Exploring L-system: Modeling of Virtual City Landscapes

Ng Hian James (HT035267U)
nghianja@comp.nus.edu.sg
School of Computing, National University of Singapore

Abstract

This is the final report for the mini-project for the course module CS6241: Advanced Topics in Computer Graphics. The mini-project is about exploring the usage of L-system in the context of automatically generating buildings to model the urban landscape of a virtual city. In this mini-project, only the work on division of small area blocks on input maps into lots for the placements of buildings and the generations of the buildings are done here, leaving the rendering (with or without texture mappings) to any rendering engine that can take in the output by the L-system.

1 Introduction

For this CS6241 mini-project, I was tasked to explore the method of reading in a planar map of a city (or a diagram representing a city map) and then using an L-system to model architectural structures (skyscrapers) on the areas allocated for buildings in the given map. Hence this is a small study-cum-implementation project which I hope to demonstrate the idea given in the current literature on the automatic generation of architectural buildings for modeling a virtual city landscape.

So why is there a need to study the automatic generation of buildings? There is a need to study it because automatic generation of architectural buildings is extremely useful to architects, city planners, residents and as well as tourists where a virtual view of a city can help them get a feel of how a city looks like now and after future development. An interactive urban planning application incorporating such technique can help urban planners and residents explore different design choices in the context of a reconstructed environment. Other applications range from research and educational purposes such as simulations to entertainment purposes such as movie-making and game development.

In this paper, I try to discuss on the approach to automatic building modeling adopted by the community and the work already done by others. I try to recreate, through naïve programming and simplifications of the ideas and concepts given, a demonstration of how modeling of buildings for a city landscape can be implemented. However, I do not concern myself with the rendering and texturing of the building models generated. I leave these to the internal display system of an L-system program or any external rendering and display engine. Finally, I record down all the difficulties I have encountered during the development of this mini-project. I also try to detail the portions of the mini-project that I am not able to complete or capable but unable to do due to certain constraints.

Hence this paper is structured as follows: Section 2 will give an overview of what this mini-project is about and how it is going to progress. Section 3 will give a short introduction to L-systems and how it can be used here. Section 4 will give the basic idea of dividing a block area, representing the land whereby a group of buildings stand, into individual lots for a building each. Here is also where I give my variant of the implementation. Section 5 will give the basic idea of modeling a building on top a lot area and my effort in accomplishing the same work as described in the literature. Section 6 will give the improvements I should have made in this mini-project. Section 7 will give a report on the difficulties I encountered in developing the programs in here as well as the assumptions made in order to complete the sample system. This is also where I justify why I do what I did and the concepts others have done which I cannot include. Then I will wrap up with a short conclusion.

2 Overview of Project

As mentioned in the introduction, this mini-project is about modeling a virtual city landscape given an arbitrary planar map. Based on the paper by Parish and Müller [1], the basic idea of the approach adopted is divided mainly into two parts as follows:

1. Creates the allotments for the placement of the buildings;
2. Generates the geometry for the buildings.

For the first part, the steps involved are primarily to have the system read in a graphic file that consists of a map, then blocks representing areas where buildings are located are selected, and finally, the blocks are sub-divided into lots. The map does not have to be a real map. It can just be a diagram representing a map of a city as viewed from the top. The file can be of any format though my choice is the GIF format. As for the sub-division of blocks to lots, the algorithm is supposed to be simple.

For the second part, a stochastic, parametric L-system is used to generate the geometry for the buildings. I will explain what an L-system is in the next section. Production rules for the L-system are designed to generate the different models of architecture with the concept of shape grammars discussed. The parameters that determine the design of a building being placed in a lot can range from zoning rules and regulations to designers' preferences.

3 L-systems

An L-system (Lindenmayer system) is defined by an axiom (start string) and a list of derivations. A derivation is given by the table (set of productions) to be applied and the number of derivation steps. It was introduced in 1968 by the biologist Aristid Lindenmayer. In Lindenmayer's and Prusinkiewicz's words, the central concept of L-systems is that of rewriting. In general, rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of *rewriting rules* or *production.* [2]

Since then, L-systems have been implemented in numerous programming languages and downloadable versions are readily available for free. The L-system of choice for this mini-project is the one called L-System 5 (http://www.geocities.com/tperz/L4Home.htm) I have downloaded (see figure 1 for screenshot). Hence the L-system is one of the most common systems around and it is best known for its application in the modeling of plants [3, 4] as well as the modeling of streets [1]. Now I am trying it out on the modeling of pre-defined architectural buildings.

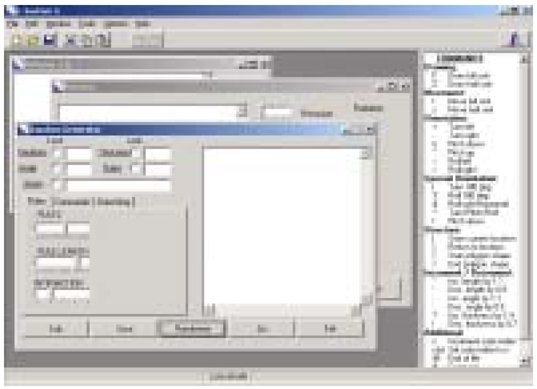


Figure 1: Screenshot of L-System 5

4 Divisions of Blocks

Before any processing or calculation can be done, I have to let my starting program read in a graphic file of the GIF format that contains my arbitrary planar map as viewed from the sky. The starting program, which is written in Java, then displays the map in a window for user selection of area blocks using the mouse (see figure 2a and 2b). For the moment, only rectangular blocks can be selected due to time and knowledge constraints as well as programming capability. The selection is made permanent when the "Record Block(s)" button is clicked, thus multiple blocks can be stored if necessary. The "Clear All" button erases all stored blocks and lots, allowing the user to begin afresh.

The next step of this portion of the system is to sub-divide each selected block into lots suitable for the placements of buildings. Based on [1], the method to divide a block is supposed to be straightforward. According to it, "a block is divided into smaller units using a simple, recursive algorithm that divides the longest edges that are approximately parallel until the subdivided lots are under a threshold area specified by the user". However, this is all the information the authors gave in the paper. I have absolutely no idea as to how the algorithm should be implemented. Hence I have to devise my own algorithm. Though Müller replied to my email regarding the algorithm (see figure 3a), I cannot comprehend sufficiently to implement.

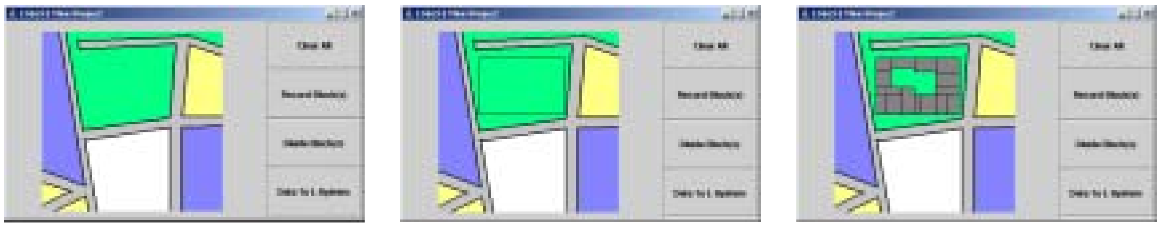


Figure 2: (a) Display of map (b) Block area selected (c) Block subdivided into lots

Since I am unable to implement the algorithm as described by Müller, I have no choice but to come up with an algorithm which can subdivide a rectangular block selected by the user into small lots. My algorithm is as follows:

1. Initiate a random number generator R.
2. Insert the selected block into an array.
3. Extract a block from the array if available. Else end division method.
4. If area of block is below threshold A, insert block into another array as a lot. Go back to step 3.
5. Else split the block into 2: split down between the 25% and 75% of the longest side of the rectangular block, with the exact point of division determined by R.
6. If any of the two new blocks is of area less than threshold B, discard the two new blocks.
7. Else insert the two new blocks into the first array defined in step 2.
8. Go back to step 3.

Using my algorithm, I am able to cut the original block of area into suitable sizes of lots in preparation for generating of building models that fit into the allocated spaces. A sample of the result given by my algorithm is seen in figure 2c. The result is stored as a list of rectangles which is used in the later stage to formalize the axioms and production rules for the L-system.

Of course, it will be terrific if I am able to implement the algorithm as described in figure 3a. I will discuss in section 6 and 7 about that algorithm, the problems in understanding it, and the possible form it will be if I am to write the code for it.

5 Modeling of Buildings

The second part of the system is to generate the geometry of the buildings. To my initial understanding, the geometry here means the shapes of the buildings that are represented by the production rules of the L-system. To be exact, I am to define the codes for the L-system to generate the shapes of the buildings and the codes will be in a text file readable by the L-System 5 application. However, an alternative understanding can be that the L-system generates the building models and outputs the geometry of the shapes as coordinates for a display engine. Both understandings are possible but I adopt the former as the basis for my work.

No, you don't need a l-system. its just a simple recursive polygon-spliting algorithm, which works as follows (as I can remember):
1. choose longest edge with most perpendicular neighbors (of a polygon/lot)
2. split it perpendicular to that length (more or less in the middle)
3. go with both resulting polygons to 1.
... quite stupid, but it worked...

(a): Müller's reply to the subdivision algorithm.

Firstly, we generated rules for a common l-system (which create some building-styles).
Afterwards, assignïng a rule/building to a lot was a statistics-function which depended on lot-area, zone (via map) & population-density (via map).
If you wanna know more about procedural modeling of buildings, check out the "Instant Architecture"-paper (Siggraph03), we did more or less the same.

(b): Müller's reply to the modeling of buildings.

Figure 3: Pascal Müller's email to my queries.

As given in [1], all buildings are modeled with a parametric, stochastic L-system with one building per lot. There are different types of buildings and for every type of building a different set of production rules is executed. Once again, the information for this part of the system is limited to a couple of paragraphs in the paper. The paper just mentioned that buildings are created by manipulating an arbitrary ground plan and the L-system consists of various modules for modeling the buildings. The final shape of the building is to be determined by its ground plan which is transformed by interpreting the output of the L-system.

Since I have no means of understanding the description written by Parish and Müller, I again try to come up with a method to suit the target of this mini-project. As mentioned before, I adopt the concept of creating a text file containing the production rules for the L-system to generate the buildings. I will start with creating the simplest of all codes to come up with just block volumes representing buildings and subsequently place them in their respective lots. Then I will follow up on Müller's advice (see figure 3b) to study the paper by Wonka *et al* [5].

In that paper, the authors had written about their work on automatic modeling of architecture with the use of a design-grammar. This type of grammar is known as "split grammars" and building designs are derived using this new type of parametric set grammar based on the concept of shape. The paper also introduces an attribute matching system and a separate control grammar, which offer the flexibility required to model buildings using a large variety of different styles and design ideas. There is a great amount of work done by the authors and a lot of information given but I am concentrating on trying to make use of shape grammars. I will give more details on shape grammars at a later part of this section.

The rest of this section is thus organized in the following way: sub-section 1 gives the preliminary development I have done; sub-section 2 gives an introduction to shape grammars, and sub-section 3 gives my final results of this mini-project.

5.1 Preliminary Development

In the beginning stage of the development, I start off by testing the L-System 5 application and finding out the necessary format the text file it reads in has to be in. The simplest set of production rules for drawing is for displaying a line and subsequently a regular block volume (see figure 4). Note that in L-System 5, the same type of block can be also produced by varying the thickness of a single line. A line is replaced by a block (or cylinder) of a specified thickness of a breadth (or radius) that is proportional to the length.

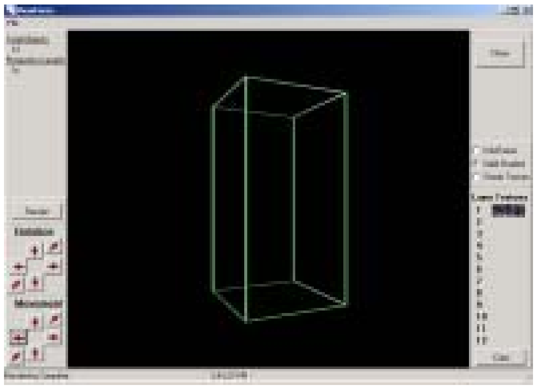


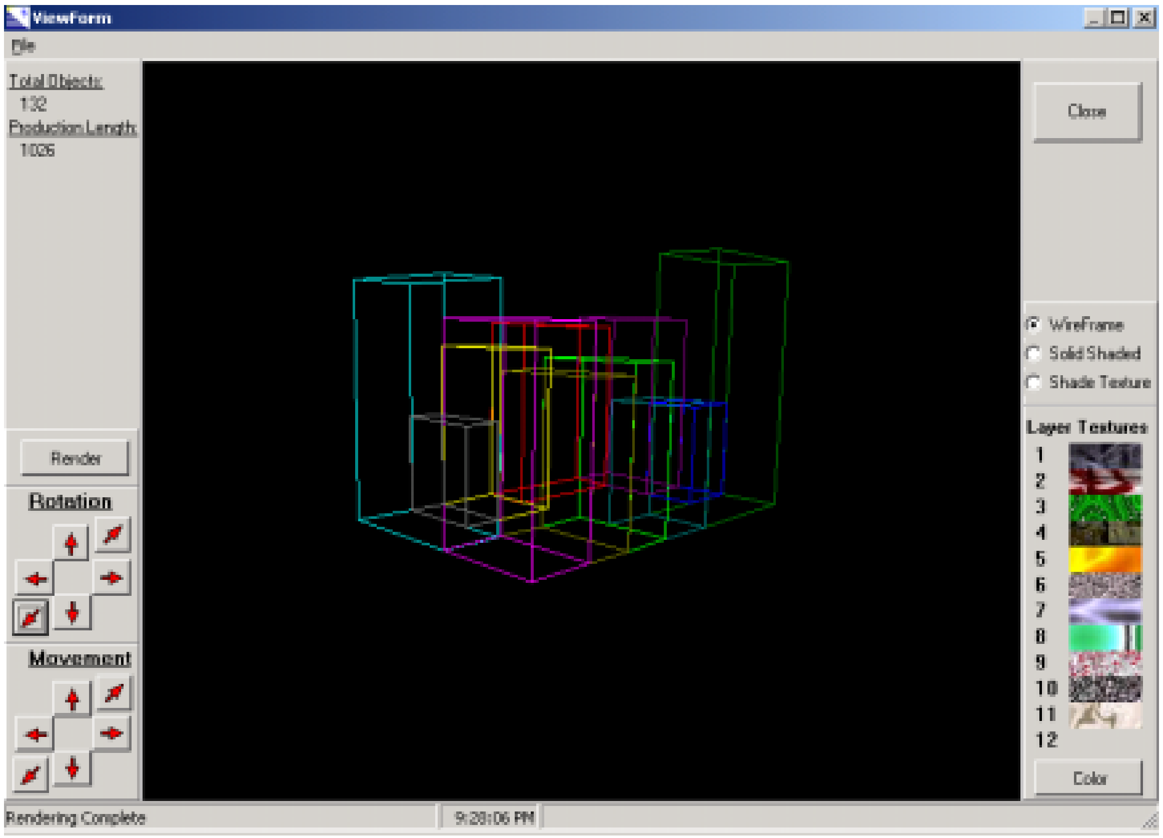
Figure 4: A regular building block volume.

Given a building block volume, I can place multiples of such volumes together to form a landscape of rectangular buildings representing a part of a virtual city. An example of this is shown in figure 5. After this is done, the next step is to map the building blocks to the lots obtained in the previous section and the codes for the production rules are generated programmatically.

However, fully automated generation of suitable codes is still not possible at this point. A certain level of hand-crafting is needed to produce the necessary text file for reading. It also means that the block volumes have heights which are fixed with respect to different base areas for the moment. It is also proper to put on record that I am using a production rule for each block volume at this point in time. Hence, I can only generate a limited number of buildings.

5.2 Shape Grammars

In [5], the authors gave a description of the spatial grammar which formed the syntactic basis for their building model. It is used to determine the spatial layout of a building. As the approach for such a spatial grammar is drawn from the work on shape grammars pioneered by Stiny [6], I therefore have to make a visit to the topic of shapes and grammars and dwell into it so that I can know of the method for my work.



Total Objects:
132
Production Length:
1026

[✓ WireFrame] Solid Shaded Shade Texture

Rendering Complete
9:28:06 PM

Figure 5: Multiple block volumes representing multiple building models.

Shape grammars are a powerful tool to express very general designs. They have properties aimed at making them especially suitable for designing, without sacrificing formal rigor. First, the components of shape rules are shapes: points, lines, planes, or volumes. Shape rules generate designs using the shape operations of addition and subtraction, and spatial transformations familiar to designers such as shifting, mirroring, and rotating. Second, shape grammars treat shapes as non-atomic entities – they can be freely decomposed and recomposed at the discretion of the designer. This liberty allows for emergence – the ability to recognize and to operate on shapes that are not predefined in a grammar but emerge, or are formed, from any parts of shapes generated through rule applications. Third, shape grammars are non-deterministic. The user of a shape grammar may have many choices of rules, and ways to apply them, in each step of a computation. As a design is computed, there may be multiple futures for it that respond differently to emergent properties, or to other conditions or goals. [7]

I now state the definitions for the following: shape, grammar, and set grammar, excerpted from [5] (originally given in [6]) as recap for the sake of understanding.

Definition 1: *A shape is a limited arrangement of straight lines in three-dimensional Euclidian space.*

A line is uniquely determined by two distinct end-points. Two lines are identical if their end-points are the same, and thus two shapes are identical if they contain the same set of lines. Shapes can be attributed with a set of labeled points. From my understanding, the shapes that are of concern here are the primitive types such as rectangles, triangles, cubic blocks, cylinders, cones and pyramids. These are the building blocks for any architectural design models and I am supposed to be able to generate sophisticated building structures out of these primitive shapes.

The L-System 5 already does provide the drawing of primitive shapes in replacement of straight lines. Figure 6 shows the four primitive shapes available for drawing in the L-System 5: cubic block volumes, cylinders, cones and pyramids. However, there is an inherent flaw with the system and it is that the primitive shapes cannot be scaled easily. Hence, I need to know how to manipulate them accordingly.

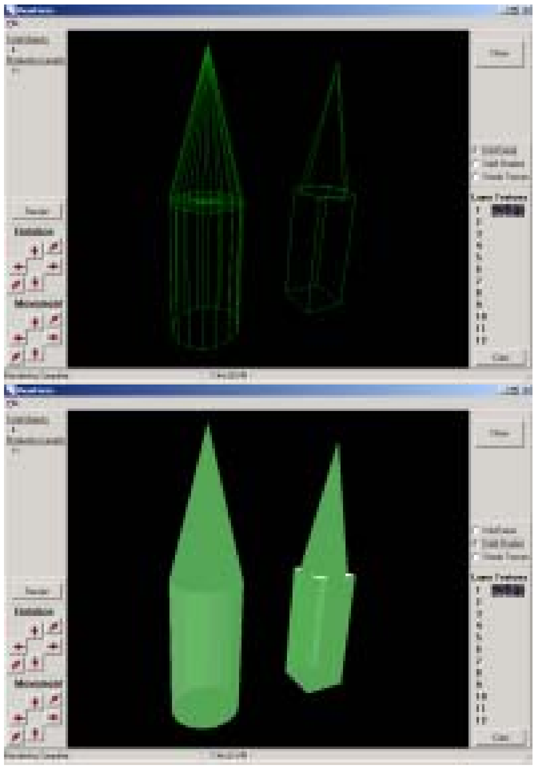


Figure 6: Building blocks composed of a cone, a cylinder, a pyramid and a cubic block. Top: In wire-frame forms. Bottom: In shaded solid forms.

Definition 2: *A grammar G = (N,T,R,I) consists of the non-terminal vocabulary* $N \subseteq U$*, the terminal vocabulary* $T \subseteq U$*, a (set of) initial object(s)* $I \subseteq N$ *and a set of rewriting rules (productions)* $R \subseteq UxU$*.*

The above is a definition for a general grammar for parameterized objects where they are operated on under a series of rules and algebra. A rule $a \rightarrow b$ is applicable to u whenever there is a transform f from F and a variable assignment g such that $f(g(a)) \leq u$, in which case, under rule application, the object v is produced, given by the expression

$$v = (u - f(g(a))) + f(g(b)).$$

This gives rise to the idea that I can design or make use of available basic shapes or polygons to generate models as long as the shapes belong to a grammar that manipulates (transforms) them through a set of algebraic operations. At this point, *set grammars* are of special interest, defined as below:

Definition 3: *A set grammar is a grammar over the vocabulary B, where the algebra U is the power set of the defining set B, the operations + and – are set union and set difference respectively, the match relation* $\leq$ *is the subset relation, and f(S) is the set {f(s)|s in S} for a subset* $S \subseteq B$*.*

With the definitions listed above and the idea of applying transformations and algebraic operations, all I need to do next is to come up with the necessary grammar (production rules for the L-system) to attach every and if necessary, different primitive polygon together as building models. I will make use of the basic polygons L-System 5 provided me with if that is possible. If not, I will craft the required shapes or polygons.

5.3 Final Results

Introduced by Wonka *et al* in [5] are split grammars, a specialized type of set grammars operating on shapes. The shapes being manipulated by the grammar are certain attributed, parameterized, labeled shapes the authors called *basic shapes*. In summary, the basic shapes are just simple building blocks of the grammar, much like basic polygons such as cubes, cylinders or prisms. A split is defined as the decomposition of a basic shape into shapes, thus a split grammar is a set grammar that has the following:

- A split rule $a \rightarrow b$ which b contains the same element as a except for the one element that is being split.
- A conversion rule $a \rightarrow b$ which transforms one basic shape into another.

One notable difference between the split and the conversion rule is that in a split rule, the elements in the sets a and b fill the same volume, whereas in a conversion rule, this is not necessarily the case. However, I will put aside the details for the time being and work on the simplest form of splitting basic shapes for modeling. As shown in figure 7, I am emulating the rules for a simple example split grammar as demonstrated in [5].

Another way of looking at the split grammars is looking at basic shapes in the reverse way. A basic shape can be seen as made up of a group of other basic shapes, thus I can derive a grammar for an L-system that combines many little shapes into one big shape. In fact, for some of the shapes in figure 7, I do use a combination of rectangles to work my way to a larger rectangle. I also derived a similar grammar for the architectural model shown in figure 8.

Both figures show examples which are very simplistic in nature. I too wish to have, to a certain extent, a realistic model, but a complex model is beyond my means. It is extremely tedious to derive a grammar for a complex model and I am even beginning to doubt that it is possible to do on a common L-system. There is only so much a common L-system can do. Hence this level-of-detail for the figures is what I can have for now (as time forbids).

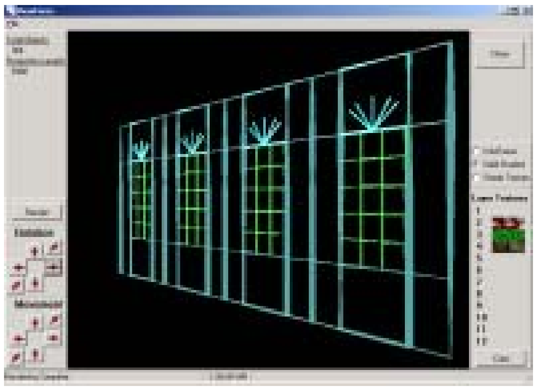
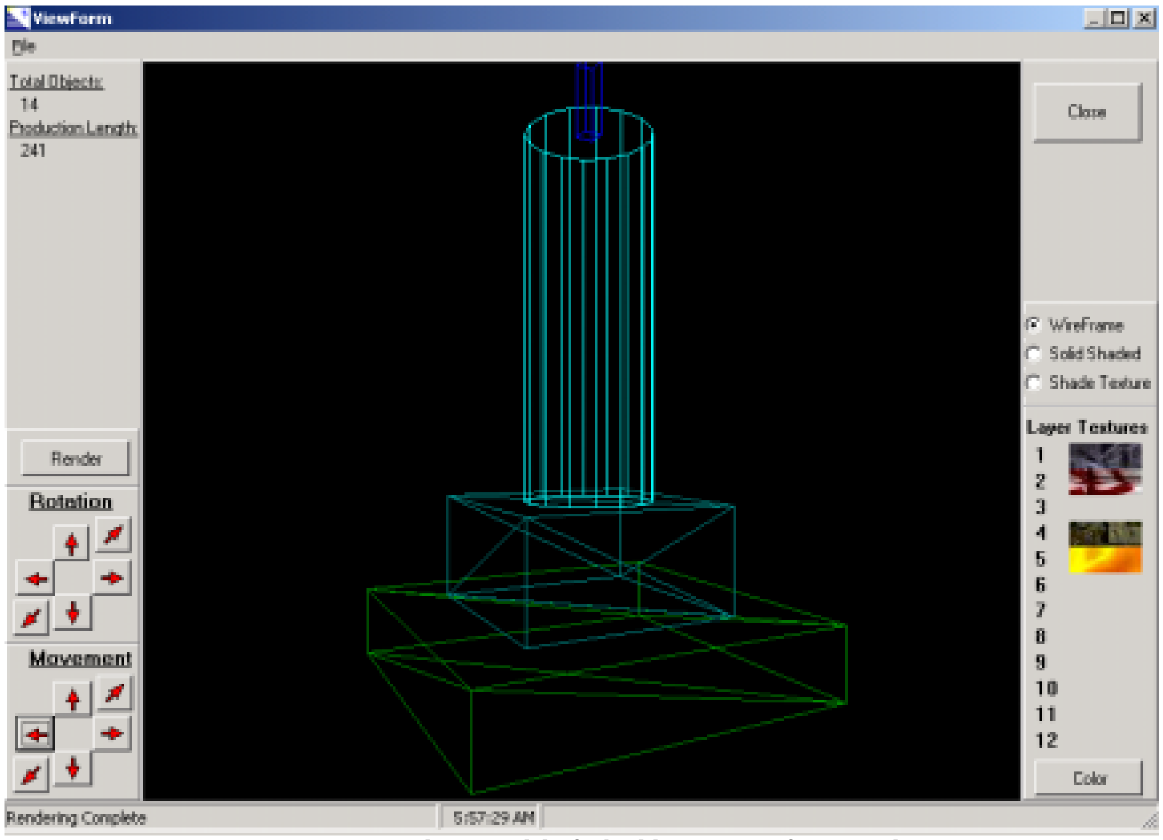


Figure 7: The result of the derivation of the simple example of a split grammar given in [5].



Total Objects:
14
Production Length:
241

[✓ WireFrame]
Solid Shaded
Shade Texture

Rendering Complete | 5:57:29 AM

Figure 8: A simplistic model of a building in wire-frame mode.

The next couple of steps are supposed to be the grafting of the results from split grammars to buildings and the moving of buildings to the allocated lots (replacing the block volumes in figure 5). Unfortunately, I am not able to complete either of them. The methods to doing both are not straightforward and from the clues given in the literature, I suspect that a specially crafted L-system is required. So I am placing a logical stop to the mini-project at this point, despite the incompleteness.

6 Improvements to Project

Due to the constraints in time and knowledge, there are various portions of the mini-project that I may but could not improve on. The lack of sufficient information and examples is a major problem to the completion of this mini-project. In this section, I will record down the improvements I would have done.

Firstly, I would have to allow the selection of polygonal block areas from the planar map. Only with polygonal blocks then can I try out the simple recursive algorithm Parish and Müller used to subdivide blocks into lots. I would have a better representation of the buildings' base areas than I have now.

Secondly, I would test out any method that is suitable for the automatic selection of blocks. A block area is bounded by the streets and roads on the map so there should be a simple algorithm to do this. One suggestion I have from the lecturer of the course this mini-project belongs to is to use a flooding algorithm.

Thirdly, I have to include a certain amount of randomness to the grammar of the L-system. Besides randomness, I have to include variables in the grammar so that parameters can be supplied. These two are important as both [1] and [5] mentioned about the use of a stochastic, parametric L-system for modeling the buildings in a virtual city. The parameters are external conditions obtainable from the planar map or user and they are values like zoning rules, types of buildings, design dimensions and aesthetic rules.

Last but not least, I should have found the way to place the building models to the allocated lots, with the base areas bounded by the size of the lots. However, I need to know how to let the grammar deriving a building model takes in the lots' coordinates as parameters first. Hence I have to fulfill the pre-requisite of reading parameters in my L-system's production rules. If not, there is still the option of tediously crafting a large database of specialized production rules for each type of buildings and for different lots' spaces.

7 Lessons Learned

The numerous setbacks I have encountered while working on this mini-project has made me realized that to automatically generate buildings for a virtual city is not an easy task. In fact, it is extremely hard as it involves a lot of aspects. There are various issues to handle and situations to control. Based on the materials I have found while reading up for this mini-project, I can summarize the problems and solutions in the following ways:

1. The problem of designing an algorithm in subdivision of blocks into lots is crucial to how the landscape of the virtual city will look like. The solution is simple and straightforward enough. The shape of a block and the intended threshold of size for an allotment determine how the algorithm is going to respond.

2. To combine architectural structures with allocated areas involves the using of parameters obtained from the planar map or an image map. For each area, the style of the building residing on it is determined by information such as the zoning rules, population density and the ground plan.

3. There is the concept of using basic shapes and a specialized type of set grammars to perform algebraic operation on the shapes in order to obtain building models. This will involve the implementation of axioms and production rules in customized versions of parametric and stochastic L-systems.

4. Production rules are controlled by a separate grammar. This grammar will base on attributes associated with the symbols in the shapes and attached to grammar symbols to perform a matching of rules. The rules matched for use will consistently give a spatial distribution of design ideas that corresponds to architectural principles.

5. The vastness in the number of differences in details when modeling a building requires a large set of production rules. This large set of production rules translates to a large demand in data storage capacity. Since the grammar searches the rule database for all rules that matches the shape under consideration, data retrieval time should take up most of the processing time.

6. The L-system does not actually display the models. It passes the geometry of the shapes to visualization systems for display and rendering with textures

mapped. The process may not be so direct. The output of the L-system can just be a string of data that is fed to another parser, which translates the string to geometry.

8 Conclusions

For this mini-project, I have shown the necessary parts of an automatic modeling system for architectural buildings. I have also implemented simplified versions of the individual parts, demonstrating how the required results can be obtained. Unfortunately, the mini-project is not as complete as it should have been. The results are also not at a high level-of-detail as I have hoped for. For this, I pinned the problem down to the lack of information and my inability to comprehend some of the crucial portions of the literature. Unfamiliarity with the L-system and the derivation of axioms and production rules also delays and hinders the development work.

References

[1] Yoah I H Parish and Pascal Müller, "Procedural Modeling of Cities". In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 301-308, 2001.

[2] http://www.stud.tu-ilmenau.de/~juhu/GX/RTEvol/DOC/LATEX2HTML/node6.html.

[3] P. Prusinkiewicz and A. Lindenmayer, "The Algorithmic Beauty of Plants". *Springer-Verlag*, 1991.

[4] I. Shlyakhter, M. Rozenoer, J. Dorsey and S. Teller, "Reconstructing 3D tree models from instrumented photographs". *IEEE Computer Graphics and Applications*, pages 53-61, May/June 2001.

[5] P. Wonka, M. Wimmer, F. Sillion and W. Ribarsky, "Instant Architecture". In *Proceedings of ACM SIGGRAPH 2003*, Volume 22, Issue 3, pages 669-677, July 2003.

[6] G. Stiny, "Introduction to shape and shape grammars". *Environment and Planning B 7*, 343-361, 1980.

[7] http://www.mit.edu/~tknight/IJDC/page_introduction.htm

[]