City generation

Mantas Aleskevicius(190344272)

May 2022

Bsc.Computer Science

Supervisor - Dr Gary Ushaw

Word Count: 5000

Abstract

Modern 3D games revolve around large open worlds with increasing demand for size and whether made by independent developers or large corporations are limited by manpower and time. What is more concerning is that games contain stories that most often require people which breeds the need for urban environments which is one of the most common settings for most games. These unlike natural environments require a lot more models which exacerbate the problem and the need for a long term solution. This dissertation describes an attempt to create a procedural generator that produces an urban environment from scratch using algorithms instead of manual modeling and level arrangement. It also summarizes the results and the algorithms used to produce them.

Declaration

I declare that this dissertation represents my own work except where otherwise stated.

Acknowledgements

I would like to thank my project supervisor Gary Ushaw for giving me guidance and for helping me reach my potential and make the best project ever could.

Table of Contents

Introduction

Introduction

This section of the dissertation covers the main aim, objectives, motivation and relevance for the project.

Relevance

In today's world, there is a big emphasis placed on the visual fidelity and realism presented to us by the media whether it is a trailer of the new movie, the newest triple A video game or even a backdrop to a news reporter. Our technological advancements raise people's expectation of the average or the normal quality of graphics and scale that they should expect from most upcoming projects, raising the bar with every subsequent release because until just recently it was a struggle to fit all of the content and geometry made for the project by the level, set and VFX designers into the game or a movie because of the lack of processing power in the CPU and GPU and being limited by the delivery medium, cannot expect the consumers to by more and more cd discs as you put more content in.

However, as the technologies have gotten more advanced and with the rise online distributing services such as steam the games and movies have gotten access to more space and more processing power, and we see that with every generation, sometimes even in the same GPU cycle studios are increasing the size of their maps 3 to 4 times and now it’s not the lack of processing power that is the issue but the amount of content that needs to be produced by the 3d artists that is the problem. For example in mid 2000s red orchestra 1 released with the environmental meshes and textures weighed in at 1.4GB a decade later a game by the same studio Killing Floor 2’s environmental meshed and textures weighed in at 17.4GB in essence increasing the costs and amount of work 12 times in a single decade.

Furthermore, the estimated costs of all art for a game is around 35-40% of the total budget and with the production costs rapidly increasing from few million in the 2000s to few hundred million in recent years there is a need for better solutions than ground up hand modeling, texturing and animation.

Motivation

As such, one of the possible solutions and the one I am proposing is procedural generation. Procedural generation has been used for years developing environments and models, although mostly natural environments and models such as terrain generation and plant modeling. This greatly lowers the production costs, and procedural generation is also what most indie studios use to cut the production costs and time. Some of the most successful indie video games have used procedural generation to greatly reduce the development costs to create a believable environment in both 2d and 3d space, such as minecraft and terraria.

My proposal is using these previously defined techniques to create more industrial and man made landscapes such as cities, towns and buildings. This combined with other procedural generation tools such as terrain generators could create quick and easy maps and sets to speed up the development process. Allowing the developers to quickly generate a basic structure to their game world, which then can be refined manually.

My work would also focus on adding more detail to the buildings themselves, providing varying styles to the building architecture, which would make it easier to create alien worlds and cityscapes by adjusting the city generator.

Aim and objectives

Hypothesis - Explore and implement algorithms and methods for asset creation by developing a procedural city generator

.

Aim: To develop a tool to reduce the development costs of film and game asset and set production.

Objectives:

- **Research and summarize published work on procedural city generation.**
  There have been several commercial urban landscape generators in the past and multiple open source projects that have released papers on their findings. My objective is to analyze and understand the work of the two commercial city generators.
- **Review the technological challenges posed by procedural generation.**
  Identify the shortcomings of procedural generation and identify a few techniques that can help solve these shortcomings.
- **Establish a list of economic, geographic and historical influences that determine the look of the cities and ways to reproduce them.**
  Document the realistic variables that determine the look of the cities and have found techniques of patterns that reproduce the similar looking result.
- **Develop an interactive tool that can create a city that looks believable and can be used as a skeleton for further refinement.**
  This task will be completed when I can create an urban environment without modeling it.

Background Research

Introduction

In this section of the dissertation I cover the background research conducted to supplement my knowledge in order to complete the goals of this dissertation.

The research was conducted on a variety of relevant sources as such I have split them up into 3 categories:

- Developer Interviews and blogs.
- Research papers.
- Media research and videos.

Developer interviews and blogs were used as inspiration and to find techniques to later research more in depth through research papers.

Research papers were the main resource that helped grasp and research more in depth and technical aspects of the dissertation.

Media mainly consisted of technical demonstrations and presentations of complex principles that I was either not able to find a coherent and relevant source or was not able to implement on my own.

Industry approaches

City creation (Spider man)

At the start of the research process even though I wanted to begin research into generation and various types of grammars I needed to understand the creative process behind level creation in the first place. As such I chose to investigate one of the biggest games in recent history spider man (2018). It is praised for its extremely accurate representation of

Manhattan and it would seem really inefficient to create the city by hand.

Figure 1: New York city Spider Man(2018).

Most of the articles about the island of Manhattan in spider man (2018) covered the creative vision and how they merged a lot of fantastic elements into the city but I dug deeper and found [1] article by "GQ". The article is an interview with the games artistic director Jacinda Chew. She describes the use of a software used widely in the films industry "houdini" that is used to procedurally create VFX for films and even cities such as depicted in Captain marvel and zootopia.The article says:

"The earliest groundwork for the game's version of Manhattan was done with Houdini, a program that could procedurally generate each of the city's blocks, complete with walking pedestrians and driving vehicles. From there, it was up to the game's artists to modify the generic gray boxes into a Manhattan that was both recognizable and fun to traverse via web. (If you're not familiar with game design, picture it like this: Houdini's job is taking a box of unpainted LEGO blocks and placing them in a series of neat, easily comprehensible rows. The artists are tasked with taking Houdini's fundamental building blocks and modifying them—via height, color, or countless other factors—until they're fun to play with.)"

As a result the process described is similar to what I proposed in the introduction of this dissertation. It places cubes in a street pattern with some parameters and 3D artists come in and add detail finalizing the city. This approach has a predetermined street map. It is not uncommon in procedural city generators to use grid pattern and cities for that matter use them as well as described in [3] however it makes the cities look too unnatural as most cities have a combination of them.

Simulation (Dwarf fortress)

Another idea of city generation that I needed to research was simulation. No need to try to recreate city features if you can simulate a whole civilisation and the best resource on the matter I found was of course Dwarf fortress. In this [2] 2008 article the game creator Tarn Adams was interviewed and explained the world generation process to the interviewer.

Let's talk about the world generator for a bit. If you wouldn't mind giving us a general overview of the process the game goes through? No need to go into great detail.

“The first overall goal of the world generator is to create enough information to produce a basic biome display. A lot of initial attempts at a world generator will start with things like "I need to lay down some forests, and some mountains, and some rivers, and some deserts..." and then when you end up with a jungle next to a desert, or a desert next to a swamp in an unlikely way, it's difficult to fix.

So the idea is to go down to basic elements. The biomes are not the basics, they arise, at least in DF, from several factors: temperature, rainfall, elevation, drainage. First, it uses midpoint displacement to make an elevation map.”

This passage of the interview is really interesting and in my opinion very applicable even in procedural generation. Something similar is talked about in a paper [4]. Where the content is not placed but organically grows and evolves with some starting parameters on a lower level.

“Next step is the erosion phase.

It picks out the bases of the mountains (mountains are all squares above a given elevation), then it runs temporary river paths out from there, preferring the lowest elevation and digging away at a square if it can't find a lower one, until it get to the ocean or gets stuck. This is the phase where you see the mountain being worn away during world creation. I have it intentionally center on a mountain at that point so you can watch.

This will generally leave some good channels to the ocean, so it runs the real rivers after this. However, some of them don't make it, so it forces paths to the ocean after that, and bulges out some lakes. Then it loop-erases the rivers, and sends out (invisible on the world map) brooks out from them. During the loop-erase it also calculates flow amounts and decides which rivers are tributaries, and names the rivers.

This gives us a world with biome data and rivers, but no actual life.”

Anyway, let's move on to what I expect was one of the biggest challenges in developing the game: the amazing pathfinding.

“Is it amazing? It's creaky and slow.”

Well it looks amazing from my end, since there's a metric ton of characters all doing it at once.

“The dwarves themselves mostly move around with A*, with the regular old street-distance heuristic. The tricky part is that it can't really call A* if they don't know they can get there in advance, or it'll end up flooding the map and killing the processor.”

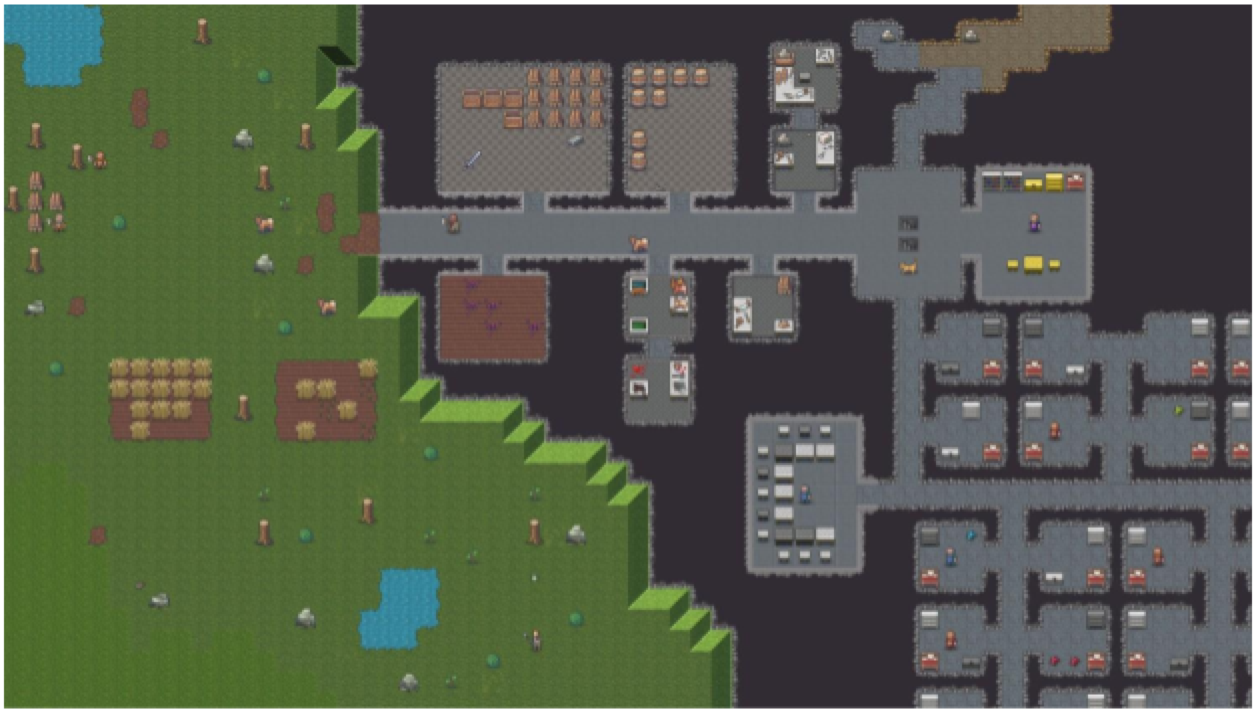

Figure 2: Settlement in Dwarf Fortress

And as Tarn Adams further elaborates he takes the concept of evolving his world from few staring variables adds AI in the terms of dwarfs with A* algorithm and continues iterating over them until he has the most common roads from which he would just need to branch out and place buildings.

This seems like a great solution for Dwarf Fortress that requires the variables for various other systems that generate wildlife and vegetation. However, it is too taxing on the hardware and inefficient when the goal is to generate a city and not to make it an open world.

City Generators

As mentioned in article [1] there are few tools that help recreate the cities for movies and games. For my dissertation I researched two city generators. That gave me an overview of the techniques and algorithms I needed as well as providing some guidance for starting the project.

The [4] paper published in 2001 by Yoav I H Parish and Pascal Müller and [5] paper published in 2007 by George Kelly and Hugh McCabe. These two papers are very similar even referencing each other and mainly focus on the same aspect of city generation - street layout. As a result I believe there is more merit in analyzing them by comparison.

The main points of the comparison relevant to dissertation:

- Initial values acquisition.
- Primary road generation.
- Secondary road generation.

- Lot division.

Initial values acquisition

Initial values in [4] were provided by 2D images that represent Geopolitical maps(elevation, water borders, etc) and Socio Statistical maps(population, zones, etc.). This approach made it trivial to add extra variables to the street generation and allowed for quick changes.

While [5] were taken from the terrain mesh and through providing UI to select the population centers.This made it far more user friendly and allowed it to be modified in real time however made it far more difficult to prototype and add new values that influence the street generation.

Primary road generation

Primary road generation [4] uses an extended Lindenmayer system (L-system) that is self-sensitive and tries to make the shortest path between the population centers provided in the socio statistical maps. The L-system itself has two bookkeeping global functions that determine the look of the city: global goals that modify the position of the streets produced by the L-system rules and local constraints that make intersections and prevent the streets from overlapping. This approach is incredibly versatile and mimics real city expansion with the ability to add and remove rules because of the two bookkeeping functions, however the L-system is only a formality as argued in [6] and severely overcomplicates the implementation.

In the [5] the population centers are connected into a single undirected planar graph. It is far simpler than the first approach and allows arguably easier division of the city into zones, however this leaves the city surrounded by a ring of highways that is in no way representative of the real world cities.

Secondary road generation

Secondary roads in [4] are generated similarly with the L-system, however they follow a superimposed pattern that allows the zones to have distinct street patterns.

Secondary roads in [5] are generated with a similar L-system, however they start generating from several points around the zone divided by the primary road graph. This leaves them less flexible as the zone is bound to have the same pattern.

Lot division

Lot division is extremely similar in both papers however [5] allows some non-convex shapes for the buildings.

Conclusion

Industry

It is clear that tools for procedural city generation or atleast templates are used in the industry however not by themselves but rather as a starting point for 3D artists as there is little trust in them producing interesting, detailed and memorable environments even after 20 years of being around. At most city generators are used as backdrops for games or for architectural projects like the city engine described in [4].

Furthermore, simulations have even bigger setbacks in their generation process as they are too technically taxing and wildly inefficient as the discussed dwarf fortress still uses ASCII interface, not to mention the end result has no more chances of being better than the procedural route. With these examples it is clear that while the simulation has its uses, procedural generation has more merit.

City Generators

The Lindenmayer system seems to be an essential part of procedural content generation, however the form presented in [6] is simpler and easier to prototype.

Implementation

Introduction

In this section I cover the technical detail of the city generator and high level overview of the implementation except where I deem more in depth clarification is necessary.

I have split this chapter into several parts as the iterative nature of the implementation lends itself well to structuring the analysis of the concepts by the versions of the project as each version added on a piece of the generation.

Development methodology

The development methodology used on the project was an iterative development model.

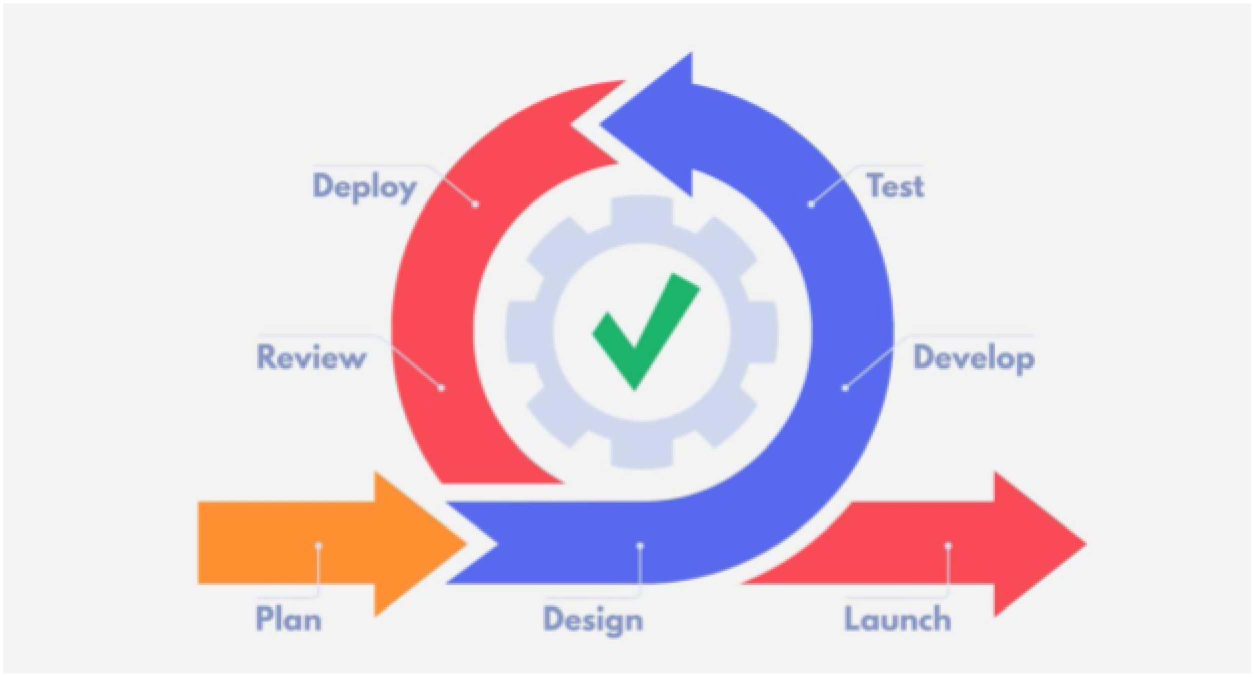

Figure 3: iterative model.

The most prominent reason was that the project lent itself really well to being developed in chunks as the city is built up of separate parts that build one on top of the other. The city needed a street map from which I could allocate lots and then generate buildings as shown in Figure 4.

Second reason is it quickly exposes defects and bugs in the code that could be fixed before advancing to the next iteration and considering the already uncertain nature of procedural content generation it can become hard to tell if the generation misbehaving because of a bug in the code or just an inconsistency in the generation.

System design

The system design can be broken up into a couple of classes further building and rewriting each other's output. It could be rebuilt as a grammar.

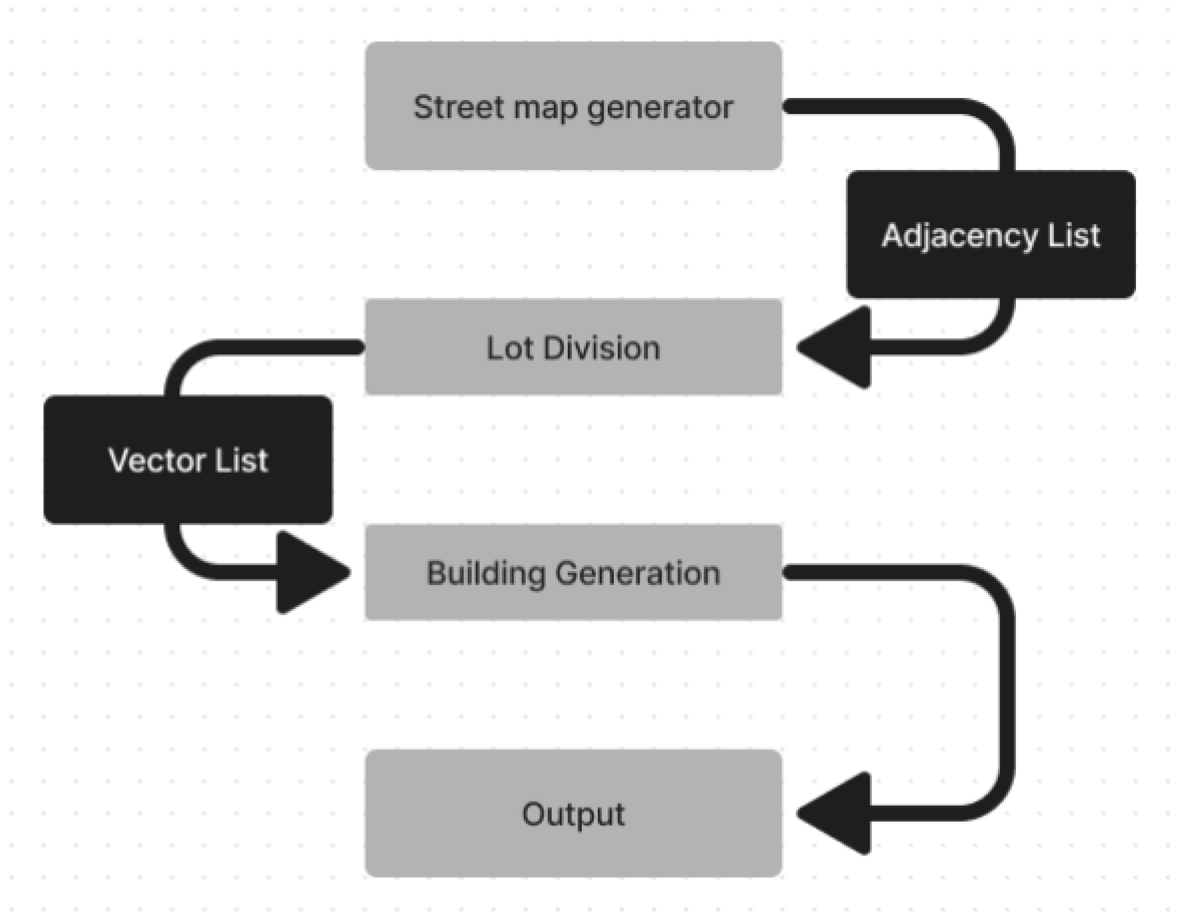

Figure 4: High-level System Design

Version 1.0 Street map

Version 0.1 Segment creation

The development process started as soon as the main research portion of the project was finished and I had enough knowledge to start implementing the first street map. At first I was implementing the L-system described in [4]. The result is shown in Figure 5. However even without the implementation of intersections the L-System seemed to be too rigid and constrained, not able to interact with already created road segments and seemed not the most efficient tool to use.

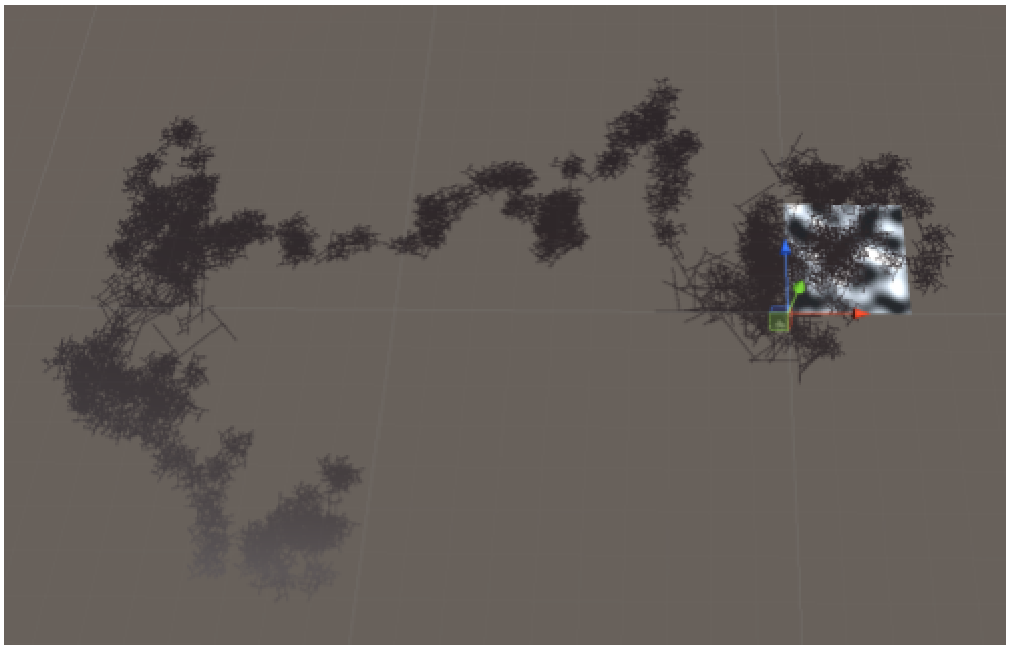

Figure 5: L-System street generation.

Version 0.2 Nothings algorithm

As explained in the research chapter and argued in [4] the L-system was just a formality and the underlying algorithm was far easier to read, implement and modify.

The algorithm created a priority queue and populated it with one segment at the start and entered a while loop for as long as there were segments in the priority queue or as many iterations as were required. In that loop it passed through two functions: global goals, that created more segments, and local constraints, that modified or deleted segments if it could not find a solution. The concept is illustrated in Figure 6.

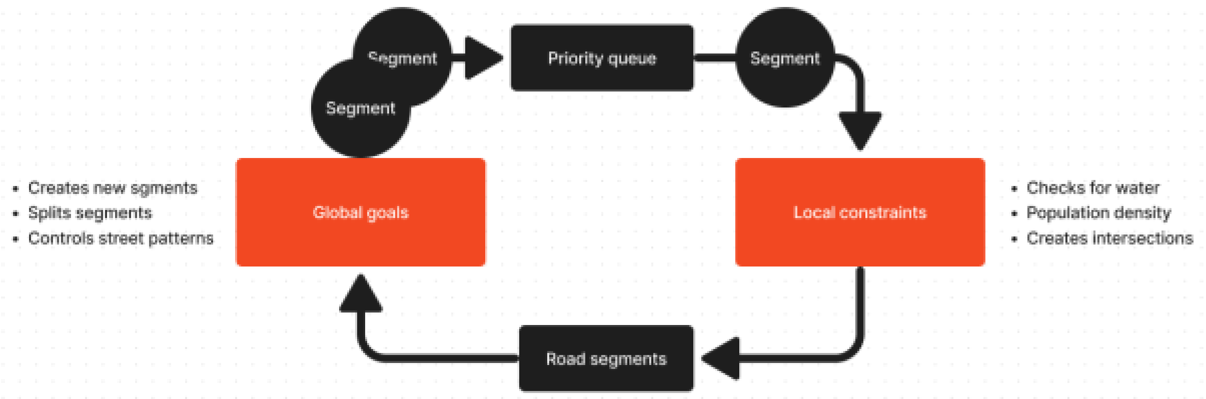

Figure 6: Simplified L-system.

Version 0.3 Global Goals

With the basic algorithm in place I created a function to create more segments in the direction of the population centers. I used a 2D image as a population map and sampled coordinates in a cone in front of the segment and assigned them a score depending on the greyscale value received and picked the lowest one creating a new segment(Figure 7).

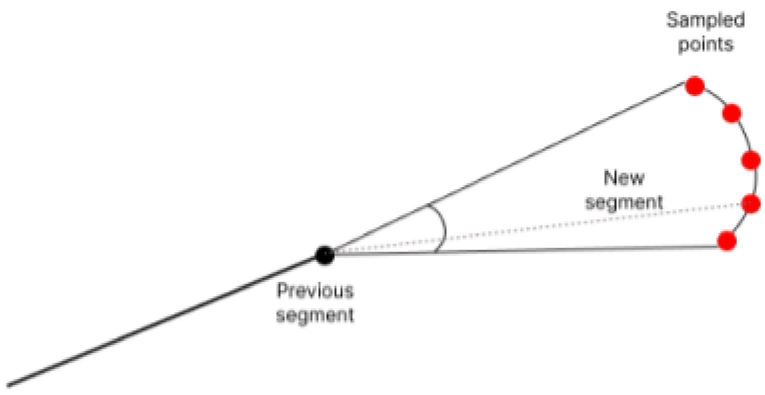

Figure 7: Point sampling.

While the side roads were created if the main road reached a certain population threshold.

Version 0.4 Local constraints

After global goals were generating segments the only function left was local constraints. In this function I added another 2D image representing the water border and a check for it and most importantly implemented a function that would check if the segment created is intersecting with any other segment or close to a road junction and creating an intersection(Figure 8).

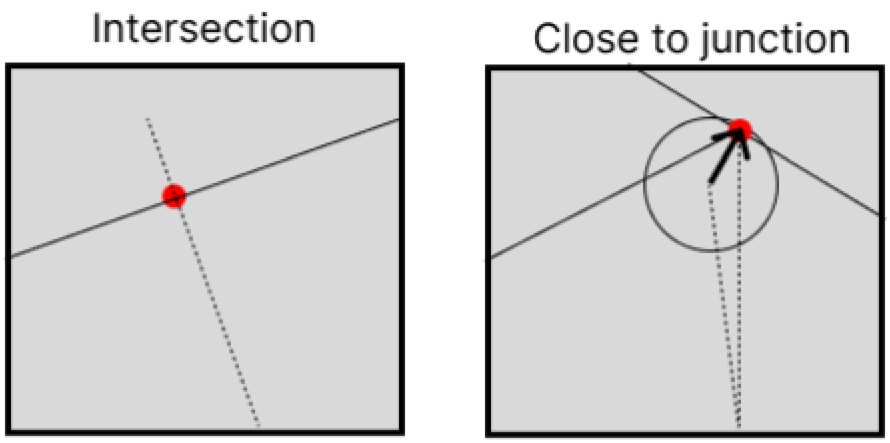

Figure 8: Intersection scenarios.

Version 0.5 Street Map

Resulting maps from the street generator (Figure 9).

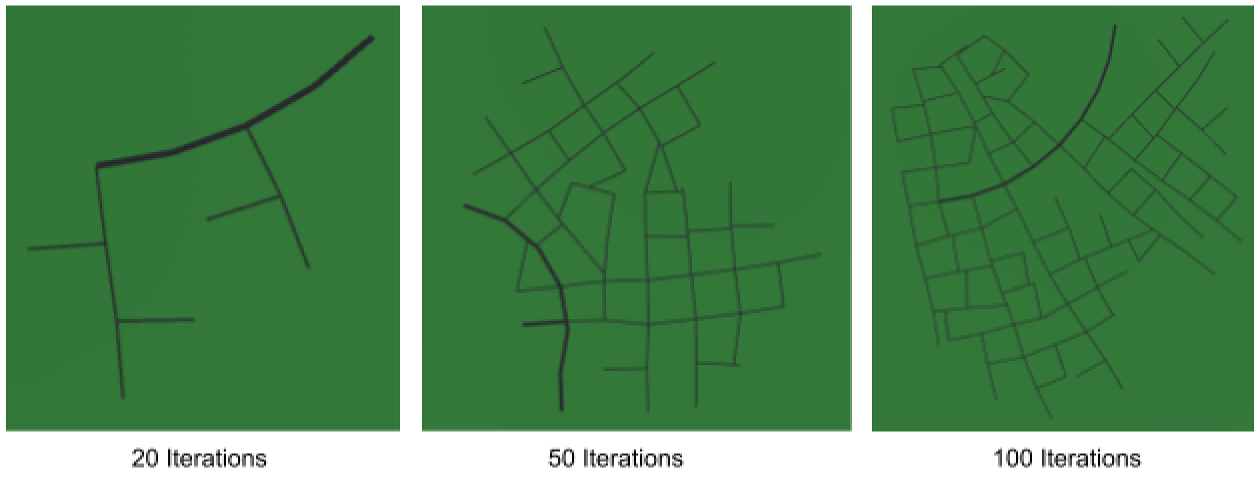

Figure 9: street map.

Version 1.1 Lot Division

Version 1.02 Block recognition

Used Ideas of [7] to implement cycle basis algorithm to locate all of the usable blocks. Usable block is an enclosed shape with no segments going through the middle of it. The algorithm used the coordinates of the graph that was formed from the adjacency list that was the output of the street generator and only went clockwise if possible till it reached the starting node.

Version 1.04 Sidewalks

Discarded the blocks that were too small.Inset the them by a fixed distance with a method described in [8] to make room for the sidewalk. However this method is flawed as it can not handle sharp dips in a straight line (Figure 10).

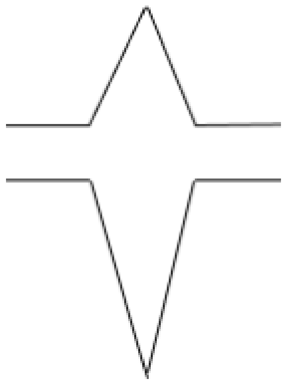

Figure 10: opposite Inset.

.

As they were rare I refiltered the blocks by comparing their sizes before and after the inset, If the size had a massive increase or decrease I discarded the block and this fixed this bug.

Version 1.05 Division

The block division process into lots was simple. I used the same method that I used to check intersections and picked the longest vector on each axis and after every set distance checked where it intersected. After that I applied the block recognition algorithm again and found all of the lots and excluded those that were again too small or did not have access to the road. Figure 11 shows the end result.

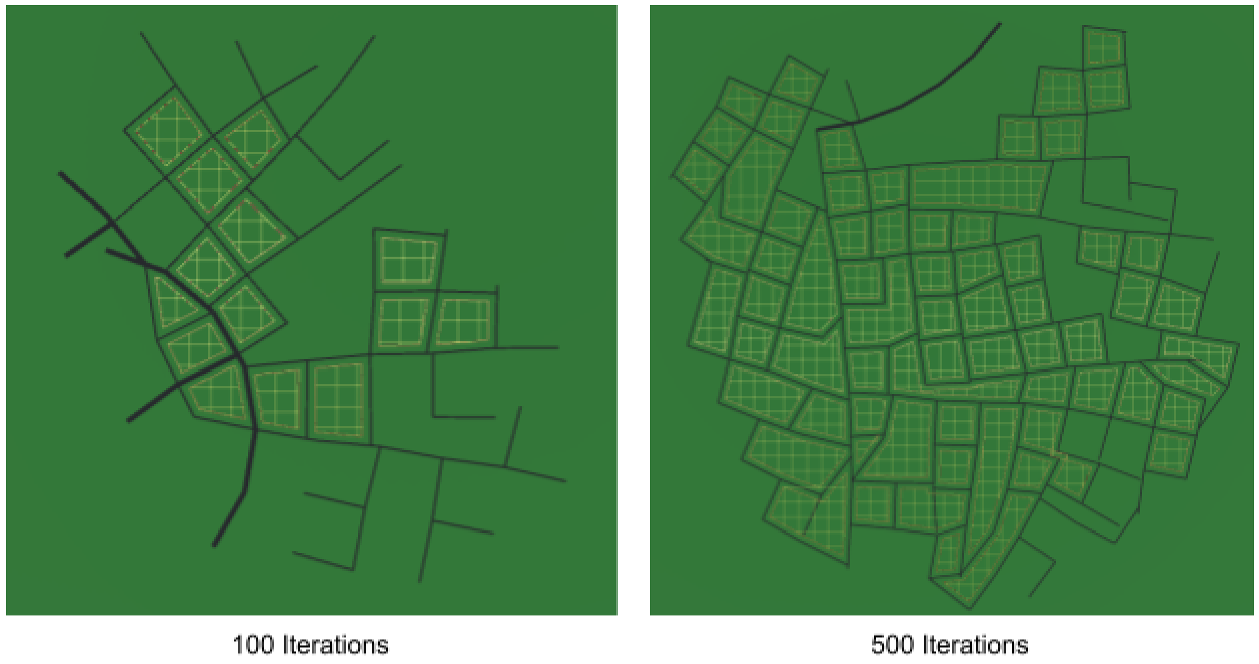

Figure 11: Lot Division.

Version 1.2 Building generation

Version 1.11 Shape grammar

Created classes according to the idea of shape grammars presented in [9]. I chose this way of handling the generation of buildings because it allows much more details and addresses a big problem in procedural generation. This is achieved by having more and more modular with the clases till the level of detail is acceptable.

Version 1.12 Creating a floor

Extruded the vertices from the base of the lot to create a floor of a building. However the top and the bottom of the floor are not always a square. As such I implemented the Bowyer–Watson algorithm to calculate Delaunay triangulation(Figure 12). With it the triangles in the top and the bottom were formed. Then I used a slightly modified version of Block recognition that only calculated triangles in one direction as such normals of the triangles would face the right direction.

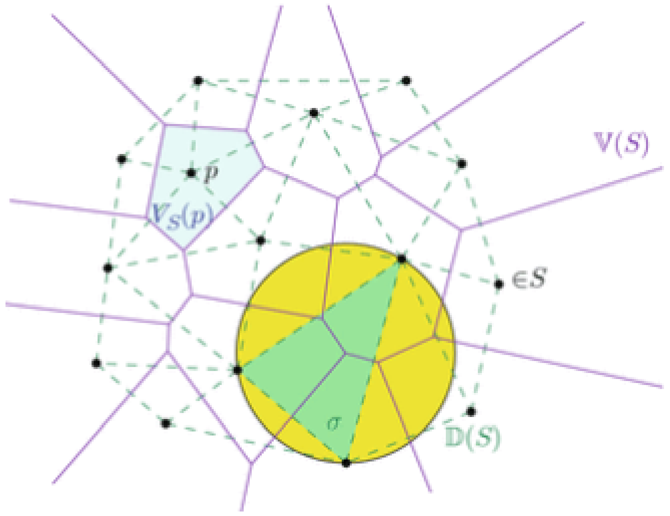

Figure 12: Delaunay triangulation.

Version 1.13 Adding details

Lastly I quickly created a few prefabs like a window, door and a ledge. That would be placed in the right position. While this does not add as much detail as would be expected it is very easily extensible to include more detail.

Conclusion

As shown in this chapter the implementation of the city generator because of the iterative development allowed me the opportunity to have a loose design structure yet have set a boundary and goals but be vague on the inner functions and algorithms that would be used.

Results and Evaluation

Introduction

In this chapter I discuss the goals of this dissertation and evaluate how well I achieved them.

Summary

Models

The model of the project was produced with prefabs made by me and as such the quality is greatly diminished with asset store assets the look could be improved. Figure 13 and 14 show the model of the city.

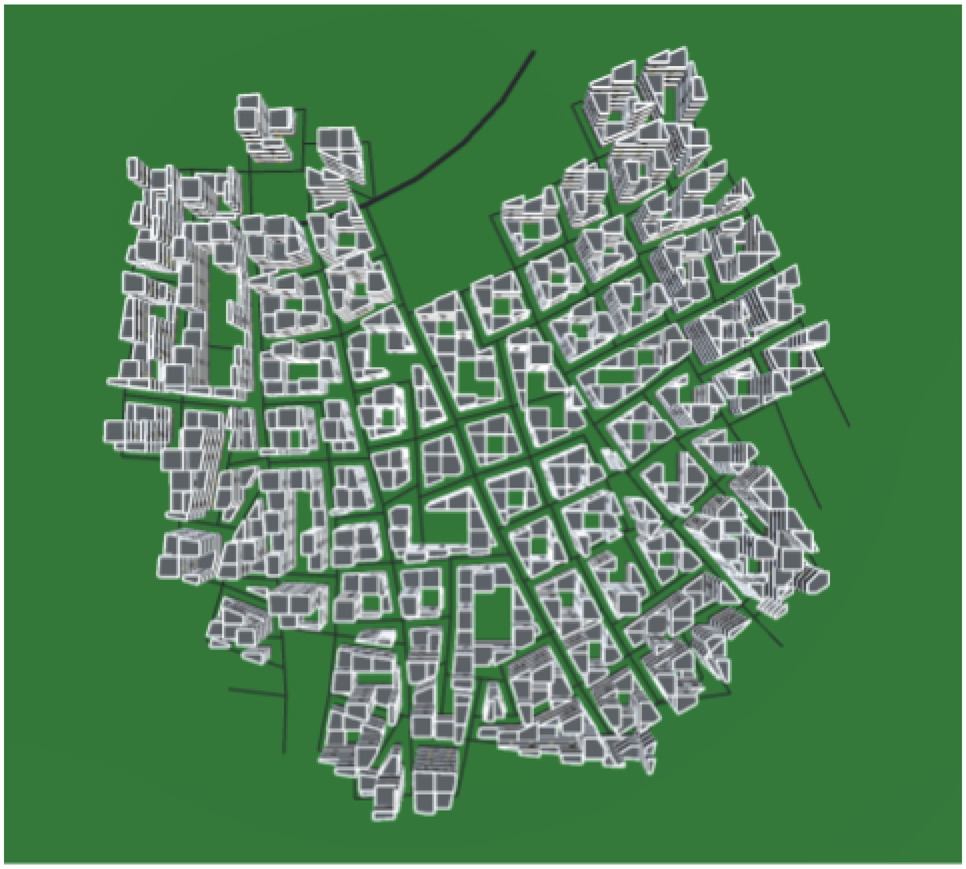

Figure 13: aerial view of the city.

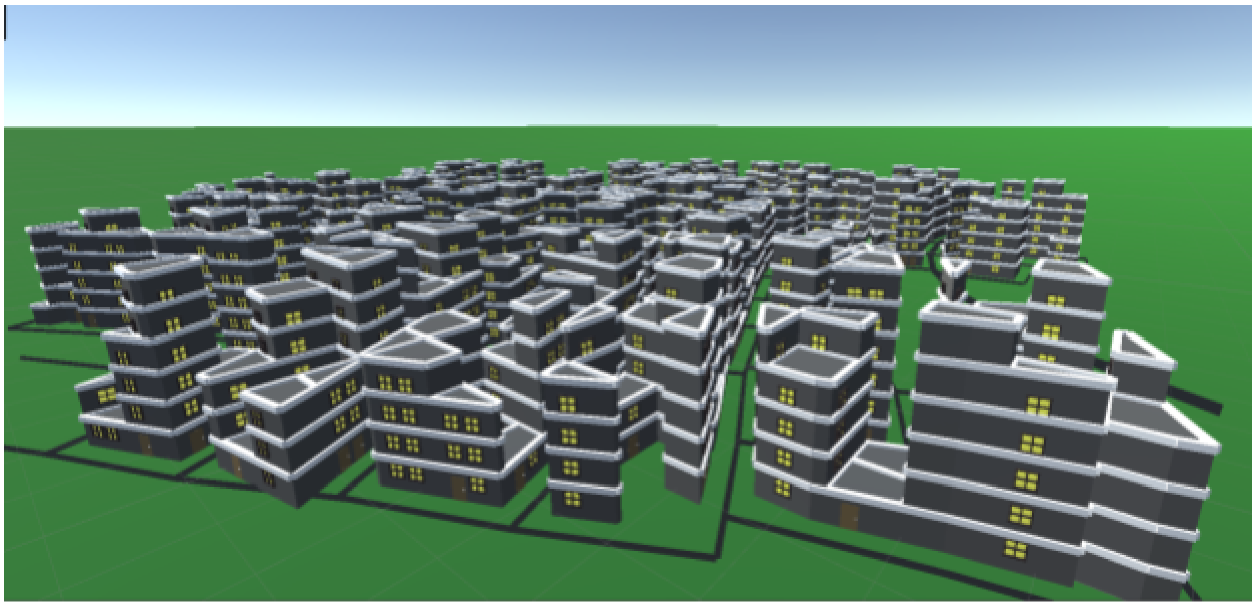

Figure 14: side view of the city.

Evaluation

- **Research and summarize published work on procedural city generation.**
  There have been several commercial urban landscape generators in the past and multiple open source projects that have released papers on their findings. My objective is to analyze and understand the work of the two commercial city generators.

  I have indeed analyzed and understood two city generators: Citygen described in the [5] paper and City Engine discussed in the [4]. As well as analyzing, I have implemented variations of several techniques that were used in either of the generators. As well as analyzing by comparing the two in the research chapter of the dissertation. However I did not engage in the texture generation part of either of the generators as I believe shape grammars create more realistic feeling.

- **Review the technological challenges posed by procedural generation.**
  Identify the shortcomings of procedural generation and identify a few techniques that can help solve these shortcomings.

  I have identified that procedural generation often lacks detail and have used shape grammars to try and counteract this shortcoming. However in my opinion this has created another problem of the content looking too similar.
  Moreover I have pointed out that another problem with procedural generation is its unreliable results - whether the content will be good, however that requires a lot more investigation on what objectively makes a believable, immersive city. This project would become more complex and I believe that I just did not have time to add extra complexity.

- **Establish a list of economic, geographic and historical influences that determine the look of the cities and ways to reproduce them.**
  Document the realistic variables that determine the look of the cities and have found techniques of patterns that reproduce the similar looking result.

  While my project tries to imitate real world cities by following a population density map and using a semi grid based street layout . I have not documented and compiled variables that influence the city's appearance. Even though [4] and [3] explain how the streets develop into looking realistic neither covers building architecture.As such I believe this objective is not complete.

- **Develop an interactive tool that can create a city that looks believable and can be used as a skeleton for further refinement.**

  I believe that I have met this objective to a considerable extent as the city is easily customizable by just swapping out assets that are used in the shape grammar and as such the look is only to represent the functionality.

Conclusions

Reflection

This project was a mild disaster from the execution to my involvement in the school activities. The concept of city generation gives association with game development as such it absolutely slipped the mind of a naive student that procedural generation is the use of mathematics as your brush and the game engine as your canvas.

The first thing that I would reflect on however is the fact that it was fun to be challenged by procedural generation. Having to relearn graph theory from scratch and struggling with vector mathematics.

As for things that I would do differently there are a lot.

First thing I would do is shift focus more to different parts of the project slowly building up all the parts at once instead of getting stuck and frustrated on one.

Another is reliance on outside help. I did not interact with my supervisor a lot.

I learned a lot of new algorithms that may come in handy when dealing with computer graphics.

Future work

This project is very extensible in several aspects.

Firstly the performance and speed of the generation can be improved with a grid system grouping close nodes into squares as such reducing the time it takes to find intersections.

Another improvement can be made In storage of the street map by using adjacency lists as proposed in [5] :
"Road networks are represented as undirected planar graphs and are implemented as adjacency lists. An adjacency list contains an entry for each node and each of these entries comprises of a list of nodes that this node is directly connected to (see Figure 1). This data structure provides an efficient way to store, edit and perform operations on graph representations of road networks."

Different direction could be expanding the shape grammars by applying them to texture generation as was introduced in [5]. This way in combination with shape grammars for geometry as well as introduction of randomness could fully address the lack of detail in procedural generation.

The most thrilling idea that I did not manage to implement was using marching cubes algorithm in tandem with shape grammars to create alien structures and cities. Redefine natural conventions of doors, windows and building shapes. This is an incredibly riveting idea however I spent too little time on the project to be able to realize it.

Overall conclusions

Having created a city generator that can create an urban environment with variable assets and prefabs I feel extremely fulfilled and proud of what I have accomplished. Furthermore I believe that this is just the start of my fascination with procedural content generation and I will carry this knowledge forward to my next career escapades.

References

1. How Marvel's Spider-Man Crafted a Perfect Digital New York City for You to Save, 2018, Scott Meslow- https://www.gq.com/story/how-marvels-spider-man-crafted-a-perfect-digital-new-york-city-for-you-to-save
2. Interview: The Making Of Dwarf Fortress, 2008, John Harris - https://www.gamedeveloper.com/design/interview-the-making-of-dwarf-fortress
3. City forms: street layouts models, Arch. Mor Temor - http://eng.mortemor.com/index.php/articles/search-good-city/city-forms-street-layouts-models/
4. Procedural Modeling of Cities (2001) Yoav I H Parish, Pascal Müller. https://cgl.ethz.ch/Downloads/Publications/Papers/2001/p_Par01.pdf
5. Citygen: An Interactive System for Procedural City Generation (2007) George Kelly, Hugh McCabe. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.688.7603&rep=rep1&type=pdf
6. L-Systems Considered Harmful, 2007, Sean Barrett - http://nothings.org/gamedev/l_systems.html
7. Constructing a Cycle Basis for a Planar Graph, 2005, David Eberly - https://www.geometrictools.com/Documentation/MinimalCycleBasis.pdf
8. Inset A Polygon By A Fixed, Perpendicular Distance, With C Code Sample, 2007, Darel Rex Finley - https://alienryderflex.com/polygon_inset/
9. SHAPE GRAMMARS procedural generation techniques for virtual cities, Rachel Hwang - https://cis700-procedural-graphics.github.io/files/shape_grammar_2_7_17.pdf