Procedural Generation of Urban Environments

Joris Kluivers
Vrije Universiteit, Amsterdam,
The Netherlands
j.b.w.kluivers@student.vu.nl

July 28, 2010

Abstract

Current computer games provide long gameplay times using large virtual world and a huge amount of content to make the worlds realistic. Creating these worlds takes time and resources. Procedural content generation will replace parts of the work done by humans to speed up the development process. This article will describe a complete pipeline used to create a virtual world. The techniques demonstrate how to create realistic looking street maps and buildings to create cities to a near infinite size. Using these techniques the role of the artist and designer will change from creating each and every detail to a more guiding role where the computer will fill in the details.

1 Introduction

As personal computers get more powerful current 3d environments are expected to grow in size and detail as well. Current 3d games for example offer hours of gameplay in huge realistic 3d worlds. Creating these environments can take large teams of designers and developers multiple years which makes this process very costly[1]. Teams of 150 developers can take over 4 years to create an entire city. Scaling this up to a virtual country or even a planet with the same amount of detail is nearly impossible.

Instead of increasing the number of developers, resources to scale a project need to be found somewhere else. Computer techniques can aid the generation of environments using a combination of algorithms using specific parameters or randomness to generate an entire city in a matter of seconds.

In computer graphics procedural generation techniques are traditionally used to model highly complex systems. These techniques target natural systems like plant growth for example. The same techniques (with some slight alterations) can be used to model urban environments. Current computer hardware is able to compute the results of these algorithms on the fly (in game for example).

This paper describes common techniques to create 3d urban environments. Section 2 describes a common algorithmic form that lies at the basis of the techniques discussed in sections 3. In section 3 several different algorithms are described that each generate a different component that together form the complete environment. Section 4 discusses how the pipeline in

section 3 can be used to create infinite environments and what role random generators play. This article concludes with a summary and remarks follow in section 6.

2 L-Systems

A Lindenmayer system (L-system) is a variant of a formal grammar invented by biologist A. Lindenmyer to study natural processes such as the growth processes of plant development and the reproduction of certain organisms. L-systems operate on strings and rewrite a simple initial object using certain rules into a more complex object.

A l-system is composed of:

- variables that can be replaced
- constants that remain fixed during rewriting
- a start symbol
- a set of rules to base the rewriting on

Given a certain start string, the rewriting process will replace variables with one or more other variables or constants based on the available rules. Implementations described in Section 3 will usually limit rewriting to a certain depth to prevent recursion. Each run of the l-system results in a different end string based on the start symbol and the selected rules for a variable (determined by either parameters and randomness). The result is a string where all variables have been replaced by constants which will be interpreted as commands by a visualizing system.

Based on what system interprets the resulting string, constants will have different meaning. A road based visualizing system will know commands such as turn left, turn right and move forward to place roads. An architecture system will operate on geometry and translate the string into commands like rotate, add shape and add height. Different applications of l-systems will be discussed in section 3.

3 Modeling a City

Creating a realistic 3d city using procedural generation requires only a few relatively simple algorithms. The algorithms are executed in a simple pipeline one after the other. More details are added after each execution.

In this section a pipeline of 4 steps will be described. The first algorithm generates the terrain. Over this terrain roads are placed. Buildings are placed in between roads. The last step is decorating the buildings using textures. This pipeline is by no means complete and can easily be extended by adding more algorithms. Instead of modeling just a city the pipeline could be extended to model a complete universe from planets to details like interiors of buildings.

Each algorithm used in the pipeline may depend on zero or more parameters. The parameters influence the resulting geometry generated by the algorithms. Take random values for the parameters and the pipeline will generate a different city each time its used. But when needed

designers can influence the individual algorithms by altering the parameters. Even though the algorithms still generate the geometry the designer can steer generation of for example commercial or highly populated areas in a city.

3.1 Terrain Generation

In a computer generated world the terrain is usually represented by values over a 2d grid, called a hightmap. Each value in the grid represents the height at that point. A height map where each point in the grid represents the same value results in a flat surface. Hills and valleys can be created by increasing or lowering the value in the grid respectively.

Lowering or raising each point in the grid at random will not result in realistic terrains. Some coordination is needed so points near each other form smooth slopes. One way to quickly generate a terrain is to pick two points at random. Take the line through these points as a virtual border. All points on one side of the line will be raised and points on the other side are lowered by a certain amount. After performing this step multiple-hundred times a usable terrain has formed.

A height map can also be seen as an image. Each point in the grid can be seen as a pixel that represents a greyscale value. Complicated height maps could for example be created using a cloud filter in photoshop. Although this wouldn't result in a realistic terrain all the time. Jacob Olsen [5] describes in his paper a way to generate eroded fractal terrains in realtime. Height maps are created using fractal algorithms. These height maps contain a lot of rough spots uncommon in the real world. The algorithm used by Olsen smoothens the terrain as erosion would have done in nature. Figure 1 shows how the same height map would look by altering the erosion factor in the algorithm.

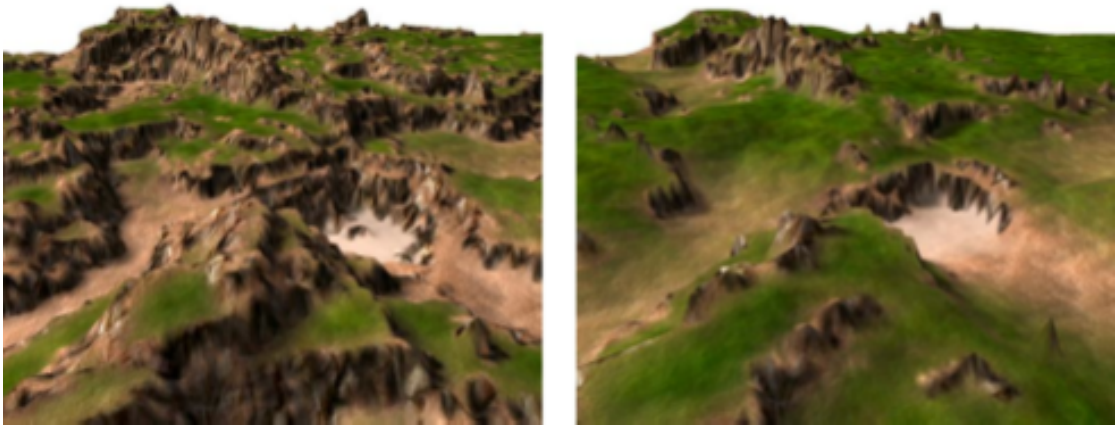
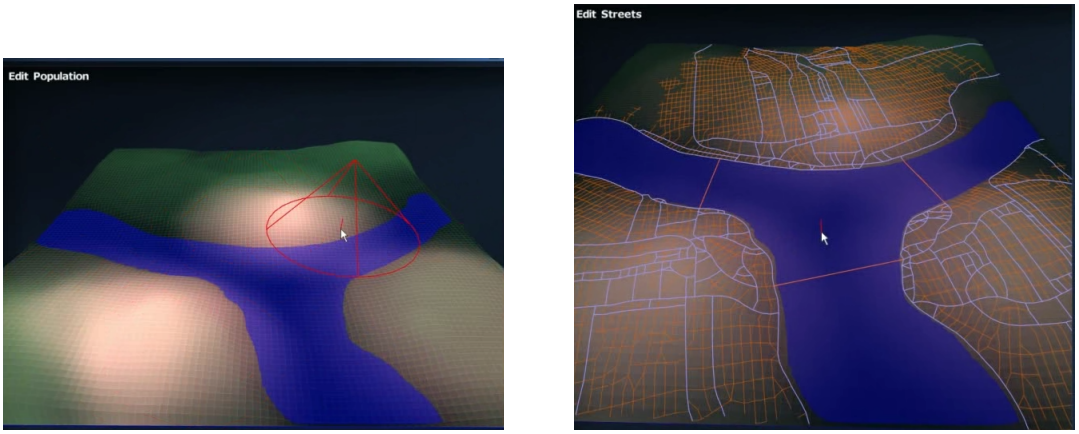


Figure 1: **The same height map rendered using two different erosion factors**

3.2 Road-Maps

When analyzing existing cities and their roadmaps different patterns can be found. Think about the square blocks in U.S. cities or a radial roadmap centered around a central landmark like Paris. These patterns all form because of different reasons. Roads are constrained by the underlying terrain: bridges will form over water and hairpins will be used to travel highly sloped terrain. Highly populated areas will contain more roads while different populated areas will be connected by highways with only a few smaller roads in between.

To simulate the creation of realistic road maps Parish and Müller use an extended L-system [3]. Traditionally L-systems are string rewriting mechanisms based on a set of production rules and parameters. When new parameters are introduced the production rules would have to be rewritten for a big part. Instead of setting each parameter inside the production rules the L-system used by Parish and Müller uses generic templates at each step. The template is called the *ideal successor*. Parameters are moved from the template to external functions.



(a) Assigning population density

(b) Resulting road pattern

Figure 2: **Generating road patterns based on population density [12]**

The parameters for each template influence how the *ideal successor* is reached. Parameters in the L-system to create roadmaps are separated in a global and local group.

- Global: paramters like population density, commerce versus suburban district and super imposed patterns like blocks or a radial pattern.
- Local: geographical boundaries like land, water, vegetation and elevation.

Using this system road maps can be created for any terrain. The global and local parameters will either be specified by a designer or taken at random. Figure 2a shows how a level designer specifies the populated areas on a terrain. Figure 2b shows the resulting road map generated using a a L-system as proposed by Parish and Müller.

3.3 Generating Buildings

Generating buildings is possible in numerous ways. It can be as simple as placing to geometric shapes together. A more completex algorithm to generate buildings is based on polygon rotations to create floorplans[2]. One or more polygons are used for the base shape of the building. For each addition to the building another polygon is added the the floorplan. Parameters like height and number of polygons will determine what the building will look like. The more height and the more polygons will be added to the floorplan the more the building will look like a skyscraper. A single cube can be used as a simple house.

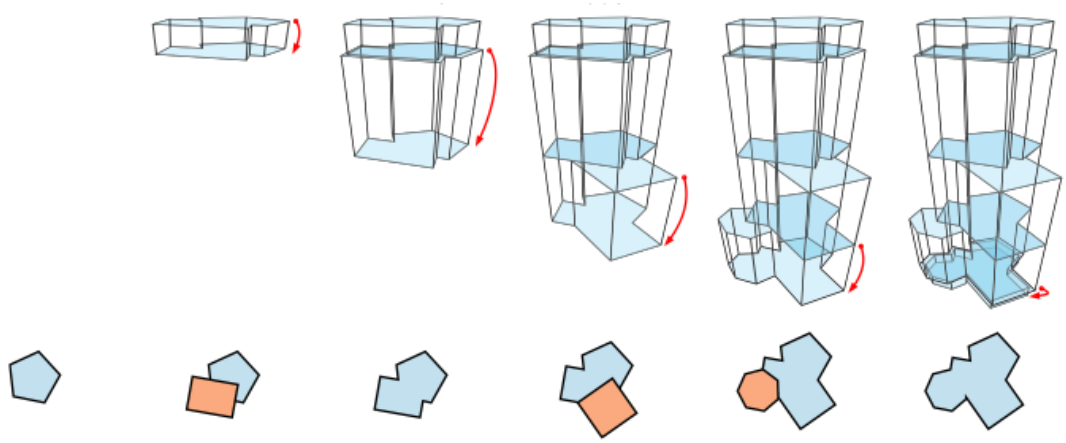


Figure 3: **Using polygon rotations to generate floorplans for different levels of the same building**[2]

The algorithm mentioned above uses shapes and rotations only. However it can be made as advanced as needed by adding more operations such as branching, extrusion and pre-build templates (for rooftops for example).

Parish and Müller use an implementation based on L-systems[3] to generate buildings similar to the example above. Three sets of production rules are used to describe different type of buildings: skyscrapers, commercial buildings and residential buildings. After rewriting the production rule using the L-system, the resulting string is translated into geometry readable by the visualization system.

3.4 Generating Building Faces

The generated buildings still are only 3d shapes without decoration. For a building to look realistic it need windows, doors and other decorations. These elements need to be placed intelligently as they were placed by an architect. A method for generating realistic building faces through a new type of formal grammar called split grammars is described in *A Survey of Procedural Techniques for City Generation*[6]. A split grammar is based on the concept of a shape.

Split grammars A split-grammer is a grammar where shapes are used as the alphabet. Shapes are replaced by one or more sub-shapes using a random selected rule. One or more rules are applied to the start shape. For the algorithm to produce unique result shapes a large number of rules are needed. Rules can be derived by taking real world architecture as an example. A window can be split in four where every quarter of the window is replaced by a different colored glass and lead pattern for example. Figure 4 shows a start shape with replacement rules for this shape.

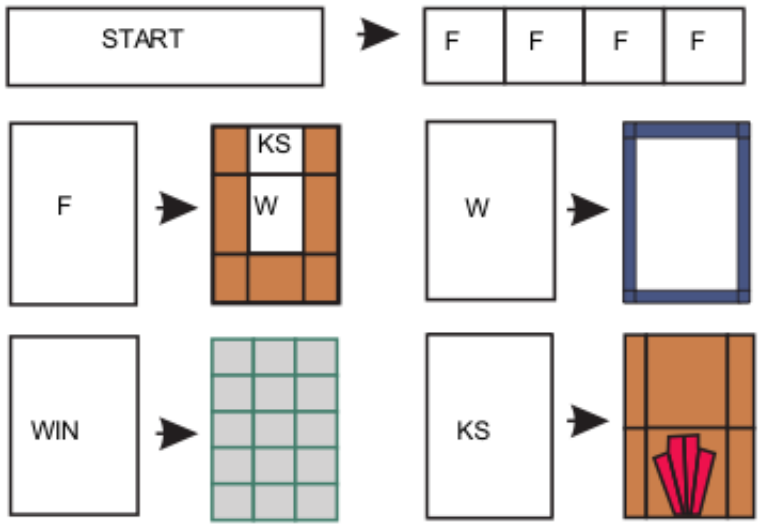


Figure 4: **A visual split-grammar example** [6]

A start shape is provided to the algorithm. This shape is split in open and closed sub shapes. Each open sub-shape is divided in more sub-shapes similar to the start shape. This process repeats itself until no open sub-shapes are present. The complete result of the split-grammar in Figure 4 looks like Figure 5.

4 Real-Time Infinite Worlds

Virtual world in games like GTA and in movies would be generated once and reused when needed. Geometry is read from a file when needed and released from memory when outdated. As the virtual world grows in size so does the file it's stored in. Cities or worlds infinite in size can't be stored in files as no infinite storage capacity is invented yet.

In case a world is too large to store on disk the world can be generated using the same techniques in real-time. Instead of reading a pre-generated building from file each time it comes into view it is generated using the same procedural algorithm when needed.

Because procedural generation algorithms are heavily based on random numbers each time you would visit the same place again a different building would be visible. Differences in results

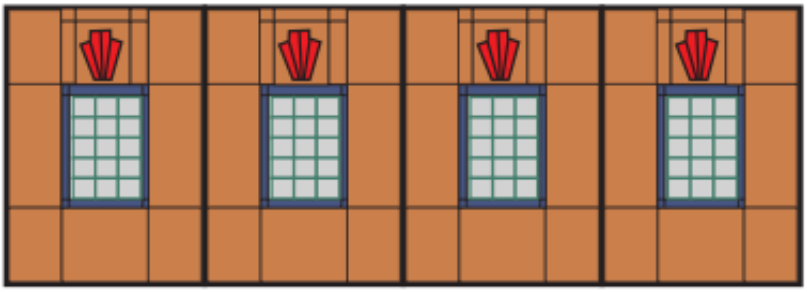


Figure 5: **The result of the split grammar in Figure 4** [6]

from the same algorithm all depend on the randomness of the sequence of random numbers. A different sequence of random numbers will result in a different outcome of the algorithm.

Sequences of random numbers are generated by the so called pseudo random number generators. These generators are initialized with a seed value. Two similar seed values result in two similar random number sequences. The same seed value for two generators results in two identical random number sequences.

To ensure the same building is shown in an infinite world the random number generator is seeded with the same number. This number is a hash value based on the x and z location of the building.

5 Conclusion

Some might argue that using these techniques the designer will be made obsolete completely when developing 3d content. One forgets in this case all content generated is influenced by parameters specified in the algorithms. Only correctly balanced parameters and algorithms will result in realistic outcomes. It's artists and engineers that will have to work together to create the perfect world.

The big future for current procedural generation techniques is in the quick generation of large amounts of content. This content can be used in various situations like computer games or computer enhanced movies. Depending on its uses the content will usually be altered because currently results still look generated too much. Games with extensive story lines will have to alter content much more to make the the story fit in the generated environment.

References

[1] Take Two Takes a Hit
http://www.forbes.com/technology/2006/12/11/games-gta-options-tech-enter-cx_rr_1211taketwo.html

[2] Stefan Greuter, Jeremy Parker, Nigel Stewart, Geoff Leach, *Real-time Procedural Generation of 'Pseudo Infinite' Cities*. Graphite 2003

[3] Yoav I. H. Parish, Pascal Müller, *Procedural Modeling of Cities*. ACM SIGGRAPH 2001

[4] Jean-Eudes Marvie, Julien Perret, Kadi Bouatouch *The FL-system: a functional L-system for procedural geometric modeling*, Visual Comput 2005

[5] Jacob Olsen, *Realtime Procedural Terrain Generation*. http://oddlabs.com/download/terrain_generation.pdf, October 2004

[6] George Kelly, Hugh McCabe, *A Survey of Procedural Techniques for City Generation* http://www.gamesitb.com/SurveyProcedural.pdf

[7] Andrew Doull, *The Death of the Level Designer: Procedural Content Generation in Games*. http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html, January 2008

[8] Mohamed Shaarawy, Ahmed Kaboudan, Shaimaa Toriah, *3D Automatic Building Footprints Generation*, Proceedings of the International MultiConference of Engineers and Computer Scientists 2009 Vol. I

[9] Sebastien Horna, Guillaume Damiand, Daniel Meneveaux, Yves Bertrand, *Building 3D Indoor Scenes Topology from 2D Arhitectural Plans*

[10] Introversion Software, *Procedural Content Generation*. http://www.gamecareerguide.com/features/336/procedural_content_.php, June 2007

[11] Introversion Software, *The Introversion Blog*. http://www.introversion.co.uk/blog/

[12] Introversion Software, *Creating a city using procedural techniques (movie)* http://download.introversion.co.uk/subv5.avi

[13] *Procedural Road Generation*, http://britonia-game.com/?p=28