Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2010, Article ID 624817, 10 pages
doi:10.1155/2010/624817

Research Article

Automatic Real-Time Generation of Floor Plans Based on Squarified Treemaps Algorithm

Fernando Marson and Soraia Raupp Musse

Graduate Programme in Computer Science, PUCRS, Avenue Ipiranga, 6681, Building 32, Porto Alegre, RS, Brazil

Correspondence should be addressed to Soraia Raupp Musse, soraia.musse@pucrs.br

Received 28 May 2010; Accepted 28 August 2010

Academic Editor: Rafael Bidarra

A novel approach to generate house floor plans with semantic information is presented. The basis of this model is the squarified treemaps algorithm. Previously, this algorithm has been used to create graphical representations based on hierarchical information, such as, directory structures and organization structures. Adapted to floor plans generation, this model allows the creation of internal house structures with information about their features and functionalities. The main contributions are related to the robustness, flexibility, and simplicity of the proposed approach to create floor plans in real-time. Results show that different and realistic floor plans can be created by adjusting a few parameters.

1. Introduction

Considering the game titles released in the last decade, the increase in visual complexity of Virtual Environments (VE) used as scenarios is noticed. Huge cities can be found in games like the GTA franchise (http://www.rockstargames.com/IV), Assassin's Creed (http://assassinscreed.us.ubi.com), and Left 4 Dead (http://www.l4d.com) which is remarkable. Apart from the cities, it may be necessary to create whole worlds, as in the case of Massively Multiplayer Online Games (MMOGs), represented by World of Warcraft (http://www.worldofwarcraft.com) and Perfect World (http://www.perfectworld.com). As a consequence, the cost and time to develop a game are also increased.

The creation of a VE requires previous knowledge in several areas of expertise. Hence, it is necessary to allocate a team of professionals to create, maintain, and reuse large VEs. Some of the main problems faced when developing interactive virtual environments are described in [1] as the nonextensibility, limited interoperability, poor scalability, monolithic architecture, among others.

A possible solution to these problems is the use of procedural generation techniques [2], which allow the creation of VE content just by setting input parameters. Such approaches may be used to generate terrains, buildings, characters, items, weapons, quests, and even stories adding a broad range of elements, but in a controlled way. A perfect example to illustrate the potential use of procedural contents generation is the game Spore (http://www.spore.com) . In this game, procedural techniques are used to create characters, vehicles, buildings [3], textures [4], and planets [5]. Even the music is created using this kind of technique.

There are some academic and commercial solutions that provide the creation of buildings with great realism. Nevertheless, there are few studies that focus on the generation of building interiors. Our model proposes a novel solution to generate floor plans of buildings using just a few parameters and constraints. After the floor plan generation, it creates a three-dimensional representation of the construction. In all generated rooms in the floor plan; semantic information is included to allow simulations involving virtual humans.

The remainder of this paper is organized as follows: in Section 2 we discuss some work found in literature, while in Section 3 we describe our model to generate floor plans. Section 4 discusses some obtained results, and the final considerations are drawn in Section 5.

2. Related Work

The process of creating large virtual cities can take considerable time and resources to be accomplished. Parish and Müller [7] present a model that allows the generation of a three-dimensional city from sociostatistical and geographical maps. The method builds the road network using an extended L-Systems. After creating the streets, the system extracts the information about blocks. Through a subdivision process, lots are created. A building is placed at each lot, generated by another module based on L-Systems. With this information, the system generates the three-dimensional geometric model of the city, and textures are added to provide greater realism to the final model.

A method to generate procedural "pseudoinfinite" virtual cities in real-time is proposed by Greuter et al. [8]. The area of the city is mapped into a grid defined by a given granularity and a global seed. Each cell in the grid has a local seed that can be used to create building generation parameters. A print foot is produced by combining randomly generated polygons in an iterative process, and the building geometries are extruded from a set of floor plans. To optimize the rendering process, a view *frustrum* is implemented to determine the visibility of virtual world objects before their generation, so that only visible elements are generated. Besides the generation of the environment, the appearance of the buildings can be improved. In this context, Müller et al. [9] propose a shape grammar called CGA Shapes, focused on the generation of buildings with high visual quality and geometric details. Using some rules, the user can describe geometric shapes and specify the interactions between hierarchical groups in order to create a geometrically complex object. In addition, the user can interact dynamically in all stages of creation.

The techniques presented previously are focused on the external appearance of buildings, without concerning their interior. Martin [10] introduces an algorithm to create floor plans of houses. The process consists of three main phases. In the first step of the procedure, a graph is created to represent the basic structure of a house. This graph contains the connections between different rooms and ensure that every room of house is accessible. The next step is the placement phase, which consists of distributing the rooms over the footprint. Finally, the rooms are expanded to their proper size using a Monte Carlo method to choose which room to grow or shrink.

An approach to generate virtual building interiors in real-time is presented by Hahn et al. [11]. The interiors are created using eleven rules that work like a guideline to the generation process. Buildings created by this technique are divided into regions connected by portals. Only the visible parts of the building are generated, avoiding the use of memory and processing. When a region is no longer visible, the structure is removed from the memory. The generation process also provides a persistent environment: all changes made in a given region are stored in a record that is accessed through a hash map when necessary.

Horna et al. [12] propose a method to generate 3D constructions from two-dimensionalarchitectural plans. Additional information can be included to the two-dimensional plans in order to support the creation of three-dimensional model. It is possible to construct several floors using the same architectural plan.

A rule-based layout approach is proposed by Tutenel et al. [13]. Their method allows to solve the layout and also to distribute objects in the scene at the same time. From an initial layout, the algorithm finds the possible locations of a new object based on a given set of rules. The relations between objects can be specified either explicitly or implicitly. The method uses hierarchical blocks in the solving process, so that if a set of elements are solved, they are treated as single block.

Besides the definition of appearance and geometry of objects, it is also necessary to specify their features and functionalities. Semantic information can be used to enhance the environment of games and simulations. This can be done by specifying features of a given actor or object, as well as functionality, physical or psychological attributes and behaviors. Tutenel et al. [14] list three levels of semantic specification: object semantics, object relationships, and world semantics. These levels can be used to create and simulate an environment. For example, information such as the climate of a region can be used to define the kind of vegetation, and the weight of an object can be used to decide if an agent can carry it or not.

The main contribution of this paper is to provide a framework for real-time procedural modeling of floor plans of houses, generating geometric and semantic information in a robust way, where all rooms can be accessible from outside the environment. Virtual agents can use the provided information, so that behavioral animation can be performed. Squarified treemaps [6] are used to generate rooms which aspect ratios approach 1. Section 3 presents the proposed model.

3. Creating Floor Plans

A common drawback in procedural generation of environments is the creation of a component that is not accessible from any other component. For instance, in a virtual city, one problem is the generation of buildings that are not accessible from the streets. Concerning internal structures of buildings, a similar problem happens when one room is not connected with any other. As far as we know, neither of the proposed procedural models to generate floor plans can solve such type of situation. Our proposed method treats this problem by adding corridors into the generated floor plan, similarly to what occurs in real life.

Our method for generating floor plans is based on Squarified Treemaps, proposed by Bruls et al. [6]. A treemap [15] is an efficient and compact form to organize and to visualize elements of hierarchical structures of information, for instance, directory structures, organization structures, family trees, and others. In general, treemaps subdivide an area into small pieces to represent the importance of each part in the hierarchy. The main difference between treemaps and squarified treemaps is how the subdivision is performed. In

squarified treemaps, authors propose a subdivision method that takes into account the aspect ratio of generated regions, trying to achieve the value 1. Figure 1 shows on the left the result of the original treemap method and on the right the squarified treemap. Both models are discussed next.

3.1. Treemaps and Squarified Treemaps. The original treemaps approach [15] uses a tree structure to define how information should be used to subdivide the space. Figure 2 shows an example. Let us consider that each node in the tree (Figure 2(a)) has an associated size (e.g., in Figure 2 the name of node is *a* and its size is 20). The treemap is built using recursive subdivision of an initial rectangle (see Figure 2(b)). The direction of subdivision alternates per level (horizontally and vertically). Consequently, the initial rectangle is subdivided in small rectangles. Further details about treemaps can be found in [15].

This method can originate subdivisions like the one illustrated in Figure 3. In this case, it is possible to see that aspect ratios of generated rectangles are very different from 1. Consequently, this approach is not adequate to tackle the problem that we want to deal with in this paper.

Squarified treemaps were proposed by Bruls et al. [6] and have the main goal of maintaining the aspect ratios of generated rectangles, defined as max(height/width, width/height), as close to 1 as possible. Their method is implemented using recursive programming and aims to generate rectangles based on a predefined and sorted list containing their desired areas (from larger to smaller areas). Then, the aspect ratio of the region to be subdivided is considered in order to decide whether to divide it horizontally or vertically. Also, the aspect ratio of generated regions in a specific step t is compared to step $t + 1$, being possible to ignore later computing regions in $t + 1$ and reconsider data from step t. Figure 4 illustrates the generation process of squarified treemaps presented by Bruls et al. [6]. We discuss this example in this paper since the understanding of our model is very dependent of squarified treemap method.

The list of rectangular areas to be considered in the example of Figure 4 is: 6, 6, 4, 3, 2, 2, 1. In step 1, the method generates a rectangle with aspect ratio = 8/3 in vertical subdivision. So, in step 2 the horizontal subdivision is tested, generating 2 rectangles with aspect ratio = 3/2 which are close to 1. In step 3, the next rectangle is generated, presenting aspect ratio = 4/1. This step is ignored, and step 4 is computed based on rectangles computed in step 2. The algorithm (described in detail in [6]) presents rectangles which aspect ratios are close to 1. In our point of view, this method is more adequate to provide generation of floor plans than original treemaps, due to the fact that rooms in real houses present aspect ratios not very different from 1. However, other problems may occur, as shown in the next sections.

3.2. The Proposed Model. The pipeline of the proposed model to build floor plans is presented in Figure 5. The process begins with the definition of construction parameters and layout constraints.

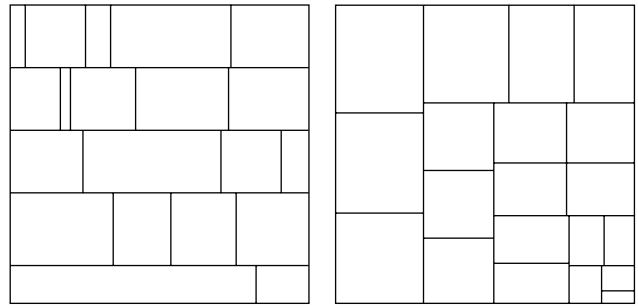

Figure 1: Original treemap (a) and squarified treemap (b).

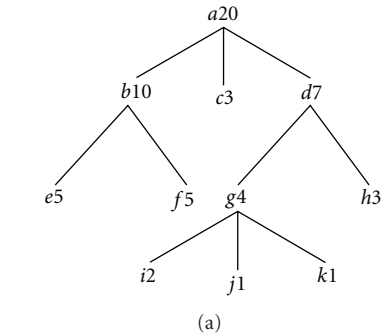

Figure 2: Tree diagram (a) and related treemap (b).

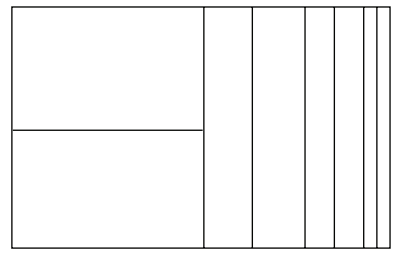

Figure 3: Example of treemap subdivision generating rectangles with aspect ratio different from 1.

Table 1: Possible connections between rooms.

	Outside	Kitchen	Pantry	Laundry room	Living room	Dining room	Toilet	Bedroom	Master bedroom	Bath room	Secondary room
Outside		X			X						
Kitchen	X		X	X	X	X					
Pantry		X		X							
Laundry room		X	X								
Living room	X	X				X	X	X	X	X	X
Dining room		X			X		X	X	X	X	X
Toilet					X	X					
Bedroom					X	X				X	
Master bedroom					X	X				X	
Bathroom					X	X		X	X		X
Secondary room					X	X				X	

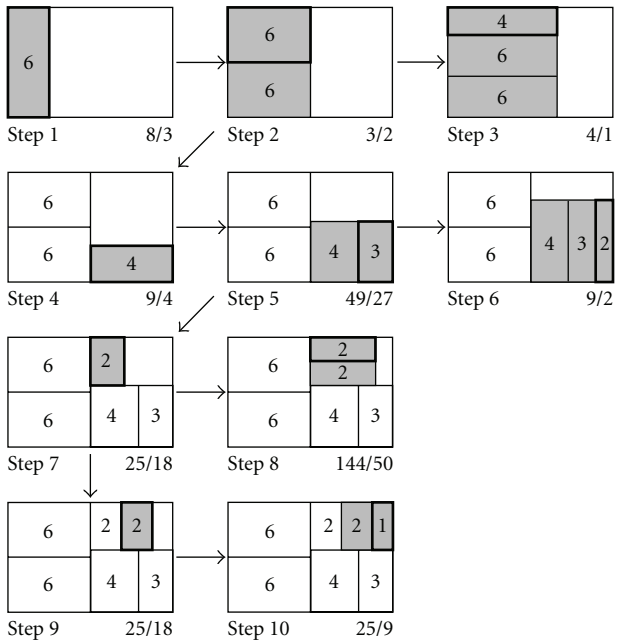

Figure 4: Example of squarified treemap process [6].

To create a floor plan, some parameters such as, height, length, and width of the building are required. It is also necessary to know the list of desired dimensions for each room and their functionalities. The functionality specifies how a particular area of residence should be used. There are three distinct possibilities: social area, service area, and private area. The social area can include the living room, the dining room, and the toilet. In the service area we can have the kitchen, the pantry, and the laundry room. At last, the private area can embrace the bedroom, the master bedroom, the intimate bathroom, and a possible secondary room that can be used in different ways, for example, as a library. This list is not fixed and can be customized by the user, and the proposed categorization is made to group the common areas.

The division of the residence area occurs in two different steps. At first, we compute the area of each one of three parts of the building (i.e., social, service, and private), and secondly, apply the squarified treemap in order to define three regions where rooms will be generated. This process generates an initial layout, containing three rectangles, each one for a specific area of the building (see Figure 6(a)).

After obtaining the positions of the polygon that represents a specific area, each polygon serves as input to run again squarified treemap algorithm in order to generate the geometry of each room. It is important to notice that we use original squarified treemap to generate each room in the building. Figure 6(b) shows the generated rooms.

With the result of the room subdivision, two more steps are required to complete the floor plan generation. Firstly, connections among rooms should be created. Secondly, rooms that are not accessible should be treated, since our environments should be used for characters animation. These two steps are further discussed in next sections.

3.3. Including Connections among the Rooms. With all the rooms generated as previously described, connections (doors) should be created among them. These connections are created using as criteria the functionalities of each room, that is, some rooms are usually not connected, for example, the kitchen and the bedroom. All the possible connections are presented in Table 1, which has been modeled based on a previous analysis of various floor plans commercially available. In this table, we consider the entry door of the floor plan as a connection from outside and two possible rooms: kitchen and living room. However, it is important to note that another set of connections can be defined by the user, to represent another style of architecture.

Geometrically, the door is created on the edge shared by two rooms that keep a possible connection. For instance, it is possible to have a connection between the kitchen and the living room, where a door can be created. The size of the doors is predefined, and their center on the edge is randomly

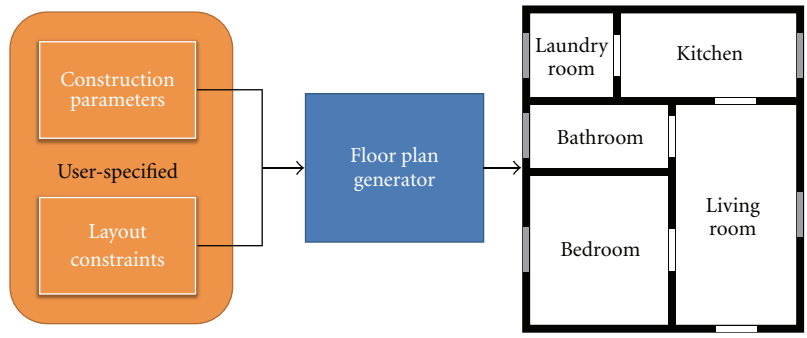

Figure 5: The generation process of floor plans. Construction parameters and layout constraints are provided by the user as input to floor plan generator.

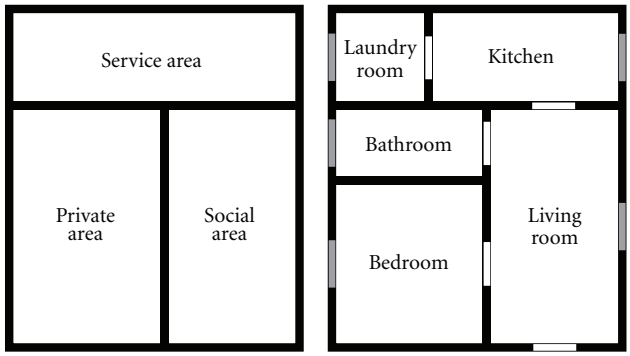

Figure 6: Dividing the total space of the house in three main areas (a): private, social, and service area. Example floor plan generated by the model (b).

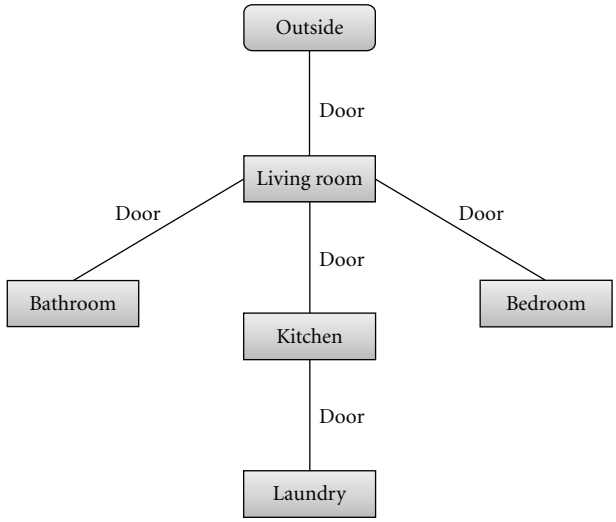

Figure 7: Connectivity graph for a generated floor plan.

defined. A similar process happens with the generation of windows, but ensuring that they are placed on the external edges. Figure 6(b) illustrates a generated floor plan, containing windows (dark rectangles) and doors (white rectangles).

After the connections between rooms are processed, a connectivity graph is automatically created (Figure 7), representing the links among the rooms. It allows checking if there is any room that is not accessible. Also, buildings and houses created with our method can be used to provide environment for characters simulation. The graph always starts from outside and follows all possible connections from the accessed room.

If any room is not present in the graph, it is necessary to include corridors. This situation can happen when there are no possible connections between neighboring rooms (rooms which share edges). This process is described in the next section.

3.4. Including Corridors on the Floor Plans. The generation of corridors is necessary in order to maintain the coherence of the generated environment and to provide valid space for characters navigation. Firstly, the rooms without access from the living room are selected. These rooms are flagged with an X, as illustrated in Figure 8(a). The main idea is to find possible edges shared by nonconnected rooms to be used to create the corridor.

The corridor must connect the living room (marked with an L in Figure 8(a)) with all X rooms. The proposed solution uses the internal walls of the building to generate a kind of circulation "backbone" of the space, that is, the most probable region to generate the corridor in the floor plan. The algorithm is very simple and has three main steps. Firstly, all external walls are removed, as corridors are avoided in the boundary of the floor plan (Figure 8(b)).

Secondly, we remove all internal segments (representing walls) that belong to the living room (Figure 8(c)). The remaining segments (described through their vertices) are used as input to the graph creation. Vertices are related to nodes, and segments describe the edges in the graph (Figure 8(c)). In order to deal with the graph, we use the A* algorithm [16], which is a very known algorithm widely used in path-finding and graph traversal. In our case, the graph is explored to find the shortest path that connects all rooms without connectivity to the living room. A room is considered connected if the graph traverses at least one of its edges. Finally the shortest path is chosen to create the corridor (Figure 8(d)).

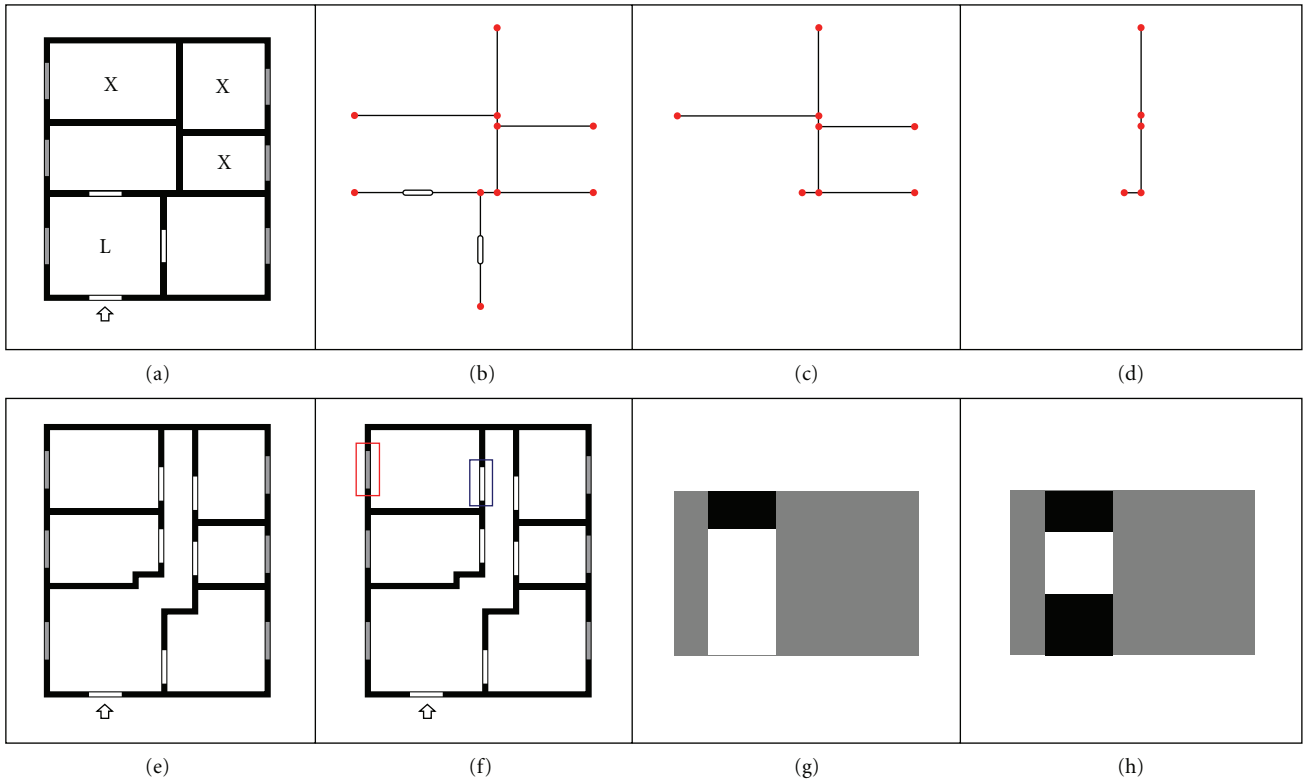

Figure 8: Steps to end the building generation. (a) Original floor plan with rooms without connectivity flagged with an X and the living room is marked with L. (b) Floor plan after external edges removed. (c) Internal walls that do not belong to living room. (d) The shortest path linking all rooms without connectivity to the living room. (e) Final 2D floor plan. (f) The red and blue rectangles highlight, respectively, the representation of a window and a door in the floor plan. Creation process of windows (g) and doors (h).

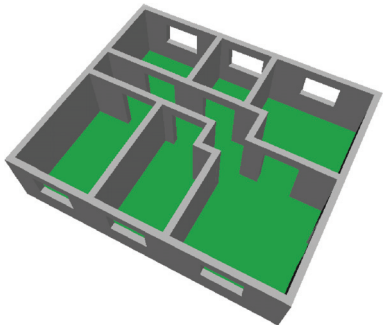

Figure 9: Three-dimensional model of 2D floor plan illustrated in Figure 8(e).

The "backbone" is the set of edges candidates to be used to generate the corridor. After the "backbone" generation, we should generate the polygons which represent the corridor, which is initially composed by a set of edges/segments. These segments must pass through a process of 2D extrusion in order to generate rectangles, allowing agents to walk inside the house. Indeed, the size of the corridor is increased perpendicularly to the edges. However, this process can cause an overlap between the corridor and some rooms, reducing their areas. If the final area of any room is smaller than the minimum allowed value, the floor plan undergoes a global readjustment process to correct this problem, as shown in 1

$$\text{area}_{\text{house}} = \text{area}_{\text{private}} + \text{area}_{\text{social}} + \text{area}_{\text{service}} + \text{area}_{\text{corridor}}. \quad (1)$$

After obtaining the final floor plan (Figure 8(e)), two simple steps should be followed to generate the final building. Initially, each 2D wall represented on the floor plan is extruded to a given height H defined by the user. Generated walls can have doors (between two rooms) (blue rectangle in Figure 8(f)) and windows (red rectangle in Figure 8(f)), that should be modeled properly. The last step of process is to add an appropriate type of window to each room, according to its functionality, from a set of models. A three-dimensional view of a 2D floor plan (Figure 8(e)) is presented in Figure 9.

4. Results

In the following floor plans, we explicitly present the parameters used in their generation, and also show the geometric and semantic information generated. In addition, the connectivity graph is shown for each house or building.

Figure 10(a) shows a floor plan generated using our model. The list of rooms and their respective areas are as follows: living room (9 m^2), two bedrooms (8 m^2 and 7 m^2),

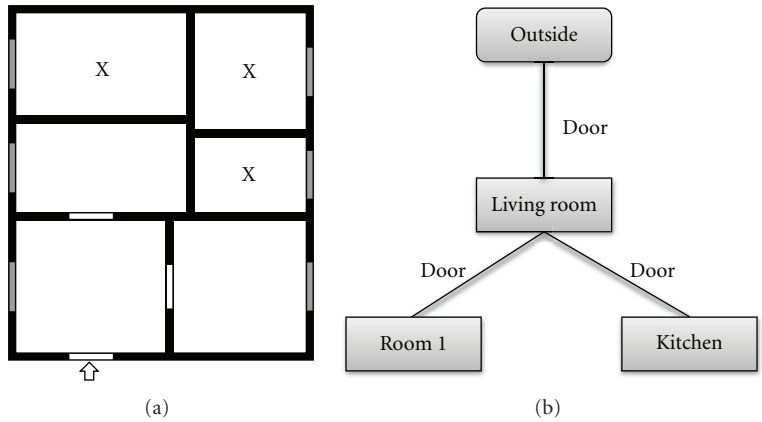

Figure 10: Example of floor plan (a) containing 2 nonconnected rooms (labeled with X) and its respective connectivity graph (b).

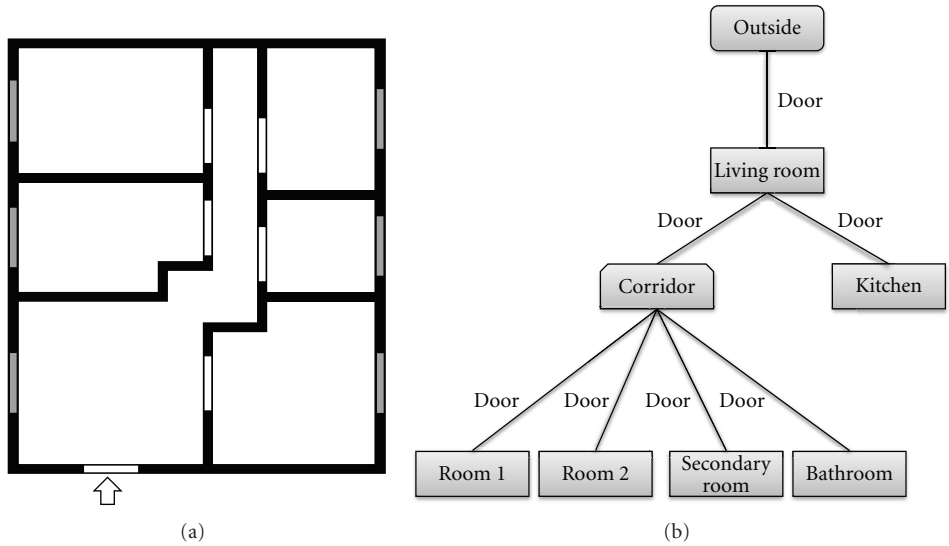

Figure 11: Example of floor plan of Figure 10 (a) containing corridors and all rooms are connected to the living room and its respective connectivity graph (b).

secondary room (6 m^2), bathroom (4 m^2), and kitchen (8 m^2). The dimensions of the house are 6 meters wide, 7 meters long, and 3 meters high. The connectivity graph can be seen in Figure 10(b). Both the bathroom and the secondary room do not have connection with any other room. This situation is corrected by adding a corridor (Figure 11(a)). The new connectivity graph is shown in Figure 11(b). All rooms are now connected, being accessible from the outside or from any other internal room.

Another case study is illustrated in Figure 12(a). This house has 84 squared meters (12 m long × 7 m wide, being 3.1 m high). The list of rooms and their respective areas are as follows: living room (22 m^2), two bedrooms (both with 14 m^2), a secondary room used as home office (12 m^2), bathroom (10 m^2), and kitchen (12 m^2). For this specific configuration, the connectivity graph that can be seen in Figure 12(b) was generated. Both bedrooms and the bathroom are not accessible from any other room. The solution is presented in Figure 13(a). Using the corridor, the new connectivity graph looks like the one presented in Figure 13(b). A generated 2D floor plan can be saved to disk and used as input to a home design software. Figure 14 shows the result of same floor plan when adding some 2D furniture.

In addition, we compared the floor plan of Figure 14 with an available real floor plan. As it can be seen in Figure 15, such real floor plan could generate exactly the same connectivity graph (see Figure 13(b)). Furthermore, the total dimension of the real plan is 94 m^2, while virtual plan has 84 m^2, and still generating rooms of similar size.

In Figure 16 we can observe a 3D model of floor plan visualized with textures.

It is important to note that for these results we determine parameters values in order to provide specific floor plans, and even compare with available houses, existent in real life. However, one can imagine having our method integrated into a game engine, and generating different floor plans in

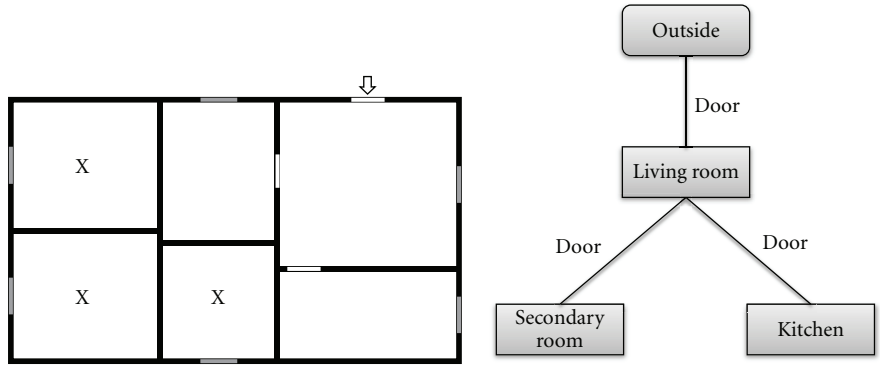

Figure 12: Example of floor plan (a) containing 3 nonconnected rooms (labeled with X) and its respective connectivity graph (b).

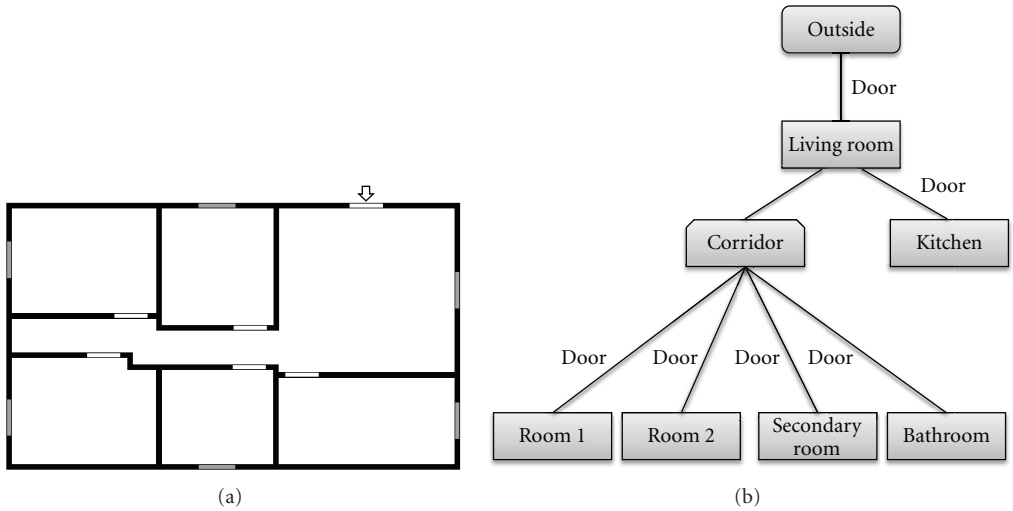

Figure 13: Example of floor plan of Figure 12 (a) containing corridors and all rooms are connected to the living room and its respective connectivity graph (b).

Figure 14: Our floor plan and included furnitures.

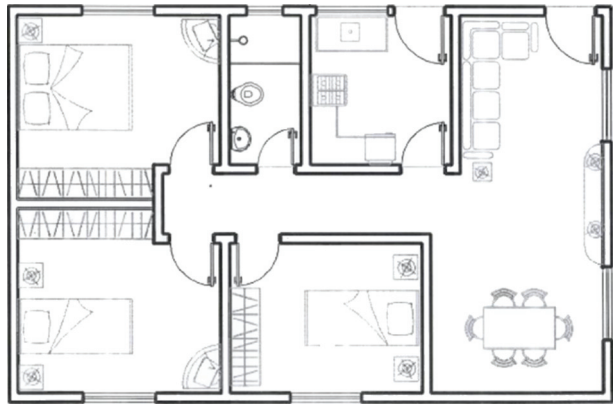

Figure 15: Real floor plan.

a dynamic way. For instance, using random functions and intervals for areas and a random number of rooms, our method is able to generate different floor plans automatically. So, during the game, the player can visit different buildings and have always different floor plans, without previous modeling.

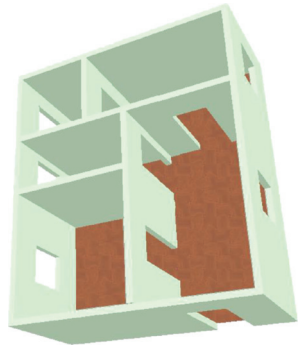

Figure 16: Three-dimensional model of a generated floor plan.

5. Final Considerations

We present a technique to provide floor plans for houses in an automatic way. The relevance of this method is the ability to generate real-time buildings for games and simulations. The generated environments are realistic as they could easily be found in real life. Besides geometric information, the technique also generates semantic information which is very useful to allow virtual humans simulation.

We compared a generated floor plan with an available commercial house in order to verify the similarity between the distribution of generated rooms and real ones. Furthermore, this work contributes to generate a connection graph which determines the navigation graph in the environment. Corridors could be created to solve problems of nonconnected rooms.

Currently there are some limitations that should be worked out later. The first concern relates to the general appearance of generated houses. Due to the squarified treemaps algorithm, created floor plans will present always a square or a rectangular shape. Depending on the initial division in social area, service area, and private area, possibly some parts may present an aspect ratio far away from 1. A possible improvement would be to change the order of the initial division of the three areas mentioned. Regarding the corridors generation, a few unnatural angles may occur due to the insertion of corridors. A possible way to solve this issue could be to align parallel straight lines of the backbone if the distance between them is smaller than a certain threshold. Probably the impact in each room area could be despicable, but this needs further investigation.

From an architectural perspective, both neighborhood and environmental factors (e.g., topography and solar orientation) should be considered when defining the house position and the apertures position (doors and windows). The introduction of a few parameters could help to generated a floor plan connected to the environment.

Future work should be focused on providing a procedural model of objects into each room (like furnitures), as well as data that allow easily behavioral simulation. For instance, we can codify in each room possible behaviors to be adopted by agents in different rooms and depending on their internal status and needs. Another issue that will be investigated is the extension of the proposed method in order to generate floor plans of an entire building or homes with more than one floor.

Acknowledgment

This work was supported by the brazilian research agency FINEP.

References

[1] M. Oliveira, J. Crowcroft, and M. Slater, "An innovative design approach to build virtual environment systems," in *Proceedings of the Workshop on Virtual Environments (EGVE '03)*, pp. 143–151, ACM, New York, NY, USA, 2003.

[2] D. S. Ebert, K. F. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing & Modeling: A Procedural Approach*, The Morgan Kaufmann Series in Computer Graphics, Morgan Kaufmann, San Francisco, Calif, USA, 3rd edition, 2002.

[3] L. Choy, R. Ingram, O. Quigley, B. Sharp, and A. Willmott, "Rigblocks: player-deformable objects," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '07)*, p. 83, ACM, New York, NY, USA, August 2007.

[4] D. DeBry, H. Goffin, C. Hecker, O. Quigley, S. Shodhan, and A. Willmott, "Player-driven procedural texturing," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '07)*, p. 81, ACM, New York, NY, USA, August 2007.

[5] K. Compton, J. Grieve, E. Goldman et al., "Creating spherical worlds," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '07)*, p. 82, ACM, New York, NY, USA, August 2007.

[6] M. Bruls, K. Huizing, and J. van Wijk, "Squarified treemaps," in *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pp. 33–42, 2000.

[7] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, pp. 301–308, ACM, New York, NY, USA, 2001.

[8] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Real-time procedural generation of 'pseudo infinite' cities," in *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '03)*, pp. 87–94, ACM, New York, NY, USA, 2003.

[9] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '06)*, pp. 614–623, ACM, New York, NY, USA, August 2006.

[10] J. Martin, "Procedural house generation: a method for dynamically generating floor plans," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2006.

[11] E. Hahn, P. Bose, and A. Whitehead, "Persistent realtime building interior generation," in *Proceedings of the ACM SIGGRAPH Symposium on Video Games (Sandbox '06)*, pp. 179–186, ACM, New York, NY, USA, 2006.

[12] S. Horna, G. Damiand, D. Meneveaux, and Y. Bertrand, "Building 3D indoor scenes topology from 2D architectural plans," in *Proceedings of the 2nd International Conference on Computer Graphics Theory and Applications (GRAPP '07)*, pp. 37–44, Setúbal, Portugal, March 2007.

[13] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker, "Rule-based layout solving and its application to procedural interior generation," in *Proceedings of the CASA Workshop on 3D Advanced Media in Gaming and Simulation (3AMIGAS '09)*, Amsterdam, The Netherlands, 2009.

[14] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. De Kraker, "The role of semantics in games and simulations," *Computers in Entertainment*, vol. 6, no. 4, pp. 1–35, 2008.

[15] B. Johnson and B. Shneiderman, "Tree-maps: a space-filling approach to the visualization of hierarchical information structures," in *Proceedings of the 2nd Conference on Visualization (VIS '91)*, pp. 284–291, IEEE Computer Society Press, Los Alamitos, Calif, USA, 1991.

[16] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.