Procedural City Generation Using L-Systems

Austin Prusik
University of New Hampshire College of Engineering and Physical Sciences

Abstract

Procedural generation is the process by which complex structures are created from a given set of rules applied to what has been generated so far. It can be used to intelligently create networks that would not be possible otherwise.

Cities are very organic structures grown by human expansion over vast amounts of time. Procedurally generating these structures in a believable fashion using a specific set of rules governed by a computer can be a difficult and interesting experiment.

Realistic city growth depends on countless factors including road access/travel times, population preferences, district zoning, elevation, and many more.

This project partially replicates some of the methods used in the 2001 paper: "Procedural Modeling of Cities" from Yoav I H Parish and Pascal Müller to create a two-dimensional road map of a city. This implements population density and a street-to-highway hierarchy using self-sensitive L-Systems with Global Goals and Local Constraints.

Specification

The city generation model used for this project creates a road network based off a population density map. In areas with high population density, smaller (street) roads are created, while larger (highway) roads connect these dense population areas.

The program produces a visualization window which displays the generated road segments.

Implementation

Possible Implementations

A detailed analysis of different techniques for procedural generation of both city road maps and buildings was created by George Kelly and Hugh McCabe in their paper: *A Survey of Procedural Techniques for City Generation*. As such, only a brief summary of the relevant road network techniques will be given here.

L-Systems

In their paper: *Procedural Modeling of Cities*, Parish and Müller describe the process of road network generation using "Self-Sensitive" L-Systems, which follows a function of Global Goals and Local Constraints.

These L-Systems can produce very complex structures and are often used for organic modeling. Because of this, they provide a good model for city generation which follows similar patterns to nature due to the rules governing its construction being highly abstracted and complex.

This can be done relatively quickly, though with large sets of segments (20,000+) it can no longer be considered a real-time approach.

Simple Grid

A simple grid structure can be used to generate a road network, with the added complexity of blocks possibly spanning multiple grid squares, however this method generates very uniform and non-realistic road systems. Real cities result from multiple structures and pattern variation, as well as unpredictable anomalies, so few actual cities remain a perfect grid for extended areas.

A benefit of the grid system, however, is that is extremely fast, easy to implement, and simple to comprehend.

Agent Based Simulation

An Agent Based approach to city generation uses several AI systems to act as city planning entities, which can create very realistic and varying city structures, with the added bonus of having a natural growth pattern, however this method is extremely slow and requires several minutes to generate relatively small road structures compared to the other methods.

Template Based Generation

Several templates based on a raster grid, a radial pattern, or a population density map can be applied to a heightmap and terrain data to create large road maps. This method generally conforms to the set template but alters road direction and shape to conform to the landscape and population.

This method produces realistic road structures for very large areas, especially in areas of small detailed landmasses, however the structure lacks much of the complexity and smaller road structure of the other methods and is better suited for highway generation only.

Pursued Implementation

The perused model implemented for this project was the L-System approach described by Parish and Müller.

Data Representation

Road segments are represented as a series of line segments, associated with metadata. Each line segment is composed of a point (Point type) and a vector (Vec type) provided in the Doodle Library for Scala (Welsh, 2019).

Each point is also stored in a kdTree of dimension 2 for faster lookup.

Once all line segments are generated, they are converted to Doodle Paths, combined into a single image, and drawn in a window. Green lines indicate roads where the population falls below the threshold to create streets. Blue lines indicate road segments labeled as highways.

A population density map is created using a triple-layered Open Simplex Noise generator (KdotJPG, 2014).

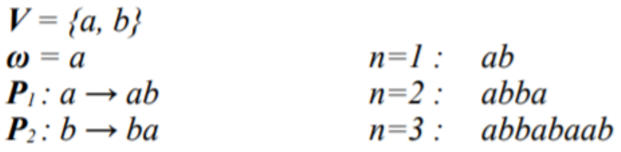
Several constants are provided at the beginning of the "citiesApp.scala" file which may be manipulated; however, some will drastically change the accuracy of the results.

General Algorithm

Parish and Müller describe L-Systems as such:

> An L-system is a parallel string rewriting mechanism based on a set of production rules. Each string consists of a number of different modules which are interpreted as commands. The parameters for these commands are stored within the modules.

Kelly and McCabe furthur explain this and offer the following table as an example:



$\boldsymbol{V} = \{a, b\}$		
$\boldsymbol{\omega} = a$	$n=1$:	ab
$\boldsymbol{P}_1 : a \rightarrow ab$	$n=2$:	$abba$
$\boldsymbol{P}_2 : b \rightarrow ba$	$n=3$:	$abbabaab$

Figure 1: An L-System: The Thue-Morse System (Kelly and McCabe, 2006)

However, it is argued (Mansfield-Williams, n.d.) that a full L-System implementation is an overcomplication which can be simplified down to the following algorithm (figure 2) which still captures the function of L-Systems, as well as Parish and Müller's Global and Local Constraint Systems.

In figure 2, r is a road segment, with arguments ti: the order of creation, ri: a geometric line-segment, and qi: any associated metadata. (Mansfield-Williams, n.d.)

```
initialize priority queue Q with a
       single entry: r(0, r0, q0)
initialize segment list S to empty

until Q is empty
  pop smallest r(ti, ri, qi) from Q
  accepted = localConstraints(&r)
  if (accepted) {
    add segment(ri) to S
    foreach r(tj, rj, qj) produced by
        globalGoals(ri, qi)
      add r(ti + 1 + tj, rj, qj) to Q
  }
```

Figure 2: A simplified version of Parish and Müller's L-System specification as used by Tobias Mansfield-Williams

Local Constraints

The Local constraints function treats both highways and streets the same. It checks each new segment to see if it ends nearby to a road intersection that has already been generated, and if it does, it connects the segment to the crossing.

If no nearby intersections are found, the segment is extended to twice it's length, and if it intersects another line segment, the road is trimmed and connected to the other segment at the intersection point.

In both cases, a road ceases to grow after meeting another road.

Global Goals

The Global Goals function generates new highways which trend towards areas of high population. When in an area of high population, these highways have a chance to branch and form new highways, as well as a much higher chance of branching into a street.

A Street will continue branching at 90-degree angles until it hits another road, or the population density becomes lower than the set threshold.

Evaluation

It can be difficult to evaluate the "correctness" of such an algorithm. In reality, there are countless factors effecting the layout of cities. However, some approximation can be formed to test at least some level of plausibility for the generated city. This approximation can be made by isolating some of the most important factors involved in the layout of a city and testing the generated model against them.

Reachability

Each vertex on the map must be reachable by every other vertex; All roads in a city are connected, and real cities generally don't have areas which are unreachable.

Efficiency

The time it takes (distance along roads) to reach any two vertices on the map should roughly scale linearly with the Euclidian distance between the two points; two points that are right next to each other should not require more movement than two points a good distance apart. This test may involve weighting the "cost" of travel depending on the complexity of the generator reached; possible weights are described as follows:

- *Main Roads* – There should be less of a travel cost for moving along main (larger) roads rather than side (smaller) roads. This mirrors actual roads and affirms the creation of main "arteries" that carry larger amounts of traffic.
- *Population Density* – If population density is included in the generation, areas of high population can have more efficiency than areas of low population; it may take longer to get between two rural locations as opposed to two urban locations even if they are the same Euclidian distance apart. This test can be accomplished by putting more weight on path cost in high population areas.
- *Point of Interest* – If POIs are used in map generation (town hall, courthouse, square, etc.), these locations should require less cost to get to from another random vertex than the cost it takes to get from one vertex to another at the same Euclidean distance. This is a soft requirement, so it should just happen most of the time, it does not necessarily need to happen in every case.
- *Elevation* – If elevation is implemented, roads should follow a path of least elevation change in general; this should have a cutoff point where small changes in elevation don't generally affect road shape. Above the cutoff, roads should never move perpendicular or close to perpendicular to the slope of an elevation change. This can be tested by checking that the roads do not exceed a certain height change per unit distance along their path.

Assessment

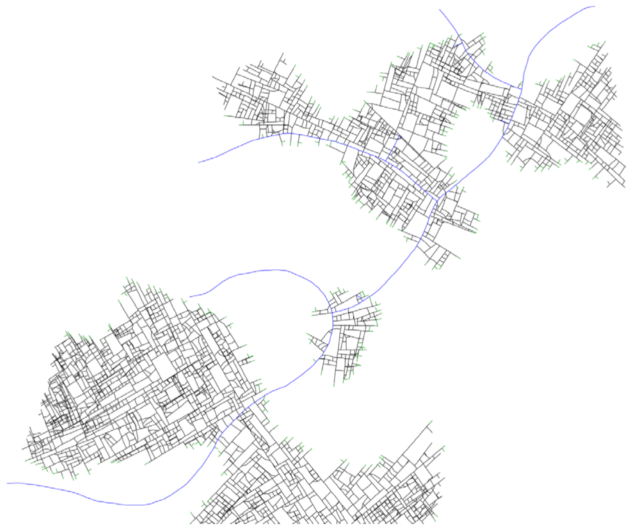


Figure 3: The road map generated with current default settings

The produced road maps (figure 3) are relatively similar to that of Mansfield-Williams' procedural city generator, with areas of high population having sets of streets connected by longer highways.

The population density map is not particularly realistic, as it is generated using 3-layer simplex noise, which is not similar to real-world population densities, but it functions well as a proof of concept.

In some areas, street generation can get messy and overlap other streets (figure 4). This is most likely due to not searching all nearby points for intersection, and only checking the closest.

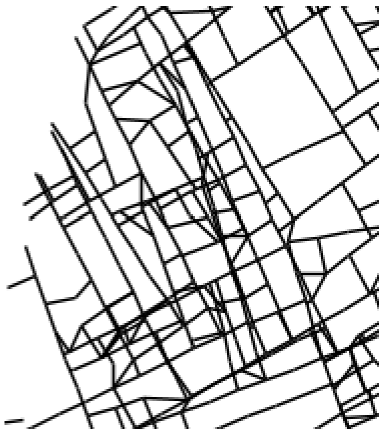


Figure 4: An example of messy street generation

Another issue comes from the localConstraints function where road connections formed from checking nearby intersections can occur at very extreme angles, which do not often occur in city blocks, and causes the creation of small "sliver-like" blocks (figure 5).

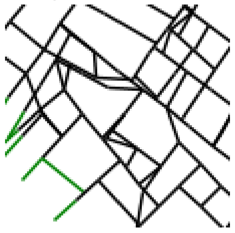


Figure 5: An example of incorrect intersection handling

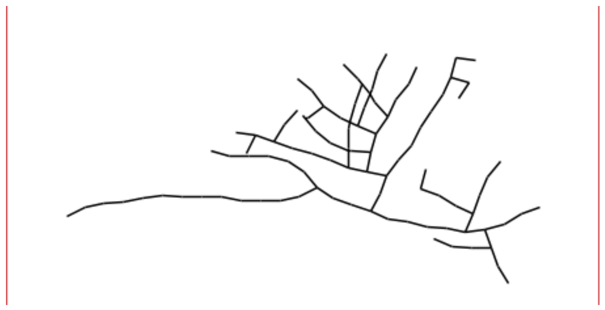


Figure 6: An example of highways with a high branching factor acting as streets. This produces something like a town map.

When generating maps using only highways with a high branching factor and no population map (figure 6), very realistic roadmaps of an alternate type can be generated, particularly within small areas.

Future Work

There is plenty of room for expansion in this project, and simply manipulating the constants like branching factors can significantly alter the output with varying interesting results.

Some extensions that can be perused is the addition of a heightmap as a road creation constraint, as used by Parish and Müller as well as water, parks, or other illegal areas on which roads cannot be generated.

Alternate patterns could also be implemented such as concentric circles or Vorni-like subdivisions instead of grid structures for streets, as well as combining these different patterns to create gradient areas.

More complex features could be implemented, such as the ability for a user to alter the generated map and provide constraints such as locked roads, and Point of Interest lots such as town halls, public squares, etc.

More robust evaluation functions could also be generated such as point-to-point navigation searching preferencing highways, and other systems described in the "Evaluation" Section.

Conclusion

Using L-Systems for procedural city generation allows for a vast amount of customization of features using the globalGoals and localConstraints functions. And many parameters can be abstracted as constants to allow for user tweaking and creating structures which are unrecognizable from each other.

L-Systems can create detail similar to that of Agent-Based methods, but with far less computational complexity and comprehension.

The Street to Highway hierarchy requires a significant amount of testing to create an accurate depiction, however a high-way only approach can be a simpler way to create good road networks, though they theoretically couldn't approach a more ideal 2-stage model with both streets and highways.

References

KdotJPG, 2014. Noise!. [Online] Available at: https://gist.github.com/KdotJPG/b1270127455a94ac5d19

Kelly, G. & McCabe, H., 2006. A Survey of Procedural Techniques for City Generation. ITB Journal, Issue 14, pp. 87-130.

Mansfield-Williams, T., n.d. Procedural City Generation. [Online] Available at: https://www.tmwhere.com/city_generation.html

Parish, Y. I. H. & Müller, P., 2001. Procedural Modeling of Cities. SIGGRAPH, p. 8.

Shaker, N., Togelius, J. & Nelson, M. J., 2016. Procedural Content Generation in Games. Cham: Springer International Publishing AG.

Watabou, 2019. Medieval Fantasy City Generator. [Online] Available at: https://watabou.itch.io/medieval-fantasy-city-generator

Welsh, N., 2019. Doodle: Compositional Vector Graphics, s.l.: Github.