The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

3D L-Systems Path-finding
Case study: Walking Agent's Explorations in Simulations

Djuned Fernando Djusdek[*1] Yoshiteru Ishida[*1]

[*1] Toyohashi University of Technology

This paper presents a technique for exploring map or surface using extended L-Systems based on L-Systems' grammar, other than L-Systems rules, we need this derivative rule for exploring a walkable area. The rule that used is flipping approach, which is a technique for changing the angle of generating grammar, then will impact the direction. The results are validated and done in simulation systems.

1. Introduction

Lindenmayer Systems or L-Systems [Lindenmayer 1968] is a great invention, not only for representing a simple living object, yet can for representing plants. Inspired by biological systems, L-Systems was developed more, i.e., Stochastic L-System and Context-sensitive L-Systems, which is used for represented plant more natural. Not only that, by combining L-System with Genetic Algorithm (GA), the plant representation is unique [Prusinkiewicz 1990] [Ochoa 1998] [Kurth 2007].

Even more than that, recent L-Systems is also used for the path-finding solver. Some research was published, e.g. [Areyan 2012]. In this research, they use Ants Colony as a base algorithm, combined with Genetic Algorithm. In case of uses share memory "*stigmergic*" that ants left as pheromones, also uses individual memory that contained in its genetic information. This genetic information stored as genes and represented by simple L-Systems' grammar.

However, not available yet for exploring the map, which has no route or path, e.g., Mars's surface. In this research, we implement a 3D exploration's simulations for walking agents path-finding solver in an unexplored land. This research objection is used L-Systems for path-finding. Hopefully, this research can support human for exploring the unexplored land.

The rest of this paper is structured as follows. Section 2 describes the previous research which relates to L-Systems path-finding. Section 3 presents the method we propose which the simulation and experimental result is provided in Section 4 and 5. Finally, the conclusion is drawn in Section 6.

2. Related Work

This research uses Cellular Automaton (CA) and combines it with L-Systems. Related to [Alfonseca 2003], in this research was explained the formal definitions of deterministic and probabilistic CA, with special attention to the problem of updating the probabilistic information each automaton in the grid. In this article, also introduced a formal notation for probabilistic L-Systems and the language generated by them.

Contact: Djuned Fernando Djusdek, Toyohashi University of Technology, 1-1 Hibarigaoka Tempaku Toyohashi Aichi 441-8580 JP, +81 80 9986 4087, djuned@sys.cs.tut.ac.jp

For support this research, there is another research, e.g. [Koopman 2016], that try to solve path-finding in 3D space for the indoor environment. Many path-finding algorithm and approach discussed in this research. Concluding to use voxel location, and use "*voxelized*" model for representing a 3D indoor environment. The agent also divided into several types, i.e., walking and flying agents.

Rather than focus on an existing problem like exploration by robots, e.g. [Shen 2012], we focus on algorithms to explore unexplored land in simulations. In this related work, the autonomous flying vehicle was used as agents for indoor exploration.

Moreover, about L-Systems exploration, this problem can be classified as space-filling curve problems. Described in [Nair 2017], they also done exploration task by implementing a systemic strategy for Hilbert's space-filling curve. The exploration task is to find obstacles (or holes) and implemented online, and their future goal is to use it for path planning and searching.

3. Method

As described in related work, we use L-Systems with CA to solve this problem and related to our previous work [Ishida 2017]. The environment that used is simulation systems that have been developing in OpenGL using Qt wrapper. The simulation environments are used to simulate a small world, i.e., land or island(s).

```
1.  algorithm 1: loadHeightMap( grayScale image )
2.  w ← width(image)
3.  h ← height(image)
4.  c ← heightDivider ← 10.0f
5.  for i ← 0…w
6.    for j ← 0…h
7.      g ← greyColor( imageAtPoint( i, j ) ) / c
8.      storeToMatrix( g )
9.  return
```

Figure 1. The Pseudocode of Height-map Loader.

In this case, we use a height-map as input for generating the land surface. This surface will see as a plain land without anything, except water area, i.e., a sea. The pseudocode for load the height-map is described in Fig. 1. For enriching the surface, we use Game of Life [Gardner 1970] for distributing the tree(s), and by randomly generated seed, and combining with the height of areas that can be placed, i.e., walkable area. The constraint to determining the walkable area is defined in Eq. 1, while the *h* is the height of the walkable area.

$$waterArea < h < mountainArea \quad (1)$$

L-Systems' grammar that used shown in Fig. 2. This grammar produces four directions in the beginning and will create a branch on each iteration. In the generating steps on each direction, will be generate a branch like described in Fig. 3, that *A* is equal to 1/4 from *B* and *B* is equal to 3/4 from start point in each iteration, and *A* plus *B* is equal to *F*. Either *A*, or *B* will produce new *F* and do similar way in each iteration. This grammar will ensure that four directions will be explored.

However, this grammar cannot guarantee that all areas will be explored if there are obstacles. To solve this problem, we introduce a flip approach to change the angle for branching implementation. The flipping technique causes the branch to grow left and then right in continues iteration, with some rules. This approach can guarantee almost all walkable area will be explored.

In this research, L-Systems is mainly used to predict the next point and the edge that is connecting from the current point, i.e., vertex, to the next point. This step is done by using CA. At this step, the CA try to connect a vertex by uses heuristic function to determine the path, which is possible to passing or not. If not appropriate, we need to move to the next point as near as possible from the previous position of next point. The pseudocode for this algorithm is described in Fig. 4, and the steps for creating edge is described in Fig. 5.

Different from [Nair 2017], in this previous research, the algorithm is based on Hilbert's space-filling curve, where just have one start and one endpoint. They modified Hilbert's space-filling curve so can do in space, which has obstacles (or holes). Comparing with this research, our approach cannot reach all area in space, i.e., walkable area. However, our approach can be run concurrently and independently, but still depend on the neighbor in each cell, i.e., point, when generating the path.

4. Simulation

The simulation systems built in MacBook Pro 2017 and Acer M5 481TG, this will ensure that our simulator can run in multi-platform. The specifications for this simulation systems are described:

4.1 Hardware Specifications

In this research, the simulation built to focus on:

- MacBook Pro (13-inch, 2017), with specifications:
  - Processor: 3.1 GHz Intel Core i5-7267U
  - Memory: 8 GB 2133 MHz LPDDR3
  - Graphics: Intel Iris Plus 650 1536 MB
- Acer M5 481TG, Windows 10, with specifications:
  - Processor: 1.7 GHz Intel Core i5-3317U
  - Memory: 6 GB DDR3 SDRAM
  - Graphics: Nvidia GeForce GT 640M LE 1 GB

$$R \rightarrow [F][+F][++F][+++F] \qquad \delta = 90°$$
$$F \rightarrow (3/4)[+F](1/4)[F] \qquad \theta = 45°$$

Figure 2. The L-Systems' Grammar.

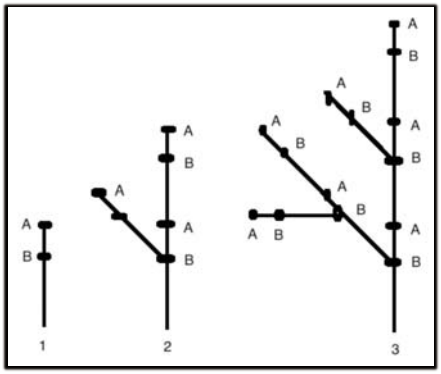

Figure 3. The L-Systems' Grammar in action.

```
1.  algorithm 2: pathMaker( current node )
2.  f ← locationOfCurrent( node )
3.  s ← locationOfCurrentParent( node )
4.  cur ← s
5.  while cur != f
6.      i, d ← findNearest( cur )
7.      if i = false
8.          return
9.      setVisited( d )
10.     storeToPath( d )
11.     cur ← d
12. return
13.
14. algorithm 3: findNearest( current point )
15. for n ← neighbor
16.     if nearestThan( n, point )
17.         t ← n
18. if isVisited( t ) || isDestination( t )
19.     return ( false, t )
20. return ( true, t )
```

Figure 4. The Pseudocode of Path-maker.

4.2 Software Specifications

This simulation system builds on:

- Mac OS 10.13 High Sierra & Windows 10 Home;

- Qt Creator 4.4.1, based on Qt 5.9.2 (Clang 7.0 (Apple), 64 bit) for Mac OS;
- Qt Creator 4.5.0, based on Qt 5.10.0 (MSVC 2015, 32 bit) for Windows;
- Modern OpenGL Shading Language (GLSL), which is implemented in Qt wrapper and native function; and
- Modern C/C++ programming language.

5. Experiments

The experiments were running on MacBook Pro and Acer M5 481TG too, with similar specifications.

5.1 Parameters

The parameters in detail:

- Game of Life generations (iterations) for determining tree(s) locations;
- The seed is for initiating Game of Life will be generated randomly. For experiment will be statically defined;
- The fixed location for start point *S*;
- The fixed locations for destination object *D*;
- Height-map that will be used;
- The height divider *div* for height-map to 3D;
- The walkable area constraint;
- The length *F*;
- The angle θ; and
- The maximum-paths that will be shown.

5.2 Environments

The environments were built refer to the approach that described in methodology, with some characteristics:

- Generated in the first run and uses the defined parameters;
- The trees, the destination and the initial point will generate by using the results of Game of Life map, which reduces to 1:5. So, in 5x5 points area will draw only one tree or destination's object or start point *R* for L-Systems;
- The destination is blindly known by *R*. The area for the destinations' object is 15x15 points; and
- The solutions paths will be drawn in 3 colors: red, green and blue.

5.3 Testing Scenario

In this research, we try to run our experiments as general as possible. First, we compare the result between Mac OS 10.13 and Windows 10. Second, we will focus on the results of L-Systems' grammar and this derivative.

To be fair, we consider using the smaller height-map and static seed for generating the environments. Then for testing the L-Systems' grammar and this derivative, we will consider the parameter θ for creating the branch and adding some rules for growing the paths.

5.4 Results

In this sub-chapter, we try to explain briefly about the experiment result. This experiment used, as described:

- Generation = 324;
- Seed = 0;

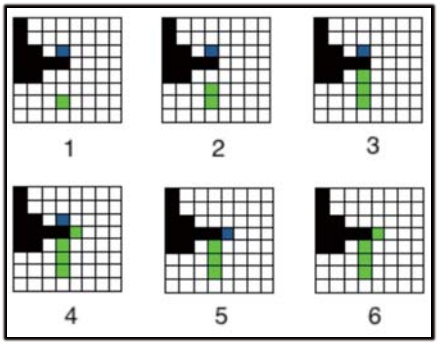

Figure 5. The Path that create by Cellular Automaton.

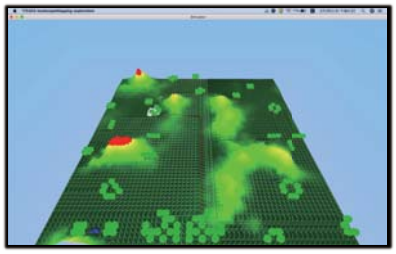

Figure 6. The Results when running in Mac OS 10.13.

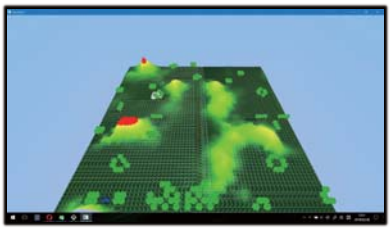

Figure 7. The Results when running in Windows 10.

- *S* = 75 (the tree at position 75th will be replaced);
- *D* = 30 (the tree at position 30th will be replaced);
- The height-map size is 257x257 pixels[1]. So, the surface will have size 257x257 points area;
- *div* = 10.0f (floating points);
- The walkable area constraints:
- Water Area < 1; and
- Mountain Area >= 6;
- *F* = 4 points (steps);
- θ = 45°; and
- The maximum-paths = 3.

First, we consider the environments that generated in Mac OS 10.13 and Windows 10. As described in Fig. 6 and Fig. 7, the environments are similar even in the different platform. This experiment gives a positive result that the paths successfully reached the destinations. For efficient time usage, we consider using Mac OS 10.13 for testing the L-Systems' grammar and this derivative rather than use both.

Consider the Fig 8, the paths also successfully reached the destinations. However, in Fig. 6, Fig. 7 and Fig. 8 cannot guarantee that all walkable areas are explored. To solve this problem, we consider changing the rule for generating branch and focus on

[1] http://www.rastertek.com/pic4005.gif

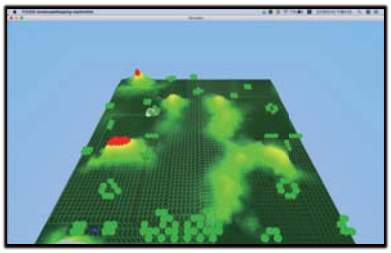

Figure 8. The Results when running in Mac OS 10.13 (θ = - 45°).

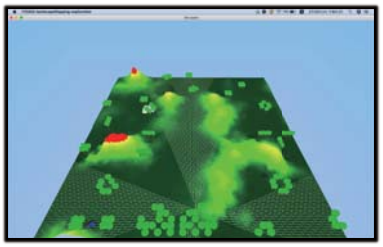

Figure 9. The Results when running in Mac OS 10.13 (θ = - 45°, flip).

angles that used. Previously, we use similar angle for generating on each iteration. Then we try to use some mechanism to make the angle flipping in next iteration and flip again. The flipping is only from -45° become 45° and vice-versa.

The results of flipping approach are described in Fig. 9, Fig. 10 and Fig. 11. The first rule that implemented in Fig. 9 is flip, that means always flipping in each iteration. Than in Fig. 10, the rule is flip-keep-flip-keep, that means in the first iteration will be flipping and in the second iteration will similar with the first iteration, then flip again and so on. Also, in Fig. 11, the rule is similar with Fig. 10, but the time for flip-keep is different, where the rule is keep-flip-keep-flip.

From Fig. 9, Fig. 10 and Fig. 11, the positive results are reached when we use the second and the third rule, which is described in Fig. 10 and Fig. 11. However, in the Fig. 9, the first rule cannot explore all area, even to create the path also impossible when surrounded by obstacles.

6. Conclusions

Path-finding in an unexplored land is a new area for developing an L-Systems based path-finding. Many possible solutions and challenges are still wide open to solve this problem.

The L-Systems' grammar that used was shown, that L-Systems can be used for exploring the surface. Then, can generate the path from source to destination. By combining with CA, growing the paths is more flexible and aware of the environments, which is if there is an impassable area, e.g., barriers such as a tree(s), or a visited location, we can reallocate the next point that will be created into a possible area. However, this grammar cannot guarantee that all walkable area will be explored. But this problem was solved by implementing the flipping mechanism that allows changing the direction of growing branch in each iteration, that can guarantee almost all walkable area explored.

Future work entails the implementation of GA as L-Systems' grammar modifier. Then, improve the algorithm so that it can be applied to the flying agent.

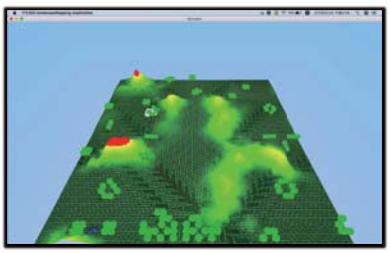

Figure 10. The Results when running in Mac OS 10.13 (θ = - 45°, flip-keep-flip-keep).

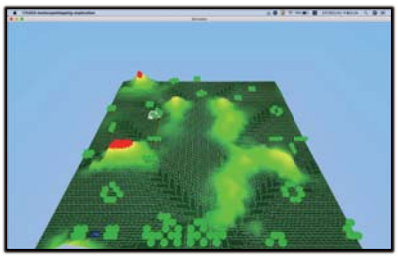

Figure 11. The Results when running in Mac OS 10.13 (θ = - 45°, keep-flip-keep-flip).

References

[Alfonseca 2003] M. Alfonseca, A. De la Puente and A. Suárez, “Cellular Automata and Probabilistic L Systems: An Example in Ecology,” 2003, 111-120.

[Areyan 2012] E. Areyan, “EvolvingAnts,” School of Informatics and Computing, Indiana University, 2012, in press.

[Gardner 1970] M. Gardner, “Mathematical Games - The fantastic combinations of John Conway’s new solitaire game “life””, Scientific American, 1970, pp.120-123.

[Ishida 2017] Y. Ishida, “Path-finding and path-finding system based on the extended L-system”, Innovation Japan, 2017.

[Koopman 2016] M. Koopman, “3D Path-Finding in A Voxelized Model of an Indoor Environment,” TU Delft, 2016.

[Kurth 2007] W. Kurth, “Specification of morphological models with L-systems and relational growth grammars,” Image – Journal of Interdisciplinary Image Science, 2007, vol.5.

[Lindenmayer 1968] A. Lindenmayer, “Mathematical models for cellular interaction in development, Parts I and II”, Journal of Theoretical Biology 18, 1968, pp.280–315.

[Nair 2017] S. H. Nair, A. Sinha and L. Vachhani, “Hilbert’s Space-filling Curve for Regions with Holes”, IEEE 56th Annual Conference on Decision and Control (CDC), 2017.

[Ochoa 1998] G. Ochoa, “On genetic algorithms and lindenmayer systems,” in International Conference on Parallel Problem Solving from Nature V, Amsterdam: Springer, 1998, pp. 335-344.

[Prusinkiewicz 1990] P. Prusinkiewicz and A. Lindenmayer, “The Algorithmic Beauty of Plants", New York: Springer-Verlag GmbH, 1990, pp.1-50.

[Shen 2012] S. Shen, N. Michael and V. Kumar, “Autonomous indoor 3D exploration with a micro-aerial vehicle”, International Conference on Robotics and Automation (ICRA), 2012, pp.9-15.