Automation in Construction 94 (2018) 405–416

Contents lists available at ScienceDirect

Automation in Construction

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

Customization and generation of floor plans based on graph transformations

Xiao-Yu Wang[a,*], Yin Yang[b], Kang Zhang[a]

[a] The University of Texas at Dallas, 800 W Campbell Rd., Richardson, TX, USA
[b] University of New Mexico, Albuquerque, NM, USA

ARTICLE INFO

Keywords:
Floor planning
Modeling
Rectangular dual
CAD

ABSTRACT

This paper introduces an approach for automatic generation of rectangular floor plans based on existing legacy floor plans with the capability of further improvement and customization. Our approach first derives a dual graph from the given input file specifying a floor plan. It then either automatically reproduces varied floor plans retaining the connectivity of the original plan or performs transformation rules to manipulate spatial relations among rooms, and to generate modified floor plans corresponding to specific requirements. Our approach introduces constraints, such as the maximum width-height ratio, to support the flexibility for various design requirements. A graphical user interface is provided for users to perform the automatic generation process. An experiment has been conducted to validate the feasibility of our approach and time taken in generating floor plans. It shows that our method is able to generate highly-customized floor plans in reasonable time.

1. Introduction

In the last few decades, researchers have developed various approaches to generate interior building layouts in styles similar to existing famous or historic design paradigms. Stiny and Mitchell [1] proposed a shape grammar method for the room arrangement of the Palladian style. They later proposed another shape grammar to generate plans following the style of Mughul gardens [2]. Koning and Eizenberge [3] developed a set of rules to reproduce the Frank Lloyd Wright's prairie style houses. Recently, we have witnessed the era of information explosion. Numerous floor plan resources are now digitized and achieved on the computers, which allows graphics and vision techniques to be applied to extract the spatial relation of rooms in the floor plan. For example, Fan et al. [4] presented a structure completion method from facade layout images, Liu and Zlatanova [5] extracted adjacency relations from CityGML, and Lin et al. [6] obtained obstacle data from the Industry Foundation Classes (IFC) data for pathfinding. Moreover, there are also works that aim to generate floor plans automatically or semi-automatically. Bhasker and Sahni [7,8] used a linear algorithm to check if there are rectangular duals and, if so, generates rectangular duals for any n-vertex planar triangulated graphs. Other researchers focus on the placement of components, e.g. Kahng [9] provided a way that places components into a space. However, his approach generates new spaces, which could change the connectivity of the original graph.

Clearly, it is highly desired to be able to reproduce well-known legacy floor plans, adjust or modify them to suit modern lifestyles while retaining the room adjacency, i.e., one of the most representative features for a given building style. To this end, we propose a Graph Approach to Design Generation or GADG, which generates floor plans based on the room adjacency from an input (as shown in Fig. 1). An adjacency between two rooms means they are connected by sharing a door or a wall. Input files could be floor plans or semantic rich languages specifying necessary requirements. With the extracted information, users are able to modify the corresponding dual graphs by applying transformation rules. To enhance usability and operability, we introduce two parametric transformation rules, the addition rule and subtraction rule, for modifying rooms. The rules are applied directly to the graph to add/remove vertices instead of rooms. Then, GADG performs an algorithm to remove isolated vertices from the graph. Since our approach only generates rectangular rooms from properly triangulated planar graphs, a validation algorithm is performed to check the existence of rectangular floor plans for the graph. Finally, with user-specified constraints, such as setting the maximum aspect ratio for each generated room and moving internal rooms to the boundary, GADG reproduces a set of new plans automatically by applying a rectangular dual finding algorithm so that the original connectivity information is preserved. With these generated plans, designers can come up with new design ideas or directly take the generated floor plans as prototypes.

GADG is featured by providing a fully automated process to generate controllable and tractable floor plans with identical connectivity. It is able to incorporate various constraints during the generation

* Corresponding author.
E-mail addresses: xwang@utdallas.edu (X.-Y. Wang), yangy@unm.edu (Y. Yang), kzhang@utdallas.edu (K. Zhang).

https://doi.org/10.1016/j.autcon.2018.07.017
Received 14 December 2017; Received in revised form 14 July 2018; Accepted 18 July 2018
0926-5805/ © 2018 Elsevier B.V. All rights reserved.

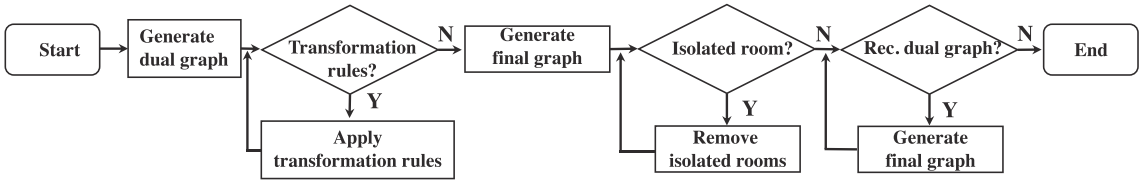


Fig. 1. The overall flow chart of the proposed floor plan generation pipeline.

Table 1
Features of different floor plan generation approaches.

Features	[3]	[10]	[11]	[12]	[13]	[7]	[14]	[15]	[16]	[17]	[4]	Our approach
Giving a floor plan to guide the generation	–	–	–	–	–	–	–	–	–	X	–	X
Generating floor plans conforming to a style	X	(–)	(–)	(–)	(–)	(–)	X	(–)	(–)	X	(–)	X
Using constraints to control generated plans	–	–	–	–	–	–	X	–	–	X	X	X
Supporting a fully automatic generation process	–	(X)	–	–	X	X	X	X	X	–	–	X
Using graphs to consider room adjacencies	–	X	X	X	X	X	–	X	X	(–)	–	X
2D/3D	3D	2D	2D	2D	2D	2D	3D	3D	2D	2D/3D	3D	2D

X: considered. (X): implicitly considered. –: not considered. (–): not mentioned.

process to generate floor plans under a wide range of design requirements. Table 1 shows a comparison of features between existing floor plan generation approaches and our approach.

2. Related work

Many researchers have proposed approaches for automatic generation of floor plans. Placing rooms into given planes is known as the space allocation problem [17]. Some researchers attempted to solve this problem by exhaustively generating all possible results, e.g., Galle et al. [10] proposed an algorithm that generates every possible building plan for any given number of rooms. The complexity of this algorithm increases exponentially with the increasing number of rooms. Others focus on generating floor plans with heuristic placements. Shekhawat [18] developed an algorithm that utilizes the given spaces inside a rectangle to find satisfactory results.

Some approaches generate floor plans based on a given room adjacency graph. Roth et al. [11] proposed an approach to turn a graph to a rectangular plan with the adjacency preserved. Their approach, however, is unable to automatically generate designs. Kozminski [13] developed an $O(n^2)$ algorithm to generate rectangular duals for given triangulated graphs. Bhasker and Sahni [7,8], then, improved this approach by providing a linear time algorithm for verifying and generating rectangular floor plans for triangulated graphs. The reproduced floor plans retain the same adjacency relations as the input graphs. This approach generates floor plans according to n-vertex planar triangulated graphs, which means that the results generated by this algorithm retain the same connectivity. Since input graphs are user-specified, without considering existing floor plans and different requirements, it is difficult to apply this approach to real-world designs.

An algorithm of Martin [15] generates floor plans by making and manipulating graphs in three main steps. The first step creates a graph to represent the floor plan. Each vertex in the graph represents a room in the floor plan. It then distributes rooms over the footprint according to the generated graph. Finally, the approach expands rooms using the Monte Carlo method to allow the rooms to grow or shrink.

Marson and Musse [16] proposed an algorithm for automatic generation of floor plans based on tree-maps provided to the program. In their approach, the initial shapes are subdivided into floor plans according to a given tree-map. Every child room in the tree-map is generated from the space of its parent. Therefore, the results maintain the hierarchical relations, while the adjacency relations between rooms are not guaranteed.

The shape grammar [19] has been widely used to generate and analyze building layouts. It is able to model spatial relations of geometries in paintings and sculptures. Shapes expand automatically with given rules in a shape grammar, providing the possibility for programs to generate different layouts by applying rules. Harada et al. [20] developed an interactive system with the shape grammar for generating building layouts.

Wonka et al. [14] proposed the split grammar, a parametric set grammar based on the concept of shapes, for automatic generation of building designs. To increase the efficiency and automation of rule derivations, a shape grammar called CGA shape [21] is proposed, which extends the split grammar by introducing the component split and the mass modeling functionality. The same authors also proposed a set of algorithms that derive high-quality 3D models from single facade images with arbitrary resolutions [22]. Wu et al. [23] later proposed an algorithm to automatically derive split grammars from facade layouts.

Instead of using a shape grammar, Merrel et al. [17] generated three-dimensional buildings with internal structures based on a given set of high-level requirements. Lin et al. [24] proposed a hierarchical approach for generating various 3D models that resemble an input piece with the same style. These approaches set a solid foundation for automatic building generation, but are unable to control the positions and adjacency of internal rooms.

In summary, none of the aforementioned approaches has been proposed to perform the complete design process for designers to generate a set of floor plans according to the required adjacency relations from the input specification. Some approaches require redundant control vertices added to the input graph or undesirable rooms would be created. Others may generate floor plans according to shape grammars, but cannot guarantee the resulting style and shapes of rooms.

3. Preliminaries

Considering a floor plan as a planar graph $\mathscr{G}(\mathscr{V}, \mathscr{E})$, where each edge represents a wall, and each vertex represents an intersection of

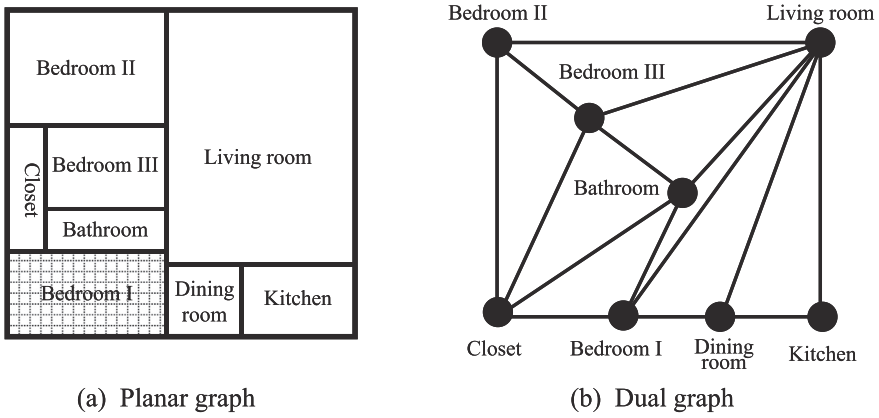


Fig. 2. A simple floor plan and its corresponding dual graph.

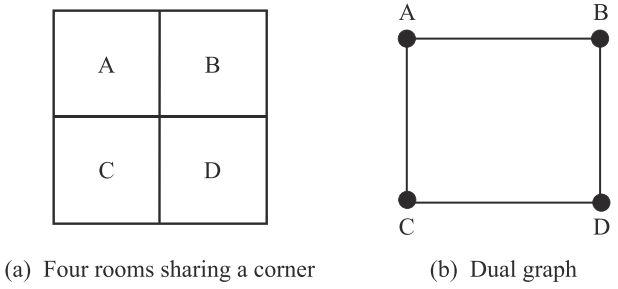


Fig. 3. A special case.

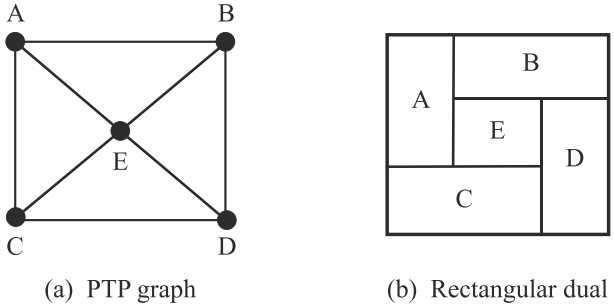


Fig. 4. A PTP graph and one of its RDGs.

walls, we can use the dual graph $\mathcal{G}_d(\mathcal{V}_d, \mathcal{E}_d)$ of $\mathcal{G}$ to clearly present the adjacency of the floor plan, as shown in Fig. 2. Each vertex in $\mathcal{V}_d$ represents a room, which is also referred to as a *shape*, in $\mathcal{G}$. An edge connecting two vertices in $\mathcal{G}_d$ implies the adjacency between the corresponding rooms.

Definition 1 (*Face [25]*). A *face* of the planar graph $\mathcal{G}$ is a maximal section of the plane in which any two points can be joined by a curve that does not intersect any part of $\mathcal{G}$.

In other words, a face is a sub-graph which contains no other vertex or edge, and the unbounded face outside the boundary of $\mathcal{G}$ is not considered. Each face in $\mathcal{G}$ represents a room in the floor plan.

Definition 2 (*Dual graph [25]*). The *dual graph* $\mathcal{G}_d$ of $\mathcal{G}$ is a graph that has a vertex corresponding to each face (room) of $\mathcal{G}$, and an edge joining two neighboring faces for each edge in $\mathcal{G}$.

For example, the shadowed Bedroom I of the floor plan in Fig. 2 (a) is represented as a vertex of the dual graph in Fig. 2 (b). Another special example is illustrated in Fig. 3 (a), where four rooms (A, B, C, D) share a corner, but A and D are not adjacent, nor B and C, as defined by the dual graph in Fig. 3 (b). The dual graph of a planar graph is also a planar graph, proved by Nishizeki [26]. Next, we generalize the definition of Properly Triangulated Planar graph (PTP) originally proposed by Bhasker and Sahni [7]. A PTP graph is a dual graph with additional constraints as defined below.

Definition 3 (*Properly triangulated planar graph*). A graph $\mathcal{G}_p(\mathcal{V}_p, \mathcal{E}_p)$, associated with a floor plan, is a PTP graph if and only if it is a connected graph with a vertex set $\mathcal{V}_p$ and an edge set $\mathcal{E}_p$ such that:

- every face in $\mathcal{G}_p$ is a triangle;
- every vertex has degree ≥ 4;
- every non-face cycle has more than 3 edges.

Definition 4 (*Rectangular dual graph [7]*). A RDG $\mathcal{G}_r$ of an n-vertex graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, is comprised of n non-overlapping rectangles with the following properties:

- Each vertex $v_i \in \mathcal{V}$, corresponds to a distinct rectangle i in the rectangular dual.
- If (i, j) is an edge in $\mathcal{E}$, then rectangles i and j are adjacent in the rectangular dual.

The adjacency here implies that two rectangles i and j share an edge. A RDG, representing a rectangular floor plan, is a generated planar graph based on the adjacency of an input dual graph.

Kozminski and Kinnen [13] prove that a graph with all faces triangular has rectangular duals if and only if it is a PTP graph. Bhasker and Sahni [8] then narrow the restriction down to only meet the first and third constraints of being a PTP graph. A PTP can be converted to many possible RDGs, and the RDG essentially maps a PTP back to a rectangular floor layout. Therefore, every face in a RDG potentially represents a room. Fig. 4 shows a PTP graph, $\mathcal{G}_p$, and a rectangular dual of the graph, $\mathcal{G}_r$. Five vertices in $\mathcal{G}_p$ are mapped to five faces in $\mathcal{G}_r$ with adjacencies preserved.

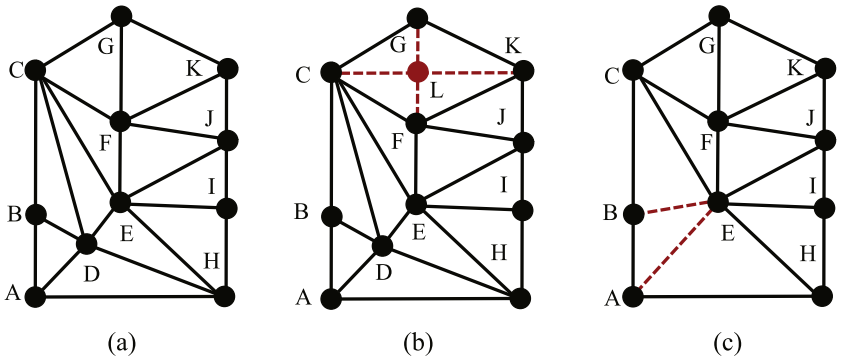


Fig. 5. An example of applying transformation rules to an input PTP.

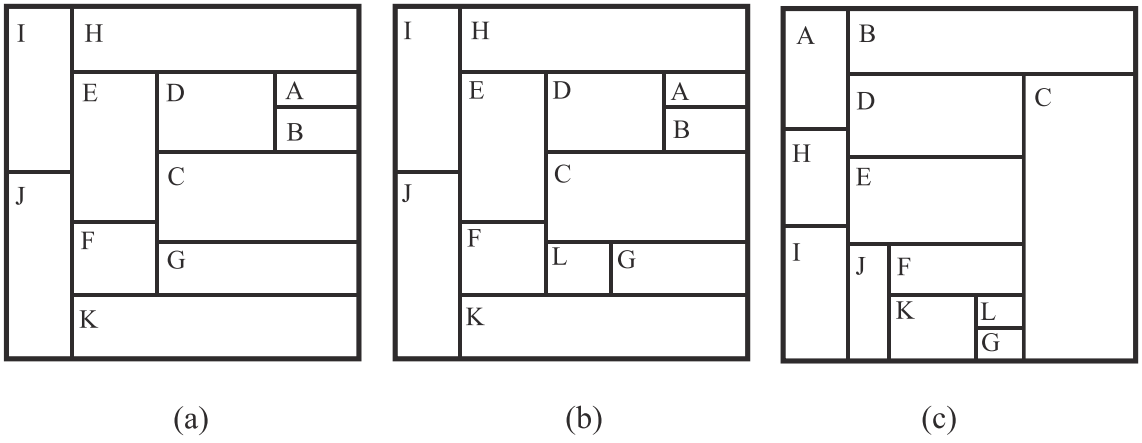


Fig. 6. Layouts corresponding to the dual graph in Fig. 3. (a) is the original layout of the input file. (b) and (c) are two possible layouts generated by inserting the room L into the layout by the addition rule.

4. Transformation rules

In this paper, we propose two transformation rules, the addition rule and subtraction rule, to GADG for automatic generation modified floor plans. The addition rule allows users to add rooms into a design with additional constraints. The subtraction rule provides a way for users to remove rooms. The transformation rules are applied to the graph structure directly, assuming the input graph to be a PTP. Then, a rectangular dual finding algorithm is introduced to map vertices in a dual graph to rectangles according to the connectivity among vertices. Each PTP possesses an associated base-layout defined as:

Definition 5 (*Base-layout*). A base-layout $\mathscr{L}$ is specified by a three-tuple $\langle b, \mathscr{R}, \mathscr{A}\rangle$, where b is the boundary of the design, $\mathscr{R}$ is a set of rooms inside b, and $\mathscr{A}$ is the set of attributes including spatial and semantic information associated with rooms in $\mathscr{R}$.

For example, the base-layout of the PTP graph $\mathscr{L}\langle b, \mathscr{R}, \mathscr{A}\rangle$ presented in Fig. 4 (a) is specified as: b is the boundary of the rectangular dual in Fig. 4 (b), $\mathscr{R}$ is a collection of rooms, i.e. r_A, r_B et al., and $\mathscr{A}$ is a set of attributes associated with rooms, for example a_A implies that r_A is on the boundary adjacent with three rooms r_B, r_C and r_E.

A base-layout of a PTP graph $\mathscr{L}\langle b, \mathscr{R}, \mathscr{A}\rangle$ will have the updated base-layout of $\langle b, \mathscr{R} + \{r_i\}, \mathscr{A} + \{a_i\}\rangle$ after being applied by the addition rule to add a room r_i with its associated attribute a_i. Similarly, the base-layout will be updated to $\langle b, \mathscr{R} - \{r_i\}, \mathscr{A} - \{a_i\}\rangle$ by applying the subtraction rule to remove a room r_i with its associated attribute a_i.

The addition rule allows a maximum of 4 adjacent vertices for a newly inserted vertex. Fig. 5 gives an example of applying the addition and subtraction rules to a PTP. Two possible layouts generated by GADG after applying the addition rule are presented in Fig. 6. Algorithms 1 and 2 report the implementation details for addition and subtraction rules in order to make them operational in a shape grammar interpreter. The interpreter is capable of automatically generating geometric products according to a given set of shapes and rules applied to the shapes [27]. We have implemented our floor plan generation algorithm and all the presented functionality into this interpreter.

The input to the addition rule is a room r_i, which corresponds to a vertex v_i to be inserted into the PTP, its attribute a_i, and its adjacent vertices $\mathscr{N}_i$. If a_i says that v_i is on the boundary, the cardinality of $\mathscr{N}_i$ must be less than 3 ($|\mathscr{N}_i| \leq 3$) and the algorithm seeks for a triangle face f_i in $\mathscr{G}_p$ such that f_i contains all vertices in $\mathscr{N}_i$ and has a boundary edge. The addition rule fails if f_i does not exist. Otherwise, we remove the boundary edge and connect v_i with all vertices of f_i. If v_i is not a boundary vertex, we must have $|\mathscr{N}_i| \leq 4$ and the algorithm attempts to find a quadrilateral cycle in $\mathscr{G}_p$ which contains all the vertices in $\mathscr{N}_i$. According to the properties of PTP, this quadrilateral cycle must consist of two triangular faces sharing an edge. The algorithm then removes this shared edge and connect v_i to all the vertices on the cycle. The layouts generated by the addition rule preserves the topology of the input PTP, and it can be proven that the resulting graph remains triangulated.

Algorithm 1. The addition rule.

Input: $\mathcal{G}_p(\mathcal{V}_p, \mathcal{E}_p)$, v_i, a_i, $\mathcal{N}_i$

1 **if** v_i is on the boundary **then**
2 // $1 \le |\mathcal{N}_i| \le 3$
3 find a face $f_i = \{v_1, v_2, v_3\}$ in $\mathcal{G}_p$ such that $f_i \supseteq \mathcal{N}_i$ and f_i has at least one boundary edge;
4 **if** f_i does not exist **then**
5 **return** *FALSE*;
6 **else**
7 $\mathcal{E}_p \leftarrow \mathcal{E}_p - \{(v_1, v_2)\}$;
8 $\mathcal{V}_p \leftarrow \mathcal{V}_p + \{v_i\}$;
9 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v1)\}$;
10 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v2)\}$;
11 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v3)\}$;
12 **end**
13 **else**
14 // $1 \le |\mathcal{N}_i| \le 4$
15 find a quadrilateral cycle $Q = \{v_1, v_2, v_3, v_4\}$ such that $Q \supseteq \mathcal{N}_i$ and $(v_2, v_4) \in \mathcal{E}_p$;
16 **if** Q does not exist **then**
17 **return** *FALSE*;
18 **else**
19 $\mathcal{E}_p \leftarrow \mathcal{E}_p - \{(v_2, v_4)\}$;
20 $\mathcal{V}_p \leftarrow \mathcal{V}_p + \{v_i\}$;
21 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v1)\}$;
22 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v2)\}$;
23 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v3)\}$;
24 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(vi, v4)\}$;
25 **end**
26 **end**
27 **return** *TRUE*;

Theorem 1. *A PTP remains triangulated after being inserted with a new vertex by* Algorithm 1.

Proof. Since the original PTP $\mathcal{G}_p(\mathcal{V}_i, \mathcal{E}_i)$ is triangulated, if a new vertex v_i is added as a boundary vertex, Algorithm 1 removes the boundary edge of a f_i (line 7 in Algorithm 1), which only destroys f_i and yields two new triangle faces. All other faces in $\mathcal{G}_p$ are unchanged. Therefore, $\mathcal{G}_p$ remains triangulated. On the other hand, if v_i is not on the boundary, the removal of (v_2,v_4) (i.e. line 19) would only destroy triangles in Q and all the other faces in $\mathcal{G}_p$ are unchanged. Therefore, lines 21–24 yield four new triangle faces into $\mathcal{G}_p$ and $\mathcal{G}_p$ remains triangulated.

Conversely, the subtraction rule removes a vertex v_i from a PTP as outlined in Algorithm 2. Let $\mathcal{N}_i$ be the set of one-ring neighbors of v_i in $\mathcal{G}_p$. Removing v_i would leave an empty polygonal face specified by $\mathcal{N}_i$. Therefore, the subtraction rule needs to perform the triangulation to guarantee the resulting graph to be a PTP. If v_i is on the boundary, we also need to find its preceding and succeeding vertices v_j, v_k on the boundary and connect v_j and v_k after the removal of v_i to prevent any internal vertex from being a boundary one.

Theorem 2. *A PTP after removing a vertex by* Algorithm 2 *remains triangulated.*

Proof. Removing v_i in a PTP $\mathcal{G}_p$ would destroy triangles incident to v_i and leave an empty polygonal face on $\mathcal{G}_p$ if v_i is internal. Otherwise, adding edge (v_j,v_k) (line 4 in Algorithm 2) closes the gap at the boundary. The final triangulation (line 11) restores all the faces to be triangles.

Algorithm 2. The subtraction rule.

Input: $\mathcal{G}_p(\mathcal{V}_p, \mathcal{E}_p)$, v_i

1 find v_i's one-ring neighbors $\mathcal{N}_i$;
2 **if** v_i is on the boundary **then**
3 find $v_j, v_k \in \mathcal{N}_i$ such that v_j, v_k are boundary vertices;
4 $\mathcal{E}_p \leftarrow \mathcal{E}_p + \{(v_j, v_k)\}$;
5 **end**
6 **for** each each $v_l \in \mathcal{N}_i$ **do**
7 $\mathcal{E}_p \leftarrow \mathcal{E}_p - \{(v_i, v_l)\}$;
8 **end**
9 $\mathcal{V}_p \leftarrow \mathcal{V}_p - \{v_i\}$;
10 **if** $|\mathcal{N}_i| > 3$ **then**
11 triangulate face $\mathcal{N}_i$;
12 **end**

Our transformation rules apply to a PTP, which is the dual graph of a floor design. In other words, they alter the topology of a floor plan. Designers have full control of results generated by GADG.

5. Method

GADG first derives a dual graph from an input floor plan (in the `IFC` format). It then applies transformation rules to the graph, validates and converts the modified graph back to a RDG. Converting from a graph to a RDG is not unique, and GADG delivers a distinct RDG each time for further improvements/modifications. GADG is versatile, allowing various parameter specifications. With different parameter settings, various customized results may be produced.

5.1. Dual graph generation from an `IFC` input

The `IFC` format is one of the most widely used templates to describe buildings and architecture designs. It is an open standard for sharing the Building Information Modeling (BIM) data among different applications. A room in an `IFC` is specified by an `ifcSpace class` and the adjacency between rooms is indicated from `ifcRelSpaceBoundary class`.

5.2. Removing isolated rooms

Kozminski and Kinnen [13] establish that there exists a rectangular dual for any graph iff the graph is a PTP graph. To check the existence of a rectangular dual graph for some special cases, the algorithm checks all vertices in the dual graph to find ones with only one neighbor, most likely to be closets in floor plans. As Fig. 7 shows that the Closet only connects to a Bedroom, implying that the Closet is contained in the Bedroom. In this case, the dual graph generated from this floor plan is a non-triangulated graph. Since there is a vertex with only one neighbor, the graph is not a PTP graph. To extend the usability and adaptability of our algorithm, we treat the vertex as an attribute of its neighbor and remove it from the graph. We then guarantee that the dual graph is triangulated even though one or more closets exist. It ensures graphs to pass the connectivity check for the aforementioned special case. During the generation process, we generate rooms for these vertices separately.

5.3. Floor plan generation

To generate floor plans from dual graphs, we extend the rectangular dual finding algorithm originally proposed by Bhasker and Sahni [7] by

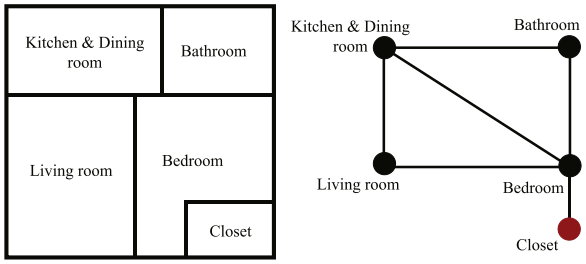
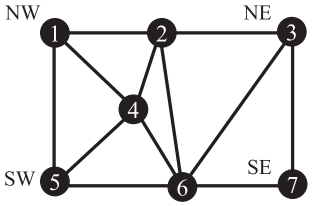


Fig. 7. The closet is adjacent only to the bedroom, which is considered an isolated room. After converting to the dual graph, the closet corresponds to a dangling vertex (in red), which will be removed. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

providing extra flexibility in choosing corner vertices. Bhasker and Sahni [7] do not consider the special case with cycles on the left boundary. Our algorithm consists of two steps, namely PTP verification and RDG generation.

PTP verification Before generating RDGs or floor plans, GADG performs a linear algorithm to check the existence of rectangular duals for the given graph. Because A graph has rectangular duals if and only if it is a PTP graph [8], our algorithm thus checks whether the derived graph possesses all of the PTP properties as defined in Definition 3. More specifically, for the dual graph $\mathcal{G}_d(\mathcal{V}_d, \mathcal{E}_d)$ of an input floor plan the algorithm first confirms that every edge belongs to at least one triangular face. We then count the total number of interior faces, which should equal to $|\mathcal{E}_d| - |\mathcal{V}_d| + 1$ if $\mathcal{G}_d$ is a PTP.

RDG generation Many algorithms have been proposed for generating RDG from graphs. Maling et al. [28] proposed a method to find optimal rectangular plans. Kozminski and Kinnen [13] presented an $O(N^2)$ algorithm for finding RDG from triangulated graphs. These methods either have high complexities or induce redundant vertices. Alternatively, we generalize Bhasker and Sahni's algorithm to produce RDGs [7,8].



Assuming the input PTP $\mathcal{G}_p$ is shown on the right, we pick four distinct corner vertices, namely NW, NE, SE and SW, out of the boundary vertices in $\mathcal{G}_p$ (vertices 1, 3, 7, and 5 in the figure). The original boundary of $\mathcal{G}_p$ can now be decomposed into four segments: top, right, bottom and left, based on which GADG generates a directed path diagram graph (PDG) or $\mathcal{G}_d$. We start with vertex NW and a vertex which is on top of all the rectangle, called HeadNode. By traversing along the left boundary, vertices 1 and 5, as well as all the edges along the path from NW to SW are added to $\mathcal{G}_d$ (and also removed from $\mathcal{G}_p$). Vertices 2 and 6 become new NW and SW corners in the updated $\mathcal{G}_p$. However, vertices 2, 4 and 6 form a cycle in $\mathcal{G}_p$, and the left edge cannot be directly incorporated into $\mathcal{G}_d$. To break the cycle, we connect vertex 4 to the last vertex in $\mathcal{G}_d$ (i.e. vertex 5), add (4,5) to $\mathcal{G}_d$, and process the remaining left boundary as before. The algorithm stops when all the vertices in $\mathcal{G}_p$ are included in $\mathcal{G}_d$. Fig. 8 reports the step-by-step result of generating $\mathcal{G}_d$ from $\mathcal{G}_p$. Clearly, $\mathcal{G}_d$ is cycle-free and builds a hierarchy of all the vertices.

To generate the RDG, the algorithm iterates all paths in $\mathcal{G}_d$ from left to right starting from the HeadNode. As shown in Fig. 9, the algorithm traverses the leftmost path 1 → 5, and converts each vertex along the path into a rectangle forming a column in the RDG. The next path is 2 → 4 → 5. According to the original PTP $\mathcal{G}_p$, vertices 2 and 4 are connected to the existing vertex 1. The algorithm creates rectangles 2 and 4, and expands rectangle 5 to provide the base for both rectangles 1 and 4. The next path is 2 → 6. Rectangle 2 already exists in the RDG, and is expanded to the next column. Since rectangle 6 needs to be adjacent to rectangles 2, 4 and 5, it is placed under 2 and closes 4 and 5. The final path 3 → 7 creates the last two rectangles and closes off all generated rectangles. The actual width and height of each rectangle in the RDG can be controlled by the user while maintaining the room connectivity.

While the user decides how to pick four corner vertices, there is an additional constraint that we need to check before generating RDG. As shown in Fig. 10, if NW, NE, SE and SW are set as v_G, v_F, v_D, v_B respectively, the left boundary becomes (v_G,v_H,v_B), forming a cycle. Since the PDG is empty at this stage, we cannot break the cycle as in Fig. 8, and the algorithm thus fails. To avoid this failure, we require that the first left boundary must be cycle-free. Otherwise, GADG asks the user to re-select four corners.

5.4. User-specified constraints

GADG is extensible and can incorporate various additional constraints to control the generated floor plans. The current implementation allows two constraints. First, GADG allows designers to set the maximum width-height ratio of every room generated so that extremely narrow rooms can be avoided. In real-world applications, rooms with the aspect ratio greater than 3 are most likely undesired. Fig. 11 reports the resulting plans with and without applying this constraint. We also allow the user to move an internal room to the boundary so that, for instance, it could have a window. Such an operation is only available for an internal vertex in the PTP that is adjacent to at least one boundary vertex[1]. Because each face on a PTP must be a triangle, this internal vertex v_I is either adjacent to two boundary vertices v_A and v_B (Example 1 in Fig. 12) or to a single boundary vertex v_A (Example 2 in Fig. 12). Since a PTP graph is guaranteed to have at least a rectangular dual, our approach is always feasible to move internal rooms to the boundary.

For Example 1, we simply remove (v_A,v_B) from the PTP and add two boundary edges (v_A,v_I) and (v_I,v_B). For Example 2, assume that v_C is the boundary vertex immediately after v_A clock-wise on the boundary. The algorithm removes (v_A,v_C) and inserts two new boundary edges (v_A,v_I) and (v_I,v_C). In addition, an internal edge (v_I,v_D) is also added for any non-boundary vertex v_D that is adjacent to v_A, and right to the edge (v_A,v_I). Doing so adds a new triangle face to the PTP for a v_D and the degree of v_D is increased by 1. But the resulting graph remains a PTP according to Definition 3. Such an operation is symmetric if one chooses to move v_I between v_B and v_A, where v_B is the boundary vertex immediately before v_A clock-wise.

The execution time of GADG for generating floor plans of increasing number of rooms is presented in Fig. 13, with and without the aspect ratio constraint. The constraint of the maximum aspect ratio of length over width of each room is set at 3. For each room number setting, we generate 20 different floor plans and calculate the average execution time. The experiment shows that GADG is scalable, since with the increasing number of rooms, the generation time increases linearly. For a floor plan with less than 10 rooms, our approach can generate a potential design in less than 5 ms. Fig. 13 also shows that the increase in the execution time with an aspect ratio constraint is almost negligible. This also implies that adding other constraints, e.g. dimensional and directional constraints, may not significantly impact the generation speed.

[1] If this condition does not hold, GADG can still swap this room to the boundary. However, doing so could significantly alter the adjacency relation in the PTP.

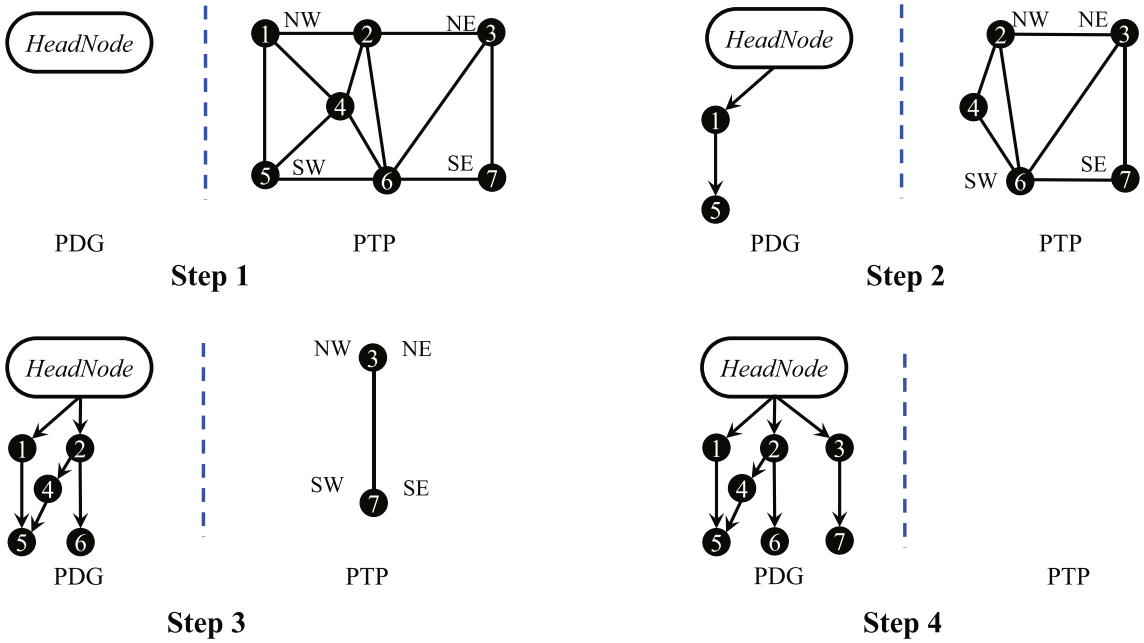


Fig. 8. Generating a PDG from an input PTP.

6. A case study

We have implemented our GADG system using `JAVA` and `Eclipse SWT Widget`. For an input floor plan with dozens of rooms, GADG generates various alternative layouts within milliseconds. This section reports the detailed step-by-step result for a case study. The input `IFC` file is generated using `Autodesk Revit`.

We [27] recently enhanced the shape grammar interpreter [29] by allowing users to import shapes and images and to use colors, among other enhanced capabilities. The graphical user interface facilitates users in importing sources and setting rule parameters, as shown in Fig. 14. In this enhanced interpreter, a dual graph can be generated from any input according to the adjacency relations of rooms and with transformation rules applied to the graph. If the graph is a PTP graph, GADG maps rooms onto the base-layout according to the improved rectangular dual finding algorithm (see Section 5). The interpreter is able to accept any parameters for customization and thus generate layouts to suit a wide range of design requirements. Users are able to

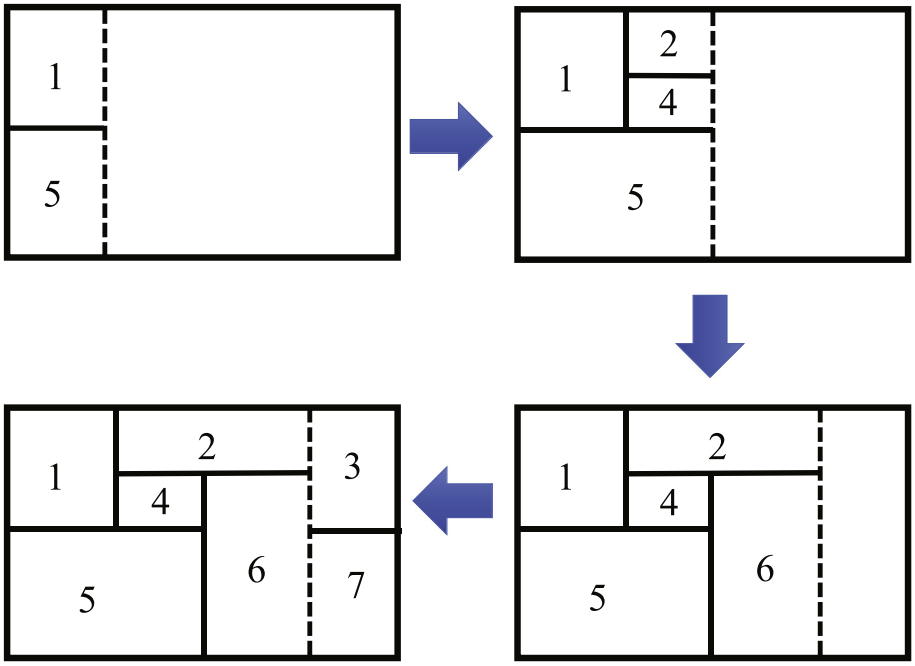


Fig. 9. Generating a RDG from the PDG and PTP.

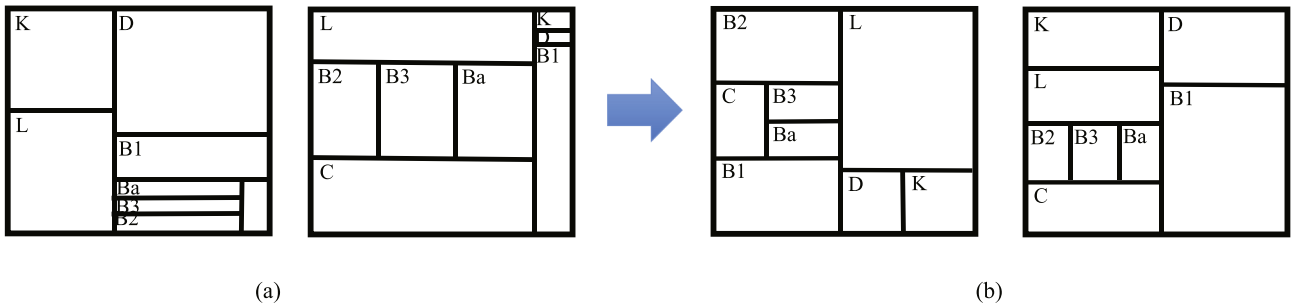


Fig. 10. The first left boundary must be cycle-free.

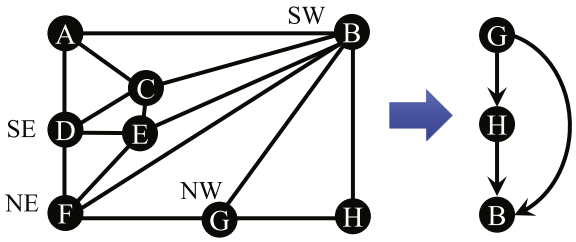


Fig. 11. Floor plans generated (a) without ratio constraint; (b) with the maximum aspect ratio of 2.5.

specify constraints, e.g. aspect ratio, base-layout size, add/remove rooms and move rooms, in the pop-up window for controlling the generated plans with different requirements.

The input floor plan consists of 7 rooms including *Hallway*, commonly seen in modern floor plan libraries as shown in Fig. 16 (a). GADG first derives the spatial relations of rooms to generate the corresponding dual graph (Fig. 16 (b)). The RDG checking algorithm is then followed to check if there exist any RDGs. If so, it generates RDGs, which represent floor plans, using the RDG finding algorithm. Otherwise, it reports an error message. Some of the results are reported in Fig. 16 (c). The generated floor plans hold the same adjacency relationships, yet with different layouts. In fact, the uncertainty of the four corner vertices in GADG and the randomness of the room generation process facilitates to generate a distinct floor plan in each iteration.

With the generated layouts, the designer is able to choose a favored design for further refinements and/or customization as shown in Fig. 16 (e)~and~(f). The 3D building model for the corresponding floor plan is shown in Fig. 16 (g).

Assuming that the designer wishes to add a new bedroom into the design and also requires it to be adjacent to the hallway, he/she may specify such requirements in the GADG user interface. GADG would then apply the addition rule to add a new room Bedroom 3, adjacent to

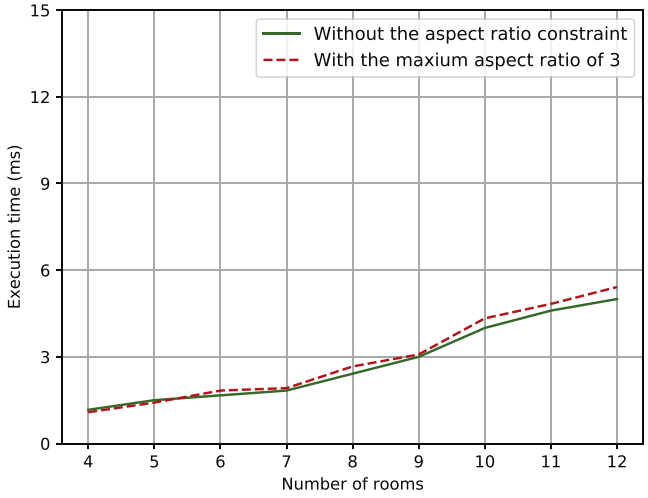


Fig. 13. The execution time for GADG to generate different number of rooms.

Hallway, into the graph, as shown on the right column of Fig. 16. GADG may also take a final design as the input to generate more floor plans, in an unlimited generation-selection-refinement design cycle.

7. Conclusion

This paper has presented a Graph Approach to Design Generation, GADG, that automatically generates floor plans, whose dual graphs are PTP graphs. The dual graph of a generated floor plan either conforms to the dual graph of an input layout or is derived from the input's dual graph by applying the transformation rules. Two transformation rules, addition rule and subtraction rule, manipulate rooms on graph structures according to spatial relations of rooms. GADG is able to generate numerous floor plans with identical connectivity. The resulting designs

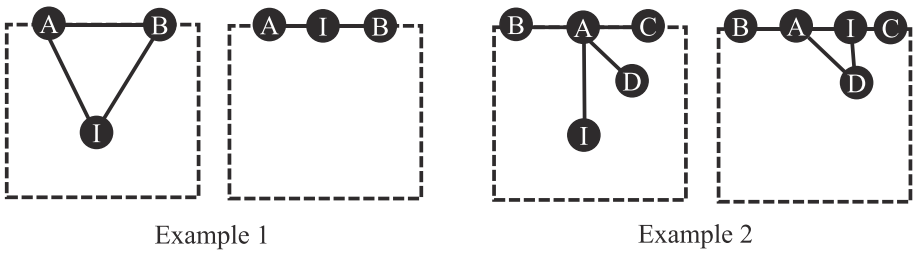


Fig. 12. Moving an internal vertex to the boundary.

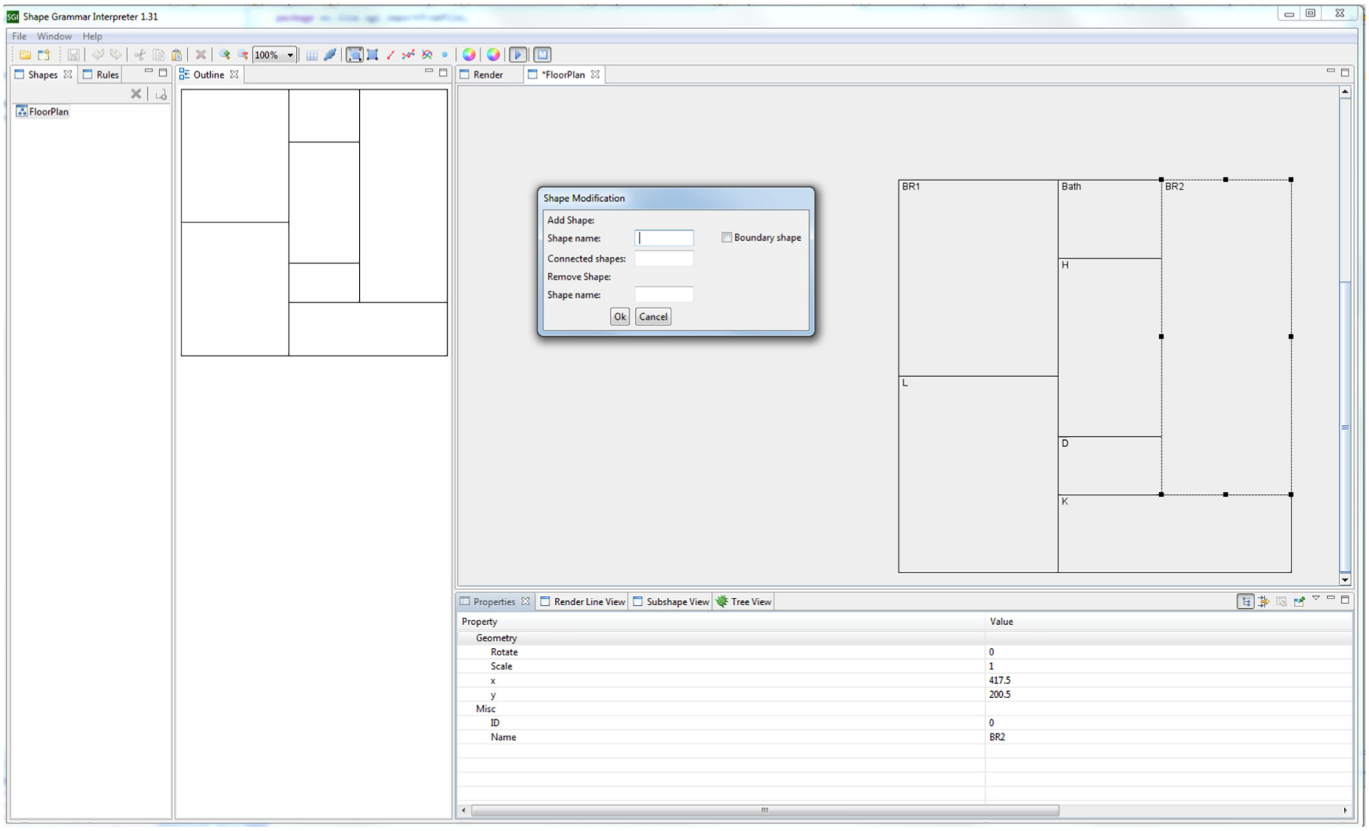
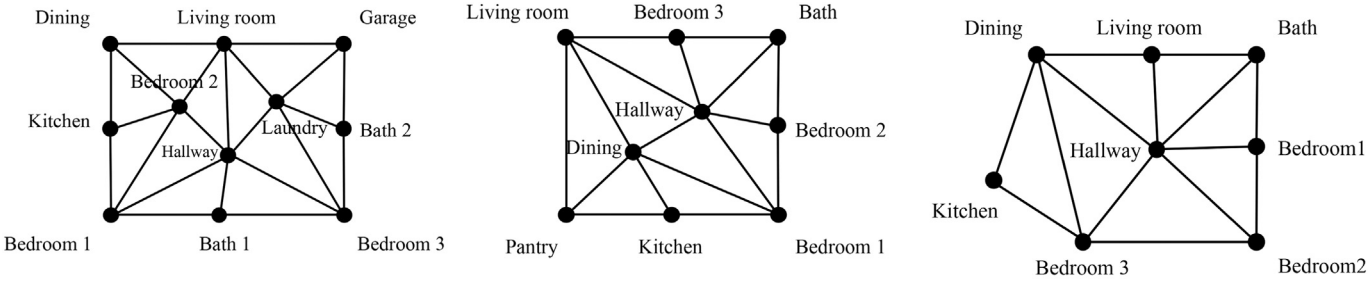


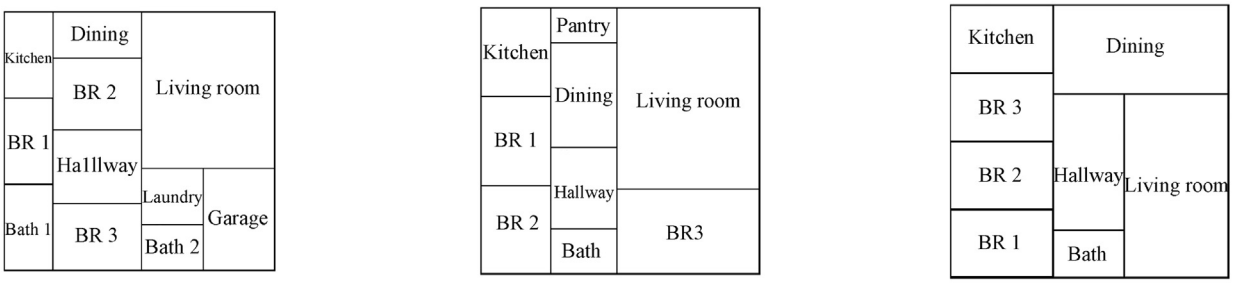
Fig. 14. An interface of the enhanced shape grammar interpreter integrated with GADG.

are controllable and tractable. Fig. 15 presents more generated floor plans with different input graphs. Fig. 15 (a) shows input graphs, while Fig. 15 (b) shows the floor plans generated by GADG.

GADG is a generic framework that may take any types of source files with proper format conversion, as long as the spatial relations among rooms are retrievable. Obtaining the spatial relations, the transformation rules allow users to modify graphs, and associated with floor plans, as desired. A rectangular dual checking and finding algorithm is used to generate various floor plans by mapping vertices in a graph to rectangles in accordance with the connectivity among vertices.



(a) Input graphs



(b) Generated Floor plans

Fig. 15. Floor plans generated by GADG.

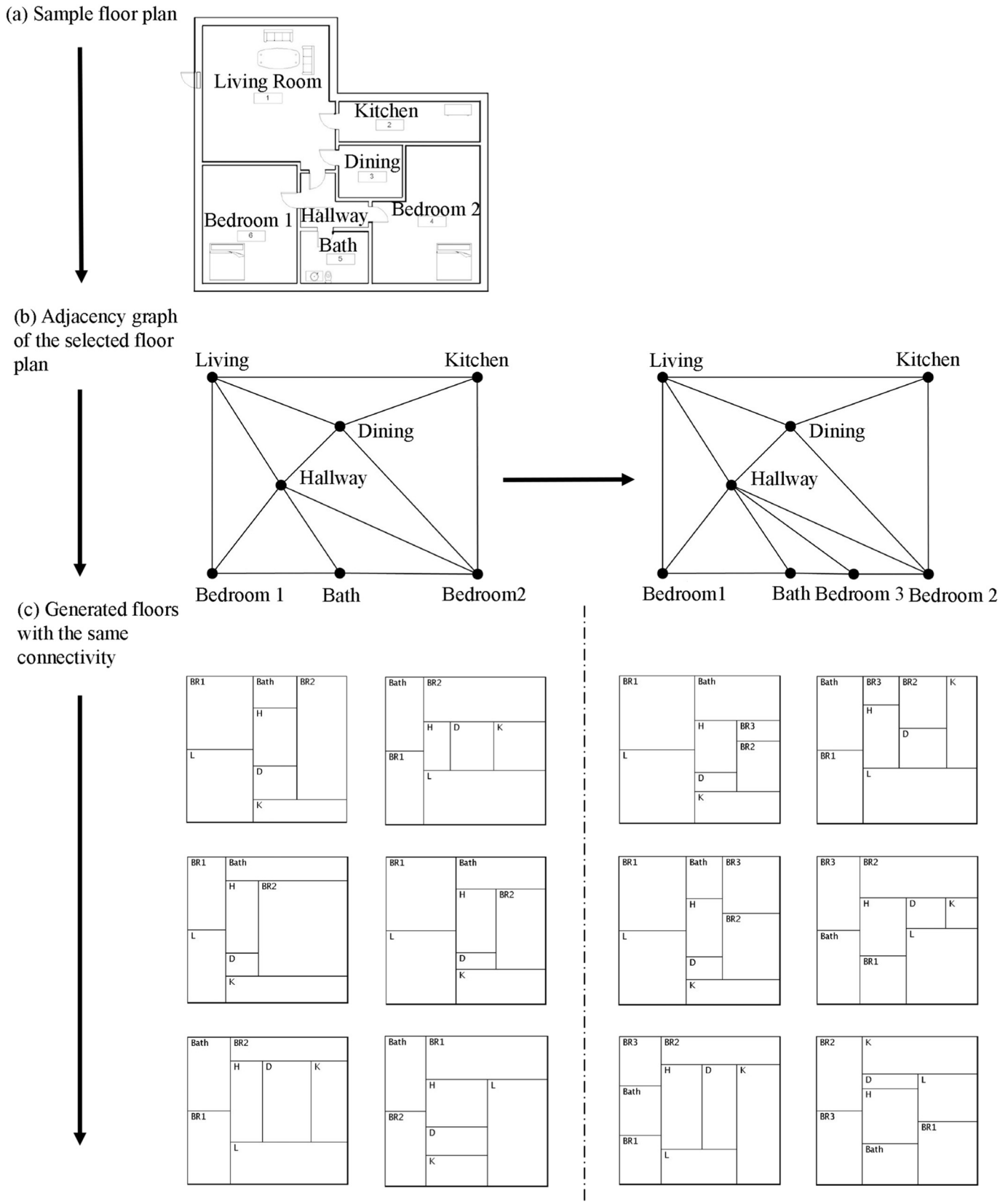


Fig. 16. Process of automatic floor plan generation. The left side presents the process of floor plan generation. The right side presents the floor plan generation by applying the addition rule to add a room *Bedroom 3*, which is connected to *Hallway*.

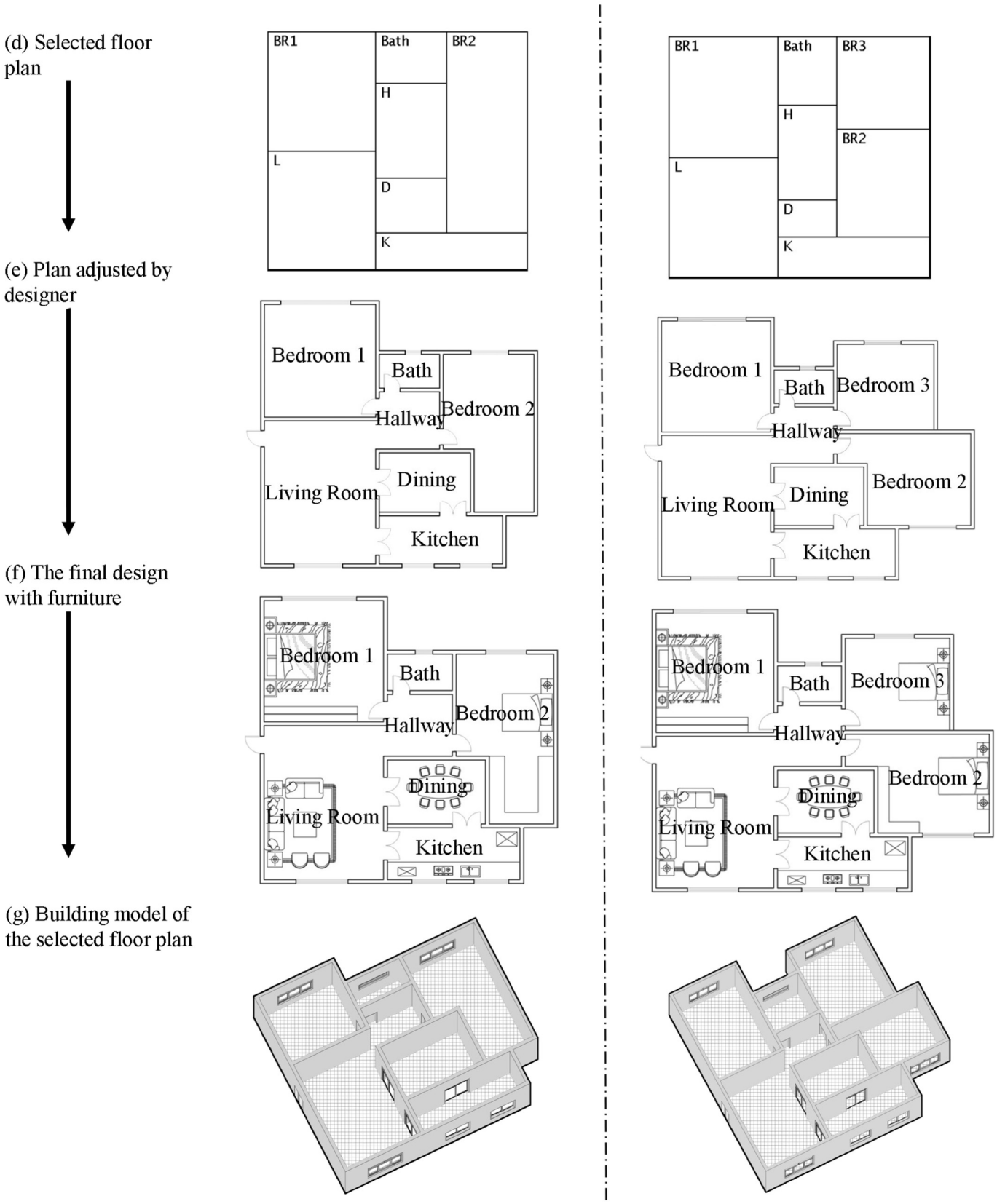


Fig. 16. (*continued*)

There are of course limitations in the current GADG implementation. For instance, it is still not straightforward to incorporate practical constraints. For example, we can only specify the connectivity among rooms without considering the rooms' directionality. For a large and complicated layout, it may also be helpful to organize rooms into regions and build a hierarchical dual graph. Furthermore, our current implementation only generates rectangular rooms from PTP graphs. All of these limitations provide us exciting new directions for the future work. Our future plan also includes an extension to generate rooms of different shapes for any given adjacency graphs. Moreover, using a hierarchical approach, large-scale buildings and houses can be generated using the shape grammar approach.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.autcon.2018.07.017.

References

[1] G. Stiny, W.J. Mitchell, The Palladian grammar, Environ. Plann. B. Plann. Des. 5 (1) (1978) 5–18, https://doi.org/10.1068/b050005.

[2] G. Stiny, W.J. Mitchell, The grammar of paradise: on the generation of Mughul gardens, Environ. Plann. B. Plann. Des. 7 (2) (1980) 209–226, https://doi.org/10.1068/b070209.

[3] H. Koning, J. Eizenberg, The language of the prairie: Frank Lloyd Wright's prairie houses, Environ. Plann. B. Plann. Des. 8 (3) (1981) 295–323, https://doi.org/10.1068/b080295.

[4] L. Fan, P. Musialski, L. Liu, P. Wonka, Structure completion for facade layouts, ACM Trans. Graph. (TOG) 33 (6) (2014), https://doi.org/10.1145/2661229.2661265 210:1–210:11.

[5] L. Liu, S. Zlatanova, Generating navigation models from existing building data, ISPRS Int. Arch. Photogramm. Remote. Sens. Spat. Inf. Sci. XL-4/W4 (4) (2013) 19–25, https://doi.org/10.5194/isprsarchives-XL-4-W4-19-2013.

[6] Y.-H. Lin, Y.-S. Liu, G. Gao, X.-G. Han, C.-Y. Lai, M. Gu, The IFC-based path planning for 3D indoor spaces, Adv. Eng. Inform. 27 (2) (2013) 189–205, https://doi.org/10.1016/j.aei.2012.10.001.

[7] J. Bhasker, S. Sahni, A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph, Proceedings of the 23rd ACM/IEEE Design Automation Conference, DAC '86, IEEE Press, Piscataway, NJ, USA, 1986, pp. 108–114, , https://doi.org/10.1145/318013.318031.

[8] J. Bhasker, S. Sahni, A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph, Networks 17 (3) (1987) 307–317, https://doi.org/10.1002/net.3230170306.

[9] A.B. Kahng, Classical Floorplanning Harmful? Proceedings of the 2000 International Symposium on Physical Design, ISPD '00, ACM, New York, NY, USA, 2000, pp. 207–213, , https://doi.org/10.1145/332357.332401.

[10] P. Galle, An algorithm for exhaustive generation of building floor plans, Commun. ACM 24 (12) (1981) 813–825, https://doi.org/10.1145/358800.358804.

[11] J. Roth, R. Hashimshony, A. Wachman, Turning a graph into a rectangular floor plan, Build. Environ. 17 (3) (1982) 163–173, https://doi.org/10.1016/0360-1323(82)90037-3.

[12] S.M. Leinwand, Y. Lai, An algorithm for building rectangular floor-plans, Proceedings of the 21st Design Automation Conference, DAC '84, IEEE Press, Piscataway, NJ, USA, 1984, pp. 663–664, , https://doi.org/10.1109/DAC.1984.1585874.

[13] K. Kozminski, E. Kinnen, An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits. Proceedings of the 21st Design Automation Conference, DAC '84, IEEE Press, Piscataway, NJ, USA, 1984, pp. 655–656, , https://doi.org/10.1109/DAC.1984.1585872.

[14] P. Wonka, M. Wimmer, F. Sillion, W. Ribarsky, Instant architecture, ACM Trans. Graph. (TOG) 22 (3) (2003) 669–677, https://doi.org/10.1145/882262.882324.

[15] J. Martin, Procedural House Generation: A Method for Dynamically Generating Floor Plans, (2006) 10.1.1.97.4544.

[16] F. Marson, S.R. Musse, Automatic real-time generation of floor plans based on squarified treemaps algorithm, Int. J. Computer Games Technol. 2010 (2010), https://doi.org/10.1155/2010/624817 7:1–7:10.

[17] P. Merrell, E. Schkufza, V. Koltun, Computer-generated residential building layouts, ACM Trans. Graph. (TOG) 29 (6) (2010), https://doi.org/10.1145/1882261.1866203 181:1–181:12.

[18] K. Shekhawat, Algorithm for constructing an optimally connected rectangular floor plan, Front. Architectural Res. 3 (3) (2014) 324–330, https://doi.org/10.1016/j.foar.2013.12.003.

[19] G. Stiny, J. Gips, Shape Grammars and the Generative Specification of Painting and Sculpture, International Federation for Information Processing Congress (2), vol. 2, 1971 ISBN: 0 7204 2063 6.

[20] M. Harada, A. Witkin, D. Baraff, Interactive Physically-based Manipulation of Discrete/Continuous Models, Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95, ACM, New York, NY, USA, 1995, pp. 199–208, , https://doi.org/10.1145/218380.218443.

[21] P. Müller, P. Wonka, S. Haegler, A. Ulmer, L. Van Gool, Procedural modeling of buildings, ACM Trans. Graph. (TOG) 25 (3) (2006) 614–623, https://doi.org/10.1145/1141911.1141931.

[22] P. Müller, G. Zeng, P. Wonka, L. Van Gool, Image-based procedural modeling of facades, ACM Trans. Graph. (TOG) 26 (3) (2007), https://doi.org/10.1145/1276377.1276484.

[23] F. Wu, D.-M. Yan, W. Dong, X. Zhang, P. Wonka, Inverse procedural modeling of facade layouts, ACM Trans. Graph. (TOG) 33 (4) (2014), https://doi.org/10.1145/2601097.2601162 121:1–121:10.

[24] J. Lin, D. Cohen-Or, H. Zhang, C. Liang, A. Sharf, O. Deussen, B. Chen, Structure-preserving retargeting of irregular 3D architecture, ACM Trans. Graph. (TOG) 30 (6) (2011), https://doi.org/10.1145/2070781.2024217 183:1–183:10.

[25] B.K. Sarkar, S.K. Chakraborty, Combinatorics and Graph Theory, PHI Learning, 2016 ISBN: 978-81-203-5173-8.

[26] T. Nishizeki, N. Chiba, Planar Graphs: Theory and Algorithms, vol. 32, Elsevier, 1988, pp. 15–16 Ch. 1 ISBN: 978-91-637-4473-0.

[27] X.-Y. Wang, K. Zhang, Enhancements to a Shape Grammar Interpreter, Proceedings of the 3rd International Workshop on Interactive and Spatial Computing, IWISC '18, ACM, New York, NY, USA, 2018, pp. 8–14, , https://doi.org/10.1145/3191801.3191805.

[28] K. Maling, W.R. Heller, S.H. Mueller, On Finding Most Optimal Rectangular Package Plans, Proceedings of the 19th Design Automation Conference, IEEE Press, Piscataway, NJ, USA, 1982, pp. 663–670, , https://doi.org/10.1109/DAC.1982.1585567.

[29] T. Trescak, M. Esteva, I. Rodriguez, A shape grammar interpreter for rectilinear forms, Comput. Aided Des. 44 (7) (2012) 657–670, https://doi.org/10.1016/j.cad.2012.02.009.