City Visualization Using Procedural Generation

Hannah Abraham[1], Shreyas Yakhob[2], Kaustav Ghosh[3] and Merin Thomas[4]
[1]*hannahabraham23@gmail.com*, [2]*shreyas.yakhob@gmail.com*,
[3]*ghosh1k4@gmail.com*
Department of Computer Science, Christ (Deemed to be) University
Corresponding author:[4]*merin.thomas@christuniversity.in*

Abstract
Cities and giant metropolis are a wonder to the average person. It takes several man hours to plan the urbanisation of an area and multiple amendments to make it fit the current as well as future demographic and political scenarios. It involves allotting areas for residential living, parks, public buildings (like post offices, hospitals, police stations etc.) and commercial areas. It also includes road networks to connect all the aforementioned places in a way that is efficient and convenient. This takes designers and planners weeks to do the right calculations, model the structures and assign them in an ideal way. Procedural generation is a method that helps to create content algorithmically and in large amounts. This paper aims to create an efficient workflow that attempts to lay out a terrain, blocks and other components to generate a full city. It consists of three work modules namely, designing the buildings algorithmically, coding life-like textures and craft a competent road network. The modules are then integrated to display a city that can be altered by editing a seed value to generate and display another blueprint. This way, the paper aims to achieve multiple city visualizations for comparison and selection. The techniques used in this paper such as Perlin noise, fractals and L-systems help to realise the aim.
Keywords: *Procedural Generation, Perlin Noise, City Visualization*

1. Introduction

A city is defined as a large human settlement, one with a relatively permanent and highly organized centre of population, of great size and importance. The size comes with the settlement but the importance comes with planning. Planning a city is an intensive task, one that requires a fairly formal education in the field of urban planning or civil engineering. Placement of hospitals, police stations, fire stations etc. within the vicinity is one of the jobs of a city planner. In case of heavy rainfall, there are drains on every nook and corner of major streets, and entrances to buildings as well as the road designs are elevated in such a way so as to direct the flow of water to these drains. If a particular area is experiencing traffic congestion, then there are alternate routes to get from point A to point B. Such carefully planned out cities require a lot of work. Procedural generation algorithms can help generate buildings, cities and large-scale settlements with ease and can do so efficiently. Such algorithms can rapidly generate large scale cities and with the right model, we can customize and tailor the generated city to our needs. This paper aims to undertake this task of developing such a procedural generative model. The objective of the paper is to simulate the construction of a city, the growth of civil infrastructure in a practical and efficient manner using procedural generation algorithms.

2. Literature Survey

This section gives an overview of related work done on procedural generation of cities. In the paper, 'Procedural Generation for Architecture' [1], Artur Alkaim lists explains a 3D modelling software that helps to generate large amounts of geometry. The road network takes land boundaries as input and a connection of roads is created. It uses an L-system that computes this network. The buildings are created using some grammar rules or a language (grasshopper 3D) similar to visual language. They are modelled by applying

techniques such as rotation, scaling and translation to basic shapes namely T, I, U, H and L. The building roofs are also generated similarly out of basic shapes.

In the paper, 'A Survey of Procedural Techniques for City Generation' [2] by George Kelly and Hugh McCabe lists out several techniques to generate geometry, textures and effects. Fractal mathematics was the first technique described in the study. They're used for their self-similarity features which help to imitate natural shapes such as clouds and mountains. Fractals are generated from a recursive algorithm which on successive recursions, gives it a more natural looking shape. The second technique described is the L-system or Lindenmayer system which was also used in [1]. It utilizes the concept of rewriting or replacing. It writes over initial objects using a set of rules. Some of its components include V (a set of symbols that represent the initial elements to be replaced), S (a set of symbols that represent the fixed elements), ω (a string that defines the initial state) and P (the set of rules or productions that guide the rewriting). Perlin noise is another technique that is explored. This technique which is also exploited in this paper uses the randomness of noise to generate various textures such as marble, wood etc. It makes use of variations in amplitude and frequency to come up with multiple effects.

Wonka et al. in their study, 'Procedural Modelling of Buildings' [3] talk about the CGS shape which is a set of rules that help geometry to evolve into a design with every iteration. The rules would first generate a raw structure called the mass model and would then go on to decorate it with doors, windows etc. The grammar works with a configuration of basic geometrical shapes, where each active shape is termed as a symbol. Then, a production rule is chosen with that symbol on the left-hand side to generate its successor. Once that successor is formed, the initial symbol is made inactive and the new one is added to the configuration. The process terminates when there are no non-terminals left in that configuration. This technique of modelling building using production rules is also severely used in the paper.

In the paper, 'Procedural Generation of Roads' [4] by E. Galin et. al. an automatic method for the generation of roads is devised. This is done using anisotropic shortest path algorithm which minimizes the cost function and searches paths formed by the joining of straight lines between assigned points. In a complex graph, the algorithm can generate tunnels and bridges too. The paper also derives formulas for continuous and discrete shortest path algorithms, cost function, surface roads, bridges and tunnels.

'Real-time Procedural Textures' [5] by John Rhoades describes a system that allows users to play around with the sharpness, transparency, diffuseness etc. of textures or images. It outlines a texture editor that uses an assembly language-like instruction set called T-codes. T-codes are of three kinds namely, generators, operators and conditionals. Generators use the famous Perlin noise function to produce various basic patterns, operators perform arithmetic operations on these basic patterns and conditionals select certain pixels to include or exclude in the entire computation. The system takes care of pixel memory management and caching too. The paper also goes on to describe some dynamic textures using time variables to give movement results such as flames or ripples.

3. Methodology

3.1. Perlin Noise

Textures: The first and simplest way to give texture to a scene is using an image and wrapping it around an object or using Photoshop to create an image and do the same. The problem with this method is that it comes at high memory cost and large execution time. What happens is that these image textures are associated with processors which all save a copy of the texture so that any free processor can load it in the scene. Although resource usability is maximum, memory wastage is too. Hence, procedural textures are used. They help to add additional detail and at the same time, solve the memory issue. Another benefit is that it helps to abstract explicit details of a scene and uses mathematical

functions instead. It also gives flexibility and parametric control. They're either made using noise of shaders in this paper.

Noise is a pseudorandom function which when manipulated is used to make multiple n-dimensional textures. Random noise is a bunch of points that have no relation with each other whatsoever whereas, Perlin noise outputs random points in a much smoother way. The function of two floating point numbers would give a random number but those specific floating point numbers would always return the same value and hence, it is called pseudorandom. The speech to text system acquires speech at run time using the microphone and the noises are eliminated using the voice recognition module which allows only the voice of the speaker to be recorded. The system is activated using the mobile application. When the user presses the button "Summarise the lecture", the speech to text system gets activated and the lecture is recorded. This recognized text is store as a .pdf file in the application which can be downloaded by the user.

Perlin noise as you can see in the figure produces a smoother result. It is comprised of three basic steps:

- Grid definition: Imagine a grid where each small square represents a pixel. The bigger blue squares have corners that each represent a gradient vector which points to a random direction. The arrows represent the gradient vector. The blue lines can be increased or decreased, therefore controlling the detail of the final result. The green arrows that point compute the distance vector. The value of the particular pixel highlighted in red is known with the help of the distance vector which is the distance between the coordinates of the corners and the coordinates of the pixel square.
- Dot Product: This is the dot product of the distance vectors and gradient vectors. It returns a number which gives maximum positive value if the vectors are pointing in the same direction, maximum negative values if they're pointing in opposite directions and zero if they're perpendicular to each other.
- Interpolation: Say, there is a table of temperatures in the x column and their corresponding humidity in the y column. The x columns have values 10, 20, 30…100. But you want the humidity of temperature 38℃. This intermediate value is known as the interpolated value. Interpolation is a method that fills up the gaps in the table. In Perlin noise, it helps to give the final colour value of the pixel. Interpolation results in smooth pixel values but visible edges. This is solved by a fade function that smoothens the whole texture.

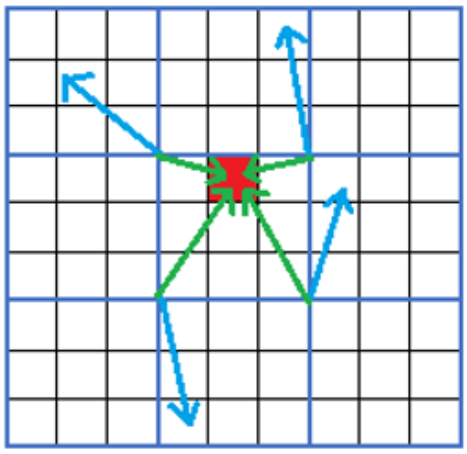

Figure 1. Perlin Noise Grid

The output of the algorithm looks like smoothened black and white gradients all over the quad, the object on which the texture was applied. This is what it ends up looking like. By changing the offset you can get a more scaled in or scaled out result.

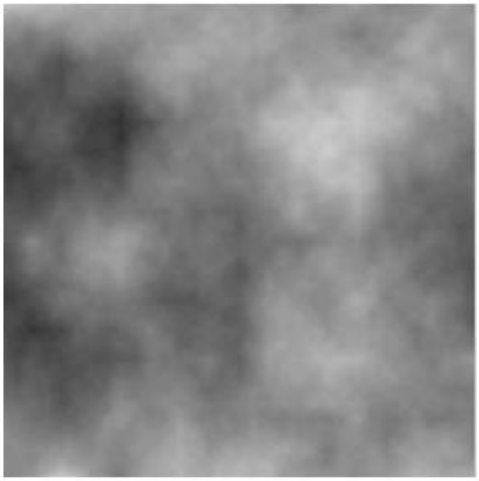

Figure 2. Perlin Noise

For marble, it is simply sin(x+y). Manipulating the RGB values changes the colour. Some power is used to make variables such as turbPower, xPeriod and yPeriod are introduced into the algorithm. turbPower is used to make twists and the other two define the angle of the lines. Changing the values of these variables gives different marble effects. For wood, the formula used is sin($\sqrt{x^2 + y}$). Here, xPeriod and yPeriod ariables are replaced by xyPeriod which denotes the number of rings. This variable and turbPower can be increased or decreased to show variations in the wooden texture.
Road networks can be generated using Perlin noise, L-systems or even some maze generation algorithms. It depends on how you want the roads to run across the city. The Perlin noise algorithm assumes an nxn grid, with each cell containing an integer from 1 to n. These numbers are the result of the Perlin noise function. Now, some of the rows in the grid are randomly picked to draw a road. Horizontal roads or x streets replace the numbers with -1 and vertical roads or z streets replace the numbers with -2. Wherever there is a crossing, the number is replaced with -3.

3.2. Building Design

Modelling of buildings manually again takes time. And a city doesn't just comprise of a single certain type of building. There's large hospitals, tall skyscrapers, residential villas, office spaces etc. Designing all of them is time-consuming. Using L-systems and CGA grammars to generate them procedurally is one way to fasten up the process.

3.2.1. L-systems

L-system or Lindenmayer system is a form of rewriting rule, a type of grammar. It consists of an alphabet of symbols that are used to make strings which are further replaced by a larger set of strings from the production rules. They're similar to fractals and create self-similar objects, they're also used in the morphology of various organisms. L-systems are defined as a tuple. Some of its applications explore botanical species, patterns and road networks.
G = (V, ω, P)

where, V – set of both terminals and non-terminals
ω – subset of symbols from V defining the initial state
P – set of production rules

L-system rules are applied iteratively from the initial state and as many rules as possible are executed simultaneously. This is what differentiates the concept from formal grammar. For producing graphical images, each symbol of the rules needs to point at an element of the drawing. L-systems comprises of two parts: Generative process which is concerned with the writing of strings in an equation and Interpretative process which uses the concept of turtle graphics to interpret and execute the strings. Some of the rules include stochastic rules where a letter is replaced by a string with a specific probability and parametric rules that include angles and incrementing factors to give a higher degree of precision. Turtle is a Python module that helps to draw objects on the screen. Think of it as a pen plotter. It moves and plots on a plane according to the commands it receives. Some basic functions of the module include forward(), backward(), clear(), penup() etc. It requires importing the turtle module, creating a turtle control and then drawing using turtle methods.

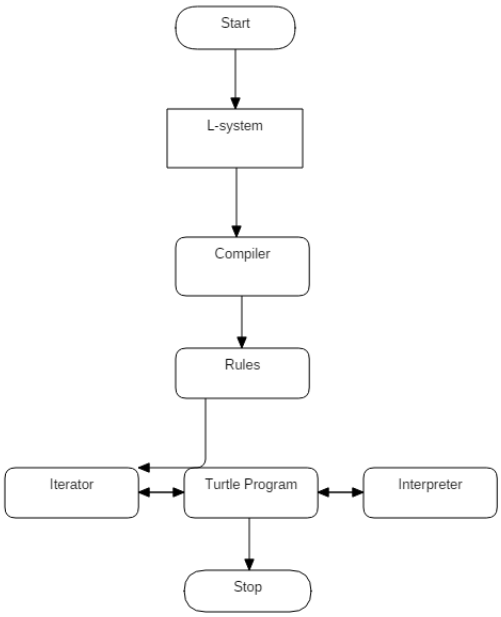

Figure 3. L-systems Flowchart

3.2.2. CGA Shape Grammar

CGA shape grammar is a set of rules included in a rule file, which is applied to the initial shape fed to the scene. The starting shape may be a 2D polygon but once the rules file is applied, it starts to build from there on. A CGA rule is of the format:

Predecessor → Successor

where, the predecessor shape is always replaced by the successor shape. The successor side of the equation contains symbolic names for the output shape along with a few operations.

For example, Lot → extrude(5) Envelope.

This rule takes the initial 2D polygon (Lot), forces out or extrudes it five times and then transforms it into an envelope.
Shape grammars are made of two parts namely, shape rules and generation engine. Now, apart from the format of the shape rules as discussed above, there must be at least three rules in a grammar: one start rule, one transformation rule and one terminating rule. The start rule is used to start the shaping process and the termination rule is used to mark the end of the process. The transformation rule is already mentioned above. The generation

engine, on the other hand, checks the existing geometry, also known as the current working system (CWS). It checks to see if the CWS on the predecessor side of the equation matches the conditions specified in the grammar. If it applies to more than a single rule, the generation engine picks one rule. Alternatively, the generation engine can first pick a rule and then check to see how many matches it can find. If there are multiple matches, the GE can either execute them serially or in parallel or just pick one of them to apply the rule on. A variation of shape grammars are the parametric shape grammars which include the RHS of the equation in the form of parameters. This helps to get more of the context in the rule. It helps when you want the grammar to respond to the width of beams or the roof structure or similar attributes.

3.3. Road Network

Road networks can be generated using Perlin noise, L-systems or even some maze generation algorithms. It depends on how you want the roads to run across the city. The Perlin noise algorithm assumes an nxn grid, with each cell containing an integer from 1 to n. These numbers are the result of the Perlin noise function. Now, some of the rows in the grid are randomly picked to draw a road. Horizontal roads or x streets replace the numbers with -1 and vertical roads or z streets replace the numbers with -2. Wherever there is a crossing, the number is replaced with -3.

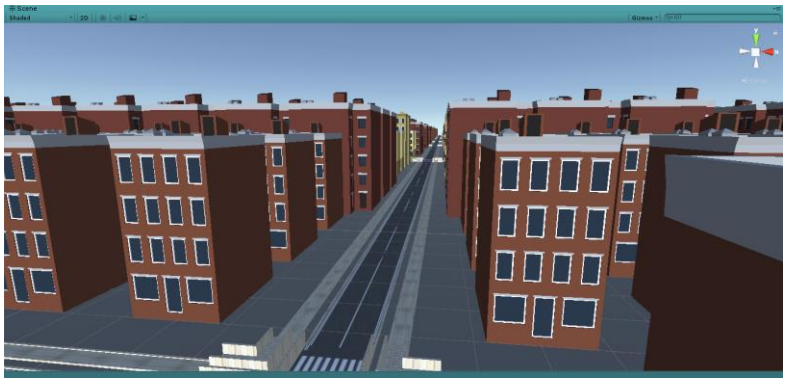

Figure 4. Road network

4. Results and Analysis

From the above implementation, we were able to generate a fairly large city with roads, parks, trees and different types of buildings. The various algorithms used help to create the many elements of the city. The algorithm that controls the map is also designed in a way that it alters the map every time the offset value is changed. This helps the designer to change the look of the city if he isn't satisfied with the previous one without actually putting in any manual work. The city shows detailed textures on zooming in. The entire terrain can be inspected and checked. The basis of procedural generation algorithms is Perlin noise and manipulating this noise is what gives us most of the city's layout. Rules, productions and formal grammar are the core of building models and other natural components placed in the map. Some of the drawbacks of the paper include:

- It currently runs on Unity only.
- Since we're using procedural algorithms, the models don't look as real as they would if they were manually designed.
- There is no plumbing system which is an important part of city planning.

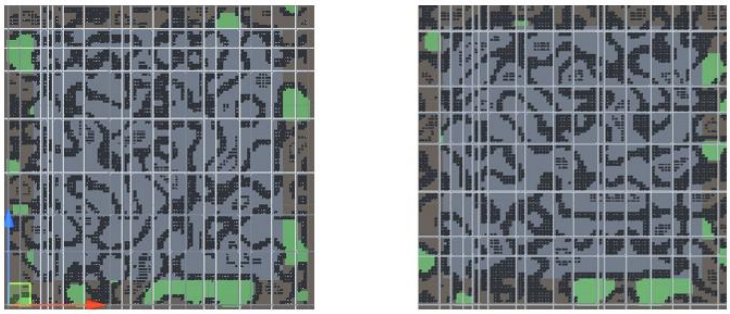

Figure 5. Top view of city with varied seed values

5. Conclusion

The report presents various procedural generation algorithms used to generate the visualization of a city. The architecture of the system along with the implementation of it for the proposed technique is also discussed. The execution time of the algorithms used is drastically reduced as compared to manually designing the entire city. Our system is flexible for usage since parcels or blocks of the city can be modified by changing the offset. Since this would change how the roads connect too, it results in a new city that looks very different than the previous one. The system also utilizes the GPU and produces much faster results. In conclusion, this model can be used for not just efficient planning of cities but also in the visual FX and gaming industry to replicate large-scale cities to heavily cut down cost and time of development as the most time-consuming part of the workflow is art asset creation. This technique will be useful for creating virtual environments rapidly using procedural generation that increases development speed of video games, fictional virtual worlds for movies or even for testing or creating a learning space for various machine learning models that drive machines in the machine's objective-specific environment.

6. References

[1] Alkaim, Artur. "Procedural Generation for Architecture."

[2] Kelly, George, and Hugh McCabe. "A survey of procedural techniques for city generation." *The ITB Journal* 7.2 (2006): 5.

[3] Müller, Pascal, et al. "Procedural modeling of buildings." *Acm Transactions On Graphics (Tog)* 25.3 (2006): 614-623.

[4] Galin, Eric, et al. "Procedural generation of roads." *Computer Graphics Forum*. Vol. 29. No. 2. Oxford, UK: Blackwell Publishing Ltd, 2010.

[5] Rhoades, John, et al. "Real-time procedural textures." *Proceedings of the 1992 symposium on Interactive 3D graphics*. ACM, 1992.

[6] Groenewegen, Saskia, et al. "Procedural City Layout Generation Based on Urban Land Use Models." *Eurographics (Short Papers)*. 2009.

[7] Vanegas, Carlos A., et al. "Procedural generation of parcels in urban modeling." *Computer graphics forum*. Vol. 31. No. 2pt3. Oxford, UK: Blackwell Publishing Ltd, 2012.

[8] Talton, Jerry O., et al. "Metropolis procedural modeling." *ACM Transactions on Graphics (TOG)* 30.2 (2011): 11.

[9] Betts, Tom. "Procedural content generation." *Handbook of Digital Games* (2014): 62-91.