Procedural House Generation: A method for dynamically generating floor plans

Jess Martin*†
University of North Carolina, Chapel Hill

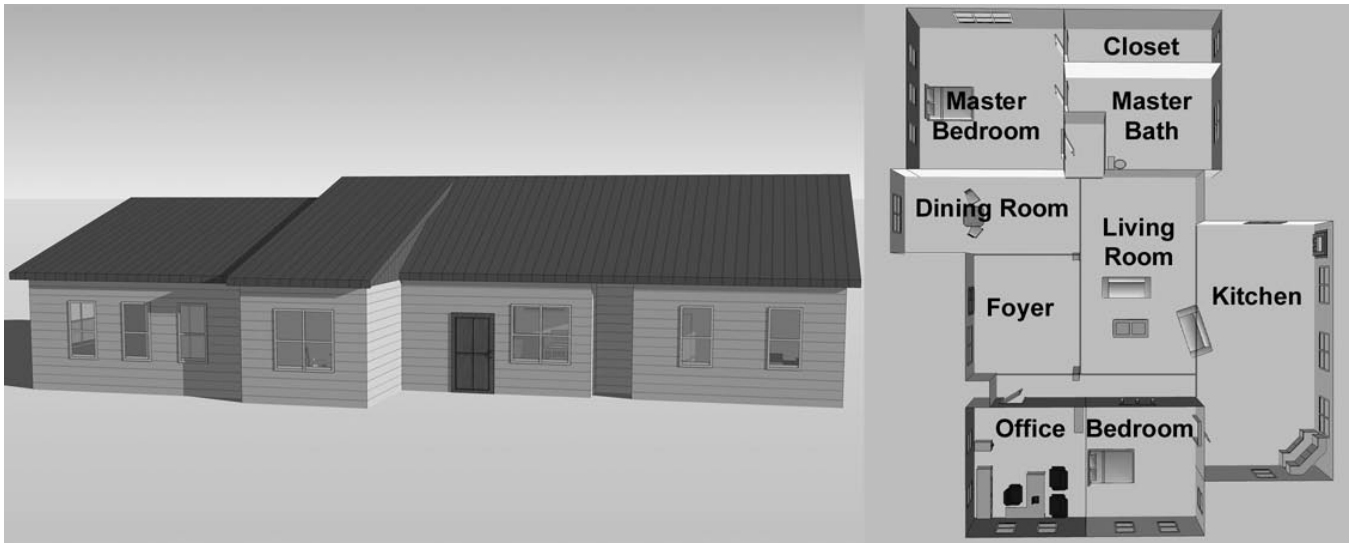

Figure 1: Left: Modeled house based on floor plan generated by algorithm. Right: Top-down view of same floor-plan.

As the capability to render complex, realistic scenes in real time increases, many applications of computer graphics are hitting the problem of manual model creation. In virtual reality, researchers have found that creating and updating static models is a difficult and time-consuming task [Brooks 1999]. New games require massive amounts of content. Creating such content is such an expensive process that it may drive smaller companies out of business [Wright 2005]. Content creators cannot keep up with the increasing demand for many high-quality models; algorithms will have to be called upon to assist.

Procedural methods, algorithms that specify some characteristic of a scene and generate a model [Ebert et al. 2002], are a possible solution to the content creation problem. With such methods, the algorithm, not an artist, determines the detailed attributes of the scene–attributes ranging from color to motion to objects.

Recently, procedural methods have been applied to generating man-made artifacts including cities. Parish and Müller used L-systems to generate a city street map and geometry and textures for buildings within the city [Parish and Müller 2001]. Focusing on city modeling, Greuter et al. describe a procedure to generate buildings based on vertical extrusion of randomly generated polygons [Greuter et al. 2003]. Lechner et al. use an agent-based simulation to generate a city map but do not attempt buildings [Lechner et al. 2003]. Though some of the buildings appear believable, each of these research efforts focused on the realistic creation of the city rather than the buildings themselves. Wonka et al. focus more on the appearance of the buildings using a "split grammar" to control the external appearance so that it conforms to architectural rules [Wonka et al. 2003].

*e-mail: jmartin@cs.unc.edu
†url: http://www.cs.unc.edu/∼eve/research/procedural/

To date, procedurally generated buildings: 1) are merely textured façades, containing no internal structure, 2) represent commercial-style buildings that would be seen in a medium-to-large city, and 3) do not reflect architectural knowledge, with the exception of Wonka et al [2003].

This poster introduces a method for procedurally generating a residential unit (a house) automatically. Combining insights from architecture and graph theory, the primary method used in generating the house is a grammar which constructs a graph of the rooms and connections between rooms. The rules of the grammar are based on architectural observations and are capable of accurately modeling houses in the style from which the rules were drawn. Rooms reach their appropriate sizes using Monte Carlo semi-deformable growth.

Our method of building generation is fundamentally different than methods previously discussed. Most other methods only undertake to generate *exteriors* of buildings with no interior structure or layout. Our method begins with the interior and uses the interior structure to dictate the exterior appearance.Most other methods focus primarily on commercial buildings–office buildings, skyscrapers, and other large, fairly regular buildings. In exploring the problem of procedurally generated buildings, we found residential buildings to be more complex due to their immense variety and irregularity and chose them as the target for our new method. We know of no other published algorithm that does so.

1 Graph Generation

We represent the basic structure of a house as a graph with each node corresponding to a room and each edge corresponding to a connection between rooms. In the first step of the procedure, a

graph encoding of rooms is generated. The graph generation phase itself occurs in four steps.

Two techniques are used in the graph generation phase: a context-free grammar and a user-defined ruleset. We found grammars to be appropriate for "growing" the structure of the graphs, but they are ill-suited to capture the semantics of the graph. To account for the semantics, we used a ruleset to maintain both local and global information about rooms in the graph.

Adding Public Rooms The graph generation phase begins by determining the structure of the public rooms. The front door of the house is added first and production progresses using a context-free grammar. This step is solely responsible for structure, not semantics. Thus, the rooms added are not yet a specific type of room.

Specifying Public Rooms Step two assigns a type (living room, dining room, etc) to each public room by using four statistics for each type of room. The use of these statistics allows the creation of more realistic spaces than using the grammar alone. Additionally, these statistics can easily be modified by a user who would like finer control over the types and connections of rooms in a house. However, in the absence of user specifications, the statistics system can still behave intelligently to mimic average American houses.

Adding Private Rooms After steps one and two, the graph has a set of public rooms. Step three randomly places private rooms adjoining public rooms until the private space is filled. Private rooms should only be accessible by means of a public room.

Adding Stick-on Rooms The final step in creating the graph consists of adding a few remaining rooms that are well-suited to being "stuck on" at the last minute, for example closets and pantries. These rooms can be added quickly and without impacting any of the other rooms already in the graph. The statistics system also aids in this step to ensure logical connections.

2 Placing Rooms

At the beginning of this stage, the types of rooms and the connections between them have been completely determined. However, the rooms are not located in space. The placement phase distributes the rooms over the footprint.

This process treats the graph as a tree with its root at the room that adjoins the front door. From that root, the root's child nodes are then evenly distributed beneath the root with each child spread an equal distance from other children and an equal distance from the root. The process is then repeated for each of the children nodes. The resulting distribution of rooms can be observed in Figure 5.

3 Expanding Rooms

Rooms are expanded to their proper size using a Monte Carlo method to choose which room to grow or shrink next. Every room in the graph exerts an outward "pressure" proportionate to the size the room should be relative to other rooms. Walls that are shared between rooms have pressure on them from both sides.

The four walls of the room are examined in turn. The decision of whether to grow or shrink a room by moving a wall is based on the difference between the pressure inside the room pushing out and the sum of the pressures pushing in on that wall from adjacent rooms. If the pressure from the inside is greater than the sum of the pressures from the outside, the room will expand by moving the wall under consideration a fraction of the footprint. If the pressure from outside is greater than that from inside, no expansion takes place.

4 Discussion and Future Work

We implemented our algorithm in Java 1.5, executing the timing trials on a PowerMac G5 Dual 2.0GHz with 1GB of RAM. The algorithm exhibits run time suitable for real-time applications, capable of generating 50,000 houses in under 3 minutes.

We have presented a procedural algorithm to generate plausible residential interiors. Our approach offers a high degree of automation with the option of precisely specifying details through the use of rulesets and statistics. Our method augments the context-free grammar often used in procedural model generation with rulesets that complement the generative qualities of the grammar by enforcing adherence to observed architectural patterns. Our method is novel because it focuses on the internal structure rather than just an external facade and it generates houses rather than more regular industrial buildings.

We believe that procedural methods for modeling man-made artifacts dramatically reduce the need for human labor in the field of 3D model design.

References

BROOKS, F. P. 1999. What's real about virtual reality? *IEEE Computer Graphics and Applications 19*, 6, 16–27.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, third ed. Morgan Kaufmann.

GREUTER, S., PARKER, J., STEWART, N., AND LEACH, G. 2003. Real-time procedural generation of 'pseudo infinite' cities. In *Computer graphics and interactive techniques in Australasia and South East Asia*.

LECHNER, T., WATSON, B. A., WILENSKY, U., AND FELSEN, M. 2003. Procedural city modeling. In *1st Midwestern Graphics Conference*.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques*, ACM Press, New York, NY, USA, 301–308.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. In *ACM Transactions on Graphics 22*, no. 3, 669–677.

WRIGHT, W., 2005. The future of content. Speech, Game Developer's Conference 2005.