An Approach for Constraint Based Heuristic Method of Generating Houses and Building Blueprints for Real-time Applications

Daniel Sanchez, Elio Lozano, and Juan M. Solá-Sloan

***Abstract*—Most city-generation computer graphic algorithms are focused on creating exteriors by extruding some random shapes. The problem with this approach is that the interiors are ignored. This paper presents a series of algorithms and heuristics to generate building blueprints of single or multi-floor houses. The type and cost of the building establish the lot space available for it. In the case of a single floor, rooms with their corresponding doors and windows will be placed on the space randomly by following a set of conditions that dictate where the rooms can be placed with respect to each other. In the case of multiple floors, the first floor contains public rooms only; the upper floors contain private rooms. The algorithm's output consists of a two-dimensional drawing of the floor plans and a three-dimensional model of the generated house. The experiments show that this algorithm can generate a large number of houses in a short period of time.**

***Index Terms*—blueprint, automatic interior design, procedural generation**

I. Introduction

Today's hardware has the capability to create virtual environments at run time. Procedural generation is a method used for generating artificial environments at run time, such as structures, buildings and parts of cities, which are used in computer games and animated movies[1]. An alternative to procedural generation consists of hiring artists that design the environment. Usually this environment follows an architectural style based on the taste of the artist. However, this method is time consuming. Artists have to modify the architectural style every time it is required [2]. In contrast, procedural generation uses fewer resources than applications that store all pre-modeled content in memory [3]. There are games that already use this approach such as *The Elder Scrolls, Oblivion* [4] and *Borderlands* [5].

During the last decade, many researchers have focused on procedural generation [6], [7], [1], [3]. In [8], [9], Martin proposed an algorithm to dynamically generate plans for single-level houses using procedural generation. This research is focused in extending Martin's work. Various algorithms for generating basic floor plans for single- and multi-level houses were created. First, the Non-Specific Room Placement (NSRP) algorithm places rooms without any constraints on the type of room to be included in the blueprint. Second, the NSRP was extended to include the architectural patterns presented by Alexander in [10]. This version of the algorithm was called the Specific Room Placement (SRP) algorithm. These algorithms are not primarily focused on constraint optimization [11], however, they implement heuristic methods to determine the placement of rooms. In addition, these algorithms have the capabilities to place windows and doors. Another set of algorithms that uses extrusion of a 2D blueprint has been included for generating the 3D respresentation of the single- and multi-level houses. Tests were conducted for the various placement algorithms varying the type of house/building, the number of houses/buildings, and the number of floors. Finally, ideas that extend this research to generate different architectural structures have been proposed. These structures vary from houses to buildings.

This article is structured as follows. Section II contains works related to research carried out. Section III presents the algorithms used to generate the floor plans for single and multiple story structures. Section IV shows the results for the different floor plan algorithms. The main conclusions and future directions of this research are presented in the final section.

II. Related Previous Work

Procedural generation is not new. Others have proposed methods for the generation of content for computer games and movies.

Lindenmayer and Prusinkiewicz [12], [13] introduced L-systems which are formal grammar used to generate building and cities dynamically. This grammar was used in [14]. However, this is not the only work that uses grammars. Shape grammars are used in [15], split grammars in [16] and a database of grammar rules in [17].

In [8], [9], Martin shows methods to procedurally generate single-level houses based on architectural patterns presented by Alexander in [10]. These architectural patterns (or rules) advise how rooms, doors and windows are organized. Among these rules are: lot space available must not be completely filled by rooms; the bedrooms need windows and should contain an outer wall; a bedroom must contain two set of windows, hallways and staircase placement. These rules were included in the development of the algorithm.

There are other classical methods used to generate architectural structures at run-time. Extrusion of 2-dimensional polygons is another technique used to generate three dimensional models of terrains [2] and buildings [6]. In [6], [7], the building

Daniel Sanchez is with Polytechnic University of Puerto Rico
Elio Lozano and Juan Sola-Sloan are with University of Puerto Rico at Bayamón, 170 Road # 174, Parque Industrial Minillas, Bayamón, Puerto Rico

DOI: 10.5176_2010-2283_2.1.154

facade is produced by choosing a random polygonal shape and extruding it. An alternative method is using systems and agent-based simulations to produce virtual cities [14], [18], [19], [6], [20], [21].

III. Blueprint Generation of Houses

An initial configuration is ran before generating a blueprint for all the algorithms. Figure 1 presents the flowchart of the blueprint generation process. First, the house size, cost range and number of rooms are chosen. The house cost is chosen at random, but this parameter determines the number of rooms. Table I shows the relationship between the house size and the cost. After the cost is set, the dimensions of the lot are calculated based on the house size and cost. Then a starting point is chosen at the bottom part of the lot, and rooms are ready to be placed.

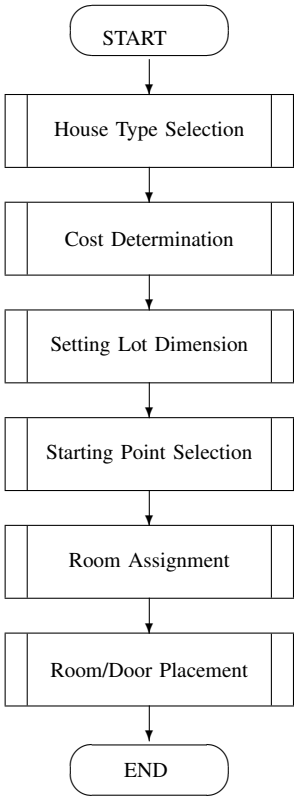


Fig. 1. House Blueprint Development Flowchart

House Size	Number of Rooms	Cost Range
Small	2	10,000 - 25,000
	3	25,000 - 50,000
	4	50,000 - 75,000
Medium	5	75,000 - 125,000
	6	125,000 - 180,000
	7	180,000 - 250,000
Large	Over 10	Over 250,000

TABLE I
House Size criteria

A. Non-Specific Room Placement

The first version of the placement algorithm is the NSRP algorithm for single-floor houses. On this algorithm, all rooms to be placed, except the front room, are symmetric. There is no specification of the type of room to be placed (i.e. bathroom, bedrooms, etc). The first step is to determine where the front room is going to be placed. A random starting point is chosen at the bottom most part of the lot[1]. Once the front room is placed, the algorithm determines the available space to the left, right, and top directions of the front room.

The remaining rooms are treated differently. These rooms are represented on the 2D blueprint by rectangular shapes of the same size. The area that constitutes a room is bound by its walls. The leftmost wall is the leftmost x coordinate of the room and the bottom wall the smallest y coordinate. These coordinates are used to determine if the room can fit in the remaining lot space.

It is very important to know where to start placing rooms. If the only room that is placed on the blueprint is the front room, this room is the *base* room or *calling room*. However, any of the rooms already placed on the blueprint could be a *calling room*. The *calling room* is an already placed room in which all of the possible rooms to be placed could be connected to. Let us call a room to be placed on the blueprint a *candidate room*.

The algorithm uses the coordinates of the *candidate room* to determine if it can fit on the remaining lot space on the right, left or top of a *calling room*. Figure 2 shows the pseudo code for the algorithm. Notice also that a *candidate room* cannot intersect a room that has already been placed on the blueprint. However, the NSRP implementation verifies if the candidate room can fit in the available space by modifying its dimensions. Moreover, the candidate room is not added if it overlaps with another room that has been already placed on the blueprint.

```
foreach calling room of the house
 spaceRight = lotRight - roomRight;
 spaceLeft = roomLeft - lotLeft;
 spaceTop = lotTop - roomTop;

 while (candidate rooms to be placed)
   if room can be placed to the right
     if room does not intersect other
       place room to right;
   else if room can be placed to the left
     if room does not intersect other
       place room to left
   else if room can be placed to the top
     if room does not intersect other
       place room to top
   else
     break
 end while
 make the next room a calling room
```

Fig. 2. Pseudo Code for the NSRP Algorithm

The generated blueprints are *organically grown*. This means that the rooms are generated based on the available lot space. Every house is build based on the lot's size. Therefore the size

[1] on a two-dimensional plane

of the house is determined by the number of rooms that can be placed on the lot.

B. Adding Doors and Windows

The Add Doors Algorithm (ADA) is integrated to the placement algorithms. This algorithm determines where the doors between the rooms must be located. It verifies which wall rooms have in common. Then, the specified coordinates of the door are calculated. If one of the rooms is smaller than the other, then the door is placed at the center of the wall of the smallest room. The door can also be placed randomly[2]. The Add Windows Algorithm (AWA) begins after all the rooms have been placed on the blueprint. The algorithm iterates room by room and places a window in the rooms which have an outer wall. A pseudo code fragment is presented in Figure 3.

```
foreach room of the house
 if no rooms connect to the right
  x = rightwall
  y = midpoint(bottom,top);
  if no other room conflicts with window
   draw window
  if no rooms connect to the left
   x = leftwall
   y = midpoint(bottom,top);
   if no other room conflicts with window
     draw window
  if no rooms connect to the top
   x = midpoint(left,right);
   y = topwall
   if no other room conflicts with window
     draw window
```

Fig. 3. Pseudo Code for the AWA

Figure 4 shows the floor plan generated by the NSRP with ADA/AWA algorithm. The rooms are placed without any constraints. Notice that the room size is the same for all rooms except for the front room of the house. Finally, we have a blueprint that can be extruded to generate a 3D representation. However, more realism can be added to the house by changing features of some of the rooms that are placed on the blueprint.

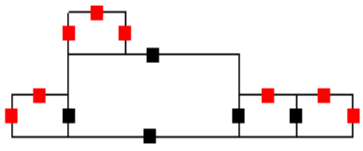


Fig. 4. Blueprint generated NSRP/ADA/AWA

C. Specific Room Placement Algorithm

The SRP algorithm is an evolution of the NSRP algorithm. This algorithm follows the rules discussed in Martin and Alexander's work [10]. The placement algorithm uses these rules for classifying the rooms in two different types: public and private. Public rooms can be accessed by multiple persons at any time (i.e. living room, kitchen, and family room). On the other hand, private rooms can only be accessed by a particular group of persons and, in some cases, by only one person at a time (i.e. bathrooms, bedrooms, and personal offices). The algorithm assigns a priority level to each public and private room. Table II presents a list of the room types and their priorities. The number of private and public rooms is divided almost evenly for small and medium houses. However, this is not true for large houses that contain over 10 rooms.

Type	Room	Priority
Private Rooms	Bathroom	1
	Bedroom	2
	Master Bedroom	3
	Office	4
	Master bathroom	5
	Walk-in closet	6
Public Rooms	Living room	1
	Family room	2
	Kitchen	3
	Dinning room	4
	Hallway	5
	Garage	6
	Laundry Room	7
	Pantry	8
	Stairway	9
	Sun Room	10
	Storage Room	11

TABLE II
PUBLIC AND PRIVATE TYPES OF ROOMS AND THEIR PRIORITY LEVELS

The SRP algorithm follows the same flowchart as the NSRP for its initialization (see Figure 1). The difference is noticed after placing the front room. Now this room is classified as the living room. Notice that the living room is a public room. The SRP algorithm chooses a candidate room based on its type and the type of the candidate room[3]. Notice that this is done before determining if a candidate room can fit next to the calling room in a specific direction. A candidate room must first be compatible with the calling room type to be placed in the blueprint.

There are some room types that do not allow for another room to be placed next to them. Among these are offices, bathrooms, laundry rooms and bedrooms. A private room cannot be connected via another private room except for the master bathroom. A room type cannot be placed multiple times on the same blueprint except for bathrooms and non-master bedrooms.

The SRP algorithm is modified to alter the room dimensions. The shape of each room remains rectangular by default; however, there are some situations in which these dimensions are modified. On the SRP algorithm each type of room accounts for a different percentage of the lot. Room dimensions may change depending on the direction in which the room is placed to help it fit inside the available space. Since these dimensions can be changed, then, the shape of the room can be modified to form a polygon. This means that some "dents" from other room walls can *invade* the candidate room area. These "dents" are known as intermediate walls in this context . Intermediate walls are walls from previously

[2]for the visual representation, the door is added on the center of the wall

[3]private or public

placed rooms that are protruding into the newly placed room. The SRP algorithm uses the intermediate walls to modify the candidate room dimensions. Then the candidate room is placed on the blueprint.

Another functionality of the SRP algorithm is the ability to Add Closets (AC) to some room types. The algorithm that places closets runs after all the rooms have been placed on the lot. It is important to notice that the closet door does not conflict with any room door. Figure 5 illustrates a floor plan generated using the room, door, window, and closet placement of the SRP algorithm. This figure shows the different types of rooms indicated by its initials. Notice that the floor plan generated has rooms with different sizes. The closets are represented with dashed lines.

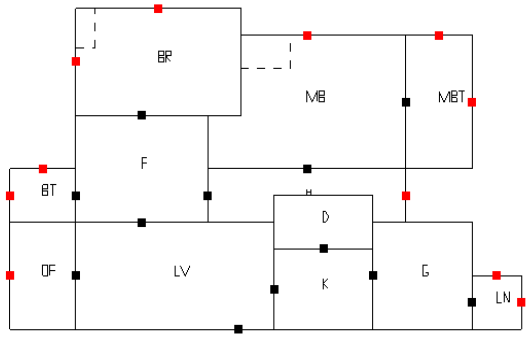


Fig. 5. Blueprint Generated Using the SRP/ADA/AWA/AC Algorithm

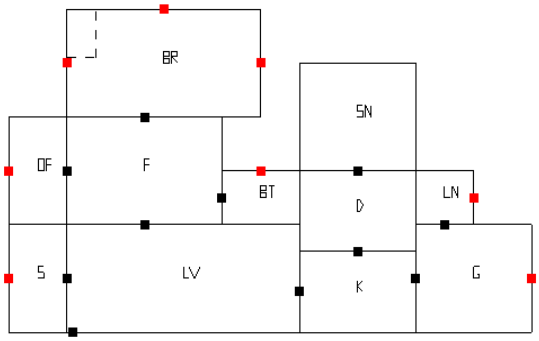
More realism has been added to the NSRP algorithm by adding room types. Moreover, the SRP algorithm has been modified to include closets and intermediate walls. This 2D blueprint can be extruded to form a 3D representation. However, multiple floor blueprints can be generated to add more realism to the algorithm.

D. Procedural Generation of Blueprints for Multiple Floor Structures

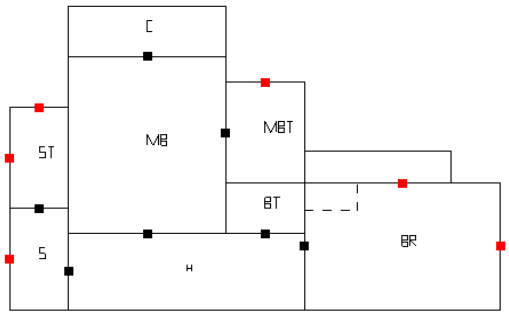
A modified version of the SRP algorithm was used to generate blueprints of multiple level houses. This version was called the Multiple-Floor Placement (MFP) algorithm. The MFP is used to generate houses that include more than 10 rooms in its blueprint . All other structures that do not fit this criterion are considered single-floor houses. Most of the public rooms are placed on the first floor for the MFP. Most of the private rooms are located on the second floor.

A few requirements emerge when dealing with multiple-level houses. First, the second floor boundaries need to be determined. This is ascertained by obtaining the house area within the lot. The algorithm constructs the second floor by using the edges of the first floor as its new limits. Therefore, rooms are placed on top of the outline of the first floor. Another requirement is that a multiple floor structure needs a stairway. This stairway is placed adjacent to the living room of the first floor. A stairway is considered as a room. The position of the stairway room on the second floor has to match the same position as the stairway room of the first floor. After placing the stairway on the second floor, the hallway is placed. The hallway is used as the first calling room when the algorithm starts to select candidate rooms to be placed on the second floor. The ADA, AWA, and AC follow the same rules as on the first floor.

1) Walk-in Closets and Balconies: The ability to add walk-in closets and balconies are features added to the MFP algorithm. The master bedroom is the only one that includes a walk-in closet. This walk-in closet is an additional room as can be seen in Figure 6. The MFP algorithm is capable of adding balconies to the master and regular bedrooms. Bedrooms can have balconies if they are attached to one of their outer walls. A balcony can only be added if it does not interfere with the placement of other rooms.



(a)



(b)

Fig. 6. (a) First Floor Blueprint. (b) Second Floor Blueprint

E. Three Dimensional Representation

The Three-Dimensional Placement (3DP) algorithm uses the SRP and extrusion to form a three dimensional representation of the structure represented on the blueprint. The first step is running the algorithm that generates the blueprint. The first floor of multiple-floor houses is the first one generated by extrusion. Then, the algorithm similarly generates the second floor. The algorithm also draws the ceiling[4]. The doors and windows that were placed by the AWA, ADA, and AC algorithms are also drawn by extrusion using their

[4]The only room in which a ceiling is not drawn is in the Sun Room

respective coordinates. Figure 7 presents a 3D representation of the house. Notice that the algorithm could also be used to generate three-dimensional representations of city blocks. However, more specific rules for building and road placement are needed for the representation of a city block.

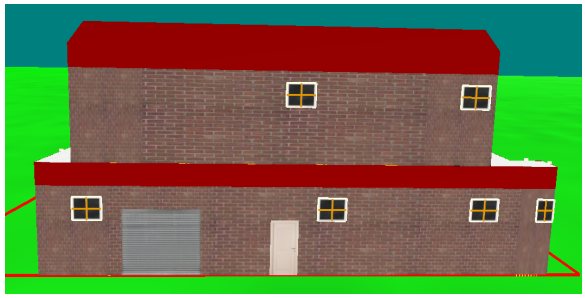


Fig. 7. Three Dimensional Representation of the House

Finally, multiple buildings are generated using previous algorithm. Figure 8 shows the result of this approach to generate a virtual city environment.

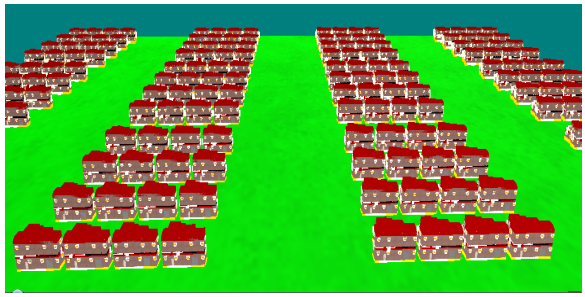


Fig. 8. Three Dimensional Virtual City

IV. Performance Analysis

A. Research Testbed

The implementation was achieved using Visual C++ 2005 with OpenGL and GLUT libraries. The hardware consists of an Intel's q6600 quad-core processor with 4GB of RAM. The video card consisted of an NVIDIA's GeForce 8800 GTS 512MB. The operating system of the host computer was Windows XP Professional.

The experiment consisted of two types of tests. The Five-Minute Test (FMT) consisted of running the program for five minutes to find out the number of houses that could be generated. The second test was the Set House Test (SHT) that consisted in determining the time that it takes the algorithm to generate a predetermined number of houses. This test was run from 1 to 10000 in increments of 1000. The SHT is similar to the test run at [8]. All of the houses generated using these tests were created randomly based on the house type.

Table III presents the results for the FMT test for all of the configurations. The number of rooms is specified after the algorithm name with a dash and the number of floors follows with a slash. Notice that the SRP algorithm generates more blueprints than the NSRP. This is due to the fact that the SRP places rooms based on its room type. The set of public and the set of private rooms are subsets of the set that includes all the rooms. The NSRP does not have that distinction. The difference in the number of houses generated from the MFP algorithm and the largest house of the SRP algorithm was 323,873 houses, 35% fewer houses generated. Overall, the 3DP algorithm generates approximately 75% fewer houses than the previous ones.

Algorithm	Num. Of Houses per second
SRP-10/1	3,376
NSRP-10/1	3,318
MFP-19/2	1,986
3DP-12/1	711
3DP-19/2	553

TABLE III
Placement Algorithms in FMT test levels

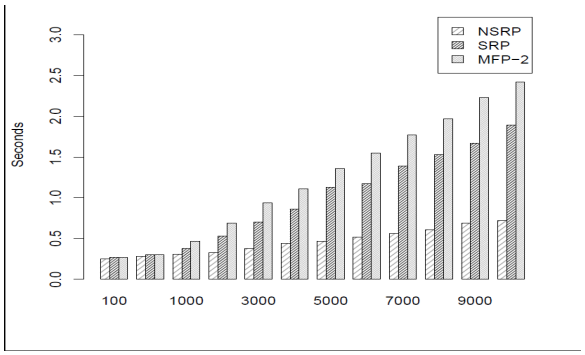


Fig. 9. SHT for three configurations of the 2D Houses

The second test performed was the SHT. Figure 9 presents the results for the NSRP, SRP in 2D and the MFP for 2 floors in 2D. Notice that the SRP and the MFP are slower than the NSRP when generating more than 1000 houses. Figure 10 presents the SHT for 3D blueprints of 1- and 10-story buildings. Notice that the 3DP-10 algorithm generated a city of 10,000 buildings in approximately 84 seconds.

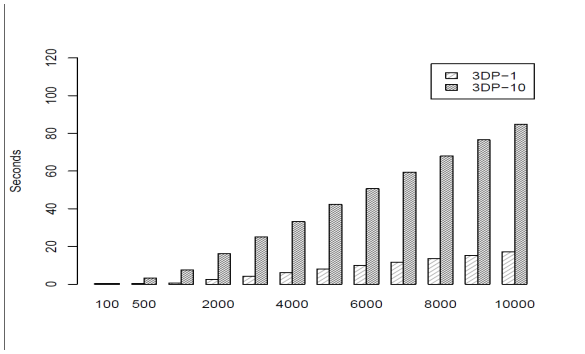


Fig. 10. SHT for 3D Houses of 1 and 10 floors

Figure 11 shows the performance results for different types of building-generation algorithms.

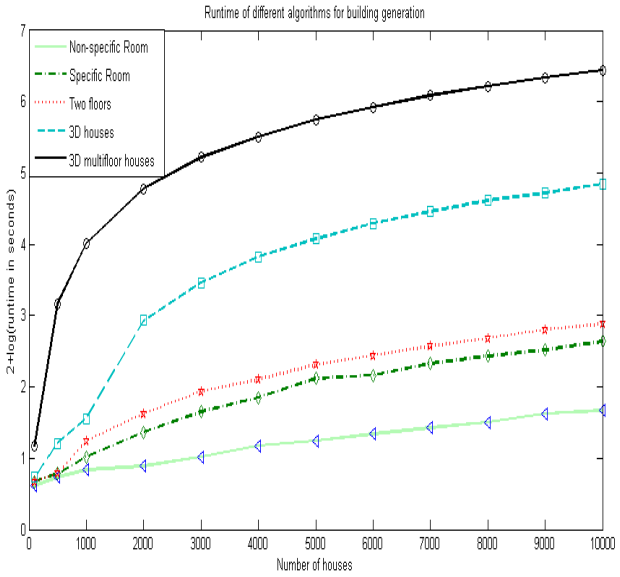


Fig. 11. Performance results for different types of algorithms for building generation

V. Conclusions

The algorithms presented on this article generate 2D and 3D representations of single and multiple floor houses. One of the proposed requirements for these algorithms was that they should be able to generate houses for real-time applications. The 3DP algorithm generates 13,000 houses with ten floors in approximately 2 minutes. Also, this algorithm generates 1,000 ten-floor houses in 4.17 seconds. The SRP algorithm generates 10,000 blueprints in 3.268 seconds. The results obtained from the NSRP, SRP, MFP and 3DP algorithms are similar to those obtained from previous investigations. This results are acceptable for real-time applications [5].

The average loading times for current games vary from a few seconds up to half a minute. All of the placement algorithms presented except for 10 floor 3D houses can generate up to 10,000 houses within that time frame. Therefore, the loading time for the placement algorithms is suitable for these applications.

VI. Future Work

The algorithms can be extended to include more types of rooms such as study, game rooms, attics, basements and nurseries. Also, the algorithms can be modified to include other types of structures such as office buildings and condominiums. However, more rules must be added to follow the architectural patterns of these types of buildings. Also, cities and neighborhoods could be generated using new architectural rules to establish the location of the buildings.

Our algorithms generates *organically grown* structures. Meanwhile, the performance is heavily dependent on the CPU capabilities to handle polygonal complexity and implementation specifics. Producing a dramatically larger number of houses would be another story and perhaps a good one if we implement the algorithm using GPUs (i.e. CUDA). This is envision as a future work.

[5]Computer games

References

[1] B. G. laugsson, "Procedural Content Generation," *Reykjavík University School of Science and Engineering*, 2006.

[2] T. Roden and I. Parberry, "From Artistry to Automation: A Structured Methodology for Procedural Content Creation," in *Proceedings of the 3rd International Conference on Entertainment Computing*, 2004, pp. 151–156.

[3] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *Institute of Technology Blanchardstown ITB Journal*, vol. 1, no. 14, 2006.

[4] B. S. U. http://www.bethsoft.com/, "The elder scrolls IV: Oblivion," [DVD, DVD-DL, Blu-ray Disc, Download], 2006.

[5] G. S. U. http://www.gearboxsoftware.com/, "Borderlands," [DVD, Blu-ray Disc, Digital Distribution], 2009.

[6] 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 Australia and South East Asia*, ser. GRAPHITE '03. New York, NY, USA: ACM, 2003, pp. 87–94.

[7] S. Greuter and J. Parker, "Undiscovered worlds-towards a framework for real-time," in *In Proceedings of the Fifth International Digital Arts and Culture Conference*, 2003.

[8] J. Martin, "The Algorithmic Beauty of Buildings: Methods for Procedural Building Generation," *Undergraduate Honor's thesis, Trinity University*, 2004.

[9] ——, "Procedural House Generation: A method for dynamically generating floor plans," *Symposium on Interactive 3D Graphics and Games*, 2006.

[10] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. USA: Oxford University Press, 1977.

[11] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. Kraker, "Rule-based layout solving and its application to procedural interior generation," *Proceedings of the CASA workshop on 3D advanced media in gaming and simulation (3AMIGAS), Amsterdam, The Netherlands*, 2009.

[12] A. Lindenmayer, "Mathematical models for cellular interactions in development," *Journal of Theoretical Biology*, vol. 18, no. 1, pp. 280–315, 1968.

[13] P. Prusinkiewicz and A. Lindenmayer, *The algorithmic beauty of plants*. New York, NY, USA: Springer-Verlag New York, Inc., 1990.

[14] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 301–308.

[15] P. Müller, P. Wonka, S. Haegler, A.Ulmer, and L. V. Gool, "Procedural modeling of buildings," *ACM Transaction on Graphics*, vol. 25, no. 3, pp. 614–623, 2006.

[16] G. Stiny, "Introduction to Shape and Shape Grammars," *Environment and Planning B*, vol. 7, no. 3, pp. 343–361, 1980.

[17] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 669–667, 2003.

[18] T. Lechner, B. A. Watson, U. Wilensky, S. Tisue, M. Felsen, A. Moddrell, P. Ren, and C. Brozefsky, "Procedural Modeling of Urban Land Use," North Carolina State University, Department of Computer Science, Tech. Rep. TR-2007-33, 2007.

[19] U. Wilensky, "Netlogo," http://ccl.northwestern.edu/netlogo, 1999.

[20] B. Weber, P. Müller, P. Wonka, and M. Gross, "Interactive geometric simulation of 4d cities," *Eurographics Workshop on 3D Object Retrieval (EG 3DOR'09) in Cooperation with ACM SIGGRAPH Munich, Germany*, vol. 28, no. 2, 2009.

[21] C. A. Vanegas, D. G. Aliaga, B. Beneš, and P. A. Waddell, "Interactive design of urban spaces using geometrical and behavioral modeling," *ACM Transactions on Graphics*, vol. 28, pp. 111:1–111:10, Dec. 2009.

[22] G. Whelan, G. Kelly, and H. McCabe, "Roll your own city," in *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*, ser. DIMEA '08. New York, NY, USA: ACM, 2008, pp. 534–535.

[23] G. Kelly and H. McCabe, "Interactive generation of cities for real-time applications," in *ACM SIGGRAPH 2006 Research posters*, ser. SIGGRAPH '06. New York, NY, USA: ACM, 2006.

Dr. Elio Lozano received B.S. in mathematics at National University of San Antonio Abad del Cusco, Perú in 2000. He received M.S. in Scientific Computing and Ph.D. in Computer and Information Science and Engineering at University of Puerto Rico, Mayagüez Campus in 2003 and 2006 respectively. Dr. Elio Lozano is with the Department of Computer Science, University of Puerto Rico at Bayamón, San Juan, PR, 00959 USA. He has developed several computer graphic programs to visualize high dimensional data. His research interests include computer graphics, data mining, and parallel computing.

Dr. Juan Sola-Sloan received B.S., Computer Science Universidad de Puerto Rico en Bayamón, 1996. He received M.S. in Computer Engineering and Ph.D. in Computing Information Science and Engineering at University of Puerto Rico, Mayagüez Campus in 1998 and 2009 respectively. Dr. Juan Sola-Sloan is with the Department of Computer Science, University of Puerto Rico at Bayamón, San Juan, PR, 00959 USA. His research interests include: Web Server Benchmarks, Data Communications, Performance Evaluation and Modeling, Queueing Theory, Web 2.0 Applications, Computer History.

MS. Daniel Sanchez received B.S., and M.S. in Computer Science in Polytechnic University of Puerto Rico in 2005 and 2009 respectively. MS. Daniel Sanchez is with the Department of Computer Science, Polytechnic University of Puerto Rico P.O. Box 192017 San Juan, P.R. 00919-2017. His research interests include: computer graphics, virtual city generation, first person shutter games.