# Real-time Procedural Generation of Personalized Façade and Interior Appearances Based on Semantics

Ivan Silveira, Daniel Camozzato, Fernando Marson
*Undergraduate Program in Digital Games*
*Universidade do Vale do Rio dos Sinos*
*São Leopoldo, Brazil*
*ivansi@outlook.com, {dcamozzato,fmarson}@unisinos.br*

Leandro Dihl, Soraia Raupp Musse
*Graduate Program in Computer Science*
*Pontifícia Universidade Católica do Rio Grande do Sul*
*Porto Alegre, Brazil*
*leandro.dihl@gmail.com, soraia.musse@pucrs.br*

***Abstract*—This article presents a computational model for procedural generation of customized façade and interior styles of buildings for use in three-dimensional virtual environments of games and simulations. The model makes use of two types of input information: geometric and semantic. The geometric information is related to the two-dimensional floor plan of a building, with its parts and dimensions, as well as positions for doors and windows. The semantic information enables the creation of architectural styles, enabling variations for materials and textures for the façade and inner parts, as well as for the shapes and dimensions of doors and windows. Changing one or more input parameters modifies the final appearance of the result. The proposed computational model can be used to generate large virtual environments, as it allows mixing different floor plans and architectural styles to achieve visual diversity. The main characteristics of the work are the real-time procedural generation of 3D buildings, the customization of façades and building interiors, as well as the use of semantic to assign meaning to the different elements of the house.**

***Keywords*-Procedural Generation; Façades Generation; Virtual Environments;**

## I. Introduction

With the continuous development of computer graphics hardware and software technologies, games are getting more complex and rich in visual details. This causes players to increase demands and expectations, hoping to find something new and interactive with each release. The virtual environments in games have become increasingly large and in open world games are a must. The manual creation of this kind of environment requires effort, consuming resources that could be applied in other areas of the game. Due to this situation, some development companies reduce the possibilities of interaction with the environment, creating scenarios with few details or buildings without any interior.

To automate the process of generating large-scale environments, game development studios have sought to use procedural generation techniques, optimizing content creation and reducing manual modeling work. Most existing techniques for creating procedural constructions require from a few seconds up to minutes to generate models that can be used in games, which provides increased productivity in the creation of large-scale environments. However, the focus is the generation and customization of the exterior of the buildings in these virtual environments and not the interior, as can be seen in Section II.

To solve this problem, this paper proposes the definition and implementation of a computer model capable of generating, in real time, a great visual variety of façades and interiors requiring only two sets of inputs. The input information is provided once and may be combined in order to produce varied results. The required data concerns the geometry and architectural style of the building to be generated. To provide higher visual variety and scalability, it is possible to modularize the elements that are part of the construction (e.g. doors, windows and roofs). The result and performance achieved in the generation of the buildings allow content to be generated on demand even while running a game or a computer simulation.

This paper is organized as follows: the next section discusses the related work, with different techniques for procedural generation of buildings and other components involved. Section III describes the proposed model and its internal processes, such as the set of necessary information and the processing to generate each part of the construction. Section IV discusses details of the implementation of the prototype and the technologies used. The results section analyzes four different cases, including a performance evaluation for mass production of content. Finally, some considerations are made about the work and about future work in Section VI.

## II. Related work

With the increasing use of large-scale environments in open-world games like Sleeping Dogs, GTA IV and V, Watch Dogs and Assassin's Creed, it becomes necessary to allocate resources exclusively for the creation of these environments. The storage of geometric models and textures for large-scale environments is also a challenge. Due to these problems, the demand for procedural content generation has grown steadily. Techniques to generate elements for use in three-dimensional virtual environments are constantly being developed and improved [1]. The possibilities range from generation of organic objects such as plants and trees ( [2] [3] [4] [5]), to geometric representations of buildings ( [6] [7] [8]) and even entire cities ( [9] [10]). Besides geometry, it is also possible to generate procedural textures ( [11] [12] [13] [14]).

[15] presents a method for automated generation of façades via a model which uses a division grammar, which is a parametric approach based on the concept of forms. It also uses a system for comparing forms and a control grammar, creating flexibility when generating different architectural styles and designs.

[16] proposes a grammar called CGA Shapes, focused on the generation of buildings with high visual quality and geometric details. Using rules, it is possible to describe, group and specify relationships for simple geometric shapes, in order to create a geometrically complex object. One of the main contributions of the work [16] is to create a massive system that maintains consistency between the volumetric shapes created. In addition, the user is allowed to dynamically interact at all stages of the process.

In 2007, [17] expands the original model proposed by [15] using the ideas presented in [16], in order to generate synthetic façades from photos of real-world building façades. The generation principle remains the same, but now a photogrammetry search is made in the image for patterns stored in a database. Once recognized, hierarchical patterns are used as input in the division grammar to start the façade reconstruction process. The façade photograph is used as a texture in the generated geometry, further increasing the realism of the result. In addition to the realism and ease of generation, another contribution of the model proposed by [17] is the linking of semantics with geometry, making it possible to specify the functions and uses of a particular geometry feature.

Another image-based approach to generate façades is presented by [18]. This semi-automatic technique uses images captured along streets and relies on structure from motion to recover camera positions and point clouds automatically as the initial stage for modeling. Firstly, a building façade is considered as a flat rectangular plane or a developable surface with an associated texture image composited from the multiple visible images. A façade is then decomposed and structured into a Directed Acyclic Graph of rectilinear elementary patches. The decomposition is carried out top-down by a recursive subdivision, and followed by a bottom-up merging with the detection of the architectural bilateral symmetry and repetitive patterns. Each subdivided patch of the flat façade is augmented with a depth optimized using the 3D points cloud.

[19] introduces a method to generate very detailed façades, presenting his work as an improvement over previous models in which the details are simpler. However, the method presented by [19], despite getting results rich in detail, requires user interaction. In similar way to earlier techniques, the method does not consider the generation or customization of the interior of buildings.

[20] proposes a procedural approach to generate consistent buildings using Bayesian networks, which are generated by machine learning of real architectural data from an extensive catalog coded manually. The focus of the work is to maintain the coherence of the interior room layout. The model allows different architectural styles to be applied through style models, but the parameterization and styles were not detailed and unknown efficiency of both. The model has limited use in real-time applications because the cost to generate each floor plan is too high.

A framework developed by [21] is able to generate many variations of a façade design that look similar to a given façade layout. Starting from an input image, the façade is hierarchically segmented and labeled with a collection of manual and automatic tools. The user can then model constraints that should be maintained in any variation of the input façade design. Subsequently, façade variations are generated for different façade sizes, where multiple variations can be produced for a certain size.

A procedural method for interactively modeling building façades is introduced by [22]. The result is a façade by composing multiple overlapping layers, where each layer contains a single rectilinear grid of façade elements described by two simple generator patterns. This way, the design process becomes more intuitive and the editing effort for complex layouts is significantly reduced. To achieve this, it is presented a method for the automated merging of different layers in the form of a mixed discrete and continuous optimization problem.

It should be noted that none of the presented techniques deals with the real-time procedural customization of both the interior and exterior appearance of buildings. In contrast, this work customizes both the interior and exterior by applying different styles according to semantic rules.

## III. Computational Model

This paper presents a computational model for customized procedural generation of façades and interiors of three-dimensional virtual buildings, using as input a two-dimensional floor plan and semantically defined architectural styles. The main purpose of the model is to facilitate the creation of three-dimensional structures for use in games and computer simulations.

The model can generate a 3D building from a real-world 2D floor plan, using the location of the elements (walls, doors and windows) contained therein. It is also possible to use floor plans generated procedurally as the input (e.g. [23] or [24]).

The method presented here, in addition to the geometric information, also uses information from user-defined architectural styles, which are semantically assigned to a building. Thus it is possible to specify, for example, what kind of material must be applied to the façade, the type of roof and the type of material it is made of, as well as the style of window and door to be used in a particular building.

Figure 1 presents an overview of the building generation process. The Builder Agent interprets the input info (a 2D floor plan and architectural style) and requests to the specialized builders the creation of three-dimensional elements to be inserted in the building (doors, windows and roof). These elements are customized according to architectural style, both in the form and the material used. The Builder Agent is responsible for consolidating the integration of the elements and walls of the building.

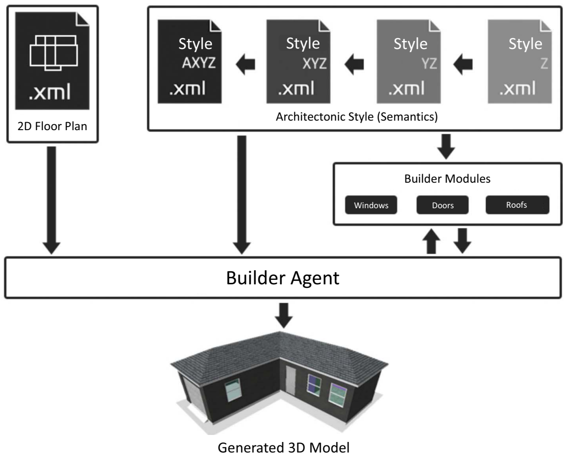

Figure 1. Architecture of the proposed model.

### A. Information input

For the operation of the model, two types of input information are required: geometric (2D floor plan) and semantic (architectural styles). In the following sections these inputs are described, as well as their uses.

*1) Geometric information:* For the three-dimensional geometry of the building to be generated, a minimum set of data needs to be supplied by the user. A list of edges and vertices must be informed to define the interior walls of each room and the exterior walls of the building.

During this step points can also be defined to represent the center of doors and windows. The dimensions of these openings (height and width) are defined as a style characteristic. Rooms and their adjoining walls are given a type (e.g. kitchen, bathroom, living room, etc.), and this information directly influences the application of semantics.

*2) Semantic information:* The second set of user input data concerns the architectural style of each building. It serves to customize the final appearance of the exterior and of each internal room in the house, assigning textures or materials and defining shapes for doors, windows and roofs given the specified style.

The same set of semantic information can generate different results, because a wide range of behaviors can be defined for each item subject to configuration. In addition to the information that can be inserted into an entry, it is also possible to create relationships of inheritance, enabling specialization as well as reuse of inherited settings.

To better understand this concept, we give as an example an architectural style that generates buildings made of bricks, with high ceilings. Given that the building is made of bricks, it can be assumed that it supports the weight of a high ceiling. It is possible to create a new style by inheriting all previous information, but including new definitions to use wood as a material, and also to present low roofs since wooden structures support less weight.

The information defined in a semantic input is as follows:

- *List of materials*: each material is characterized with a type (e.g. *OutsideWall*, *Ceiling*, *Glass*) and a list of room types, this last being used to restrict the types of material used in each room (e.g. Material1 {Type: Ground, RoomType: Kitchen, Bathroom}, in this case Kitchen and Bathroom can use the same *Material1* type of material for the ground).
- *Window styles*: defines dimension information (width and height), opening direction, depths of the opening, room types to which it applies and materials. This option allows overwriting materials in the master list of materials described above (e.g. only bathroom windows may have frosted glass).
- *Door styles*: similar to window styles, kept separate to provide scalability for future work, so that different characteristics can be considered (e.g. doors with a rounded top, sliding doors, doors with glass panes, etc.).
- *Roof styles*: roof types that can be used in the building, defining allowable dimensions, materials and shapes.
- *Builder modules*: used in the construction of specific parts, such as doors, windows and roof. This parameter allows the definition of custom modules to meet the specific needs of each architectural style, as detailed in Section III-C.

A semantic definition is not simply a definition of appearances, it gives meaning to what is being created. It enables informing that a material is more resistant than another, that a specific room type is associated with a characteristic type of window and that a given architectural style requires that the roof be of type *Hipped*. This semantic information can be used in simulations and games, to help guide characters in moving from one room to the other and in performing specific actions. For example, to make a character use the bathroom, it must move to the correct room and this is possible if the room is assigned with a specific type.

### B. Builder Agent

The geometric and semantic information provided by the user are used as an input in the main generation process, which is performed by the Builder Agent. If the creation of buildings with different styles is required, the previous set of semantic rules is cleared using the redraw operation. Otherwise, the same rules are used and the result will retain exactly the same style of the last building generated.

The redraw operation was created to allow the Builder Agent to repeat the generation with the same semantic combinations, allowing the application of the same architectural style to different buildings, as seen in Figure 6. Once the semantic style to be used is defined, the Builder Agent handles other steps such as the construction of exterior and interior walls, as well as the ceiling and the floor, storing the partial results hierarchically starting from a root node representing the building as a whole.

In addition to this processing performed by the Builder Agent itself, some other specific processes are handled by subprocesses handled by builder modules, for example the generation of the openings. Such builder modules are discussed in Section III-C. At the end of every processing cycle, the results of the subprocesses are attached to the main result and this is arranged in a scene as a three dimensional object.

## C. *Builder modules*

Some parts of a building have a greater need for detail and their generation may require specific characteristics, depending on architectural style. To facilitate the management and application of these specific needs of each style, the model makes use of builder modules, which are responsible for generating the different elements of the building. At the planning stage of this work, we identified three types of elements that potentially have this kind of need: windows, doors and the roofs.

If there is a need to use an external module to generate a new element, which was not predicted, it must be informed alongside the semantic data and a new builder module must be defined following the rules set in the implementation section. In the following sections, the three default builder modules are presented.

*1) Roof:* The builder module for roofs used in this work generates three types of roofs: *Hipped*, *Flat* and *Mansard*. The *Hipped* type is built based on the Straight Skeleton algorithm [25]. This algorithm allows the creation of a skeleton calculated according to the contour of a polygon, which in this case is the contour of the building, as seen in Figure 2, calculated on the contour of the floor plans used in Section V. The technique used was based on the optimized generation process proposed by [26].

*2) Window:* The process of procedural generation of windows can be as simple or as complex as desired, because there is a great visual diversity, even in a single architectural style. To demonstrate one possibility as well as keeping the method real-time, we opted for the generation of simple windows, without architectural details like those seen in Figure 4. The window model created by the builder module follows the principles suggested by [27], which define a window as the recursive subdivision of an area in two others. However, up to two levels are considered, with a total of three combinations for the frames of the windows (a single external frame, or two children frames, either vertical or horizontal). If a higher level of customization is necessary, the creation of custom builder modules is possible, generating different models for windows.

[2]Images obtained at http://buildingwatch.blogspot.com.br/2009/11/47-french-provincial.html
and http://responsive-ins.com/wp-content/uploads/2013/02/Hip-Roof.jpg

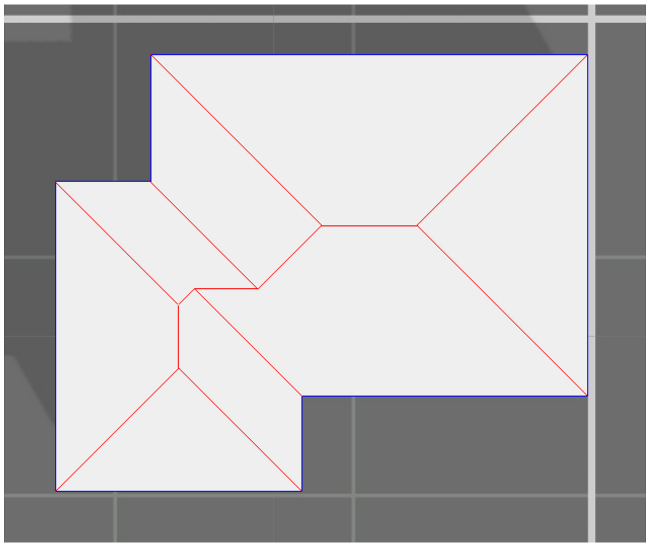
Figure 2. Example of skeleton calculated using the Straight Skeleton algorithm. In blue, the contour of the building, and in red the resulting skeleton.

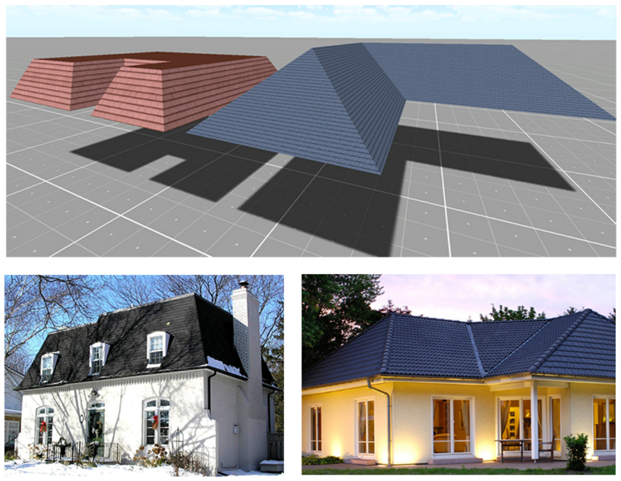
Figure 3. Example of the Mansard and Hipped roof models generated by the builder module and its real world equivalents. [2]

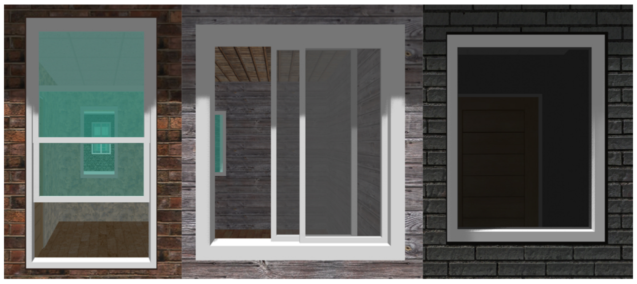

Figure 4. Example of window models generated by the builder module.

*3) Door:* The door models follow a simpler principle than that of window models, considering the external frame and the door, as presented in Figure 5. As in previous modules, for any necessary customization, a custom door builder module can be used.

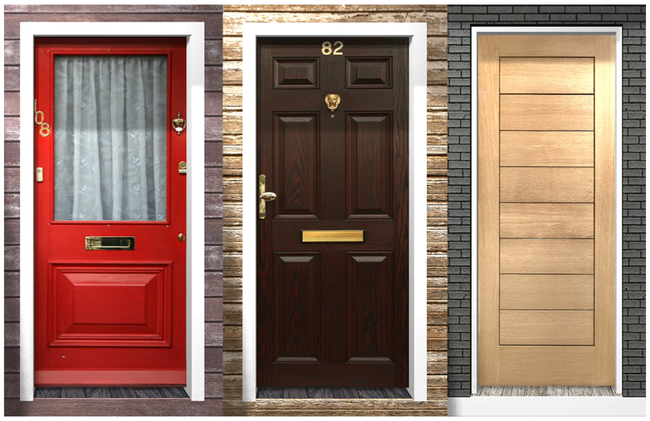

Figure 5. Example of door models generated by the builder module.

## IV. Model implementation

To implement the prototype the game development engine Unity has been used [3]. The choice for a game development engine was made because our method is suited for use in games, but also due to the ease of prototyping. The development environment was integrated with the Builder Agent to reveal some features seen in Section III-B. Among these features are the preloading of style and geometric information files, executing the method using a set of preloaded information and forcing a redefinition of combinations of semantic files before generating a new building.

### *A. Data input*

To acquire the geometric and semantic information provided by the user the XML format has been chosen [4]. The choice for this format is due to the clarity in the hierarchical organization of the data. This characteristic enables automation in the acquisition of the input, facilitating integration with any other source of floor plan files (CAD programs, for example).

*1) Structure of the geometric information:* The geometric information used as input is as follows:

- *Plan* - root node of all the geometric information, representing the floor plan.
- *Points* - contains all the points shared by the edges of each interior or exterior wall.
- *Floor* - the data structure is prepared for the inclusion of multiple floors in a future work, but for now only one floor entity is supported.
- *Wall* - Each floor has a list of walls. The walls are edges formed by the points defined in *Points*, referenced by attributes p1 and p2. To differentiate interior and exterior walls we set *IsInternal = "true"*. As for the exterior walls, the walls of the floor must be defined in sequential order, following either an anti-clockwise or a clockwise direction, forming the perimeter of the floor.
- *Windoor* - the walls can contain a list of openings, which can be configured to behave as a door (*isDoor = "true"*) or a window. This opening instance must contain a point in the 2D space of the floor plan. If this point does not lie upon the parent wall, the opening is ignored. As an alternative, the opening can be centered in the wall using the *centralize = "true"* option, but this is only considered in cases where there is only one opening in the wall.
- *Room>Wall* - different from the walls defined for the floor, the walls defined in a room are references to walls already defined for the floor. To create the list of walls for a room the same standard used for exterior walls must be followed, that is, they must be set in sequential order in either counterclockwise or clockwise direction, forming the outline of the room.

*2) Structure of the semantic information:* The semantic information used as input is as follows:

- *style* - root node of all the semantic information and representative of a style. In this element, the *inherit = "..."* attribute defines whether the style inherits information from another style. Circular references are not allowed in the definition of inheritance and a style without a parent is defined as a child of the default style.
- *Materials* - a list with the definition of all materials used by the style. For each material a type can be defined with the *type* attribute. The types of room in which the material is allowed should be defined in the attribute *RoomTypes*, separated by ','. For the material to function, its definition must contain in the attribute *name* the exact name of the material in the asset library.

[3]http://www.unity3d.com
[4]http://www.w3.org/XML/

- *BuilderModules* - in this entity the modules responsible for the construction of doors, windows and roof are defined. In the attribute *builderType* one of the three types must be specified and in the attribute *assembly* the full path to the script implementing the module.
- *FloorStyle* - defines general styles for the floor. The attributes *minGroundOffset* and *minTopOffset* let you set a minimum distance that the opening must have to the top and bottom of the floor.
- *WindowStyle* - a list of specific styles of windows in rooms. Each window style must contain at least one room type for which it is intended and optionally a list of allowable materials, which overwrites materials of the same type defined at the root of a style. For each window style the type of decorative frame can be configured using the *decorationFrameType* attribute, however the standard window builder module prototype only supports two types, the *None* and *SimpleBordered* types.
- *DoorStyle* - defined the same way as *WindowStyle*, except using unique properties of each.
- *RoofStyles* - in this element the types of roof available and their properties are defined. Some properties are common to all roof types, while others are specific to a roof type. For example, the attributes *minOuterOffset, maxOuterOffset* and *minInnerOffset, maxInnerOffset* define the length of the roof extending outward from the wall, and from the top to the base, respectively. These attributes are consumed by the types of roof *Mansard* and *Hipped*, for which they are meant, but ignored by other types of roof.

### B. *Builder modules*

To make modularization possible, an interface was created to define the method *Build* used by the Builder Agent. Every module developed must implement the interface *IBuilderModule* present in the source code of the prototype, otherwise it is impossible for the agent to create an instance of the module. Once the interface has been implemented, the module must be defined in the styles for which it is used. The module definition is performed in the *BuilderModules* element of the semantics file.

## V. Results

To evaluate the functionality and efficiency of the implementation, four floor plans were created, each in its respective XML file, and two style files which were inherited from the default semantic file and applied as semantic input.

With these input information defined, four case studies were developed. In the first, a single semantic is applied to four different buildings. In the second case the same four buildings receive multiple semantic variations. In the third, a floor plan receives many variations in style and finally, in the fourth, a performance test is performed to verify the efficiency of implementation.

All tests were run on a computer with the following configuration: Intel Core i5 2500k, 8GB of RAM and a NVidia GeForce GTX550Ti graphics card.

### A. *Case 1*

In the first case study a single predefined style was applied to four different buildings, in order to evaluate if dependencies exist between the semantic and geometric information. As can be seen in Figure 6, the same visual identity has been kept for all houses and there has not been problems in the application of the style. The total time to generate the four houses was 61 ms.

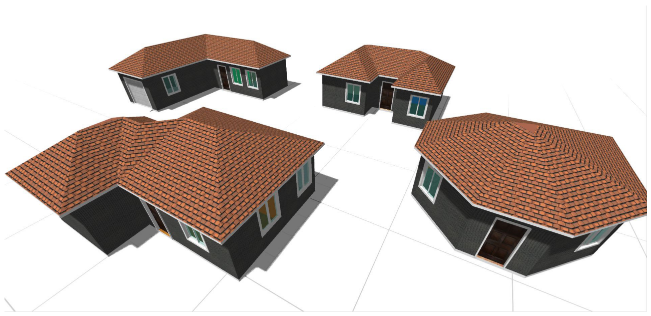

Figure 6. Same semantic applied to four different floor plans.

### B. *Case 2*

For the second case study, prior to generating each building the redraw operation has been performed (see Section III-B), so that every new three-dimensional model was generated with a new semantic variation of the two style files. The result is 4 houses with different styles. Figure 7 presents the result of using different styles with the same geometric information of the previous case. The total time to generate the buildings was 64 ms.

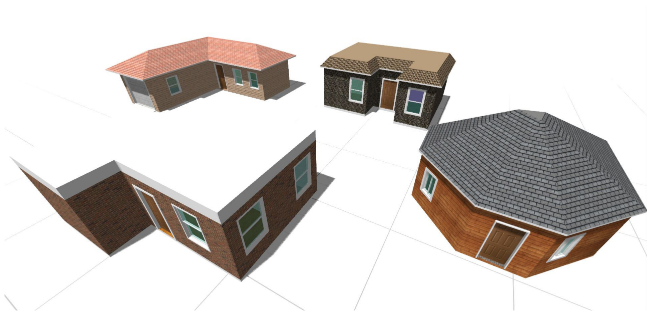

Figure 7. Variations of two semantic styles applied to four different floor plans.

## C. Case 3

In the third case, different styles are applied as variations for a single floor plan, to check the visual consistency in the application of these different styles. The results can be seen in Figures 8, 9 and 10. Examples of interior style variation can be seen in Figure 11. The total time to generate the buildings was 90 ms.

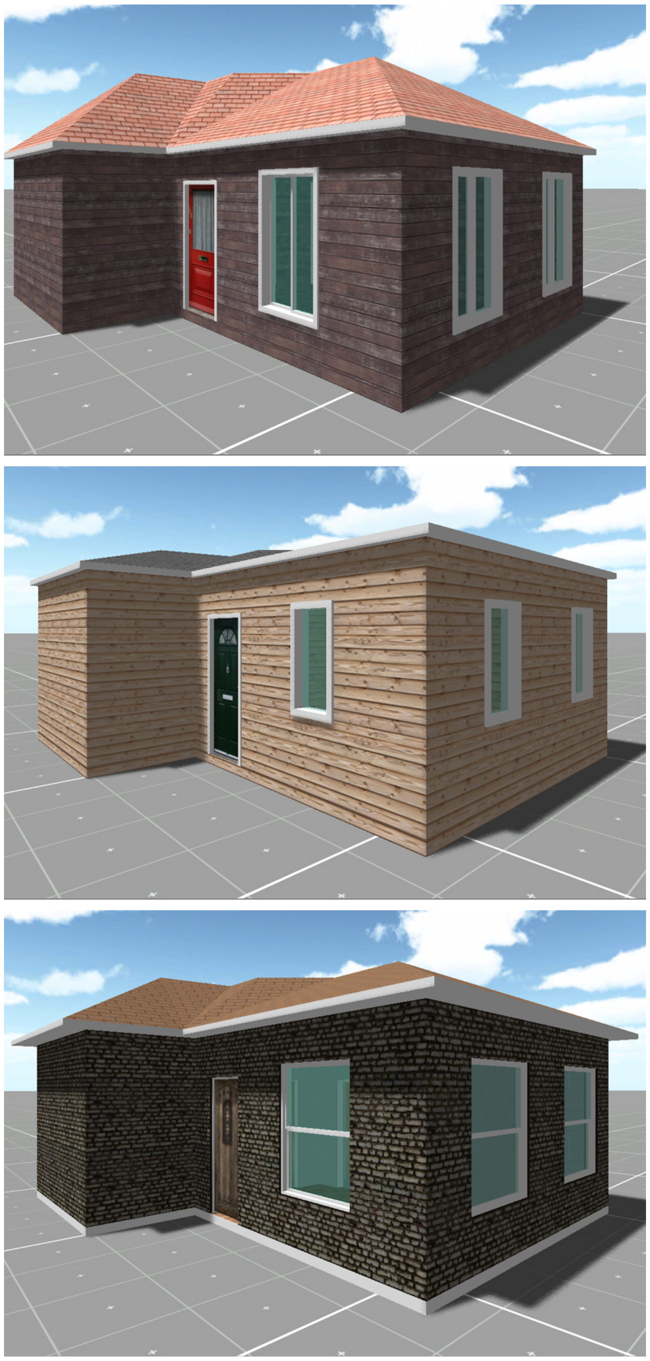

Figure 8. Style variations applied the same floor plan.

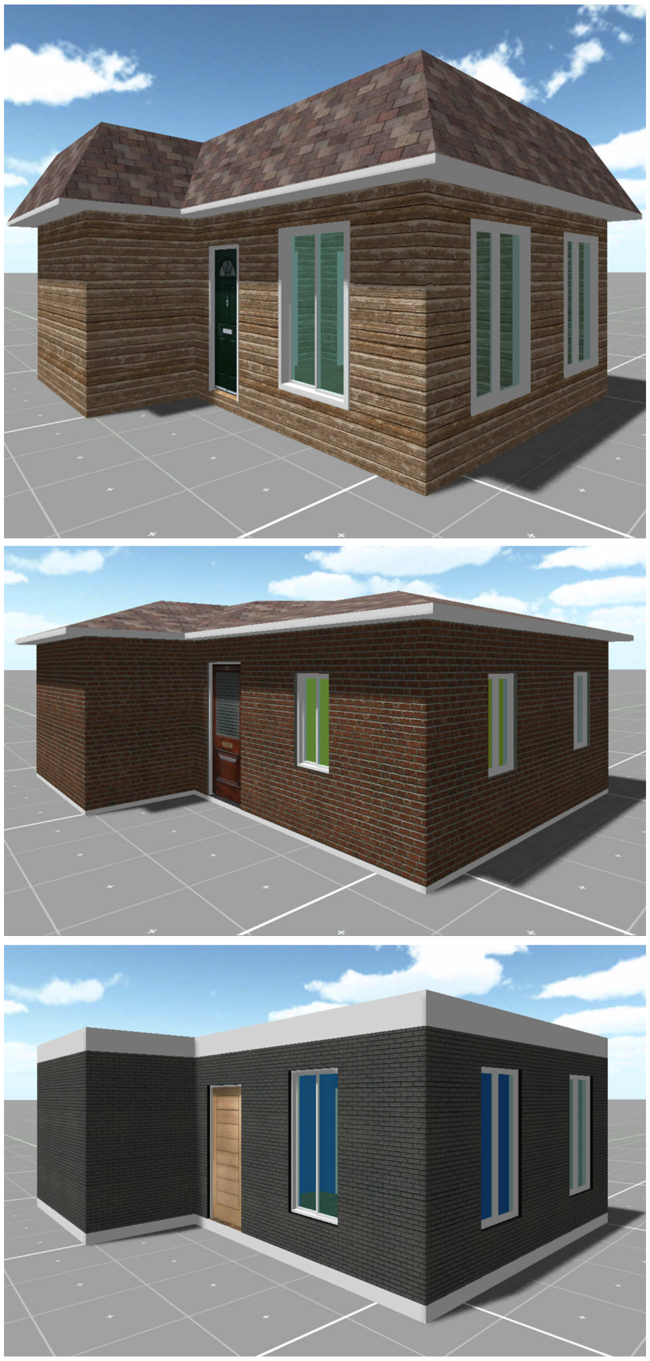

Figure 9. Style variations applied the same floor plan.

## D. Case 4

The fourth case study was generated in order to demonstrate the model's performance in the massive generation of buildings. For this case the same settings of the previous cases have been used, the only difference being the number of buildings which have been generated, 100 building distributed in a grid of 10 x 10 units. The 100 buildings were

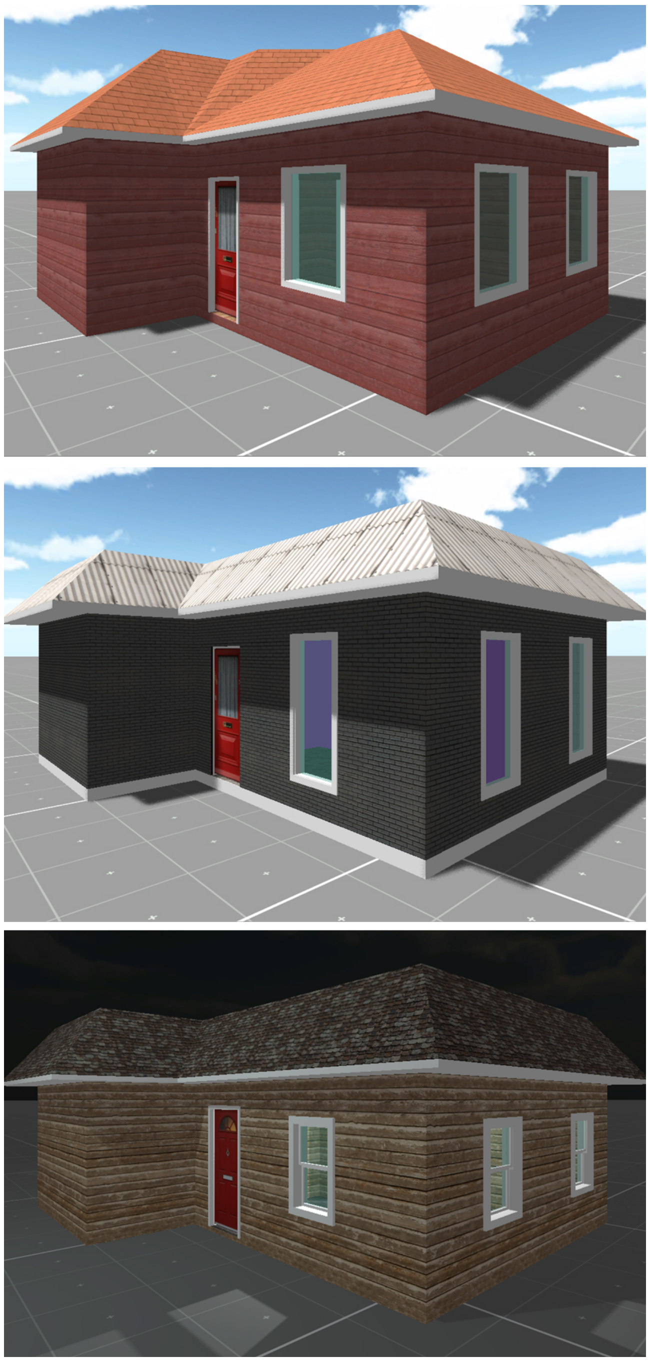

Figure 10. Style variations applied the same floor plan.

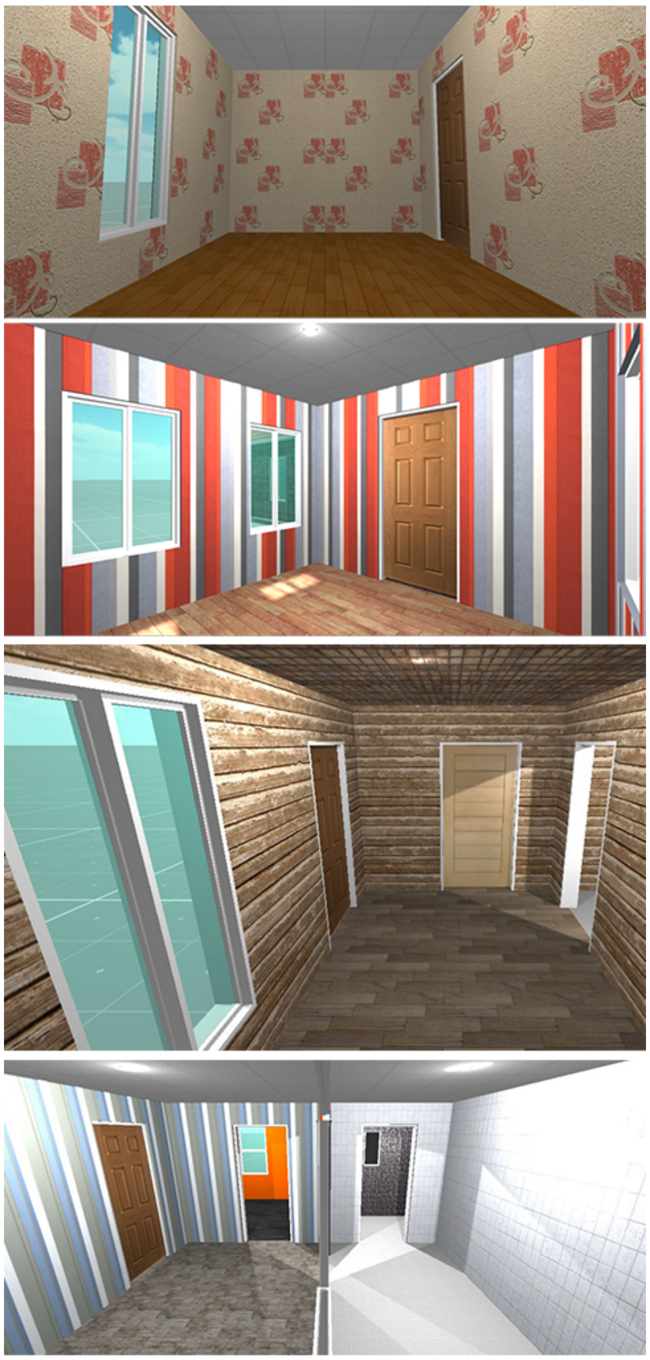

Figure 11. Example of interiors generated with style variations.

generated in 4.7 seconds. Figure 12 showcases the diversity obtained with four floor plans and two style files.

## VI. Final Remarks and Future Work

In this paper we present a computational model for procedural generation of customized façades and interior for buildings for use in three-dimensional virtual environments in games and simulation. The generation process uses the geometric specifications of a floor plan and a semantic specification of the visual styles and features of each building element. A builder agent is in charge of interpreting the input data and requesting from specialized builder modules the generation of different items, such as doors, windows and roof, which are integrated into a virtual building. Using architectural style configuration files, different looks can be

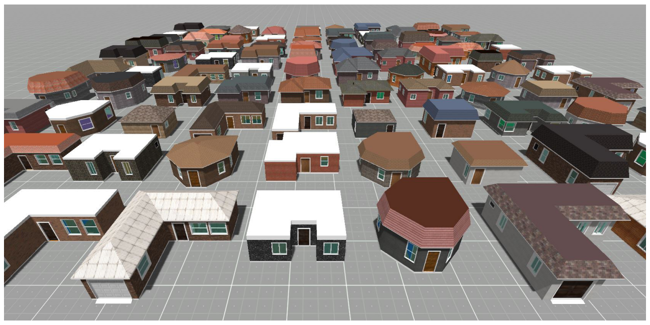

Figure 12. A hundred buildings generated from four floor plans and two style files.

generated for the same geometry. The main contributions of the proposed method are:

- Automating the process of creating three-dimensional virtual buildings for use in games and simulations;
- A semantic format to specify architectural styles and building features;
- Generating style variations for the same building in real-time;
- Possibility to expand the model using different specialized builder modules.

For future work, some improvements can be made in order to raise the level of detail of the generated models. One possibility is the implementation of the method proposed by [19], adding the possibility to generate curves and ornaments which are characteristic of some architectural styles.

An issue to be handled is the creation of buildings with multiple floors, enabling the creation of apartment buildings or houses with multiple floors. This requires the creation of a builder module to generate stairs and elevators.

## REFERENCES

[1] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, pp. 1:1–1:22, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2422956.2422957

[2] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 275–286. [Online]. Available: http://doi.acm.org/10.1145/280814.280898

[3] J. Lluch, E. Camahort, and R. Vivó, "Procedural multiresolution for plant and tree rendering," in *AFRIGRAPH '03: Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*. New York, NY, USA: ACM, 2003, pp. 31–38.

[4] K. Onishi, S. Hasuike, Y. Kitamura, and F. Kishino, "Interactive modeling of trees by using growth simulation," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '03. New York, NY, USA: ACM, 2003, pp. 66–72. [Online]. Available: http://doi.acm.org/10.1145/1008653.1008667

[5] L. Streit, P. Federl, and M. Sousa, "Modelling plant variation through growth," *Computer Graphics Forum*, vol. 24, no. 3, pp. 497–506, 2005. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2005.00875.x

[6] H. Ali, C. Seifert, N. Jindal, L. Paletta, and G. Paar, "Window detection in facades," *Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on*, pp. 837–842, Sept. 2007.

[7] J. Benner, A. Geiger, and K. Leinemann, "Flexible generation of semantic 3d building models," in *1st International Workshop on Next Generation 3D City Models*, Bonn, 2005.

[8] J. Dollner and H. Buchholz, "Continuous level-of-detail modeling of buildings in 3d city models," in *GIS '05: Proceedings of the 13th annual ACM international workshop on Geographic information systems*. New York, NY, USA: ACM, 2005, pp. 173–181.

[9] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *SIGGRAPH '01: Proceedings of the* $28^{th}$ *Annual Conference on Computer Graphics and Interactive Techniques*. Los Angeles, CA, USA: ACM, August 2001, pp. 301–308.

[10] 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*, ser. GRAPHITE '03. New York, NY, USA: ACM, 2003, pp. 87–ff. [Online]. Available: http://doi.acm.org/10.1145/604471.604490

[11] G. Turk, "Texture synthesis on surfaces," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 347–354. [Online]. Available: http://doi.acm.org/10.1145/383259.383297

[12] L. Lefebvre and P. Poulin, "Analysis and synthesis of structural textures," in *Graphics Interface 2000*, May 2000, pp. 77–86.

[13] B. Chan and M. McCool, "Worley cellular textures in sh," in *SIGGRAPH '04: ACM SIGGRAPH 2004 Posters*. New York, NY, USA: ACM, 2004, p. 18.

[14] J. Dorsey, A. Edelman, H. W. Jensen, J. Legakis, and H. K. Pedersen, "Modeling and rendering of weathered stone," in *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*. New York, NY, USA: ACM, 2005, p. 4.

[15] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," in *ACM SIGGRAPH 2003 Papers*, ser. SIGGRAPH '03. New York, NY, USA: ACM, 2003, pp. 669–677. [Online]. Available: http://doi.acm.org/10.1145/1201775.882324

[16] P. Muller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," in *ACM SIGGRAPH 2006 Papers*, ser. SIGGRAPH '06. New York, NY, USA: ACM, 2006, pp. 614–623. [Online]. Available: http://doi.acm.org/10.1145/1179352.1141931

[17] P. Muller, G. Zeng, P. Wonka, and L. V. Gool, "Image-based procedural modeling of facades," *ACM Trans. Graph.*, vol. 26, no. 3, p. 85, 2007.

[18] J. Xiao, T. Fang, P. Tan, P. Zhao, E. Ofek, and L. Quan, "Image-based faade modeling," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 161:1–161:10, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1409060.1409114

[19] D. Finkenzeller, "Detailed building facades," *IEEE Comput. Graph. Appl.*, vol. 28, no. 3, pp. 58–66, May 2008. [Online]. Available: http://dx.doi.org/10.1109/MCG.2008.50

[20] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," in *ACM SIGGRAPH Asia 2010 Papers*, ser. SIGGRAPH ASIA '10. New York, NY, USA: ACM, 2010, pp. 181:1–181:12. [Online]. Available: http://doi.acm.org/10.1145/1866158.1866203

[21] F. Bao, M. Schwarz, and P. Wonka, "Procedural facade variations from a single layout," *ACM Trans. Graph.*, vol. 32, no. 1, pp. 8:1–8:13, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2421636.2421644

[22] M. Ilcik, P. Musialski, T. Auzinger, and M. Wimmer, "Layer-based procedural design of facades," *Computer Graphics Forum*, vol. 34, no. 2, pp. 205–216, May 2015.

[23] F. Marson and S. R. Musse, "Automatic generation of floor plans based on squarified treemaps algorithm," *IJCGT International Journal on Computers Games Technology*, vol. 2010, pp. 1–10, January 2010.

[24] D. Camozzato, L. Dihl, I. Silveira, F. Marson, and S. Musse, "Procedural floor plan generation from building sketches," *Vis. Comput.*, vol. 31, pp. 753–763, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00371-015-1102-2

[25] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Grtner, "A novel type of skeleton for polygons," 1995.

[26] P. Felkel and S. Obdrzalek, "Straight skeleton implementation," in *Proceedings of Spring Conference on Computer Graphics*, 1998, pp. 210–218.

[27] P. J. Birch, S. P. Browne, V. J. Jennings, A. M. Day, and D. B. Arnold, "Rapid procedural-modelling of architectural structures," in *Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage*, ser. VAST '01. New York, NY, USA: ACM, 2001, pp. 187–196. [Online]. Available: http://doi.acm.org/10.1145/584993.585023