geometric primitives. This aim implies the development of algorithms to analyze geometry and obtain semantic features that can be used on Building Information Model (BIM) systems, GIS and city models [4–7].

The rest of the paper is structured as follows: first, some previous works are summarized in Section 2. Sections 3–7 explain the wall detection process using wall adjacency graphs (WAGs). Section 8 describes the detection of joint points amid walls and openings and the search of intersection points amid walls. Finally, Sections 9 and 10 present the results, conclusions and future work.

2. Previous work

There are several studies related to the processing of architectural floor plans. They can be grouped according to the kind of input and the goals of the process:

- Some interesting works introduce methods to recognize special symbols from a floor plan. These methods typically apply Computer Vision and Pattern Recognition techniques

* Corresponding author. Tel.: +34 953212902.
E-mail addresses: bdmartin@ujaen.es (B. Domínguez), algarcia@ujaen.es (Á.L. García), ffeito@ujaen.es (F.R. Feito).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2011.12.009

on scanned floor plans to obtain their results. Ah-Soon and Tombre [8] use networks of constraints on geometrical features to find symbols that represent doors and windows, defining a formal grammar to describe what have to be recognized. On the other hand, Dosch et al. [9] describe a complete system for symbol recognition involving Computer Vision tools such as segmentation, vectorization and feature detection; the result is used to create a 3D reconstruction of the floor (without topology information) by means of extruding the recognized 2D geometry. Other interesting works include the one from [10], who apply the Hough transform to recognize symbols from hand-drawn architectural floor plans, and the one from [11], who deal with the recognition of structural objects and the classification of wall shapes.

- CAD vector drawings are the input data considered by other researchers that propose methods to generate 3D building models. These include the work from [12], who define the generalized-maps to represent adjacency relations amid geometric elements, and use them to extract 2D topology and 3D volumes from a floor plan, given some assumptions on the quality of the floor plan and some considerations about the structure of a building that allow them to define constraints on the geometry; occasionally, the user intervention can be necessary to provide semantic associations to the geometry, and curved walls are considered as polylines. As far as we know, no further advances on this approach have been published up to now. Other works are the one by Mas and Besuievsky [13], based on the extrusion of planar polygons from CAD vector drawings to generate 3D building models used for light simulation, and the one by Paoluzzi et al. [14], who use 3D reconstruction for security modeling of critical infrastructures.
- A third group of works introduce methods to retrieve topological information from CAD vector drawings, like the one from [15], who use graphs to store adjacency relations amid floors, rooms and stairs during the design stage. Other related work is the one from [16], who build a topology graph to describe the distribution of walls and openings in a floor plan, and search for a set of fundamental loops to find corridors and rooms, so that an evacuation plan can be created from that information; this work emphasizes the loop search, but gives very few details on the graph building process.

3. Problem overview

The problem we face here is the extraction of topological information from a building floor plan. Typically, building designs are stored as vector drawings created with CAD software applications and composed of low level graphical primitives, like lines, polylines, circular arcs, etc. Optionally, blocks can be created as combinations of primitives and/or other blocks; this allow the user to easily create and instantiate representations for common elements like doors, windows or furniture.

Moreover, CAD applications allow the designer to split the graphical information in layers to group related elements together, and whose visibility can be switched on or off in order to emphasize what is really important in every situation.

In order to get the topology from a floor plan, it is necessary to obtain the information about the walls and openings it contains. Although there exist several standards on how the layers should be organized in architectural design [17–19], CAD applications do not force the users to follow any of these standards, and therefore it is possible to find CAD drawings where the information regarding walls and openings is mixed with other data or divided into several layers. The second situation is easily solved by mixing the contents of the layers that keep the desired information, whereas the first situation is still an open problem, because typically there is no additional data (apart from line attributes like color or thickness) to support automatic extraction of primitives from the layers without involving the user. Here we consider that the walls are stored in one or several layers, but mixed with nothing else than (optionally) the openings, and that blocks represent the openings.

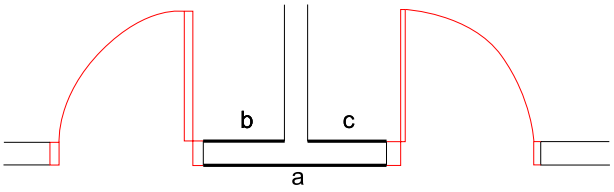


Fig. 1. The mapping between pairs of segments and walls is not bijective, e.g. the walls *ab* and *ac* share the segment *a*.

One additional problem that appears sometimes when processing CAD vector drawings is related with precision: overlapping segments or duplicated vertices can appear if the user is not careful enough when creating the floor plan. These situations have to be detected and corrected by checking all the geometry and merging faulty vertices and segments, and may require the user intervention to set a precision threshold. To be precise, the following tasks have been implemented successfully in our test application [20]: segments with zero length are removed from the floor plan, partially overlapping segments are replaced by a unique segment obtained by merging them, and in those cases where there are two copies of the same segment, one of them is removed. Occasionally, these changes may produce new precision related problems; therefore, the process of checking and correcting these situations has to loop automatically over the geometry until there are no more changes to be applied.

Every wall in a vector drawing of a floor plan is represented by two parallel line segments, but the mapping between pairs of segments and walls is not bijective, i.e. two consecutive walls may share one segment (see Fig. 1).

Furthermore, there is no information about which segments match to make up a wall. Therefore, the wall detection process involves searching for *wall-prone* pairs of segments using a threshold ε (maximum wall thickness), and split them in order to obtain wall pairs. The following definitions clarify these concepts:

Definition 1 (*Wall-Prone Pair of Line Segments*)**.** Let a and b be line segments in $\mathbb{R}^2$. Let r and s be the lines containing a and b respectively, and a' and b' the projections of a and b onto s and r. Given a fixed threshold ε, the pair (a, b) is wall-prone (represented by the predicate $prone(a, b, \varepsilon)$) if and only if all these conditions are held:

C1. a and b are parallel: $a \parallel b$
C2. a' and b (and therefore b' and a) overlap: $a' \cap b \neq \emptyset$
C3. The distance between r and s is less than or equal to the threshold: $d(r, s) \leq \varepsilon$.

Note: In this work line segments are considered open (their end points do not belong to them), to avoid the special case where two line segments with consecutive projections hold condition C2.

Definition 2 (*Wall Pair of Line Segments*)**.** Two line segments a and b are said to form a wall pair, given a fixed threshold ε (represented by the predicate $wall(a, b, \varepsilon)$), if they hold the conditions to be a wall-prone pair, and their projections onto the line that contain each other match. Therefore, a new, more restrictive condition is added to C1, C2 and C3:

C4. a' and b (and therefore b' and a) are the same: $a' = b$.

Some properties of these predicates are immediate:

- *prone* does not depend on the segment order: $prone(a, b, \varepsilon) \Leftrightarrow prone(b, a, \varepsilon)$

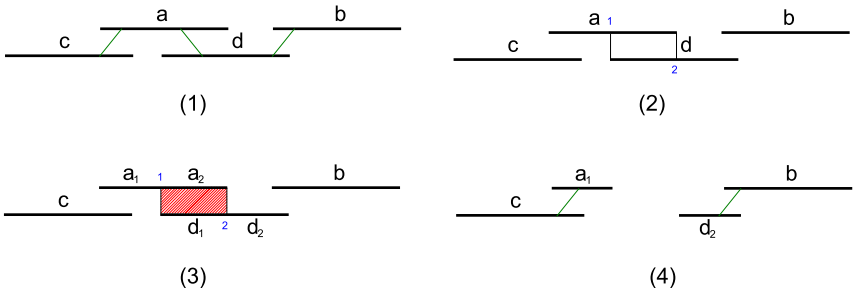


Fig. 2. Iteration of the algorithm. (1) Initial set of segments and relations. (2) Projection of the end points. (3) Segment splitting and wall extraction. (4) Updated segments and relations.

- *wall* does not depend on the segment order: $wall(a, b, \varepsilon) \Leftrightarrow wall(b, a, \varepsilon)$
- *wall* is a restriction of *prone*: $wall(a, b, \varepsilon) \Rightarrow prone(a, b, \varepsilon)$.

Given the above definitions, this is the outline of the algorithm to compute the walls represented by a set of segments in a floor plan (see example in Fig. 2):

(1) Find all the wall-prone pairs. In Fig. 2(1), these pairs are (a, c), (a, d) and (b, d).
(2) For each wall-prone pair, apply the following steps (in this example, we take the pair (a, d)):
  (2.a) Compute the projections of the end points of one segment on the other segment. Applying this step to the wall-prone pair (a, d) results in points 1 and 2, as shown in Fig. 2(2).
  (2.b) Split each segment using the computed projections and check the new set of segments for wall pairs: some segments will form wall pairs while the rest will become *single* (their projections do not overlap). In Fig. 2(3), the resulting segments are a_1, a_2, d_1 and d_2. The new detected wall pair is (a_2, d_1), and the single segments are a_1 and d_2.
  (2.c) Update the set of segments by removing the old segments (a and d in this case) and adding the single ones (a_1 and d_2). Then, update the wall-prone pair set; in this example, wall-prone pairs (a_1, c) and (b, d_2) are added, as shown in Fig. 2(4).

In order to make the processing of wall and wall-prone relations amid segments easier, and keep record of the hierarchical relations between each line segment and the pieces it is split into, the Wall Adjacency Graph is proposed as a data structure to give support to this process. The following sections give a detailed description of this structure and its application to the problem.

4. The Wall Adjacency Graph

The Wall Adjacency Graph (WAG) is a graph whose nodes represent the line segments from a floor plan that are involved in the representation of walls, and whose edges represent relations between those segments. In order to represent the walls drawn in a floor plan, three kind of relations between segments are defined as follows:

Definition 3 (*Wall-Prone Relation*)**.** Given a finite set of line segments $\mathcal{A}$ and a fixed threshold ε, the wall-prone relation over $\mathcal{A}$ is defined as the set

$$PR(\mathcal{A}, \varepsilon) = \{(a, b) \in \mathcal{A} \times \mathcal{A} \mid prone(a, b, \varepsilon)\}.$$

Definition 4 (*Wall Relation*)**.** Given a finite set of line segments $\mathcal{A}$ and a fixed threshold ε, the wall relation over $\mathcal{A}$ is defined as the set

$$W(\mathcal{A}, \varepsilon) = \{(a, b) \in \mathcal{A} \times \mathcal{A} \mid wall(a, b, \varepsilon)\}.$$

As the relations defined above are based on Definitions 1 and 2, the following properties hold:

- $PR(\mathcal{A}, \varepsilon)$ is symmetric: $(a, b) \in PR(\mathcal{A}, \varepsilon) \Leftrightarrow (b, a) \in PR(\mathcal{A}, \varepsilon)$
- $W(\mathcal{A}, \varepsilon)$ is symmetric: $(a, b) \in W(\mathcal{A}, \varepsilon) \Leftrightarrow (b, a) \in W(\mathcal{A}, \varepsilon)$
- $W(\mathcal{A}, \varepsilon) \subseteq PR(\mathcal{A}, \varepsilon)$.

As the wall-prone pairs are being processed, their segments are split into pieces. For each segment a, the set of pieces it is split into form a partition $P(a)$ of that segment, because they do not overlap.

In order to store in the Wall Adjacency Graph the information about partitions, one more relation is defined as follows:

Definition 5 (*Hierarchical Relation*)**.** Given a finite set of line segments $\mathcal{A}$, the hierarchical relation over $\mathcal{A}$ is defined as the set

$$H(\mathcal{A}) = \{(a, b) \in \mathcal{A} \times \mathcal{A} \mid b \in P(a)\}.$$

Unlike the wall and wall-prone relations, the hierarchical relation is obviously not symmetric.

Once all the elements that are involved in the Wall Adjacency Graph are defined, a formal definition of this structure follows:

Definition 6 (*Wall Adjacency Graph*)**.** Given a finite set of line segments $\mathcal{A}$ and a fixed threshold ε, the Wall Adjacency Graph (WAG) associated with $\mathcal{A}$ is a graph $G(\mathcal{A}, \varepsilon) = (\mathcal{A}, PR(\mathcal{A}, \varepsilon) \cup H(\mathcal{A}))$.

For a given floor plan, its WAG is therefore formed by the line segments it contains, together with the relations amid them. A WAG is not necessarily connected, and this is not a requirement for the algorithms to work successfully.

Due to the fact that $W(\mathcal{A}, \varepsilon) \subseteq PR(\mathcal{A}, \varepsilon)$, it is necessary to distinguish the set of *strict* wall-prone segment pairs that do need to be processed to get wall pairs; therefore, the set $\overline{W}(\mathcal{A}, \varepsilon)$ is defined as the complement of $W(\mathcal{A}, \varepsilon)$ with respect to $PR(\mathcal{A}, \varepsilon)$, i.e. $\overline{W} = PR(\mathcal{A}, \varepsilon) \setminus W(\mathcal{A}, \varepsilon)$.

From now on, WAG edges representing wall and wall-prone relations will be notated (when necessary) as unordered pairs $\{a, b\}$, as these relations are symmetric; similarly, WAG edges representing hierarchical relations will be notated as ordered pairs (a, b).

Example 1 (*Initial WAG*)**.** Fig. 3 shows an example of (a) an initial set of line segments and (b) its associated WAG. Edges representing wall-prone relations are drawn with single lines, while edges representing wall relations appear as double lines. The elements defining the WAG are:

$$\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$$
$$PR(\mathcal{A}, \varepsilon) = \{\{a_1, a_4\}, \{a_2, a_4\}, \{a_3, a_5\}\}$$
$$H(\mathcal{A}) = \emptyset.$$

The pairs in $PR(\mathcal{A}, \varepsilon)$ can be grouped into wall and wall-prone sets:

$$W(\mathcal{A}, \varepsilon) = \{\{a_3, a_5\}\}$$
$$\overline{W}(\mathcal{A}, \varepsilon) = \{\{a_1, a_4\}, \{a_2, a_4\}\}.$$

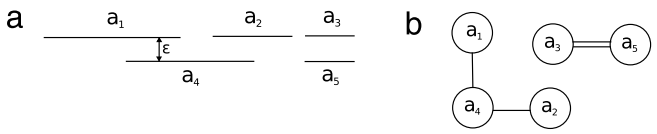


Fig. 3. Example of (a) a set of line segments (b) its associated WAG using a threshold ε. Single lines represent wall-prone relations, while double lines represent wall relations.

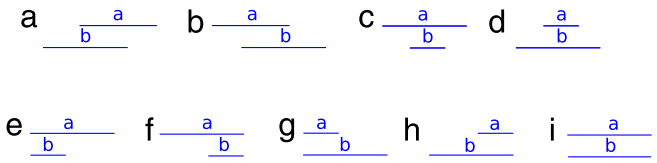


Fig. 4. Possible layouts when processing wall-prone pairs of line segments: (a) *intersection1*, (b) *intersection2*, (c) *contained1*, (d) *contained2*, (e) *common1*, (f) *common2*, (g) *common3*, (h) *common4* and (i) *matching*.

5. Wall detection algorithm

In Section 3, an outline of the wall detection algorithm was described. Now that description will be revised, including the WAG as underlying data structure for this algorithm.

The input of this process is the set of line segments that represent the walls in a vector drawing of a floor plan. As mentioned in Section 3, these segments are considered as the unique contents of one layer of the drawing, or alternatively, the unique result of the mixture of several ones. Given this set, it is possible to estimate automatically the value for the threshold ε as described in [21], or the user can be required to fix this value, as it is essential for the rest of the process.

As a preliminary step, it is necessary to build the initial WAG for the segments. Its elements ($\mathcal{A}$, $PR(\mathcal{A}, \varepsilon)$ and $H(\mathcal{A})$) are assigned as follows:

- $\mathcal{A}$ is defined as the initial set of segments.
- $H(\mathcal{A})$ is empty.
- To build the initial set of wall-prone relations $PR(\mathcal{A}, \varepsilon)$, each possible pair of line segments (a, b) has to be analyzed to determine whether it holds the conditions to be a wall or a wall-prone pair. For the convenience of the wall detection algorithm, this set is subdivided into the subsets $W(\mathcal{A}, \varepsilon)$ and $\overline{W}(\mathcal{A}, \varepsilon)$, defined in Section 4.

Once the WAG has been created, the wall-prone pairs from the subset $\overline{W}(\mathcal{A}, \varepsilon)$ are processed in turn to obtain wall pairs. There are nine possible types of wall-prone pairs; the way they are processed will be detailed later.

The processing of each wall-prone pair results in a series of changes in the WAG:

- As described in Section 3, it could be necessary to divide the segments from the pair and use their pieces to build new wall and wall-prone pairs; new nodes would therefore be added to the WAG to represent these pieces. The set of added nodes will be notated as $\mathcal{A}^+$.
- Including new nodes in the WAG implies changing graph edges to represent the new wall and wall-prone pairs that are created. This results in the deletion of some wall-prone edges and the addition of new ones. The sets PR^- and PR^+ represent these edges. Moreover, a wall-prone pair is always deleted to produce a wall pair, and therefore a wall edge w has also to be added to the WAG.
- In order to keep track of the origin of each node, edges representing the hierarchical relation between the nodes corresponding to the segments that are split and the nodes representing the pieces they are split into are added to the graph. The set H^+ represents these edges.

At the end of the process, there only exist wall pairs in the structure, therefore $PR(\mathcal{A}, \varepsilon) \equiv W(\mathcal{A}, \varepsilon)$. Algorithm 1 formally describes the wall detection procedure using the WAG as supporting data structure.

Algorithm 1 Wall detection algorithm

Input: $\mathcal{A}$, ε
Build $\overline{W}(\mathcal{A}, \varepsilon) = \{\{a, b\} \mid a, b \in \mathcal{A} \wedge prone(a, b, \varepsilon) \wedge \neg wall(a, b, \varepsilon)\}$
Build $W(\mathcal{A}, \varepsilon) = \{\{a, b\} \mid a, b \in \mathcal{A} \wedge wall(a, b, \varepsilon)\}$
Build $H(\mathcal{A}) = \emptyset$
for all $\{a, b\} \in \overline{W}(\mathcal{A}, \varepsilon)$ **do**
  Study the layout of a and b
  Build $\mathcal{A}^+$, PR^-, PR^+, w and H^+ depending on a and b
  $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}^+$
  $W(\mathcal{A}, \varepsilon) \leftarrow W(\mathcal{A}, \varepsilon) \cup \{w\}$
  $\overline{W}(\mathcal{A}, \varepsilon) \leftarrow \overline{W}(\mathcal{A}, \varepsilon) \setminus PR^-$
  $\overline{W}(\mathcal{A}, \varepsilon) \leftarrow \overline{W}(\mathcal{A}, \varepsilon) \cup PR^+$
  $H(\mathcal{A}) \leftarrow H(\mathcal{A}) \cup H^+$
end for
return $G = (\mathcal{A}, W(\mathcal{A}, \varepsilon) \cup H(\mathcal{A}))$

The result of the processing of each wall-prone pair depends on the relative position between the segments that form the pair. Fig. 4 shows the nine possible cases that may appear. The changes that have to be applied to the WAG differ from one case to the other. In order to make the description of these changes easier, the set of wall-prone edges incident to a node a will be notated as $E(a)$, while the set of nodes adjacent to a node a will be notated as $V(a)$. The formal definition of these sets follows:

$$E(a) = \{\{a, x\} \in \overline{W}(\mathcal{A}, \varepsilon)\}$$
$$V(a) = \{x \in \mathcal{A} \mid \{a, x\} \in \overline{W}(\mathcal{A}, \varepsilon)\}.$$

Fig. 5 shows how the segments from each particular layout are split to get wall pairs. The nodes representing segments in green in the figure are connected using the new wall edges that are added to the WAG, while the nodes representing segments in red in the figure are connected with the wall-prone edges that are modified. Table 1 shows a detailed description of the contents of $\mathcal{A}^+$, PR^-, PR^+, w and H^+ for each particular case.

The following example focuses on the processing of a wall-prone edge of a WAG.

Example 2 (*Wall-Prone Edge Processing*). Fig. 6(2) shows the initial WAG for the set of line segments in Fig. 6(1). This WAG has three wall-prone edges which must be processed sequentially to find the corresponding wall pairs.

The segment pair represented by edge $\{a, d\}$ (in red in Fig. 6(2) corresponds to the case *intersection2* (Fig. 5(b)). Then, the endpoints of segments a and d are projected onto each other, resulting in points 1 and 4, and these points are used to split the segments into a_1, a_2, d_1 and d_2. Applying the definition of sets $\mathcal{A}^+$, PR^-, PR^+, w and H^+ from Table 1 results in:

- $\mathcal{A}^+ = \{a_1, a_2, d_1, d_2\}$
- $w = \{a_2, d_1\}$
- $H^+ = \{(a, a_1), (a, a_2), (d, d_1), (d, d_2)\}$
- $PR^- = \{\{a, d\}, \{a, c\}, \{b, d\}\}$
- $PR^+ = \{\{a_1, c\}, \{d_2, b\}\}$.

Fig. 6(3) shows the WAG after the set H^+ and the wall edge w have been added, while Fig. 6(4) shows the WAG after the set of edges PR^- has been deleted, and the set of edges PR^+ has been added.

Regarding the algorithm behavior, some interesting facts related to the graph structure follow:

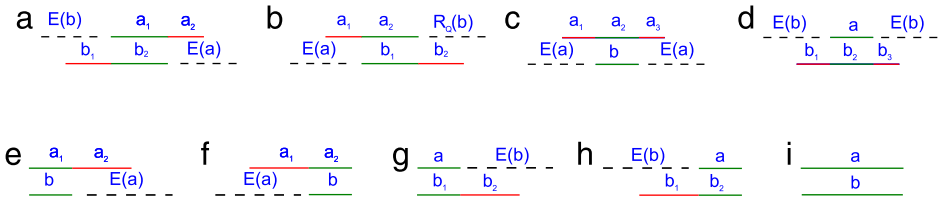


Fig. 5. Segment splitting for each particular case from Fig. 4.

Table 1
WAG modifications for each particular case in Fig. 5.

Layout	$\mathcal{A}^+$	w	H^+	PR^-	PR^+
Intersection1 (Fig. 5(a))	$\{a_1, a_2, b_1, b_2\}$	$\{a_1, b_2\}$	$\{(a, a_1), (a, a_2), (b, b_1), (b, b_2)\}$	$\{a, b\} \cup E(a) \cup E(b)$	$\{\{a_2, x\} \mid x \in V(a)\} \cup \{\{b_1, x\} \mid x \in V(b)\}$
Intersection2 (Fig. 5(b))	$\{a_1, a_2, b_1, b_2\}$	$\{a_2, b_1\}$	$\{(a, a_1), (a, a_2), (b, b_1), (b, b_2)\}$	$\{a, b\} \cup E(a) \cup E(b)$	$\{\{a_1, x\} \mid x \in V(a)\} \cup \{\{b_2, x\} \mid x \in V(b)\}$
Contained1 (Fig. 5(c))	$\{a_1, a_2, a_3\}$	$\{a_2, b\}$	$\{(a, a_1), (a, a_2), (a, a_3)\}$	$\{a, b\} \cup E(a)$	$\{\{a_1, x\} \mid x \in V(a) \wedge prone(a_1, x)\} \cup$ $\{\{a_3, x\} \mid x \in V(a) \wedge prone(a_3, x)\}$
Contained2 (Fig. 5(d))	$\{b_1, b_2, b_3\}$	$\{a, b_2\}$	$\{(b, b_1), (b, b_2), (b, b_3)\}$	$\{a, b\} \cup E(b)$	$\{\{b_1, x\} \mid x \in V(b) \wedge prone(b_1, x)\} \cup$ $\{\{b_3, x\} \mid x \in V(b) \wedge prone(b_3, x)\}$
Common1 (Fig. 5(e))	$\{a_1, a_2\}$	$\{a_1, b\}$	$\{(a, a_1), (a, a_2)\}$	$\{a, b\} \cup E(a)$	$\{\{a_2, x\} \mid x \in V(a)\}$
Common2 (Fig. 5(f))	$\{a_1, a_2\}$	$\{a_2, b\}$	$\{(a, a_1), (a, a_2)\}$	$\{a, b\} \cup E(a)$	$\{\{a_1, x\} \mid x \in V(a)\}$
Common3 (Fig. 5(g))	$\{b_1, b_2\}$	$\{a, b_1\}$	$\{(b, b_1), (b, b_2)\}$	$\{a, b\} \cup E(b)$	$\{\{b_2, x\} \mid x \in V(b)\}$
Common4 (Fig. 5(h))	$\{b_1, b_2\}$	$\{a, b_2\}$	$\{(b, b_1), (b, b_2)\}$	$\{a, b\} \cup E(b)$	$\{\{b_1, x\} \mid x \in V(b)\}$
Matching (Fig. 5(i))	$\emptyset$	–	$\emptyset$	$\emptyset$	$\emptyset$

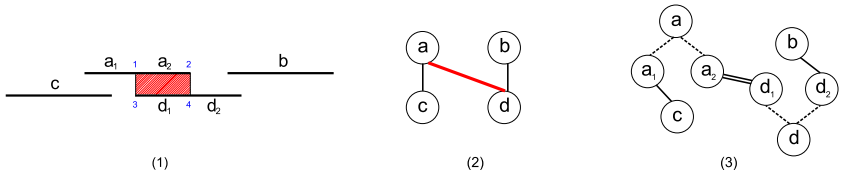


Fig. 6. Example of one step of the algorithm. (1) Initial set of segments. (2) Wall-prone edge $\{a, d\}$ is processed. (3) WAG after the appropriate changes have been applied.

1. The initial WAG does not contain hierarchical edges.
2. When a segment is split and new nodes and hierarchical edges are created to represent this changes, the resulting structure can be viewed as a (sub)tree whose root is the node that represents the split segment. All the wall-prone edges incident to the root are changed for wall-prone edges incident to the leaf nodes. Thus, there are never wall-prone edges incident to non-leaf nodes (initially all the nodes can be considered as leaves).
3. Two nodes connected with a wall edge cannot be connected with any other node with a wall-prone edge, because their projections onto each other overlap (see Fig. 4(i)).
4. As a consequence of the above points, nodes connected with wall edges are leaf nodes, because the segments they represent will not need to be split. Thus, wall edges always connect leaf nodes.
5. Although only leaf nodes are necessary after the algorithm has processed the WAG, non-leaf nodes are kept for further queries about the original geometry.

The following list shows other interesting remarks about the algorithm behavior and performance:

1. The algorithm execution always ends, because one wall-prone edge is removed during each iteration, and the other wall-prone edges incident to the nodes that are connected by the edge that is removed are replaced by the same amount of wall-prone edges. Thus, the number of iterations equals the number of wall-prone edges.
2. The execution time of the algorithm is $O(n)$, with respect to the initial number of wall-prone edges. Operations executed on each iteration are $O(1)$, as they do not depend on the size of the set of wall-prone edges.
3. The initialization of $W(\mathcal{A}, \varepsilon)$ and $\overline{W}(\mathcal{A}, \varepsilon)$ is $O(n^2)$ with respect to the number of segments in the floor plan, since all the pairs of segments have to be studied. This stage could be optimized using spatial indices to avoid comparisons amid distant segments; this possibility will be studied for future work. However, for real floor plans the number of line segments is of the order of 10^3, and for this size of the problem, the initialization of the WAG is executed approximately in half a second, and the wall detection algorithm is executed in a few milliseconds with current CPUs. See Section 9 for more details.
4. The set of walls can be obtained directly from the set of wall edges.

6. Generalized Wall Adjacency Graph

The algorithm in Section 5 is based on the assumption that no more than two parallel segments are close enough to be considered as a wall-prone pair. However, there are situations where more than two parallel segments may appear in a small enough range that the algorithm does not handle appropriately, producing runtime inconsistencies. See Fig. 7(a). The initial WAG for this segment layout is $G = (\mathcal{A}, PR(\mathcal{A}, \varepsilon), H(\mathcal{A}))$, where $\mathcal{A} = \{a, b, c\}$, $PR(\mathcal{A}, \varepsilon) = \{\{a, b\}, \{a, c\}\}$ and $H(\mathcal{A}) = \emptyset$, as shown in Fig. 7(b). $PR(\mathcal{A}, \varepsilon)$ is in turn divided into $W(\mathcal{A}, \varepsilon) = \emptyset$ and $\overline{W}(\mathcal{A}, \varepsilon) = PR(\mathcal{A}, \varepsilon)$.

The first loop iteration of the wall detection algorithm takes the wall-prone edge $\{a, b\}$ and applies the appropriate rule to modify the graph structure as described previously. The result of these modifications is shown in Fig. 7(c), and the resulting WAG elements are:

$$\mathcal{A} = \{a, a_1, a_2, b, b_1, b_2, c\}$$
$$\overline{W}(\mathcal{A}, \varepsilon) = \{\{a_1, c\}\}$$
$$W(\mathcal{A}, \varepsilon) = \{\{a_2, b_1\}\}$$
$$H(\mathcal{A}) = \{(a, a_1), (a, a_2), (b, b_1), (b, b_2)\}.$$

This WAG contains two semantic errors with respect to the segment layout after splitting segments a and b:

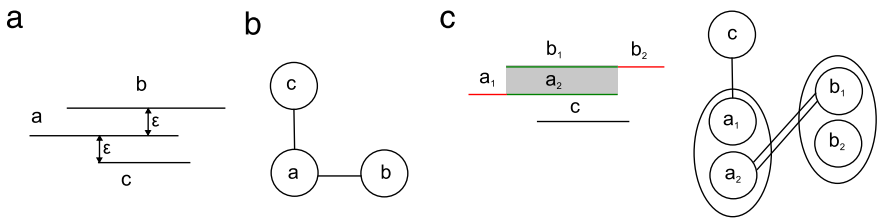


Fig. 7. (a) Three rows of parallel segments. (b) Initial WAG. (c) Result of applying the wall detection algorithm to the wall-prone edge $\{a, b\}$. (Note: the hierarchical relations are represented here as ovals that group the nodes that represent the pieces a segment has been split into.)

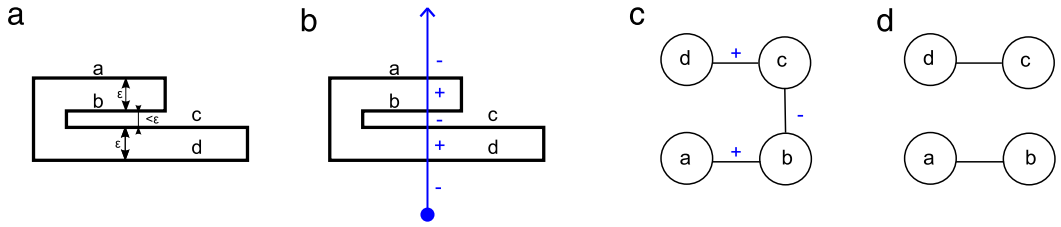


Fig. 8. (a) Narrow spaces between close walls can lead to a wrong WAG; (b) Jordan curve theorem based solution: ray tracing and intersection labeling; (c) labeled WAG; (d) corrected WAG.

- The wall-prone edge $\{a_1, c\}$ has been added to the graph. However, these segments do not form a wall-prone pair.
- The segments a_2 and c form a wall-prone edge, but there is no wall-prone edge in the graph to represent this situation.

There are two possible strategies to appropriately handle situations like the one described above, as it is not reasonable to force the user to create floor plans taking into account this issue:

(1) Study situations with multiple rows of segments making up wall-prone pairs, like the ones shown in Fig. 7, that lead to a wrong initial WAG, in order to set up an automatic filter for the segments from the floor plan. This strategy is overviewed in Section 6.1
(2) Modify the WAG structure and the wall processing algorithm, so that more than one partition per segment is accepted. This strategy leads us to introduce the generalized WAG, covered in Section 6.2.

6.1. Handling of multiple rows of segments

It can be observed that when there exist narrow empty spaces between close walls, the corresponding WAG contains connected components whose nodes represent segments contained in more than two parallel lines. Thus, we can assume that:

1. A WAG connected component always represents an even number of parallel rows of segments (except for irregular designs). Each pair of segments thus determine either an interior of a wall or an exterior area between close walls.
2. There is always an empty space between two walls; respectively, there is always a wall between two empty spaces.
3. When a pair of segments is wrongly considered as a wall-prone pair, the initial WAG will contain an edge that has to be removed.

Therefore, we can apply a process based on the Jordan curve theorem [22] to remove wrong wall-prone edges from the initial WAG (see Fig. 8):

1. For each WAG connected component, a ray perpendicular to the segments represented by its nodes is traced from an external point. Each intersection is labeled as positive (incoming) or negative (outgoing). The first intersection is considered always as positive (Fig. 8(b)).
2. The corresponding WAG wall-prone edges are labeled according to the sign of their intersections with the ray (Fig. 8(c)).
3. Wall-prone edges labeled as negative are removed from the WAG (Fig. 8(d)).

After this process, the wall detection algorithm can be executed on the WAG without runtime inconsistencies. However, this approach has significant drawbacks:

- The WAG needs to be analyzed after its construction to detect connected components whose nodes represent segments contained in more than two parallel lines. This additional analysis can be complex, and reduces the advantages of constructing a scenario-independent WAG.
- As the starting point of the ray has to be changed for each WAG connected component, it is not always a straightforward task to place it correctly. Computing correct locations for the starting points implies additional geometrical analysis of the floor plan, therefore increasing the execution time.

6.2. Generalizing the wall detection process

The problem described above with a segment layout like the one shown in Fig. 7(a) is due to the fact that each iteration of the wall detection algorithm removes from the WAG those wall-prone edges incident to the node that represents the segment that is split, and substitutes them by wall-prone edges connecting the nodes that represent the pieces the segment is split into, as shown in Fig. 7(c). In order to avoid these problems, the WAG structure and the wall detection algorithm are modified in the following way:

- The only wall-prone edge that is removed in each iteration of the algorithm will be the one that is being processed. Therefore, for each iteration of the algorithm, $PR^- = \{a, b\}$ (given the notation used in Table 1).
- The graph will allow more than one partition for the same segment.
- Wall-prone edges will be allowed to connect non-leaf nodes.

Given these modifications, the resulting graph is notated as *Generalized* Wall Adjacency Graph.

Using the generalized WAG and the modified wall detection algorithm, the segment layout shown in Fig. 7(a) is handled as

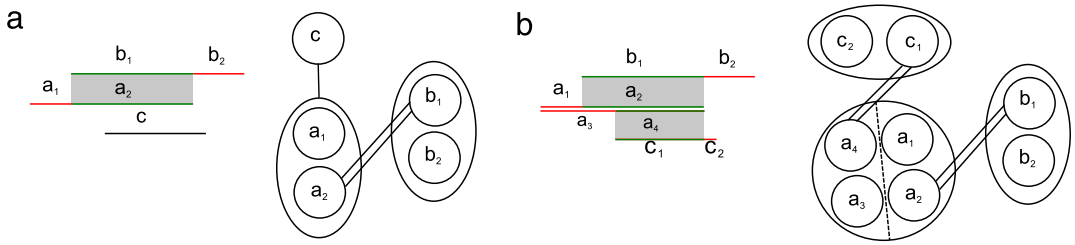


Fig. 9. Processing the segment layout from Fig. 7(a) using a generalized WAG. (a) Result of processing the edge $\{a, b\}$. (b) Result of processing the edge $\{a, c\}$.

shown in Fig. 9. The elements of the generalized WAG after processing the wall-prone edge $\{a, b\}$ are the following (Fig. 9(a)):

$$\mathcal{A} = \{a, a_1, a_2, b, b_1, b_2, c\}$$
$$\overline{W}(\mathcal{A}, \varepsilon) = \{\{a, c\}\}$$
$$W(\mathcal{A}, \varepsilon) = \{\{a_2, b_1\}\}$$
$$H(\mathcal{A}) = \{(a, a_1), (a, a_2), (b, b_1), (b, b_2)\}.$$

Finally, the processing of the wall-prone edge $\{a, c\}$ results in the situation shown in Fig. 9(b). The elements of the final generalized WAG are:

$$\mathcal{A} = \{a, a_1, a_2, a_3, a_4, b, b_1, b_2, c, c_1, c_2\}$$
$$\overline{W}(\mathcal{A}, \varepsilon) = \emptyset$$
$$W(\mathcal{A}, \varepsilon) = \{\{a_2, b_1\}, \{a_4, c_1\}\}$$
$$H(\mathcal{A}) = \{(a, a_1), (a, a_2), (a, a_3), (a, a_4), (b, b_1), (b, b_2), (c, c_1), (c, c_2)\}.$$

The changes made to the wall detection algorithm do not substantially change its behavior:

1. The algorithm execution always ends, because only one wall-prone edge is removed in each iteration of the algorithm. The number of iterations of the algorithm is therefore equal to the number of wall-prone edges, and the execution time is $O(n)$. However, it is expected to be slightly lower than the time taken to process the *old* WAG, because the amount of graph changes per iteration is smaller.
2. Each node from the initial graph can be the root of *several* subtrees, each of them representing one partition of the segment represented by the root node. Wall edges always connect leaf nodes, while wall-prone edges can connect any node.
3. The set of walls can be obtained directly from the set of wall edges.
4. Possible graph cliques would be processed without problems, as the generalization of the WAG allow the presence of more than two parallel segments in a small region of a floor plan. This could indeed result in a graph clique, but the possibility of creating more than one partition for a segment allows the algorithm to cleanly process the segments.

7. Processing curved walls

Current architectural designs often contain not only straight walls, but also curved walls in order to create more expressive, dynamic and human-friendly spaces. Typically, curved lines in CAD designs are represented as circular arcs (defined by their circle center, their radius and their angular interval) or as polylines that approximate the curved lines. Wall detection algorithm handles the second case successfully, as the final representation is a set of straight segments. Here we describe the changes in the wall detection algorithm to process walls defined using pairs of circular arcs.

Table 2 summarizes the criteria to form wall-prone pairs of straight segments, and shows how they are adapted to circular arcs.

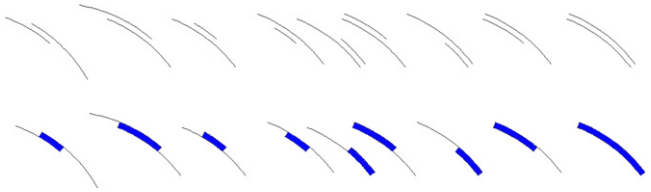


Fig. 10. Up: pairs of circular arcs. Down: detected walls.

Using these criteria, the wall detection algorithm has been modified to work not only with straight segments, but also with circular arcs, using angular values to split the arcs as necessary to create the corresponding wall pairs. The remaining parts of the process are analogous to the ones using straight segments. Fig. 10 shows some examples of the processing of circular arcs.

8. Building the floor topology

As stated before, the set of walls from a floor plan can be obtained simply by checking the wall edges from the result of applying the wall detection algorithm to the WAG associated to the floor plan segments. However, the walls are not the only element that forms the topology of a floor, since the openings (typically doors and windows) also play an important topological role as links between the spaces defined by the walls. Moreover, the intersections amid walls are not detected by the algorithm, and therefore it is necessary to process the detected walls to build these intersections.

In order to get a simplified representation of a floor plan that represents its topology, each detected wall is represented by a single segment placed in between its corresponding wall pair of segments. In order to keep the coherence of this representation, the openings have to be also represented by single segments. The process of finding the openings in a floor plan and computing the endpoints of the segments that topologically represent them is described below. After that, the steps to compute the intersections amid walls will be detailed.

8.1. Openings

Openings are typically represented in a vector CAD drawing of a floor plan as instances of blocks, defined as compositions of single primitives (lines, polylines, circular arcs, etc.) and/or other blocks. Block components are given coordinates with respect to the reference point of the block, which can be freely located by the designer; therefore, each block has its own coordinate system and a name.

When a block instance is inserted in a CAD drawing, the designer defines its position, rotation and size. These transformations are applied to the block coordinate system, and then its components are translated, rotated and scaled as necessary. All the block instances share the same block name.

In order to add the information of the openings to the floor topology representation, it is necessary to define a segment for

Table 2
Comparison of the criteria to form wall-prone pairs of straight and circular arcs.

Straight segments	Circular arcs
The segments must be parallel	The segments must share the same circle center
The distance between segments must be less than a threshold ε	The difference between the circle radii must be less than a threshold ε
The projection of one segment onto the line that contains the other one overlaps with it	The angular intervals that define the segments overlap

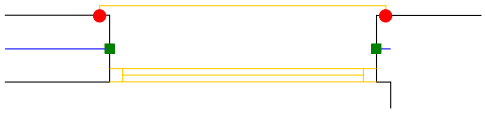


Fig. 11. Instance of a window block (in yellow). Black lines represent walls and their topological representation. The desired endpoints for the representative segment of the block are drawn in green, while red points are intersection points between the block and the walls drawing. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

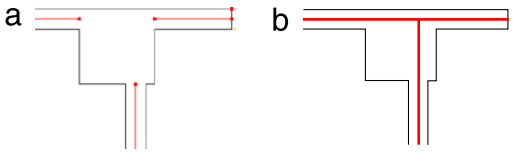


Fig. 12. (a) Result of the wall detection process. (b) Intersection point that has to be computed.

each block representing an opening in the floor plan. The user intervention may be necessary to select the name of the blocks that represent openings in the floor plan, as current CAD applications do not force the designers to use standardized names.

Once the blocks representing the openings have been selected, it is not enough to find the intersections between the block instances and the walls drawing, as this may lead to wrong points (see Fig. 11). Instead of that, the following steps are applied to compute the representative segment for each block instance:

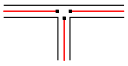
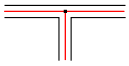
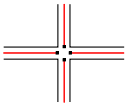
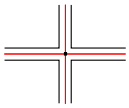
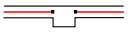
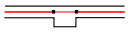
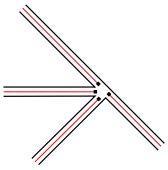
1. Compute the bounding box of the block instance. This can be done by applying the instance transformations to the bounding box of the geometry of the block definition.
2. Given the center of the bounding box, notated P, search for the nearest endpoint of a topology segment that represents a wall. This point will be notated as Q_1, and will be considered as the first endpoint for the new segment.
3. Compute the vector $\overrightarrow{v}_1 = Q_1 - P$.
4. Search the endpoints of the topology segments for the nearest point Q_2 such that given the vector $\overrightarrow{v}_2 = Q_2 - P$, the cosine of the angle between $\overrightarrow{v}_1$ and $\overrightarrow{v}_2$ is less than zero.
5. Create the segment that topologically represents the opening using Q_1 and Q_2 as endpoints.

8.2. Wall intersections

The joint result of the wall detection algorithm and the processing of the opening blocks is a set of segments, most of them are already connected into polylines representing topology. However, due to the way the wall detection algorithm works, the intersections amid walls are still not found (see Fig. 12). It is therefore necessary to process the polylines representing the topology to compute these intersection points and complete the topological representation of the floor plan.

The proposal presented here divides the process of computing the wall intersections in two steps: first of all, areas with a high concentration of polyline endpoints are sought in the topology representation; after that, a representative point is computed for each of these areas, as the topological representation of the wall intersection.

Table 3
Cases and solutions.

Case	Layout	Wall intersection point
T-crossing		
L-crossing		
X-crossing		
I-crossing		
Default (centroid)		

In order to detect high endpoint concentration areas, the user intervention can be necessary to define a search radius (although a fixed value of one meter gives good results for typical floor plans). Then, the distances between every two endpoints are computed, and those endpoints placed at a closer distance than the search radius are included into the same set. In the end, the sets with more than one endpoint represent high concentration areas. Algorithm 2 summarizes this process.

Algorithm 2 Detection of high endpoint concentration areas

```
Input: a set L = {L_1, ..., L_n} of n polylines representing
topology; a search radius ρ
Build a set for each endpoint of the polylines in L
for i=1 to n − 1 do
  {p_1, p_2} ← endpoints(L_i)
  for j=i+1 to n do
    {q_1, q_2} ← endpoints(L_j)
    for all (p, q) ∈ {p_1, p_2} × {q_1, q_2} do
      if dist(p, q) < ρ then
        merge the sets p and q belong to
      end if
    end for
  end for
end for
return the remaining sets of endpoints
```

The search radius has to be big enough to avoid problems due to the presence of columns in the floor plan, as shown in Table 3 and Fig. 12.

Once the high endpoint concentration areas have been detected, it is necessary to compute a topological intersection point for each of these. Depending on the floor plan layout, the intersection points are computed in a different way (see Table 3):

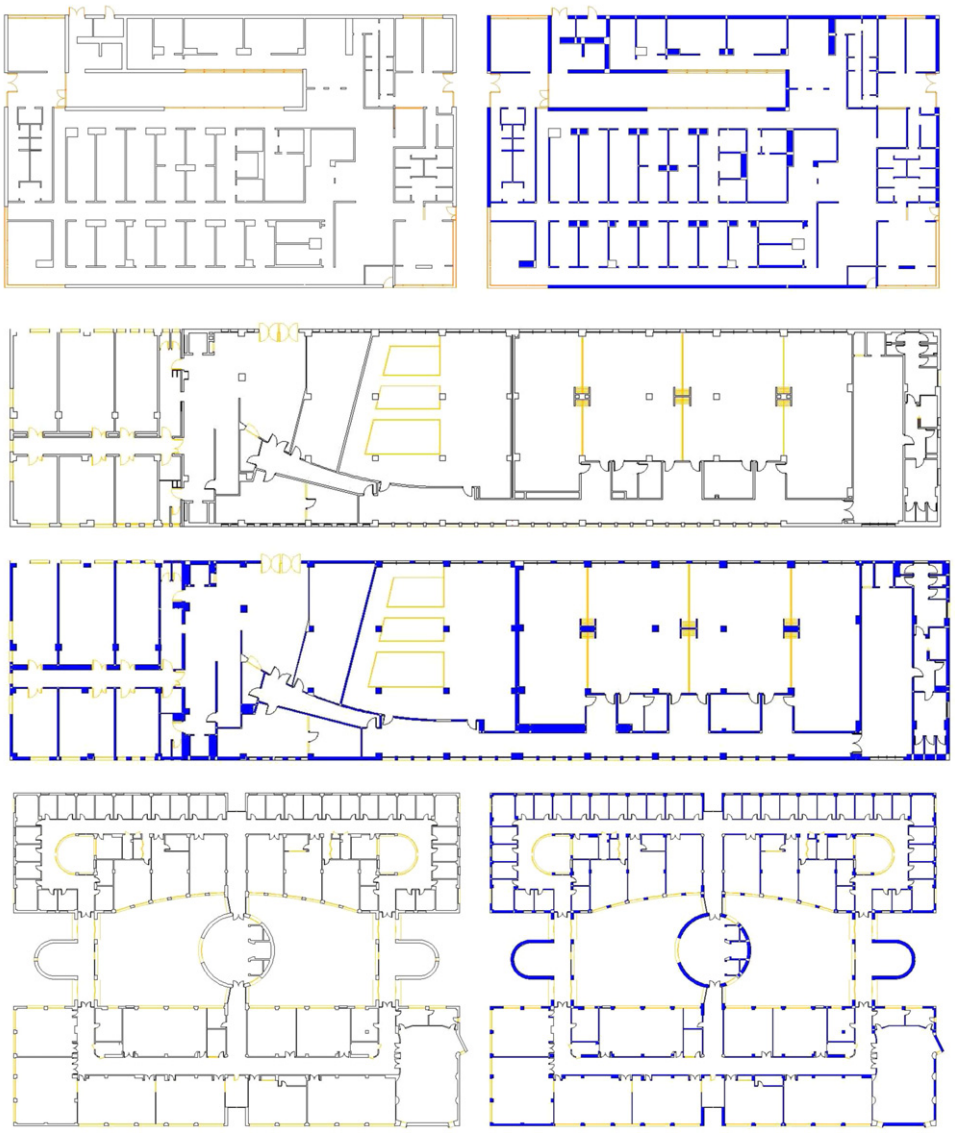


Fig. 13. Real floor plans used to test our algorithms, together with the results of applying the wall detection algorithm. Detected walls are drawn in blue. The layers containing windows and doors are shown, but have not been processed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

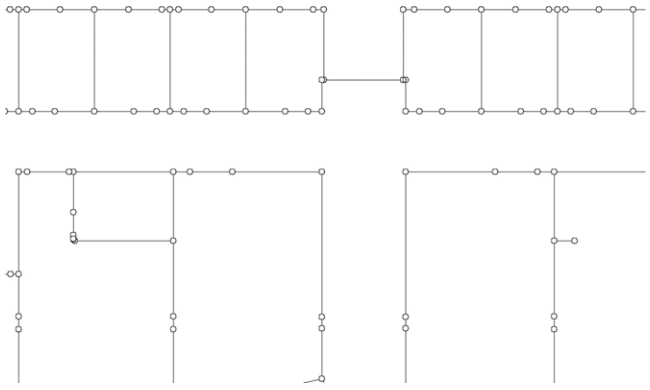


Fig. 14. Topology representation from a portion of a CAD vector floor plan.

- *T-crossing amid walls*

  Three topology segments appear in an orthogonal position, as shown in Fig. 12(a). The intersection point is computed as the intersection of the lines that contain the segments. Fig. 12(b) shows the result for this situation.
- *L-crossing between walls*

  In this layout, two topology segments are placed in an orthogonal position, representing a corner in the wall structure of the floor. The intersection point is then computed as the intersection of the lines that contain the walls.
- *X-crossing amid walls*

  In this case, four walls meet orthogonally in a point, making up a cross-like intersection. The representative wall intersection point is computed as the intersection point between the lines that contain the wall segments.
- *I-crossing between walls*

  This kind of crossing is due to the presence of a column in between two parts of the same wall. When the wall detection algorithm processes this geometry, two unconnected wall segments are detected, and it is necessary to join them to form the topology representation of the original wall. Therefore, instead of a wall intersection point, a new wall segment is created for this purpose using the endpoints of the high concentration area.

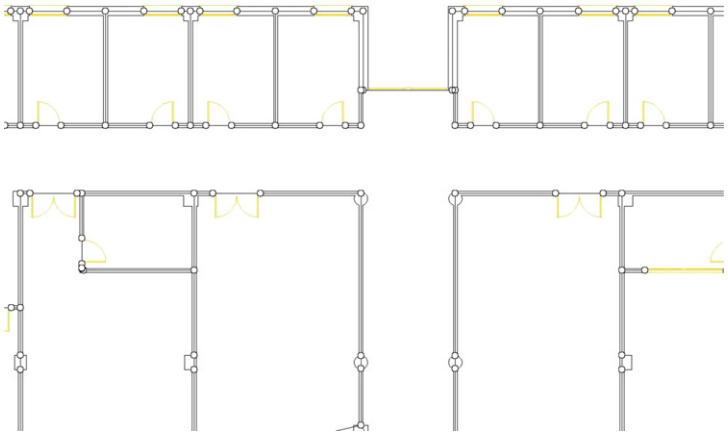


Fig. 15. Topology representation on top of the original CAD vector floor plan.

Table 4
Numerical data from the wall detection tests with real floor plans.

Plan	A1	A2	ε	B1	B2	C1	C2	D1	D2	E1	E2	F	G
1	643	0	0.4	193	0	56	0	249	0	36.10	0	118.50	1.09
			0.5	199	0	58	0	257	0	35.00	0	119.17	1.10
			0.6	203	0	60	0	263	0	33.90	0	122.03	1.16
			0.7	236	0	62	0	298	0	26.59	0	136.92	1.35
2	1148	14	0.4	482	9	159	0	641	9	32.50	11.10	612.21	3.99
			0.5	549	9	206	0	755	9	25.10	11.10	755.40	5.08
			0.6	628	9	244	0	872	9	16.00	11.10	937.65	6.38
			0.7	692	9	287	0	979	9	10.10	11.10	1161.43	8.37
3	1678	79	0.4	442	20	93	0	535	20	47.73	54.43	696.45	4.45
			0.5	455	20	104	0	559	20	45.76	54.43	785.69	4.91
			0.6	637	33	176	0	813	33	26.04	26.58	922.89	5.21
			0.7	667	33	190	0	857	33	22.76	26.58	957.27	5.56

A1–A2. Number of segments/arcs in the floor plan respectively.
ε. Threshold used to form segment pairs.
B1–B2. Number of initial wall-prone edges linking segments/arcs respectively.
C1–C2. Number of initial wall edges linking segments/arcs respectively.
D1–D2. Number of final wall edges linking segments/arcs respectively.
E1–E2. Segments/arcs that do not form walls respectively (%).
F. WAG building time (ms).
G. WAG processing time (ms).

- *Default case*
  If the wall layout does not correspond to any of the cases described above, the representative wall intersection point is computed as the centroid of the set of points included in the high concentration area.

Once the wall intersection points have been computed, the endpoints of the topology polylines are replaced by the intersection points obtained from the high concentration areas they belong to, so that a completely connected representation of the topology of the floor plan is obtained.

9. Results

A standalone Java application has been developed as a platform to test all the algorithms that have been described [20]. The application is able to load and show a DXF file containing the vector drawing of a floor plan, and provides the user with tools to fix some precision related problems as described in Section 3, select the layer (or layers) with the walls and openings information, choose the values of the parameters for the detection algorithms and show and store the resulting topology representation.

The algorithms have been tested in a computer with a 2.4 GHz Intel® Core™2 Quad processor with 4 GB of RAM using real floor plans of buildings of our university. Fig. 13 shows three of the floor plans used and the result of applying the wall detection algorithm to them; the detected walls are drawn in blue. Although the algorithm has been only applied to the representation of walls, other layers are shown with different colors to give a better understanding of the floor plans.

Table 4 shows some numerical results obtained in our tests using different threshold values (the results appear in the same order the results are shown in Fig. 13). For each test, the table shows how many straight segments and circular arcs were used as input for the algorithm, the value of the threshold ε used, the amount of wall and wall-prone edges in the initial generalized WAG that link respectively segments and arcs, the final number of wall edges at the end of the algorithm execution, the percentage of segments and arcs that do not belong to any wall in the result, the time spent to build the initial generalized WAG and the time the algorithm took to detect the walls. As can be seen, 1 ms is the lower limit for the execution of the wall detection algorithm. The final number of wall edges is equal to the sum of the initial number of wall edges and the number of wall-prone edges; this was the expected result, given the way the algorithm works.

The time spent in the construction of the initial WAG is in general less than a second for floor plans with no more than 1700 segments, although the complexity of the drawing has a significant influence on this task. Fig. 14 shows the representation of the topology of a portion of a real floor plan as obtained with the algorithms described in this paper, while Fig. 15 shows the topology representation on top of the original floor plan.

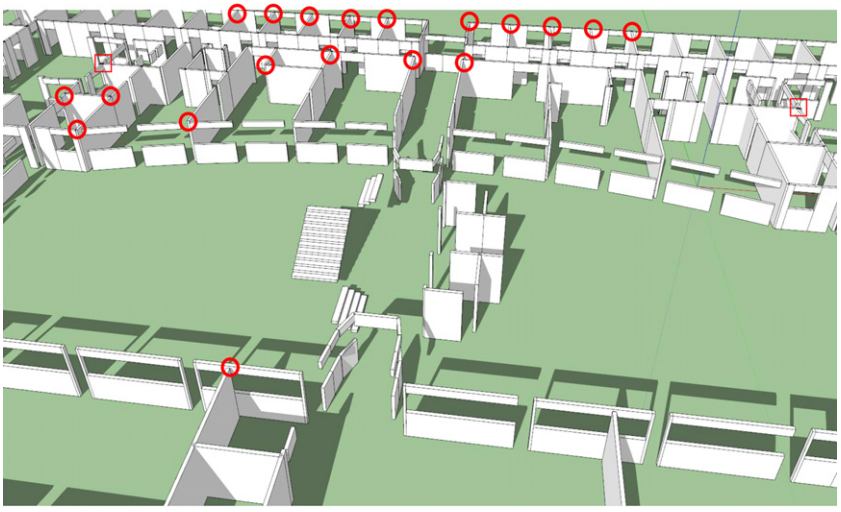


Fig. 16. 3D reconstruction of floor plans.

It is interesting to remark that using different values for the threshold ε produces significant changes in the results: the bigger the threshold, the more complex is the structure of the generalized WAG, because the number of relevant pairs of segments increases, and the time spent to build and process the graph is therefore higher. On the other hand, the number of segments and arcs that are not considered as part of a wall pair at the end of the wall detection algorithm decreases as the value of the threshold increases. It could be interesting to study more carefully this issue in order to find a way to compute dynamically the optimum value of the threshold for each floor plan.

10. Conclusions and future work

A method to extract topological information from 2D CAD vector floor plans has been presented. The problem is divided into several stages: first of all, the wall detection algorithm produces the basic wall topology; after that, the blocks representing the openings are processed in order to link their representation to the wall topology; finally, the wall intersections are computed to complete the topology representation of the floor.

The Wall Adjacency Graph (WAG) has been presented as the supporting data structure of the wall detection algorithm. It stores the topological relations amid the segments (represented as graph nodes) that compose the drawing of the wall structure of the floor plan, using three kind of edges: wall, wall-prone and hierarchical edges. An initial WAG is directly built from the floor plan; then, the wall detection algorithm processes the graph in linear time with respect to the number of wall-prone edges. The topological information about walls is obtained directly from the final WAG in an easy way. All this process typically takes less than one second using real floor plans.

In order to handle appropriately some particular cases, a generalized, less restrictive version of the WAG has been presented that allows a more flexible detection process.

Future work include the detection of additional elements like stair flights, the use of spatial indices to speed up the WAG building process, the study of a way to compute the optimum threshold value for a given floor plan, as well as the inclusion of semantic information in the same process, so that the user intervention can be minimized, and applying graph theory to the study of wall adjacency graphs, in order to analyze the meaning of cycles, shortest paths and graph isomorphism. The long term goal is to integrate the topology detection process into a complete system for 3D building reconstruction (see Fig. 16).

Acknowledgments

This work has been partially supported by the Andalusian Government, the Spanish Ministry of Science and Innovation and the European Union (via ERDF funds) through the research projects P07-TIC-02773, TIN2007-67474-C03 and TIN2011-25259.

References

[1] Brutzman D, Daly L. X3D. Extensible 3D graphics for web authors. Morgan Kaufmann; 2007.
[2] Arnaud R, Barnes M. COLLADA. Sailing the gulf of 3D digital content creation. A.K. Peters; 2006.
[3] Walsh AE, Bourges-Sévenier M. Core web3D. Upper Saddle River (NJ, USA): Prentice Hall PTR; 2001.
[4] Li Y, He Z. 3D indoor navigation: a framework of combining BIM with 3D GIS. In: 44th ISOCARP congress. 2008.
[5] Benner J, Geiger A, Leinemann K. Flexible generation of semantic 3D building models. In: Proceedings of the 1st international workshop on next generation 3D city models. 2005.
[6] Müller P, Wonka P, Haegler S, Ulmer A, Van Gool L. Procedural modeling of buildings. ACM Transactions on Graphics 2006;25(3):614–23.
[7] Choi JW, Kwon DY, Hwang JE, Lertlakkhanakul J. Real-time management of spatial information of design: a space-based floor plan representation of buildings. Automation in Construction 2007;16(4):449–59.
[8] Ah-Soon C, Tombre K. Architectural symbol recognition using a network of constraints. Pattern Recognition Letters 2001;22(2):231–48.
[9] Dosch P, Tombre K, Ah-Soon C, Masini Ga. A complete system for the analysis of architectural drawings. International Journal on Document Analysis and Recognition 2000;3(2):102–16.
[10] Lladós J, López-Krahe J, Martí E. A system to understand hand-drawn floor plans using subgraph isomorphism and Hough transform. Machine Vision and Applications 1997;10(3):150–8.
[11] Lu T, Yang H, Yang R, Cai S. Automatic analysis and integration of architectural drawings. International Journal on Document Analysis and Recognition 2007; 9(1):31–47.
[12] Horna S, Meneveaux D, Damiand G, Bertrand Y. Consistency constraints and 3D building reconstruction. Computer-Aided Design 2009;41(1):13–27.
[13] Mas A, Besuievsky G. Automatic architectural 3D model generation with sunlight simulation. In: Ibero-American symposium on computer graphics. 2006. p. 37–44.
[14] Paoluzzi A, Milicchio F, Scorzelli G, Vicentino M. From 2D plans to 3D building models for security modeling of critical infrastructures. International Journal of Shape Modeling 2008;14(1):61–78.
[15] Medjdoub B, Yannou B. Separating topology and geometry in space planning. Computer-Aided Design 2000;32(1):39–61.
[16] Zhi G, Lo S, Fang Z. A graph-based algorithm for extracting units and loops from architectural floor plans for a building evacuation model. Computer-Aided Design 2003;35(1):1–14.
[17] National CAD standard, v.4.0. 2008. http://www.nationalcadstandard.org.
[18] ISO 13567: organization and naming of layers for CAD. ISO. International Organization for Standardization. 1998. http://www.iso.org.

[19] Davies N. et al. AEC (UK) CAD standard for basic layer naming, v.2.4. 2005. http://www.aec-uk.org.
[20] Domínguez B, Conde F, García ÁL, Feito FR. On the specification and interface design of a semiautomatic tool for the detection of semantic elements in a floor plan. Tech. rep. Departamento de Informática. Universidad de Jaén. 2011.
[21] Domínguez B, García AL, Feito FR. An open source approach to semiautomatic 3D scene generation for interactive indoor navigation environments. In: Proceedings of IV ibero-American symposium on computer graphics. 2009. p. 131–8.
[22] Salomon KB. An efficient point-in-polygon algorithm. Computers & Geosciences 1978;4(2):173–8.