

# IMPLEMENTATION OF L-SYSTEM IN PROCEDURAL CITY GENERATION USING JAVA

**Surya Sujarwo; William Salim; Ferry Yuwono**

Information Technology Major, Computer Science Faculty, Bina Nusantara University  
Jln. Kebon Jeruk Raya No 27, Kebon Jeruk, Jakarta Barat 11530  
surya.ss@binus.edu<sup>1</sup>; wsalim@binus.edu<sup>2</sup>; fyuwono@binus.edu<sup>3</sup>

## ABSTRACT

*Article discusses about the design and implementation of procedural content generation using java, especially the generation of virtual city. It is applied by using L-System to generate the elements of the city and also using some images as the base models. This method is proven to be more effective because it can produce almost unlimited variations of city in short amount of time without any needs to modify the application. The result of this application is a road map which can be used in many areas such as virtual reality, games, or other related purposes.*

**Keywords:** *L-System, virtual city, procedural content generation, java*

## ABSTRAK

*Artikel ini mendiskusikan tentang perancangan dan implementasi prosedur isi umum penggunaan Java, terutama berhubungan dengan kota virtual. Program tersebut diaplikasikan menggunakan L-System untuk menghasilkan elemen-elemen kota dan juga menggunakan beberapa gambar sebagai model. Metode ini dibuktikan agar lebih efektif karena bisa memproduksi beberapa variasi kota yang banyak dalam waktu sedikit tanpa kebutuhan memodifikasi aplikasi. Hasil dari aplikasi ini adalah sebuah peta jalan yang bisa digunakan di berbagai are seperti virtual reality, games, dan sebagainya.*

**Kata kunci:** *L-System, kota virtual, prosedur isi umum, java*

## INTRODUCTION

In gaming industry, the game contents are considered as a very important part. Most of great games usually come with detailed and complex contents. However, it should be noted that the level of complexity of the content itself will determine the creation time needed. The more detail content is, more time will be needed to generate.

Limited amount of time usually becomes an issue in the game industries, which always run for the published deadline especially the one with a lot variation of contents. Just lately the Starcraft II that just launched needed almost 7 years to be generated due to the detail content. Usually a game should have detailed contents and also should be produced as fast as possible. For example, a game which taken cities or wide area as the places where the character is required to walk or explore, will take a great amount of time and a lot of work to do for the artists need to model every details of every road or area of those cities.

A better approach (than modeling every detail) is by using aerial imagery of the real world cities, and then the object (roads or buildings) can be extracted by using computer vision. It is indeed a great method as it can create realistic contents and will save a great amount of time. But it has its own limitation as it can only produce the real cities based on the images, whereas some games usually need a virtual one.

Based on the problems, the offered solution is using procedural content generation. The procedural content generation that will be discussed in this article uses fractal as its base concept. Fractal allows the contents to be generated in the natural forms which cannot be done by any conventional mathematic models. It can also produce something with no limitation on details. Combined with some randomness, it can produce huge number of variations output from a single input. The fractal method which will be used in this research is the L-system model.

The goal of this research is to use the concept of procedural content generation to produce roadmaps which will contain the vertices and edges information of the generated virtual cities which can be applied directly to be used in the gaming industry. During the process, it will utilize the drawings in the form of height map, water map, population density map, blocked areas map, and pattern map as the bases in the roadmap generation. This research then can be used as the reference in the game development to produce more detailed contents in the least amount of time as possible.

### Literature Review

Procedural content generation (PCG) is a method to generate the contents of the game by using some specific algorithms which executed procedurally. It usually uses random or pseudo random numbers to produce almost (pseudo) infinite unique variations of game contents. Although it is stated to be random in some ways, procedural content generator can be controlled to produce the exactly same output by using its random seed value. The same random number sequences will be produced by using same random seed (because computer uses pseudo random number); therefore it is possible to predict the output. Procedural content generator usually takes some amount of parameter to produce large amount of contents.

The word *procedural* in procedural content generation means a process to execute some specific functions by using a predefined set of rules. Procedural content generation uses many kind of methods to generate the contents. One of the methods is Lindenmayer System or also known as L-system.

In 1968, a biologist named Arstid Lindenmayer introduced a new string rewriting mechanism which then popular as L-System. In general, rewriting is a technique to define a complex object by changing the object's parts (which are originally more simple) into a more complex parts consecutively by using a set of rewriting rules and productions. The classic example of the graphical object that is defined by using the rewriting rules is the snowflake curves, which was proposed in 1905 by Von Koch (1870 - 1924).

Benoît B. Mandelbrot restated the construction of snowflake curves which is started with two shapes, one act as the initiator, and the other act as the generator. The generator is an oriented broken line made of  $N$  lines with equal sides of length  $r$ . Each construction phase started with a broken lines which then continued by replacing each straight line with a copy of the generator. The copy is rescaled and reoriented so that it has the same start and end points as the replaced line. The construction concept of snowflake curve is shown in figure 1.

As it is easier to study the rewriting system by using string than a graphical representation, there were several concepts of rewriting system appearing, including Chomsky grammar and L-system. The main difference is how the productions are applied to the existing string. Chomsky grammar replaces the 'string in a production' sequentially whereas L-system replaces 'all the string' simultaneously in parallel.

The simplest class of L-System which are deterministic and context free, called DOL-System. For example, assume that a string is formed by two letter, 'a' and 'b', which can appear multiple times in a string. Every letter has its own rewriting rule. The rule  $a \rightarrow ab$  means that every letter 'a' will be replaced by string 'ab', and the rule  $b \rightarrow a$  means that every letter 'b' will be replaced by string 'a'. The rewriting process starts from a string called axiom. Assume that the axiom contains the letter 'b', in the first rewriting phase, axiom 'b' is replaced by 'a' using production  $b \rightarrow a$ . In the second derivation, 'a' is replaced with 'ab' from the production  $a \rightarrow ab$ . String 'ab' has two letters, both of them will be replaced simultaneously and it will produce 'aba'. Using the same method, 'aba' will produce 'abaab', then it will become 'abaababa', then 'abaababaabaab', and so on. The detail of replacement process is shown in Figure 2.

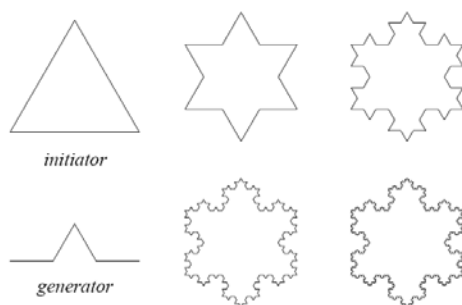


Figure 1 Construction of snowflake curve

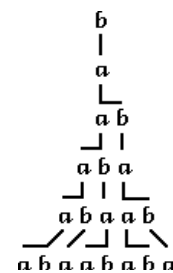


Figure 2 Example of a derivation in a DOL-system

Assume that  $V$  is an alphabet,  $V^*$  is a set of every words of  $V$ , and  $V^+$  is a set of non-empty words of  $V$ . A string in OL-system is an ordered triplet  $G = \{V, \omega, P\}$  where  $V$  is the alphabet,  $\omega \in V^+$  is a non-empty word which is called axiom, and  $P \subset V \times V^*$  is a finite set of productions. A production  $(a, \chi) \in P$  written as  $a \rightarrow \chi$ . The letter 'a' is a predecessor and the letter  $\chi$  is the successor of the production. It is assumed that for any letter  $a \in V$ , there should be at least one word  $\chi \in V^*$  such as  $a \rightarrow \chi$ . If there is no production specified for a given predecessor  $a \in V$ , the identity production of  $a \rightarrow a$  is assumed as the part of production  $P$ . An OL-system is deterministic (noted DOL-system) if and only if for each  $a \in V$  there is exactly one  $\chi \in V^*$  such that  $a \rightarrow \chi$ .

Let  $\mu = a_1 \dots a_m$  be an arbitrary word over  $V$ . The word  $\nu = \chi_1 \dots \chi_m \in V^*$  is directly derived (or generated by)  $\mu$ , noted  $\mu \Rightarrow \nu$ , if and only if  $a_i \rightarrow \chi_i$  for all  $i = 1, \dots, m$ . A word  $V$  is generated by  $G$  in a derivation of length  $n$  if there exists a developmental sequence of words  $\mu_0, \mu_1, \dots, \mu_n$  such that  $\mu_0 = \omega$ ,  $\mu_n = \nu$  and  $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$ . The sample implementation of DOL-System is shown in Figure 3.

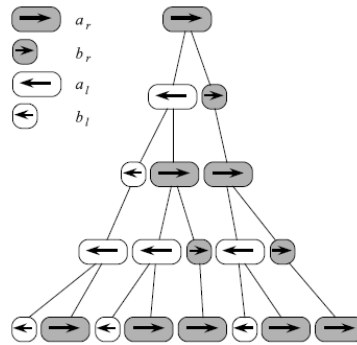


Figure 3 Development of a filament (Anabaena catenula) simulated using a DOL-system

L-system can be used to visualize the structures by embedding graphical symbols in the string that can be later used for rendering. Turtle commands can be used to describe and visualize a wide range of L-systems including Koch's snowflake, plants and branching structures. The concept behind Turtle Graphics is like a simulation to give a 'turtle' certain given instructions relative to its current position and as it moves it leaves a pen line mark (trail) behind it. By using turtle graphics, shapes, drawing and structures can be defined in the terms of an L-system. Using a bracket extension to Turtle Graphics, L-systems can support the branching structures such as trees that are predominant in nature. The interpretation of turtle graphic is shown in figure 4.

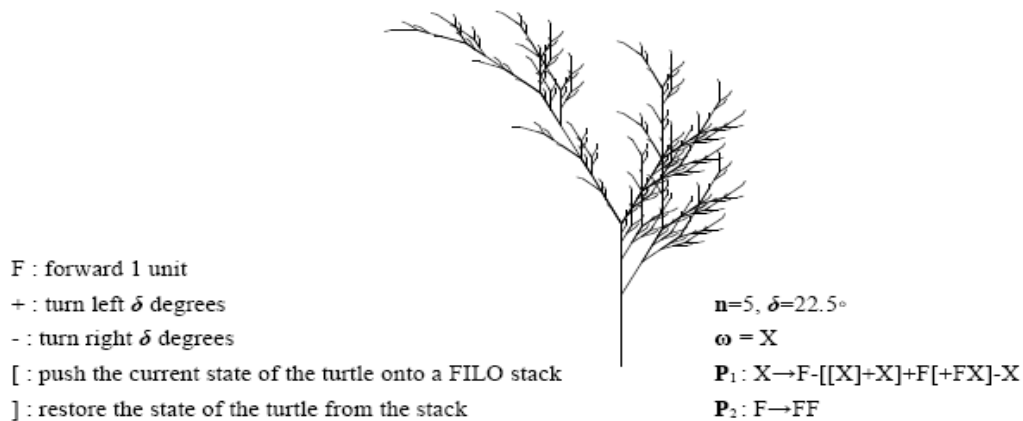


Figure 4: Tree formation generated with via the turtle graphics L-system interpreter.

## METHOD

This research process applied two methods, literature and laboratory study. Literature study is done by finding the resource from internet, documentations, and articles which are connected to the

algorithm that is discussed in this research. Laboratory study is done by testing the software speed to generate contents on two computers, each with different specification.

## DISCUSSION

### Designing The System

The system is designed to allow automatic construction of roadmap based on some predefined procedures and predetermined input. It will form a workflow as shown in figure 5. The system itself will begin its operation by creating the base road map which consists of the main roads and the streets. It is done by using extended L-system.

The extended L-system only provides the basic template at each stage of production. The modification of parameters is done by the external functions. This way, the system grows more flexible. It allows the separation of global goals and local constraint. Usually, when a predecessor produce its successor based on the production rule, the parameters haven't set yet. When the global goals are applied to the successor, it will determine the values of the parameters. The system then will apply the local constraints to modify the parameters based on specific environment. This way, the produced successor will have different parameters according to its environment.

The next stage is to remove bad edges. Bad edges include the roads that are intersecting the other roads, which shouldn't happen, and also the roads that have the dead end. From the produced roadmap, the curbs are generated to give thickness to the roads. It will create blocks. The blocks then subdivided into smaller spaces which then are used to place the buildings.

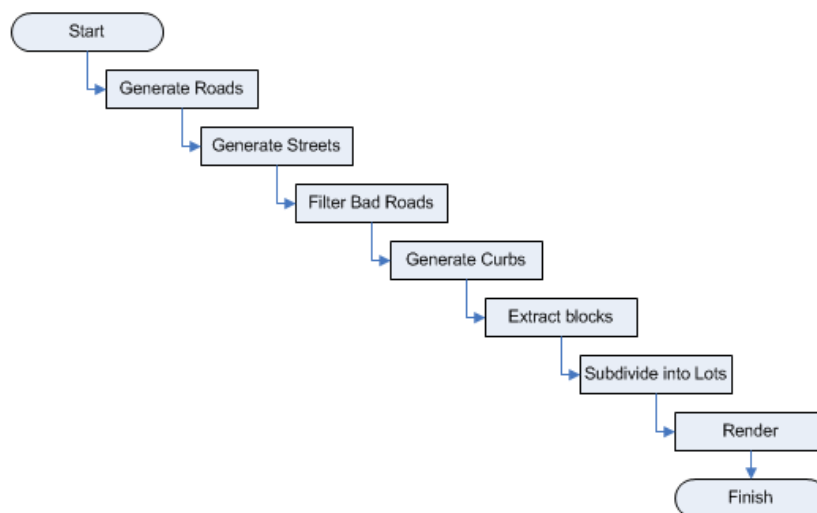


Figure 5 The workflow of procedural city generation

This research will use simple production rules to produce the desired roadmap. It will have two alphabets R and B. The letter R will have 4 parameters and letter B will take 5 parameters. The production rules are written as follows:

$$\omega : R(0, 1, \text{ROAD}, \text{ACCEPT})$$

$$p1 : R(\text{from}, \text{to}, \text{type}, \text{state}) : \text{state} == \text{DELETE}$$

$$\rightarrow \varepsilon$$

```

p2 : R (from, to, type, state) : state==ACCEPT
    → B (d[0], from,dirX[0], dirY[0], t[0])
      B (d[1], from,dirX[1],dirY[1], t[1])
      B (0, from, dirX[2], dirY[2], type)
p3 : B (delay, from, dirX, dirY, type) : delay > 0
    → B (delay, from, dirX, dirY, type)
P4 : B (delay, from, dirX, dirY, type) : delay == 0
    → R (from, to, type, state)

```

The first production will be used to terminate the production of the predecessor string. It will only happen when the state changed by the fourth production. The second production will produce three strings B. One of them has to be the main road, while the other two can be either main road or the street. The direction of each branch is generated based on the previous branch. The local constraint will be applied to modify this parameter. It will include the checking on the provided height map, population density map, and pattern map. It will re-calculate the direction of generated road based on the provided map.

The third production will decrease the delay parameter on each predecessor without changing the other parameters. The delay is used to ensure the system to apply the fourth production to the successors from the same predecessor at different time. The fourth production is responsible to modify the parameters based on the current condition. For example, if the current location is already occupied by another road or if the road intersects with the other, then it has to be readjusted by rotating or rescaling the road. It also decides the state of the successor string based on whether the new road is successfully inserted or not.

The detailed flowchart of procedural city generator is shown in figure 6. It includes all the procedures taken to create a roadmap. In the flowchart, the L-system is applied repeatedly until there's no change detected in the roadmap. Then the L-system is reapplied with slightly different parameters to create the street and fill the remaining space on the map. In the figure 6, it appears that before generating the edges, the L-system will try to attach the current vertex to the nearby vertex. If the system doesn't find any vertex, then it will look for an intersection with other edge. If the intersection is found, the new edge will be cut and intersected edge will be split.

The next step is to clean the produced map from unwanted roads, which begin with removing duplicate roads (the roads with same start and end points). There are some intersecting roads left which should also be removed. The road production usually produces the dead ends or unused vertices. Those edges and vertices won't have any use because there won't be any building generated on them. After the roadmap is clean, the curbs must be generated. This operation requires a little mathematic computation to generate vertices on the correct locations based on the position of the neighboring road. The generated vertices are then connected to form the curbs.

Usually, the curbs are connected to the other curbs, and they will form a polygon. The system must extract this information to get the list of polygons. Because it is easier to operate on convex polygon than the concave one, every concave polygon should be split into the convex one.

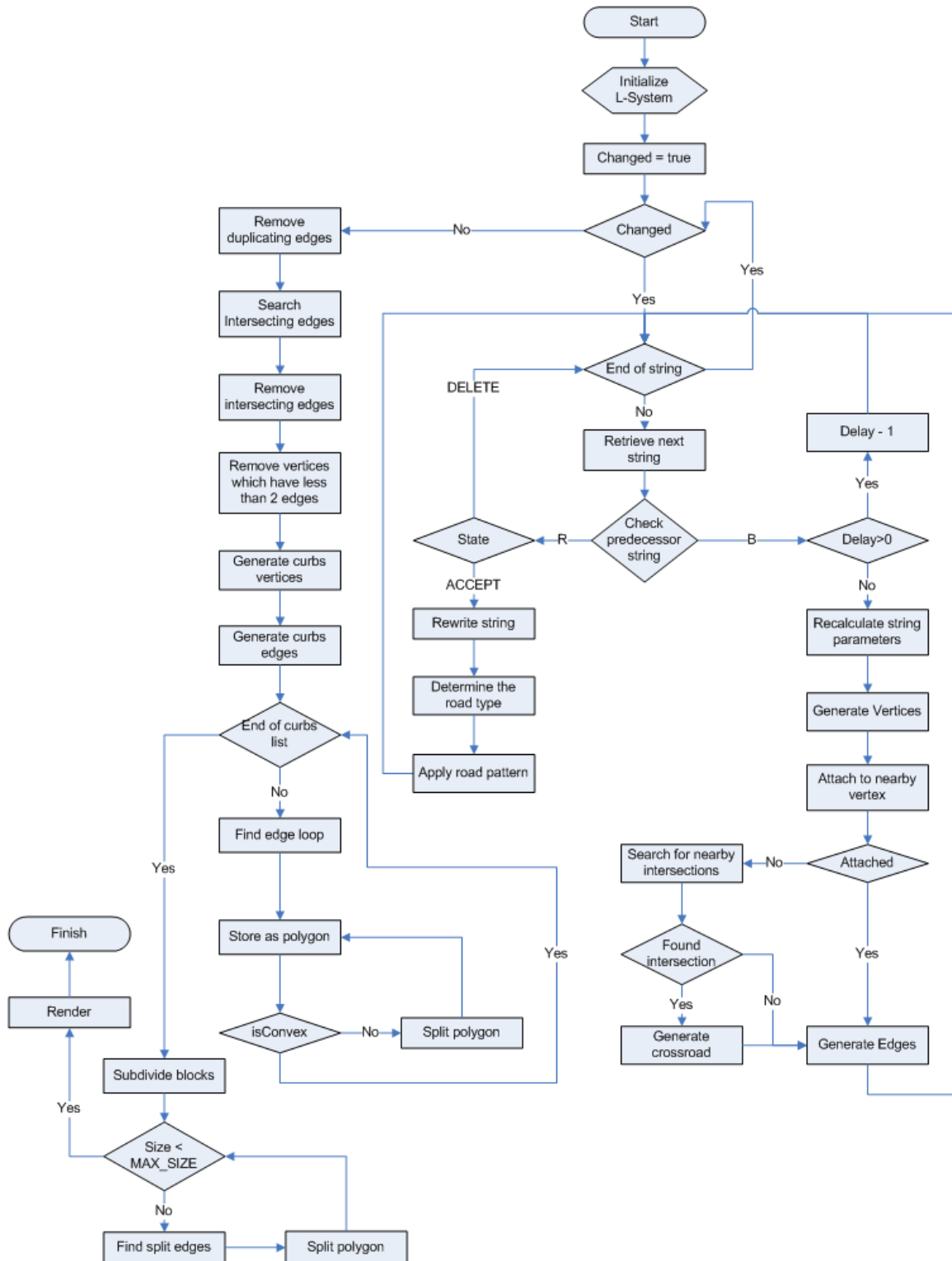


Figure 6 The detailed flowchart of procedural road generation

The last step in procedural city generation is to generate the allotments. The allotments are created by subdividing the extracted polygons. It also uses random numbers to create allotment with

different size. The created allotment usually will form a quad or triangle depending on how the original polygon shaped. The procedural city generator also has to take into account for the building which doesn't have any direct road access. Such a building has to be removed. The generated data then can be used to draw the texture or saved for future use. In this research, the generated roadmap will be used to render the images and saved as XML file so it can be used easily by the other programs.

## Implementation And Evaluation

This system is implemented by using java language. The program itself doesn't really have any interface. It only has one window to show the city generation progress. It will redraw each time if there is any change in the road map. Since the redraw operation is taking a lot of time, this feature can be toggled off to save time in the process. The program will require some images as mentioned before, height map, water map, population density map, blocked areas map, and pattern map. These maps will be the guideline so that the system knows where the roads should be generated.

This program also needs a file containing the basic parameters for road creation such as the length of road segments, random seed, initial road position, sampling rate, road width, pattern color key, etc. These parameters will affect the process entirely. For example, if the length of road segments is changed, there will be fewer roads since every road segment will take more space. Another example is if the random seed is changed then a completely different roadmap would be generated. It means the user only has to change the parameters or the maps to create any variation of roadmap. The program will start with loading all the required files including the images. Then the main roads are generated from the initial road defined in the parameter file. It runs through the L-system algorithm. After the process finished, it will generate the street. The resulting images of these two operations are shown in figure 7.

After the roadmap is generated, the system will filter the roadmap so there is no invalid road. The curbs are then generated by creating new edges around the old one. Then each block will be subdivided into smaller area that is called allotment. Notice that the allotment which doesn't have any road access is removed from the produced map. All of these procedures are implemented without using L-system. It uses general vector algorithm to find any intersection and to calculate the position of each curb's vertices. And finally, the system uses some geometry algorithm to subdivide the blocks and rescaling each allotment. The results can be seen in figure 8.

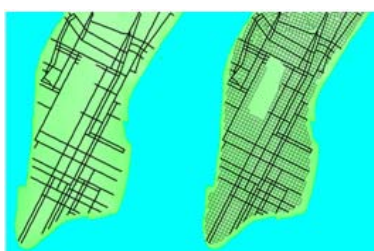


Figure 7 Generated main roads (left image) and generated streets (right image)

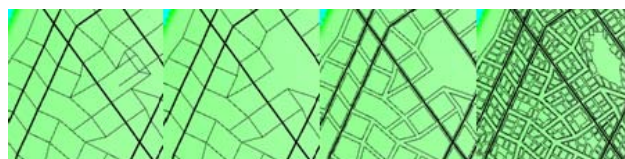


Figure 8 Original roadmap (top left), filtered roadmap (top right), generated curbs (bottom left), and generated allotments (bottom right)

As mentioned before, extended L-system is easier to expand. The parameter modification can be expanded as needed without the need to alter the production rule at all. Therefore, it is possible to apply more than one pattern to generate the roadmap. The roadmap generated in figure 7 uses raster pattern which form a grid. The system can be also programmed in some way that it will generate a



population based pattern or gradient based pattern. It also possible to combine more than one pattern in one roadmap based on the pattern map as shown in figure 9.



Figure 9 Gradient based pattern (left image) and hybrid pattern (right image)

This system is tested on two computers with different specification: (1) Computer I: Processor: Pentium 4 2.80 GHz, Memory: 1.93 GB DDR, VGA: ATI Radeon Xpress 200 64MB, Harddisk: Seagate Barracuda 80 GB, OS: Windows XP SP2; (2) Computer II: Processor: Core 2 Duo 2.66 GHz, Memory: 2.0 GB DDR2, VGA: NVIDIA Gforce 7300 GS 256MB, Harddisk: Seagate Barracuda 250 GB, OS: Windows 7.

The test is done by running the application and let it generate some variations of roadmap, then recording the time required for generating each roadmap. The time divided into a smaller part to see how long each stage of procedural city generation needs to finish its job. It includes the time to generate main road, generate street, filter road, generate curbs, extract blocks, subdivide blocks, and rendering the final images. The test is done five times on each computer. Each test will use different random seed. But the other parameters are left as they are. So there will be five unique roadmaps generated in the test. The result of the test is shown in table 1 and table 2 below.

Table 1: Test result on computer I (in millisecond)

No	Generate Road	Generate Street	Filter road	Generate Curbs	Extract block	Subdivide	Render	Total Time
1	844	2562	2751	3468	63	422	2234	<b>12344</b>
2	969	2531	2781	3360	62	422	2328	<b>12453</b>
3	891	2625	2859	3453	78	438	2297	<b>12641</b>
4	750	2594	2781	3219	63	421	2219	<b>12047</b>
5	735	2500	2734	4484	47	438	2234	<b>13172</b>
<b>Average</b>	<b>837.8</b>	<b>2562.4</b>	<b>2781.2</b>	<b>3596.8</b>	<b>62.6</b>	<b>428.2</b>	<b>2262.4</b>	<b>12531.4</b>

Table 2: Test result on computer II (in millisecond)

No	Generate Road	Generate Street	Filter road	Generate Curbs	Extract block	Subdivide	Render	Total Time
1	687	2273	3183	1215	22	143	1060	<b>8583</b>
2	738	2375	3192	1130	20	144	1076	<b>8675</b>
3	693	2279	3230	1203	21	144	1078	<b>8648</b>

4	676	2282	3201	1245	21	144	1080	<b>8649</b>
5	697	2282	3188	1150	21	144	1073	<b>8555</b>
<b>Average</b>	<b>698,2</b>	<b>2298,2</b>	<b>3198,8</b>	<b>1188,6</b>	<b>21</b>	<b>143,8</b>	<b>1073,4</b>	<b>8622</b>

This result shows that the system only needs about 12.5 seconds on lower specification computer to produce a roadmap. This is a lot faster than letting the artist to model the city manually, though it will resulting a better result than the one generated by computer. On computer II, it takes around 8.6 seconds, 31.2 % faster than the process on computer I. But for some reason, filter stage on computer II takes longer than expected.

The test also records on how many vertices and edges are generated in each unit time. The results are shown in the figure 10 and figure 11. These data show the overall progress. It shows when the vertices and edges of the roadmap are added or deleted.

All the tests above are done using five 512x512 pixels images as the base maps. The generated final rendered image is approximately 1600x2100 pixels. However another test using different maps (resulting image shown in figure 9) which is approximately 2000x2000 pixels in size takes longer time, around 50 seconds, because the maps is almost full of land so there are more places to put the road on which means more vertices and edges to process. The roadmap also uses hybrid pattern. This pattern consists of raster pattern and gradient based pattern. The gradient based pattern usually requires more time to compute since they add more steps to compute the gradient of the map for each road vertex generated.

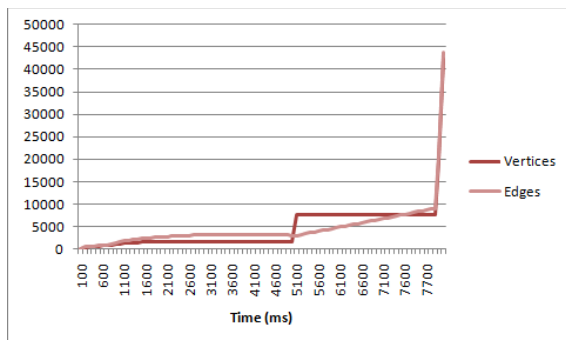


Figure 10 Generated vertices and edges on computer I

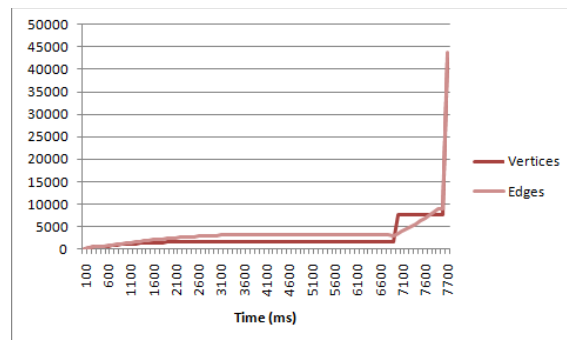


Figure 11 Generated vertices and edges on computer II

## CONCLUSION

Based on the test result, extended L-system is a very good method in the procedural content generation. It can create realistic road pattern based on the global goals and local constraints defined. The style or pattern of the generated roadmap can be changed by using different sets of global goals and local constraint. Moreover, the shape of the resulting roadmap can be altered easily by changing the parameters. The processing time is also low, the L-system itself only takes around 3 seconds to finish. However, this research is done by using java language which is still slower than C or C++.

For future research, it is recommended to create more complex and realistic road pattern by using different sets of production rules and different global goals and local constraint. A 3D visualization and some generated buildings using L-system can also be implemented to create more realistic scene.

## REFERENCES

- Greuter, S., Parker, J., Stewart, N., & Leach, G. (2003). *Real-time Procedural Generation of 'Pseudo Infinite' Cities*. In Proceedings of GRAPHITE 2003, ACM Press, 87-95.
- Kelly, G., & McCabe, H. (Dec 2006). *A Survey of Procedural Techniques for City Generation*. ITB Journal Issue: 14.
- Parish, Y. I. H., & Müeller, P. (2001). *Procedural modeling of cities*. In Proceedings of ACM SIGGRAPH 2001, ACM Press / ACM SIGGRAPH, New York, 301–308.
- Prusinkiewicz, P., Lindenmayer, A., & Hanan, J. S., et al. (1990). *The Algorithmic Beauty of Plants*. New York: Springer-Verlag.