Procedural City Generation
with City Zoning

BSc Computer Science

James Lethem (071505415)
Project Supervisor Dr Graham Morgan

Abstract

This dissertation looks at the modification of the Blended Cities script authored by Jérôme Mahieux to include city zoning. It also looks at the differing opinions of those in the film, game and city generation industry on the usage of such tools providing justification for this project.

The motivation for this project stems from a large interest in the open source community and 3D modelling with the application 'Blender'. This dissertation gives back to this community through the addition of city zoning for an open source script.

The resulting script from this dissertation allows more artistic control over the way that procedural cities were previously generated as it allows users to paint the probability of zones through the use of colour, allowing multiple districts to be created within the city.

Declaration

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

Acknowledgments

I would first of all like to thank my project supervisor Graham Morgan for allowing me to partake in this project. He has helped with the organisation of the project and has always given helpful advice. I would like to thank Jérôme Mahieux for helping me modify the Blended Cities script that he authored. Lastly I would like to thank my friends and family who have been there when I needed them most. To them I dedicate this dissertation.

Table of Contents

Acknowledgments 4

Table of Figures 7

1 Introduction 9
- **1.1 Purpose** 9
- **1.2 Project Aim and Objectives** 9
- **1.3 Dissertation Outline** 10

2 Research 11
- **2.1 Research Strategy** 11
- **2.2 Background Research** 12
  - 2.2.1 Procedural Content Generation 12
  - 2.2.2 City Generation 21
  - 2.2.3 City Zoning 23
- **2.3 Interviews and Correspondence** 25
  - 2.3.1 Chris Preston – Engineering Manager at Ubisoft Reflections 25
  - 2.3.2 Chris White – Visual Effects Supervisor at Weta Digital 27
  - 2.3.3 Steve Rotenberg – CEO of PixelActive 30
  - 2.3.4 Igor Raffaele - Operations Director of Interwave studios 32

3 Methodology 36
- **3.1 Overview** 36
- **3.2 Planning** 36
- **3.3 Elicitation** 36
  - 3.3.2 System behaviours and constraints 36
  - 3.3.4 Functional Requirements 36
  - 3.3.5 Non-Functional Requirements 37
- **3.4 Tools** 38
- **3.5 Development** 42
  - 3.5.1 Version 0.1 42
  - 3.5.2 Version 1.0 44

4 Testing & Results 49
- **4.1 System analysis** 49
  - 4.1.1 Environment testing 49
  - 4.1.2 System Usage 49
- **4.2 Black box testing** 50
- **4.3 Results** 53
  - 4.3.1 Environmental Testing 53
  - 4.3.2 System Testing 53
  - 4.3.3 Black Box Testing 55
  - 4.3.4 Conclusion 55

5 Evaluation 56
- **5.1 User Testing** 56
- **5.2 Fulfilment of objectives** 59

6 Conclusion 60
- **6.1 Personal Development** 60
- **6.2 Future Work** 60

6.3 Future Releases **61**

References **62**

Glossary **64**

Appendix **67**

Testing **67**

All Black 67
All Blue 69
All Red 70
All Green 71
All white 72
Colour Mix 1 74
Colour Mix 2 76
Colour Mix 3 78
Colour Mix 4 80
Colour Mix 5 82

Interviews and Correspondence **85**

Table of Figures

Figure 1: Vegetation in Elder Scrolls 4, Oblivion 13
Figure 2: Screenshot of LOVE 14
Figure 3: Screenshot of LOVE 14
Figure 4: Galaxy generation in Infinity: The Quest For Earth 16
Figure 5: Procedural landscape in Infinity: The Quest For Earth 17
Figure 6: Procedural landscape made with Terragen 2 18
Figure 7: Forrest in Avatar grown with l-systems 20
Figure 8: Sim city zoning 24
Figure 9: Procedural city from King Kong film 27
Figure 10: Procedural city from King Kong 28
Figure 11: City made with CityScape by PixelActive 29
Figure 12: Screenshot of Nuclear Dawn by Interwave 32
Figure 13: Screenshot of Nuclear Dawn by Interwave 33
Figure 14: Perspective view of procedural city made with version 0.1 42
Figure 15: Orthographic view of procedural city made with version 0.1 42
Figure 16: Version 1.0 in action 44
Figure 17: RGB Colour chart showing how colours mix 43
Figure 18: GUI for version 1.1, Inputs tab 47
Figure 19: GUI for version 1.1, Libraries tab 47
Figure 20: Test scene, 1st view 51
Figure 21: Test scene 2nd view 51
Figure 22: Test scene 3rd view 52
Figure 23: System usage when idle 53
Figure 24: Memory usage when running 54
Figure 25: System usage when running 54
Figure 26: Users who would use it for future projects 57
Figure 27: Users who think that it should be ported to Blender 2.5 57

Figure 28: City Zoning article in Blendernation.....58

Figure 29: Painting zones on street map 61

1 Introduction

This section seeks to introduce the project to the reader and provide them with the structure of this dissertation.

1.1 Purpose

As urban environments, cities are often used within the digital medium. As film sets vie to provide their audience with a more realistic performance, the more they need to rely on the use of technology. This is also true with that of the game industry.

This process is expensive due to the large time scale associated and the employment of large numbers of artists in order to achieve a highly detailed virtual environment.

Procedural content generation systems seek to lower this cost by reducing the amount of time needed and the expenditure to create assets for virtual environments. [1]

Procedural city generation is where urban landscapes can be generated to provide environments for the digital medium. For a greater sense of variety and realism, this dissertation looks at the inclusion of city zoning within an existing city generation tool as well as providing the reader with a greater understanding with existing tools and their usage.

1.2 Project Aim and Objectives

The aim of the project is to:

"To advance understanding and techniques in automated city generation within a digital medium"

The project aim is split into 3 objectives:

- *"Investigate how existing tools and techniques are used for city generation"*
- *"Investigate the structure of an existing city generation tool"*
- *"Design, Implement and Document a city zoning algorithm with the Blended Cities script"*

1.3 Dissertation Outline

Introduction

An introduction to the dissertation detailing the problem domain and an explanation as to the project's aim and objectives

- Purpose
- Project aim and objectives
- Dissertation outline

Research

Research into the project area including interviews and correspondence from experts in industry

- Research strategy
- Background research
- Interviews and Correspondence

Methodology

A detailed account of what was carried out in this project and why

- Elicitation
- Tools
- Development

Testing

An analysis and summary of the project results

- System analysis
    - Environment testing
    - System usage
- Black box testing

Evaluation

An evaluation by those who would use the tool

- User Testing

Conclusion

- Fulfilment of objectives
- Summary of achievements
- Personal development
- Future work

2 Research

This section shows the research taken on this dissertation. Note that this is one of the larger sections of the document due to the research objectives for the project.

2.1 Research Strategy

Research was conducted in three main areas:

- Community
- Professionals
- Papers & Magazines

Research through the community was through posting in forums and discussing on IRC. The main research found through the community was finding their opinions on city generation and its uses. It was also a means of finding professional developers and artists who were familiar with this concept.

The main sources from the community were:

- www.blenderartists.org/forum
- IRC, freenode - #gamedev, #unity3d, #blendedcities, #blender, #blendercoders #blenderpython

Research from professionals was by contacting companies and developers within the digital media industry or with those developing generation tools; directly in person or by email.

This provided information as to the kind of clients of existing software, how they are developed as well as opinions within the digital media industry on the strengths and weaknesses of using such a tool.

The professionals contacted were:

- Chris Preston
- Chris White
- Steve Rotenberg
- Igor Raffaele
- Robert Shinka

Research on papers about city generation and generation techniques in general was by searching for through the use of the university library, google scholar and organising through the use of Mendeley.

Research from magazines came from reading ‘gamedeveloper’.

2.2 Background Research

2.2.1 Procedural Content Generation

This sub-section looks at procedural content generation and how it is used from the point of view as other projects as case studies.

There are 2 different kinds of procedural content generation [1]:

1. Online
2. Offline

Online refers to procedural content generation 'on the fly'. Examples of this are in-game effects such as fire and water, vegetation and textures.

Offline refers to where content is generated for an artist to use as a starting point.

These 2 kinds are at opposing sides of the spectrum, it is important to note that procedural content generation is a vague term. This dissertation will be focusing on an offline open source procedural city generator.

Speed tree is a middleware application, which has been used in many games such as Elder Scrolls 4 Oblivion to create the lush vegetation that can be found in the environment. It would be nigh impossible for an artist to create the landscape present in this game by hand where they would have to place every single plant.

"Using SpeedTree allowed us to put vegetation in our game like never before," said Todd Howard, Executive Producer of Oblivion. "It saved us time in implementation and the SpeedTreeCAD tool was essential for our artists to model the hundreds of foliage variations in the game." [2]

Vegetation must be carefully placed. By example this would mean that desert plants such as cacti would not be seen in a tropical environment.

Figure 1: Vegetation in Elder Scrolls 4, Oblivion

"Eskil Steenberg is an independent game and software developer and researcher. He has worked on a wide range of projects like the network protocol Verse, and the 3D modeler Loq Airou. He has a past in the OpenGL Architectural Review board and has released his procedural multiplayer adventure game "Love"" [3]

"LOVE is a cooperative online first person adventure game. You play as a scavenger on a small planet who together with other scavengers will build a settlement by placing a Monolith some where in the world. This Monolith makes the ground lose so that you can shape the environment around it in to what ever you want. Build walls, catacombs, houses and shape your settlement any way you want." [4]

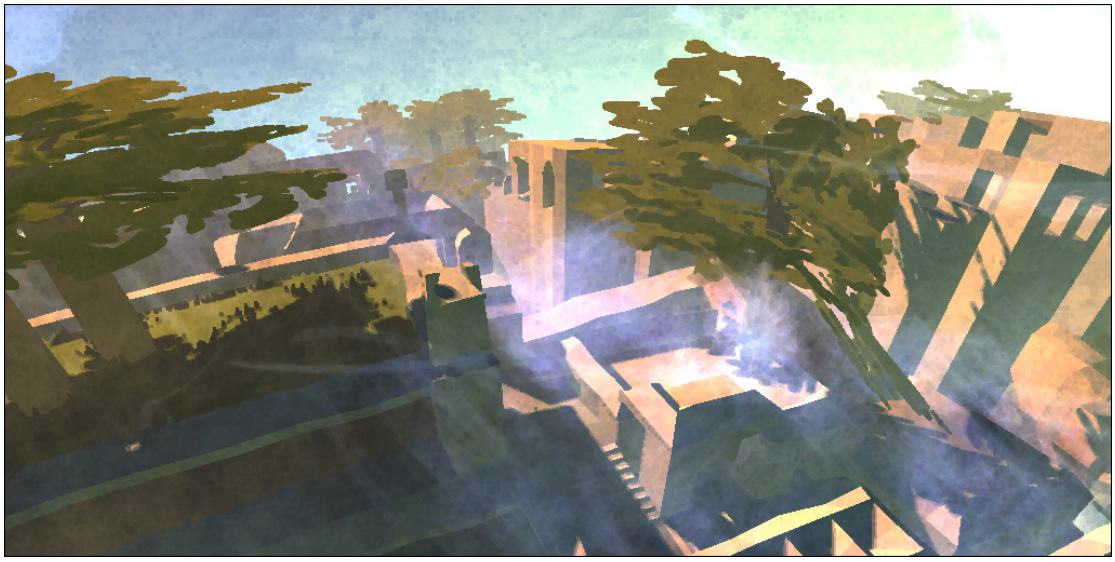


Figure 2: Screenshot of LOVE

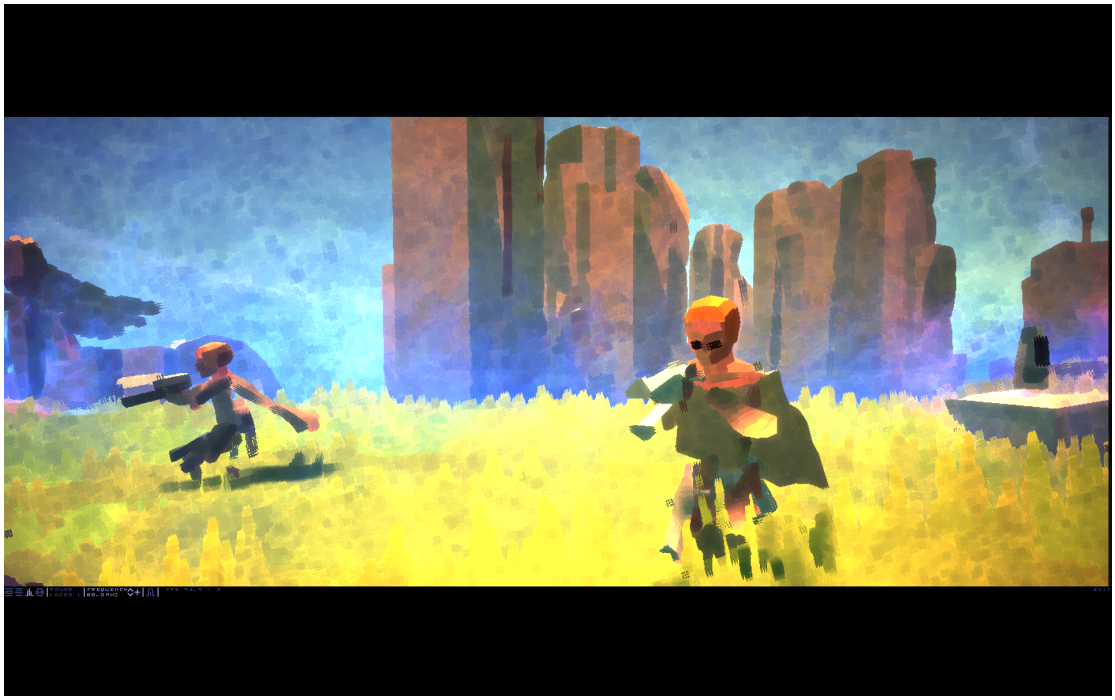


Figure 3: Screenshot of LOVE

"The earliest games to have procedural generation used it to create massive amounts of content on machines that neither had the memory or storage to keep them. Rescue of fractalus, Populous and Elite where early procedural games that managed to do huge games on very limited hardware.

The problem is that the generation of content needs to be deterministic. This means that you must be able to determine the state of a particular peace of data with nothing but the position of that data. (you simply don't have the memory or anything else so it cant depend on whatever is around it) This means that it is ridiculously hard to make any interesting content and since the content is generated while the game is being played it needs to be perfect since no game designer can go in and fix it. More then anything the main benefit to

this approach is that you need very little storage and ram, but these problems have largely been overcome by modern hardware.

So whats the point of Procedural generation? The main problem of game development today is not lack or ram or storage, it is the cost of filling it with content. By moving the procedural content creation in to our tools we can solve that problem. The tools will output data during development that we can use in any normal engine that that is data driven and the designers can properly test and tweak any content that is generated before shipping it." [5]

It is important to realise that ram or storage space are no longer the same problem when it comes to designing tools for developing games. In recent times it is more of a problem to use these areas efficiently rather than not having enough. Procedural content generation is very cheap for designing huge worlds; this can be seen more in a game currently in development called Infinity: The Quest For Earth.

"In terms of Infinity: The Quest For Earth, the game is built around many thousands of lines of procedural code (mathematical algorithms, etc). Imagine these thousands of lines of code as a big machine: you put a number in and it produces a galaxy complete with stars and planets. If the program was given a different seed, then the resulting galaxy would be entirely different. This system works on various levels as well: the seed that determines the layout of the galaxy will in turn produce seeds which determine the layout the solar systems. If a solar system is generated and you explore it and leave, then the next time you come back it will be exactly the same due to the fact that the seed is the same.

When put into terms of physical memory usage, procedural programming actually takes up LESS space than traditional programming. Only the physical models of starships and the textures for the ships and planets are stored on your hard drive (as well as the thousands of lines of procedural code, which only take up a few megabytes at most). Once the seed is provided, your computer essentially turns into a galaxy generating machine. When you arrive at a new star system, your computer will generate that star's planetary system for you. Everyone else that is online with you will have the same code and seed as you, so they will see the exact same star system that you do. The generation is all done on your computer; only the seed is loaded over the internet, which frees up valuable bandwidth.

There are downsides to this technique: while every planet and solar system will be different, there will be many similarities between them. However, this should not be entirely surprising; Earth-like planets will generally be similar in atmospheric composition and fundamental biology and will therefore look similar from a distance. It will instead be the details of the planets that separate them (landscape, continent shapes, etc). In some ways, this is the great advantage of procedural generation: we can make incredibly vast environments, and even though much of it will be empty and repetitive, some areas will be interesting and noteworthy. Still more, you will even occasionally find something that is absolutely stunning and extraordinary!" [6]

This is a very good way of using procedural generation with a real time game and is somewhere in the middle of an online, offline procedural system.

It is important to research how procedural systems work in games as it aids development of algorithms for city development. For instance it could be possible to

design an algorithm for a game where the city is generated in a similar fashion to how the galaxy is generated in this game. However the artist can only control the design of such a galaxy through changing the constraints and parameters. It may be easier for an artist to be able to interact with the data and have the freedom to change it. This may not be possible in this game as the procedural code is much like an algorithm where the single seed generates everything. It is also not needed for the players as they will not be changing the game state nor should they be allowed to have this level of control over the game.

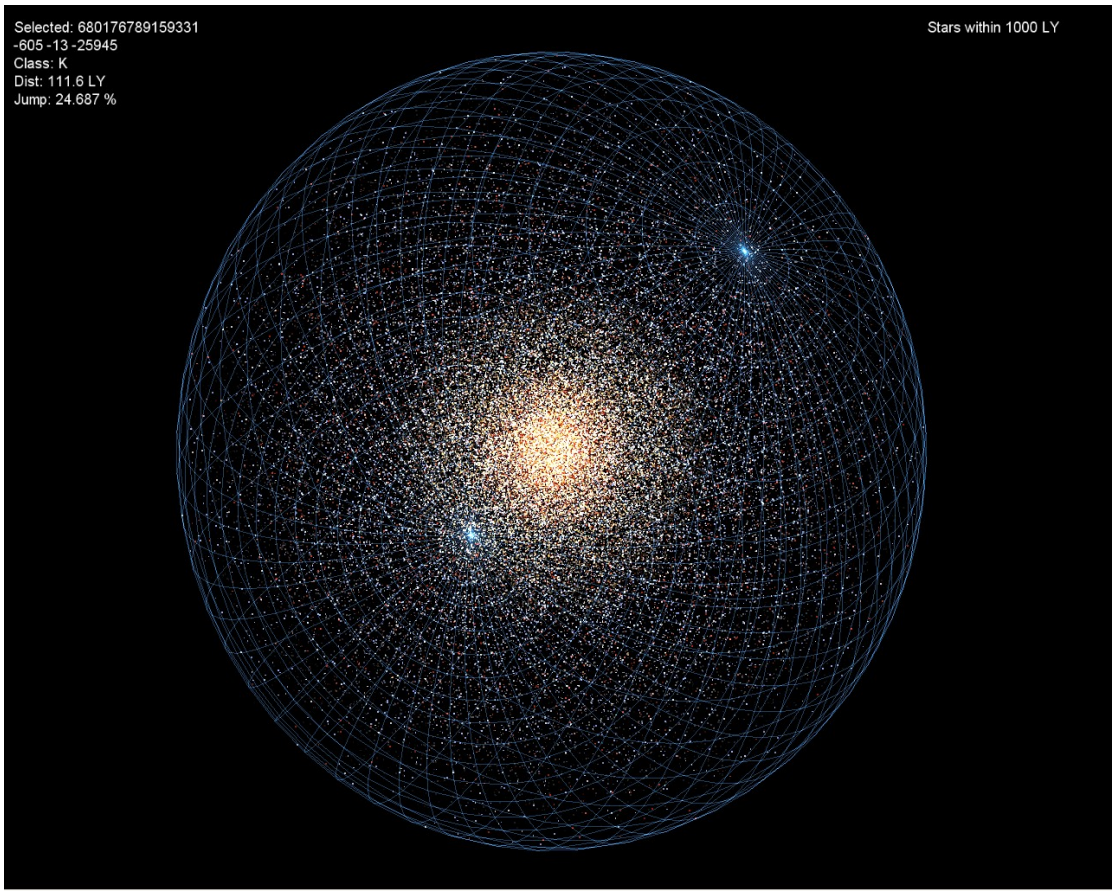


Figure 4: Galaxy generation in Infinity: The Quest For Earth

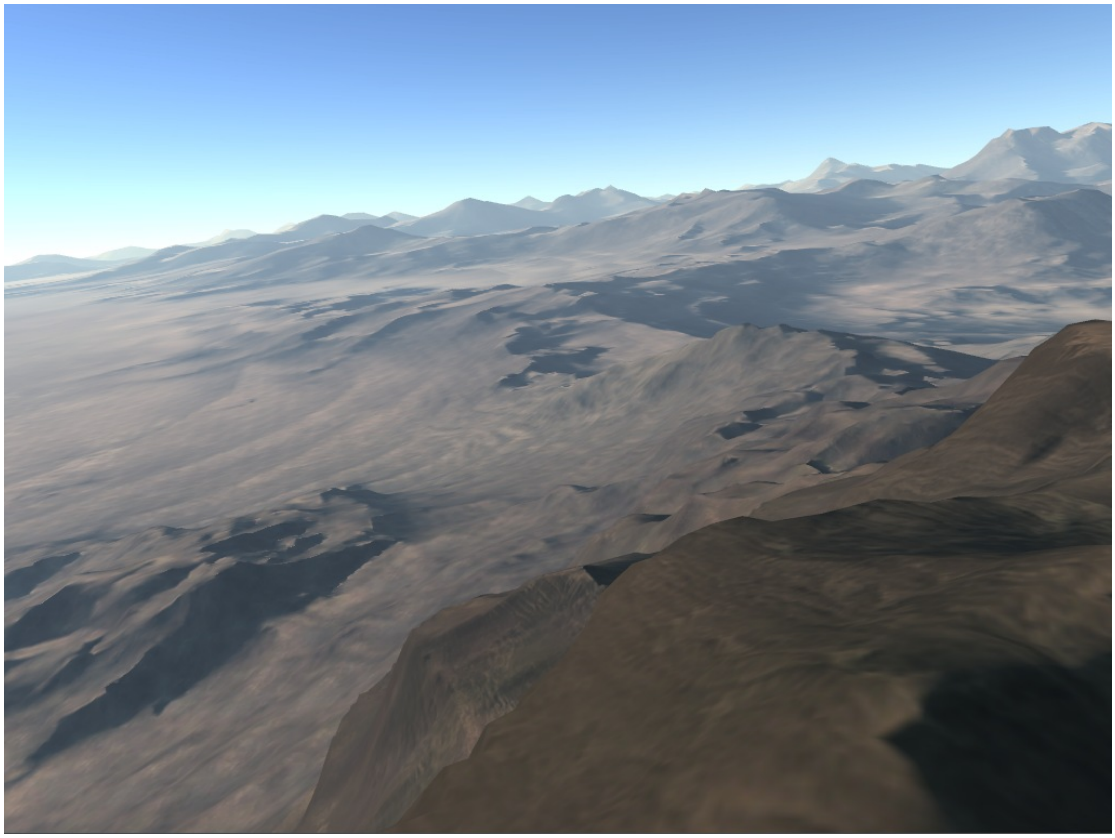


Figure 5: Procedural landscape in Infinity: The Quest For Earth

Infinity space also used procedural techniques when creating planets, this can be seen in figure[] with the landscape. A tool which allows the artist to create terrain is Terragen.

"Terragen™ 2 is a powerful solution for rendering and animating realistic natural environments.

Terragen is not a game engine; it has a sophisticated film- and broadcast-quality renderer and procedural modelling tools designed so that you can create the most realistic images possible without taking a photograph." [7]

Terragen allows the artist to interactively make the landscape in a point and click approach.

Figure 6: Procedural landscape made with Terragen 2

Since space seems to be a problem with producing large numbers of content, it is important to look at the work of '.theprodukkt'.

".theprodukkt is a project in generative computer graphics. we spent the last few years developing non-interactive demonstrations under a strict artificial size-limit for both program code and data.

We want to create a computer graphics content creation system that:

- *Combines textures, materials, meshes, scene management, lighting, animation, composing and everything else you need to create 3D computer graphics in one tool.*
- *Does this within a procedural and modular framework.*
- *Is able to import data from other tools where procedural approaches won't cut it.*
- *Is highly productive for the trained artist.*
- *Integrates aspects needed for game development, like AI and interaction.*
- *Integrates sound and music.*
- *Scales to large worlds*

" [8]

"The secret behind all this is procedural content generation. in a nutshell, instead of storing a movie as-is we're storing a certain number of mathematical formulas for image and audio manipulation as well as the "recipe" how to apply those small bits of code in a way that what you see comes out. And while it's running your computer's graphics card cares for the actual displaying ot the world, just like in modern 3D games. Formulas+recipe are of course way smaller than the end result, and with a few additional sprinkles of magic

compression fairy dust we get down to under 200 kilobytes, which is way less than one single high quality image of this show would take."[9]

This is similar to that of Infinity Space, by making everything in algorithms and making it run at real-time. This project shows us that we can make detailed systems with just code with small file sizes. For premade buildings Blended Cities makes instances of one object. This whilst still taking a lot of memory up at the beginning helps decrease the render time of a scene. Procedural buildings are also used where they are generated when the city itself is generated. One of the more complicated procedural buildings (littleneo – residential) is only 25 KB in size.

Avatar also uses procedural generation for the creation of vegetation.

"It was very interesting. You could actually watch a forest grow in real time with this solution, and any TD could grow just by painting colors on the terrain." With this elegant solution, the big trees would grow first, then the smaller trees would die off as the big trees took away the light, the smaller trees would fight for position, the ground cover would fill in where it could get light."

By painting colours on the terrain, artists were able to use this as input for the program. This could then be used to determine how the trees would interact with each other. The use of colour is important, as this is an easy way of allowing artists to control the outcome, as it is a simpler concept to understand than changing code or sliders and parameters to affect the outcome.

"A lot of our modeling techniques were procedural, so we wrote a tree building L-System type that allowed us to build lots of variations on trees, plants, and ground covers in a very efficient way. They came out essentially rigged so we could do dynamics and interactions." [10]

L-Systems are an efficient way of creating multiple variations of vegetation. But they can also be used with the generation of cities.[11]

Figure 7: Forrest in Avatar grown with l-systems

2.2.2 City Generation

This sub-section looks at city generation and the different ways on how it can be used.

One way of designing a procedural city generator would be to allow the input of a small set of statistical and geographical data, which means that the application would be highly controllable. Maps in the form of images would control certain aspects of the city [11];

- Geographical Maps
  - Elevation
  - Land/Water/Vegetation

- Socio-statistical Maps
  - Population
  - Zone
  - Street pattern
  - Height

Müller uses L-systems for creating the road infrastructure of the city. Robert Shinka, a bio- technician from UL Studios describes L-systems:

"The current trend for L-systems use is as a form of fractal generation, which (although similar to Lindenmayer's original usage) does not accurately model the "evolutionary" processes involved in an organisms life cycle. In the case of an algae, as new cells are generated, they're primarily an addition to the previous state, until the prior generations have matured sufficiently and begin to die off. The population decline is usually factored into the rules for growth rather than explicitly including decay, keeping the systems simple.

Deterministic context-free L-systems are useful for a simplistic modelling of, for example, the shape of a plant, or a pattern of roads within a city, but the other attributes (such as the thickness of a branch or a cells walls) are usually left to another system to describe. When coupled with the observable repeating patterns typically present in fractals, this separation of attribute generation causes the model to appear inconsistent and usually very unrealistic.

In a city, there is usually a discernible growth pattern where traffic on roads, the size of roads, the space reserved for parking, the number of residents, and the amount of, types of, and proximity of business are all very closely interrelated. An otherwise well-generated city may appear unrealistic if the building density is too sparse along the most easily-travelled intersections. Similarly, the vessels in a multi-cellular organism must agree with the needs of the surrounding tissue, roads leading to nowhere are seldom built, and leaves on a severed branch will quickly wither and die. Accurate modeling of problems such as these require a feedback loop that context-free systems lack.

Although they are typically more complicated, none of these problems need be present in context-sensitive L-systems that consider (and possible alter) prior generations, and the increase in complexity is made up for by ability of these systems to cleanly operate on a large number of state variables. Given enough rules, each of which can be expressed in an independent constraints-logic clause, you can run elaborate simulations on a set of values (such as a city) until it evolves to an acceptable level. In order to avoid an obvious repetitions of patterns, these systems can alter the existing dataset by transforming individual nodes (or groups of nodes), then use the resulting state to source dependent variables for the next iteration.

Another advantage of these systems is that they can be easily implemented via any pre-existing graph-rewriting system simply by repeatedly applying the entire set of rules to the graph, which also happens to lend itself extremely well to functional approaches. A

context-free system can achieve greater performance by keeping separate track of edge nodes, but doing so actually requires an overall more complex implementation, despite the lesser expressivity of the production rules.

The only significant disadvantage to the approach is the increase in resources needed to perform each iteration: in addition to generating the next generation of neighbouring nodes, each previous generation of nodes must also be transformed. The cost is more than made up for by the increased realism granted by a quality implementation, particularly in the case of static content generation where there is no need to conserve cpu resources."

Games such as EA's Need for Speed racing games can benefit from procedural generation of urban environments[12]

"Game artists aren't looking for a one-button procedural solution. Instead, they're interested in procedural methods that help with tedious tasks and provide results that adjust to gaming constraints. Procedural methods should free artists to spend time creating and polishing, rather than performing mundane, repetitive, and time-consuming tasks."

This is an important point as Blended Cities is not just a 'one-button procedural solution'. Artists using the tool can adjust settings to achieve what they need. With zoning, it is important that it is developed to be useful for the artist and help them with the tedious task of building placement.

This paper also describes the challenges faced when authoring terrain in urban gaming environments:

- Providing correct shapes, locations, and sizes of foundations for buildings;
- Minimizing polygon density to meet memory budgets;
- Minimizing unique textures; and
- Having a uniform UV mapping across the entire terrain, including building footprints.

Recent Need for speed games must aim for 30 frames per second. Because of this, efficiency is very important.

The applications that city generation can be used for are as follows [13]:

- Tourism
- Marketing
- Architecture
- Town planning
- City climate
- Noise propagation and environmental research
- Navigation systems
- Training
- Mapping

[14]

- Films

2.2.3 City Zoning

This sub-section looks at city zoning, the definition and how it is used in the game SimCity.

Professor Michael Pacione published a book by the name of: The City: Land use, structure and change in the Western city. He refers to city zoning as:

"the first fundamental step in any city to establish a practical basis for constructive growth."

He mentions the three different zones:

- Residential
- Commercial
- Industrial

And how there should be separation between these zones to allow safe places for industries to grow and the protection of residential zones.
As an artist, zoning is also important as it gives a layer of control as to the placement of buildings in a city. But it can be more than this, city-zoning affects so much more.

"*We cannot discuss residential zoning, the subject originally assigned for this paper, without first considering its objects within relation to industry retail business. From practical experience in several cities, we found that the best way to draw boundaries of residence is to work out first the business and industrial boundaries – what is left will naturally form the residential zones..*"

This is important as it recognises that all zones should be realised with how they interact with each other. The statements below emphasise this point:

"Industries must have contented home condition for employees"

"Demand protected home neighbourhoods for the poor as well as for the rich"

"Industries must have a safe place to invest"

As the book suggests, it is unreasonable to place residential housing in close proximity to industry, as they can be seen as 'offensive neighbours'. The location of residential zones also directly affects the prosperity of commercial zones.

Zoning can also be found in the game Sim City;

"It was released in 1989, being the first computer game simulation to accurately model a city and its inner workings. Using the software, players could build their own utopias of bustling megalopolises or quaint little towns. The game provided intricate detail at the time: a realistic traffic model which featured roads and rail, a zoning system made up of residential, commercial, and industrial zones, and a complex simulator that made it all run together." [16]

Players place zones down on the landscape that virtual citizens can choose to populate if they wish to. In the original game there were 3 zones:

- Residential
- Commercial
- Industrial

By placing these zones and having them occupied, players could create realistic models of cities. However there are some problems from a zoning perspective in that you cannot mix zones. This could be because it would be detrimental to game play and would probably break the game mechanics if zoning was designed too complicated.

[17]

However this problem must be recognized for developing a city generator with city zoning.

Figure 8: Sim city zoning

2.3 Interviews and Correspondence

These interviews and correspondence look at the opinions of those within the industry and their views with regards to city generation.

2.3.1 Chris Preston – Engineering Manager at Ubisoft Reflections

Bio

Chris Preston is Engineering Manager at Ubisoft Reflections, the Newcastle-based developers of the Driver series. He joined the studio over 7 years ago as a Tools Programmer but now manages the engineering team on the studio's next AAA game for home consoles. A Computer Science graduate, he got his big break in the games industry with Codemasters in the mid-nineties, working mainly on the original Colin McRae Rally. He has nearly 15 years professional experience of software engineering and technical management.

Discussion

Preston mentions that there are two main arguments against the usage of procedural content generation and one argument in favour;

Against:

"It's extremely hard to procedurally generate the kind of subtlety, detail and uniqueness that a human designer can put into an object. Whether that's gameplay nuances, visual detail or whatever. Not impossible, just very hard, and there aren't many examples of it having been done really well. It's the opposite of randomness or natural effects - objects in our world are specifically and carefully designed by people to deliver on a given purpose. For example, so-called 'new towns' were effectively procedurally generated - a strict pattern of development for their estates was laid out in central guidelines. Result - soulless places that only became nice as successive overseers made their personal marks."

"It's completely different from how most developers are used to working and thinking, and it's an area of sufficient complexity (if you want great results) that it's very hard for teams to make the jump. They set a certain quality bar with their manual processes and it's hard to guarantee they won't go backwards if they change to procedural generation."

In favour:

"Modern hardware is extremely good at doing the kind of work involved in procedural generation. If you want to make maximum use of a PS3, you really need to be using some procedural systems to prevent you being limited by the memory size - it can process a lot more data than it can hold in memory."

The first argument brings forward a very important point. The player needs to feel immersed within the game. This is not possible if the environment feels wrong to them. An artist can leave a personal touch on their creations, a program can only do this if

programmed in a very specific way. It may be possible in the future for computers to create designs similar to an artist.[18]

His second argument points out that artists have a huge creative threshold that would be difficult for a computer to replicate. This also draws back into the first argument where computer created designs are not up to par from that of artists. This point also suggests that developers are not used to thinking in such a way. The workflow of using these tools is completely different if the artist is relegated to using the tool and modifying the output, rather than making everything. This is important to realise as such a change in work flow would need retraining of all staff for this technology to be used successfully.

His final argument points out that modern hardware already allows this technology to be used and with the limitations of memory in console systems procedural systems need to be used to give the player a quality performance from their game.

2.3.2 Chris White – Visual Effects Supervisor at Weta Digital

Bio

"Chris White is a Visual Effects Supervisor at Weta Digital in New Zealand. His recent work consists of Avatar, The Lovely Bones and The Day the Earth Stood Still. Chris began his work in visual effects over 15 years ago at George Lucas' Industrial Light & Magic. Working as intern, White began writing dynamics software to be used for the film Twister. After being asked to continue on at ILM to work as a R&D Technical Director, he developed special effect looks and techniques for the next seven years. From exploding "pods" and asteroids in Star Wars Episode I and II, unruly oceans in The Perfect Storm, to battling dinosaurs in Jurassic Park III, he supervised on a variety of challenging projects." [19]

Figure 9: Procedural city from King Kong film

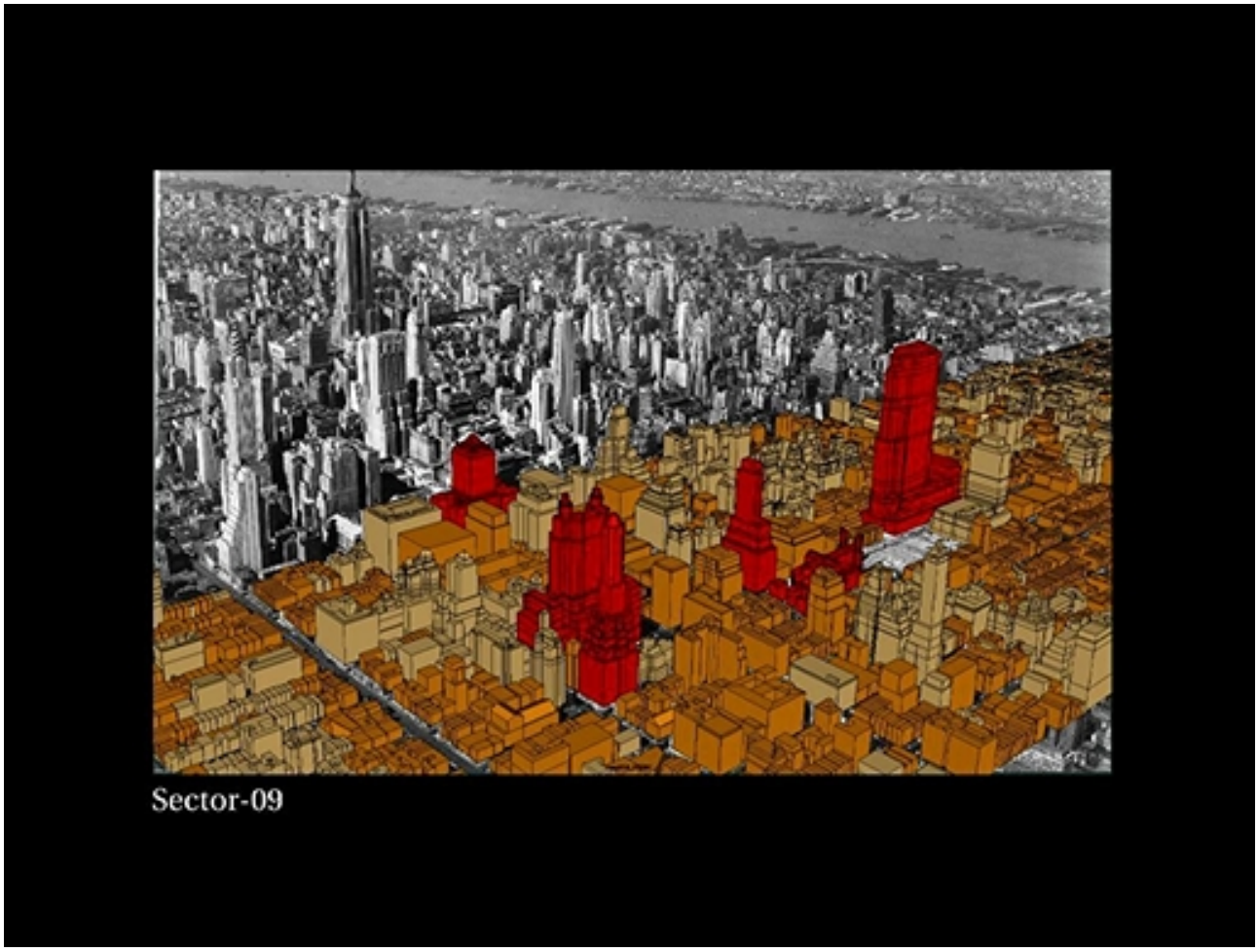


Figure 10: Procedural city from King Kong

Discussion

"CGA: What was the brief you received to design and plan the architectural shots of New York?

CW: *Our goal was to create a photorealistic environment as accurate to 1933 New York as we could. Coinciding with our digital work, there was also a New York set built in New Zealand. This set was built up to the first story, with digital extensions to be added later to complete the scene. We were also asked to make it all 3D, so the director would have complete freedom to send the camera anywhere in the city.*

CGA: Tell us about how you obtained architectural data for the buildings for pre-1933 New York and how you went about determining what would be modeled.

CW: *For specific landmark buildings, we worked off blueprints and period photo references. Each of these buildings would be hand modeled. For the thousands of other buildings the task would be much too time consuming to model by hand.*
For these I began developing a piece of software called the "CityBot". This software created procedural 3D buildings to be rendered using Renderman. Before the project would be completed, we would create over 90,000 3D buildings. All of Manhattan Island would be 3D as well as parts of New Jersey, Brooklyn, and Queens. Every building on Manhattan was

unique and built to the finest level of detail. To bring the city to life we also had traffic, boats, and working chimneys, factories, and el trains."[20]

It is important to note that the director wanted freedom where he was able to send the camera anywhere in the city. This could be so that he could take the shot from any angle. This would have been impossible if the city had been made through a matte painting instead of being modelled.

When asked about the strengths and weaknesses of city generation, White answered:

"The strength is the ability to create massive amounts of buildings with lots of detail. Something that could never be done by hand. It gives the director the flexibility to take his camera anywhere. It also adds more dimension to the final image since it is 3D, and not a fixed painting.

The weaknesses with any procedural system, whether it by city generation, particles, or textures is art direct-ability.

When the client wishes to break the rules, you need to have to ability to adjust things by hand. A procedural system also will not give you a exact match to the existing building on site, since it is using rules to construct buildings. So for signature New York landmarks, we had to still model those by hand."

In the film King Kong, White uses city generation (CityBot) to create over 90,000 buildings [20]. This would have been impossible for an artist to complete within the time period that his team had. Another way this could have been done would be for he artists to use matt painting for the background, but as White suggests; dimensionality provides the scene a greater visual impact.

The first weakness he suggests is that if the director requests a change to the scene they would need to alter the models by hand. This may either be on the scale of one building or many. If it is one building then they could edit the building mesh with an artist. However with this approach, regeneration of the building with the generator is not possible. It would be possible though if the parameters were changed within their tool.

The second weakness that he points out is that when generating certain buildings it is hard to get the rules that you set out to work in such a fashion that you get an exact visual match. This is more noticeable if the building has stylised architecture or warped symmetry. In this case the buildings must be done by hand.

2.3.3 Steve Rotenberg – CEO of PixelActive

Bio

"Steve Rotenberg has worked in the computer graphics and video game industry for over 15 years. He served as both Director of Software and Director of Research during his 10 years at Angel Studios from 1992-2002. During that time, he was involved in development of real-time technologies such as physics simulation, character animation, rendering, and played a key role on many video game titles including the Midtown Madness, Smuggler's Run and Midnight Club franchises (MobyGames).

Since 2003, Steve has taught as a lecturer in the computer graphics department at the University of California at San Diego (UCSD) covering topics such Computer Animation, Computer Graphics and Video Game Development."

[21]

CityScape

"CityScape is the leading-edge rapid urban modeling solution for real-time environment generation. Using proven game technology and an intuitive interface, CityScape allows artists and designers to quickly iterate and refine complex urban data sets in a fully rendered real-time 3D environment."

[22]

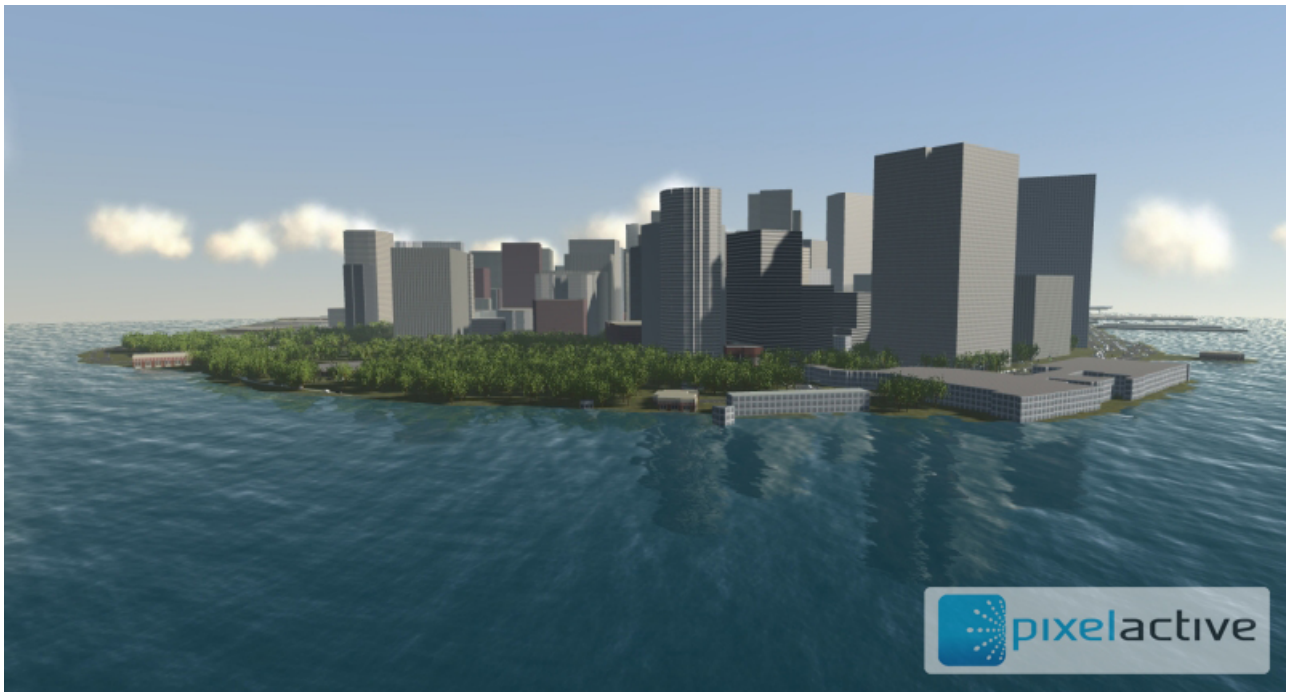


Figure 11: City made with CityScape by PixelActive

Discussion

When asked about the clients that use of CityScape Rotenberg answered:

"Originally, our product was designed for video game development. Currently, the industry we do most business in is actually mapping/navigation, with military/simulation/training coming second, games third, and urban/civil planning last."

It is important to note that the gaming industry is third from this list of clients. This could suggest that either the technology has not yet matured enough for developers to take on board.

"We keep up with all current research in city modelling and have followed Pascal's work for many years. I would say that our primary motivation has been to build a tool that allows you to interactively and explicitly model a city exactly as you want it, rather than a tool that builds a 'random', or 'rule-based' city. This has caused us to focus more on fast interactivity, powerful user interface, reliable undo/redo capability, handling of very large environments, support for multiple simultaneous users, and other key features that our customers want. As I mentioned earlier, nobody has ever asked us for something to generate a random or rule-based city, as neat as that stuff might be. Generally, our customers don't want to write scripts or use grammars to generate cities. They want to point and click with a mouse."

Here we have the main point that artists need to have control when designing cities. With this in mind, CityScape also allows multiple users to build the city together. It should be mentioned that in this program the artist paints the surface of the world similar to how they would paint a picture, however they are in fact describing to the program the parameters for which the program will use to generate the environment.

"Do you see a future for this area of development?

- Absolutely. Especially in the urban planning/management, civil engineering and transportation engineering fields.

Any particular future plans?

- Bigger & bigger worlds. We're working on supporting Earth-sized models."

With Earth-sized models, games may evolve to ever-greater heights.

"One more thought: on the subject of 'procedural' city generation... our customers fall into two categories: ones that want to interactively design their own custom city, and those that want to build the real world as it exists today. In the 7 years we've been in business, no one has ever asked us for a 'random' city generation tool, 'rule-based' generation, or anything related to that."

This is an important point; that random city generation is not really wanted as it will not give a designer what they want as it is very hard to control. For this reason Rotenberg concentrates on creating software which allows users to interactively make the city through a point and click approach.

2.3.4 Igor Raffaele - Operations Director of Interwave studios

Bio

"Igor Raffaele came to the games industry from a career in marketing, PR and gaming journalism. After talking about games, selling games and writing about games, the only possible evolution of his career was to do some actual work, and start making games. Interwave Studios, brimming with new ideas and prospects was the perfect place to start. Igor is currently involved in the design and marketing of Nuclear Dawn, along with several as yet unannounced projects."

Nuclear Dawn

"Nuclear Dawn is an upcoming Source Multiplayer game that combines the visceral action of a First Person Shooter with the deep branching gameplay of a Real Time Strategy title.

Forge your own destiny on the field of battle with every tactical decision, and ensure your faction's survival with each headshot, capture buildings and bring the fight to your enemies' strongholds riding the future's machines of destruction.

The world has changed, it has been swept clean by the winds of change. Do you think you can survive it?"[23]

Figure 12: Screenshot of Nuclear Dawn by Interwave

Figure 13: Screenshot of Nuclear Dawn by Interwave

Discussion

"My main point is that procedural generation is more or less useful, in relation to the type of game you're making.

I replied, as an example, with Nuclear Dawn, our own game, which is a multiplayer online shooter set in real-world cities, but after a devastating third world war.

In such a context, procedural generation is of extremely limited use. It cannot be applied to the levels themselves for a number of reasons:

1) It cannot be used for main level blocking (since we're modelling on real cities)

2) It cannot be used to effectively create interactive objects such as cover or debris, since it would require an extreme amount of constraints - so much so that it would essentially be limited to a mechanism that only produces a few patterns that conform to the same requirements."

It is important to note that he mentions that the usefulness of procedural generation depends on the type of game that is being made. Shooting games require large amounts of detail close to the player to help make the game more believable. This is in contrast with racing games where the player is not so interested with minute details close to the vehicle.

Point one suggests that as the cities are based on real life examples, a randomly generated building is not ideal. It is possible to develop models for procedural generation of buildings with large numbers of constraints to use in this scenario but if a

change is needed later on in development, it is easier as an artist to change the structure of an object rather than add new rules and constraints for the generation of a building.

Point two suggests that the large number of constraints needed for the generation of cover and debris would only allow a small number of original patterns that conform with the requirements. It could be possible for computer-aided design to be used in some kind of fashion for designing these objects where the user teaches the computer what in essence is a good design.

"Finally, such placement would have to be FUN - which is not something you can leave to a procedural algorithm. Each single prop in our game has been moved (often a few inches at a time) to be placed where they would best conduct specific gameplay situations (furious, fluid firefights, defence, last stand scenarios and so on)

In such cases, procedural generation is too random, and not intelligent enough to make the most out of each situation. In order to be appealing, such procedural systems would have to be highly controllable, with a large amount of input variables. It is my opinion that once you're done inputting all the variables, you might as well have created a number of equally valid scenarios by hand."

The most important part of a game is the gameplay. As he suggests, the environment is modelled with the player in mind and how they will interact with it. He notes that procedural generation is random and not intelligent enough. This goes back to a similar argument before where the computer could be taught how to design certain aspects of generation such as the placement of objects.

When asked about the future for this technology and the features that would be needed for him to consider using it in development, he answers:

"I already see applications for your technology today. In the right kind of game, it would be an invaluable prototyping tool. In order to be commercially appealing, you would have to provide a great amount of control in the procedural generation (such as the definition of weight maps, zone definition, cutoff points and several other parameters).

I think that procedural generation will be much more appealing once modern engines move to a true real time lighting model. Once that happens, it will be feasible to generate levels that strike a balance between variations and playability, but that maintain the highest standard of graphical appeal."

He mentions that the tool could be an important asset for prototyping worlds for particular game types. He also mentions that for it to be appealing, greater control needs to be added. Thus allowing the artist to influence the scene to their liking.

He notes that lighting is a major problem as:

"It's hard to bake an impressive lightmap when you don't know where your objects will be."

With this in mind, once modern engines develop a better lighting model then procedural generation would be able to offer more in terms of quality and variety.

Summary

Preston, White and Raffaele all talk about the importance of details, which can be hard to gain when using city generators. White manages to do this in King Kong by collecting city data on New York over a period of 2 years. This is difficult for a game developer to do and so city generation needs to mature more before being used to create levels with vast amounts of detail.

Raffaele mentions that he wants artists to have complete control over the tool and so in this case perhaps CityScape used as a point and click tool would be useful. It may also be ideal as CityScape also allows maps to be imported in, this would allow an artist to rapidly prototype cities through the use of geographical and socio-statistical maps.

The various sources found in background research correlate to the views found in the interviews and correspondence where procedural content generation is a good tool to use for specific needs.

3 Methodology

This section details what was done in the project and why. It must be mentioned that the contribution was made to an open source project and so some of the work done was rewriting existing sections of code. In later versions of the script the work that was contributed was mostly designing, as code was rewritten by the script author for greater efficiency. The work accomplished will be made clear with each version.

3.1 Overview

The main tasks in this project were:

- Research into the project area
- Planning the project
- Software design and development
- Evaluation

3.2 Planning

Planning for this project was difficult, at first it was decided to use a waterfall model but within weeks it was realised that an iterative plan was needed due to the contact that needed to be made with Jérôme Mahieux and the community at regular periods. This was sometimes a problem with time commitments on both sides.

The majority of background research was done in the first semester whereas interviews and correspondence were given in the second semester.

The implementation of version 0.1 was done early in the second semester with the more recent version 1.0 over Easter. Version 1.1 is still being developed as of now and is still in an alpha stage without public release.

3.3 Elicitation

3.3.2 System behaviours and constraints

The system was designed with artists in mind and so has low technical skill requirements. This meant that the interface had to be easy to understand so that the artist would be able to use it.

3.3.4 Functional Requirements

FR1 City Zoning

FR1.1 The user will be able to choose what zone a building library is situated in

FR1.2 The user will be able to choose whether a building library is to be used

FR1.3 The user will be able to mix the zones together

FR1.4 The user will be able to choose the effect of a zone

FR2 Documentation

FR2.1Code

FR2.1.1 The code will contain documentation

FR2.2 User

FR2.2.1 A user guide will be created

FR2.2.1.1 A video tutorial will be recorded showing the use of the script

FR2.2.1.1 A written guide will be created to show how the interface works

FR2.2.1.1.1 The written guide will be translatable

FR3 Script

FR3.1 The addition will work with the existing mechanics within the script

FR3.2 The addition will work for the supported systems of the script

FR3.3 The addition will follow the GUI design set by the script

3.3.5 Non-Functional Requirements

NFR1 City Zoning

NFR1.1 The script will be easy to use for the artist

NFR2 Documentation

NFR2.1 Code

NFR2.1.1 The code will be easy to understand

NFR2.2 User

NFR2.2.1 The documentation will be easy to understand and follow

3.4 Tools

This section looks at the different tools used within this project and what was learnt for their usage.

Blended Cities

Blended Cities is an open source python script for Blender written by Jérôme Mahieux for City Generation.

"Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License."[24]

By using python, a developer can access Blender's API to interact with the system.

At the beginning of the project it was very important to learn the language Python and how the Blender API worked. This was done by making small scripts, the first of which was aiding someone in the community [25] and the second was a script that ported objects made in Blender to pure OpenGL code. This was especially useful for other pieces of graphics coursework given out this year. [26]

To begin developing with the script it was important to understand how the script works and for this, correspondence was made with the Author:

> If possible could you include a short bio on who you are and your motivation for blended cities?

"Between two jobs as a network consultant, I discovered Blender by curiosity about 3d modelling and animation. 4 years later, it becomes a passion and almost a job today. as I was experimented in programming and as Blender supports Python, I begun to learn this language, first to answer some special animation needs.

I looked several time these last years for an open source 3d tool able to build a big city according to some global properties. Ideally I was looking for a good Blender script since it's the 3d suite I'm the more experimented in. unfortunately I didn't find anything that matches my needs.
then Arnaud Couturier (Piiichan) published Suicidator City Engine 0.1 (sce)[27]*, that was able to produce realistic background cities very quickly.*

I had begun to modify his script in an informal way, adding some features here and here. This work started as a simple hack to answer my own need for another project. The first aim was to add streets and roads in it, the other first aim what to learn more about python…
As Arnaud had no time to develop it further during the year, and as the code and my thoughts go on alone, it becomes little by little a real personal project, and after almost one year, it became blended cities.

My motivations for this script are very various: create cities for games, architectural rendering or animation, traffic simulation… The main idea is to allow an artist/game developer/architect to focus on his main project, by designing relatively quickly a world around it that can answer most of his needs."

> Blended Cities:

>
> When you started developing this script, where did you start?

"As I said it start as a sce modification (fork ?) : I added functions to produce streets and sidewalks starting from a simple image. But there was major limitations with images: it was hard to interpret curved streets from pixel curve, and limited to interpret road widths (and especially hard to interpret width of curved streets..). Also the generated buildings were always aligned to XY world coordinates and shaped rectangular.
So I begun to turn everything as vectors: I worked on a different building lot algorithm, and though the streets as a network, made of intersections and lines.
I also started very soon to design a modular concept, that would allow people to add their own building to the generator, or to create a new data source, or to append a third party code that can use the inner variables for specific need. I think modularity is primordial for that kind of open source big project in order it can live by its own."

> Any particular Inspirations?

"Unfortunately I didn't have the occasion to test another city generator except sim city 3000 ;) and soon cityXL, whom game renders looks nice and realistic.
I like the intuitive, 3d way the objects are appended and edited in cityscape as shown in their video on the website. as far as I know the most impressive and complete city generator application I know is the Pascal Mueller city engine (the Procedural City Engine) especially, among numerous other thing, for his street and building algorithmic generation and the procedural language it uses."

> Where did you have the most complicated problems?

"I think it was the way to build irregular crossings made of roads with different widths, and the fight is still not over.
the building lot builder gives me also some severe headaches."

> What is the structure of the script?
> eg) How are things modularised, data structures etc?

"Data structures tend to become a coherent system, in order to meet the requirements for data exchanges. Python dictionaries and XML will be the ways.
as the project started as a fork of a little script, data structure was not an identified need, but is now mandatory.

The code is split for different reasons:

the same core code should be used in both blender 2.4 and 2.5. This to permit a smooth transition between them and maintaining them accordingly, as to my point of view blender 2.5 stable and complete versions won't appear before September and 2.49b will still be used a lot next year. as the blender function calls and the gui scheme differ I created to directories (/bin/24 and /bin/25) to group any blender python specific functions.

To allow users to add their own city objects, procedural building codes and premade meshes are apart from the core code, grouped by author or themes, and called through a unified library system.

A lot of procedural mesh creation function has been written is this script, and also a lot of common, multi-purpose functions, both can be reused in some other script; these functions are apart, in two different files.

The script is translatable, so there is different language files apart from the core.

I also would like to split the lot builder algorithm and the sources parsers and interpreters from the core, in order one can use other ones easily:
ideally the core script would be an empty shell, only responsible for the gui's in/out, the city data structure file operation, and creation process management, and would ask building algorithms, parsers, parametric objects to different modules: call this parser and interpret that street data source, retrieve streets data, build streets using this library, etc...
I think it is a good way to ensure the script evolution on a long-term perspective."

> What would you like the script be in the future?

"A real community project, with a huge building and urban object library, able to build any kind of cities on undulating terrain, with bridges, railways and tunnels."

> Would you like it to be ported to Blender 2.5?

"Not only ported but taking advantages of the new 2.5 features. I think that 3d city edition as in cityscape and micro edition would be easier to develop in 2.5, as the python script integration is 'by default' in 2.5, on the whole gui."

> If so, will the script need to be rewritten as to how it works?

"I hope that the modular concept will help the 2.5 port."

> Do you think City Zoning helps users?

"Absolutely, without zoning, building choice decision is based on the building lot properties: area size and height of the lot determine which kind of building to use. These lots attributes don't depend on their geographic position in town but essentially on the building elevation map, the block sizes, and the main lot properties defined by the user, so a kind of zoning can be made by playing with the building elevation map, but it's not the better way since elevation and zoning are two different city properties. for example one can have the flat and long shop of a car reseller and the 8 floors building of a big shop that both should grown in the 'commercial' areas of the city, but can't with the non-zoning way.
in order the zoning integration to be completed, the different zone must influence the building lot attributes : in other words, the used building library should be known before to create the lot. Nevertheless, using parametric/procedural buildings or flexible enough premade buildings it does the job already."

Blended Cities Structure

Bin:

24 - Blender 2.4x Gui

25 - Blender 2.5x Gui

img - GUI images

bc_common.py - Common functions

bc_fileops.py - File system functions

bc_primitives.py - Blended cities mesh functions

preset_IN.py - load presets

preset_OUT – save preset

blended_cities_0433.py - Main python script

Hook:

Mac/Linux

Windows

Libraries:

Author Name:

name.blend - Contains mesh data for library

name.xml - Contains xml info on how to construct library

Locale:

Language files

Plugins:

External plugins

Presets:

User city presets

Tmp:

Images used for generating city

Through looking at this structure of the script it was clear that the libraries and the main script would need to be modified.

3.5 Development

This section looks at the different iterations that this project went through to form a modified script with city zoning.

3.5.1 Version 0.1

Design

The first version zoned buildings with regards to how close they were to the origin. Buildings closer to the origin were taller than those further away. This version was to test how much influence I had over the properties of buildings through modification of the libraries.

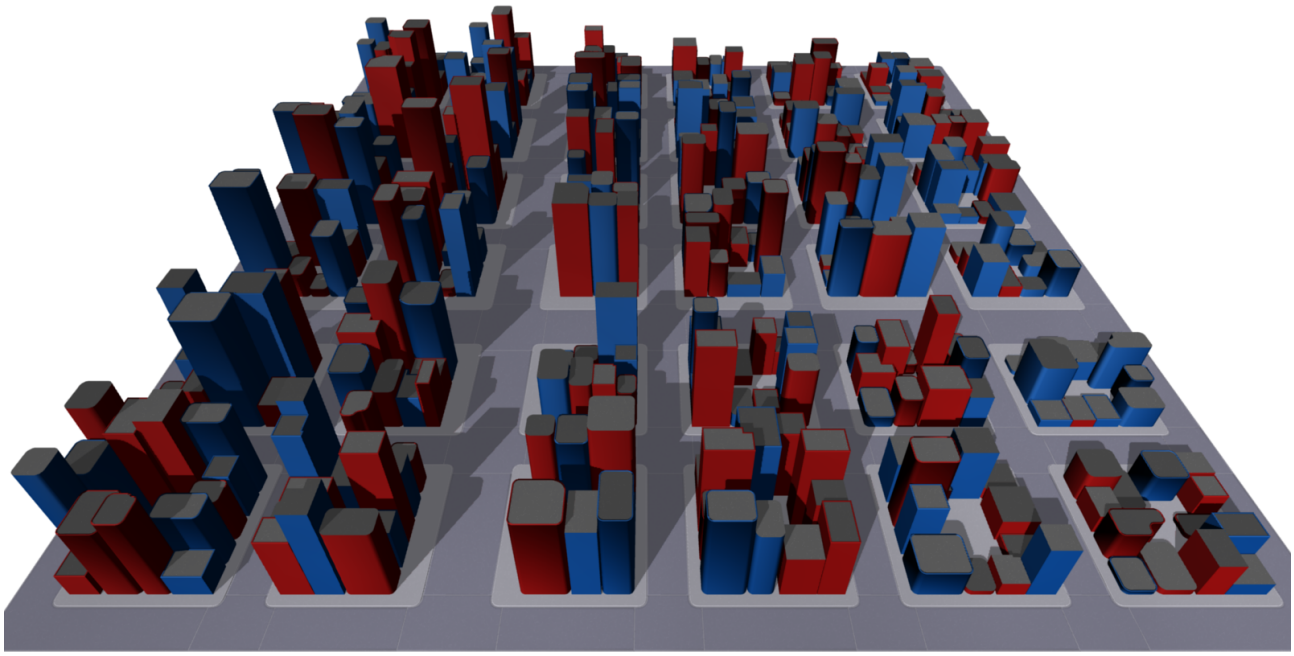


Figure 14: Perspective view of procedural city made with version 0.1

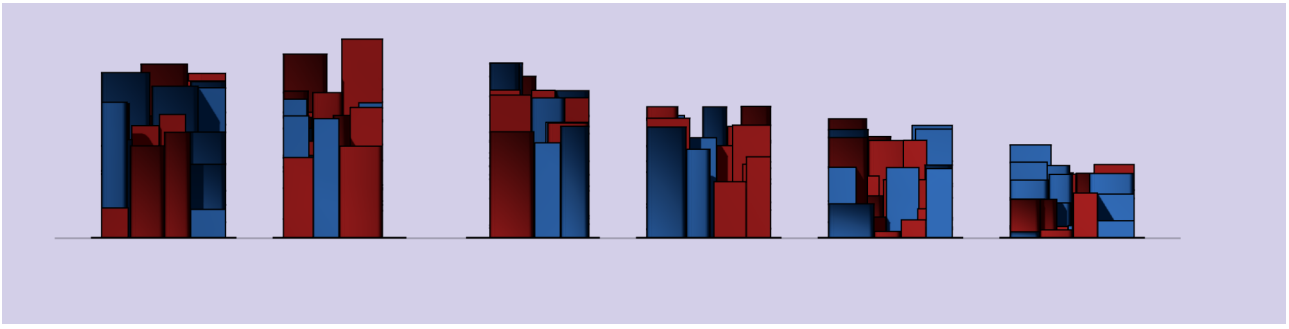


Figure 15: Orthographic view of procedural city made with version 0.1

The properties in the library can override all the properties changed in the GUI. This means that zoning could be implemented on a per library basis as to how the artist wants. This is more difficult though as an artist would need to know the language python to add any complicated zoning through the modification of these libraries.

It also means that more complicated zoning with multiple libraries is not possible. However it did show that additional variables in the library could aid the main script to zone.

Functional / Non-functional Requirements

None of the requirements were fulfilled with this version.

Code

Lines 67-75 of /Code/version0.1/blended cities/libraries/simple cube/ building.lib

Summary

The problems with this version was that the code for zoning this way was hard coded in a library and would have had to change with each library. It would also have only worked for buildings that were procedurally created, as premade buildings would not have had a height parameter to change.

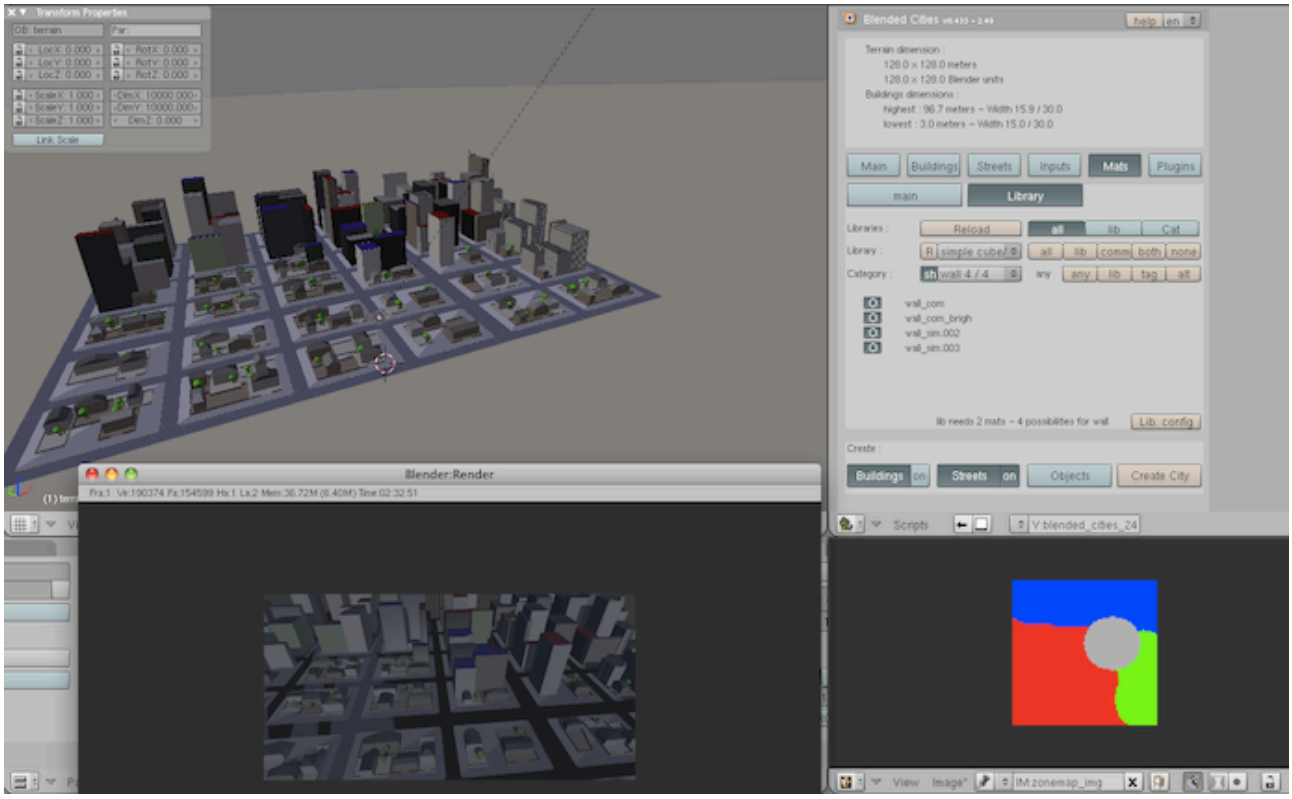
It is too simplistic and does not allow the user to have much control over the city. There is no sense of realism either.

However it was good to see the limitations of adding zoning through modification of the building libraries.

3.5.2 Version 1.0

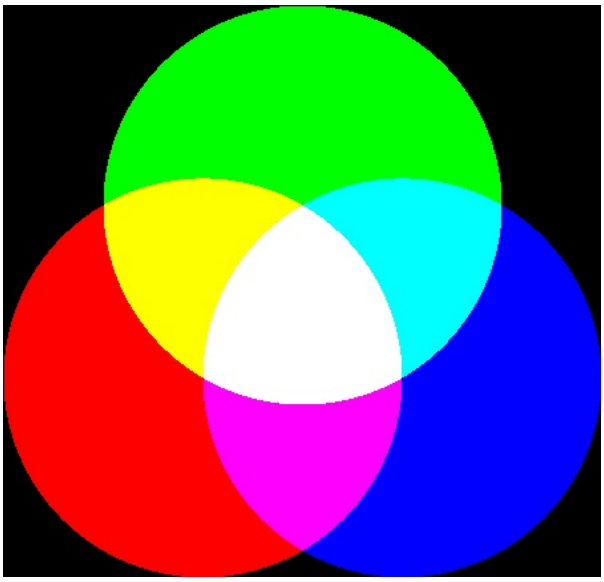
This version uses images for zoning as other systems have suggested [11]. An image is often easier to understand than using multiple sliders. By changing the resolution of the image, the user can change the amount of control they have over the city with regards to zoning.

Figure 16: Version 1.0 in action



Here the user paints an image inside Blender as to where the zones are in the city.

Figure 17: RGB Colour chart showing how colours mix



There are 3 zones:

- Residential
- Commercial
- Industrial

A zone can hold one or more building libraries.

Each colour represents a zone:

- Red – Residential
- Green – Commercial

- Blue – Industrial

When the image is analysed, the value of each colour is taken and stored.

The value of each colour channel determines the probability of a zone to be used. This allows zones to be mixed. For example the colour Blue has the value (0.0,0.0,1.0), this means that the probability for the industrial zone is 100%.

However if the colour was white, all three zones would have an equal probability of occurrence and so any zone could be chosen.

Colours and images were chosen to be used in this way because of how they interact with each other. It is one simple familiar way of allowing an artist to 'paint' the zones of the city.

This version was posted on a popular Blender site as part of the evaluation. Because of this, a video tutorial was made to show the user how to use this addition. [][]

Functional / Non-functional Requirements

Functional requirements for city zoning FR1.2, FR1.3, FR1.4, FR2.1.1, FR2.2.1, FR2.2.1.1, and FR3.2 were achieved in this version.

Non-functional requirement NFR 2.2.1 was achieved.

Code

Lines 32 -112, 3893 – 3983 of /Code/version1.0/bin/blended_cities_0433.py

Created library files for buildings by user jrs100000 in /Code/blended cities 0.5a/libraries/jrs100000

Summary

In this version, zoning worked really well and so this version was released to the community as an alpha to try out. Colour was a really good way of allowing artists to have greater control over the city.

3.5.3 Version 1.1

This version was designed in collaboration but implemented by Jérôme Mahieux as the code base needed a major upgrade to turn it to using python dictionaries rather than lists. The GUI for city zoning was also written in this version. However at the time of submission this version has not had public release due to the number of bugs present in the version and so testing was done with the previous version.

If one looks at similar line numbers to that of version 1.0 they will see the modifications that Jérôme Mahieux has made to the code to make it more efficient.

Design

To give the user more control we thought that perhaps we could take the brightness of a colour into account. Where the colour value was the probability for a zone before, now it also decides what kind of library to use from within a zone.

The argument for this is that the brightness can stand for the population density of a location.

This is because it is not realistic to have large residential blocks next to a few semi-detached houses. More often cities have districts with a type of building dominating the area and a few outliers.

An example of how this could be used would be how commercial zones have shops of different sizes. In a residential zone, it is more likely to have a corner shop or a few grocery shops situated on the main road than a large shopping mall. In these circumstances, a dark green could represent a shopping mall and a light green a corner shop.

For the colour 'black' we thought that we could add an extra zone; countryside. Whilst not an urban zone, this would represent an area without any buildings for the script to add vegetation.

GUI

In this script, the GUI was updated to be more logical than previous versions with the Inputs tab moved to 2nd place in the navigation structure. We felt that this made more sense and would help a new user navigate the different options present in the script.

GUI for zoning has been added so that the user can turn it on and off [image 1] or choose the colour to be used for a particular category [image 2].

Functional / Non-functional Requirements

Functional requirements for city zoning FR1.1, FR1.2, FR1.3, FR1.4, FR3.1, FR3.2 and FR3.3 were achieved in this version.

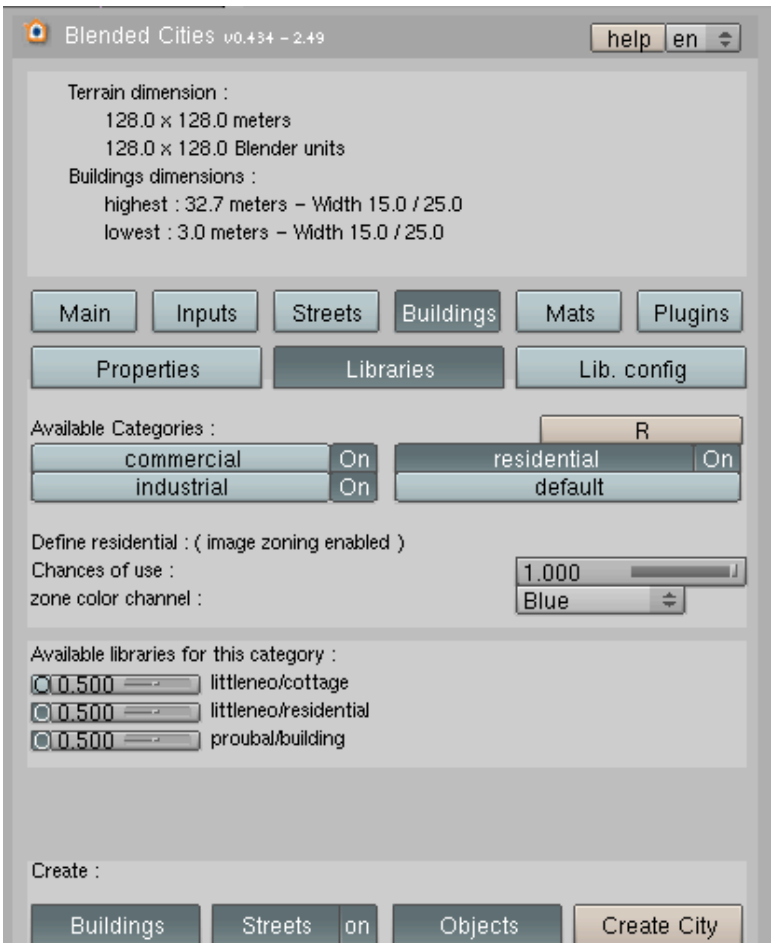
Non-functional requirement NFR 1.1 and NFR 2.2.1 were achieved.

Summary

Documentation was not done on this version as it has not yet been publicly released due to the number of bugs present n the script. The GUI in this version now allows users to change settings for zoning whereas in the previous version, zoning was always on.

Figure 18: GUI for version 1.1, Inputs tab

Figure 19: GUI for version 1.1, Libraries tab



4 Testing & Results

Because version 1.0 was evaluated by the community and version 1.1 is unstable it was decided that version 1.0 should be the version to be tested. Note that this version is incomplete only due to the lack of its GUI, it performs zoning as intended although upgrades to zoning will be implemented at later stages.

4.1 System analysis

4.1.1 Environment testing

It was important to test the different operating systems in which blender can be used, so as to check that the modification to the existing script did not break the original script.

	Linux	Windows	Mac
Operating System	Fedora 8	Windows 7	Snow Leopard
Processor	Intel(R) Core(TM)2 CPU 6300 @ 1.86GHz	Intel Atom N270	2.4 GHz Intel Core 2 Duo
Memory	512MB	1GB	2GB
Hard Disk	30GB	160GB	160GB
Graphics	Nvidia GeForce 6400	Intel GMA 950	GeForce 8600M GT
Form	Desktop	Netbook	Mac Book Pro

4.1.2 System Usage

Testing was using a Mac Book Pro with the same specification as the one shown in the table above. The system application 'Activity Monitor' was used to check both the memory and processor Usage. Other than the Terminal and Blender, no other applications other than the needed system applications were currently running whilst testing. The laptop was plugged into the mains.

The base scene can be found on the attached disk, the city preset James was used, all other settings were default.

4.1.2.1 Memory Usage

City generation is a memory intensive operation and so it was important to measure how much was needed.

4.1.2.2 Processor Usage

City generation requires a large number of calculations to be made when generating the city. The calculations made can be seen when looking at the console traces on the disk.

4.2 Black box testing

The modified script was exposed to a number of different scenarios in how the user decided to zone their city with colour. 10 diverse tests were chosen:

1. All
    a. Black
    b. Blue
    c. Green
    d. Red
    e. White
2. Mixes
    a. 1
    b. 2
    c. 3
    d. 4
    e. 5

Tests in 1 include a zone map of one colour only; tests in 2 include a zone map with many different colours.

When testing, 3 images were saved to show what the city looked like. The 3 images below represent the 3 different views from which the city can be displayed for evaluation:

Figure 20: Test scene, 1st view

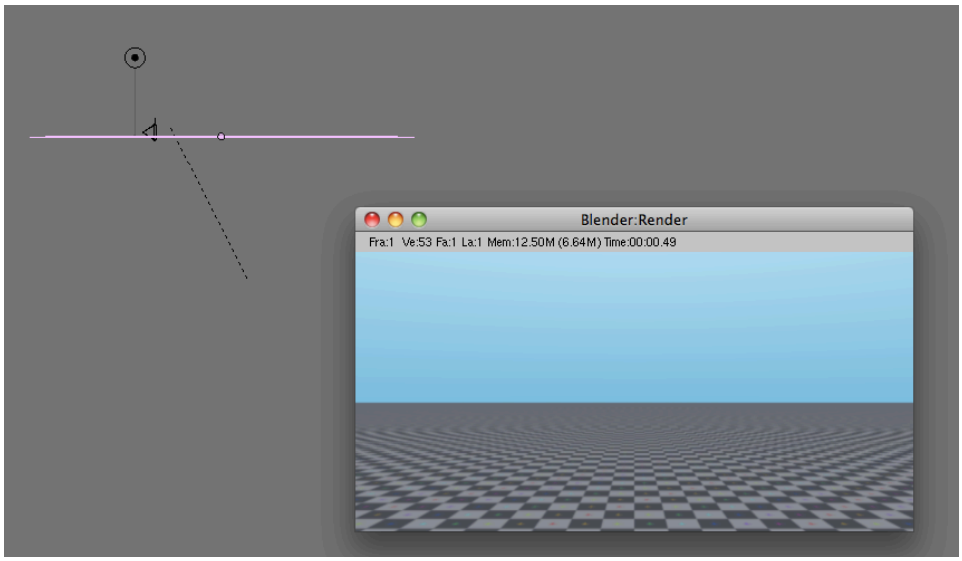


Figure 21: Test scene 2nd view

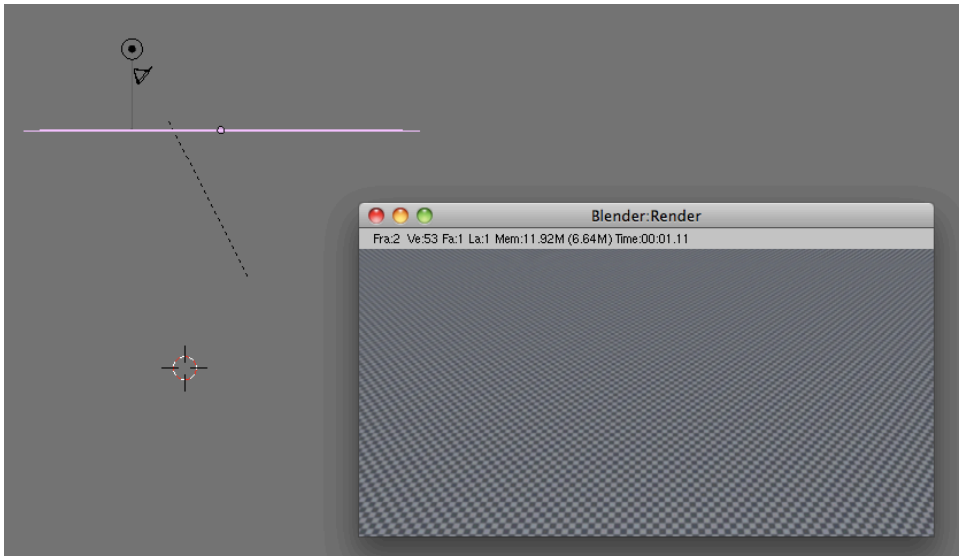
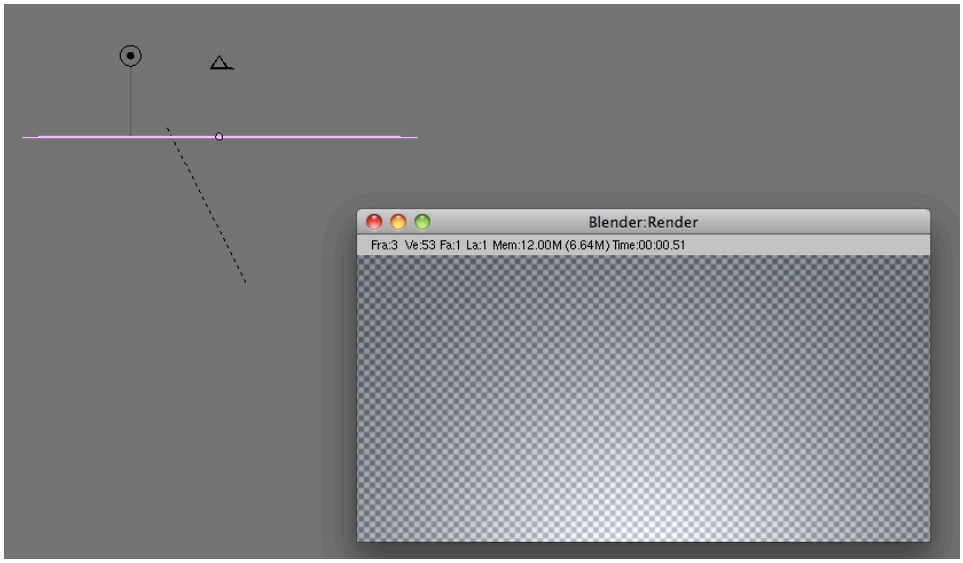


Figure 22: Test scene 3rd view



A zoning map is included for tests where there a mix of colours. A console trace of the terminal window which records the functions used when generating the system has been placed on the disk with this dissertation as these files are substantial in length.

4.3 Results

4.3.1 Environmental Testing

The modified script worked on all 3 main operating systems with no problems. This was also seen with no problems in the community when it was evaluated.

4.3.2 System Testing

Figure 23: System usage when idle

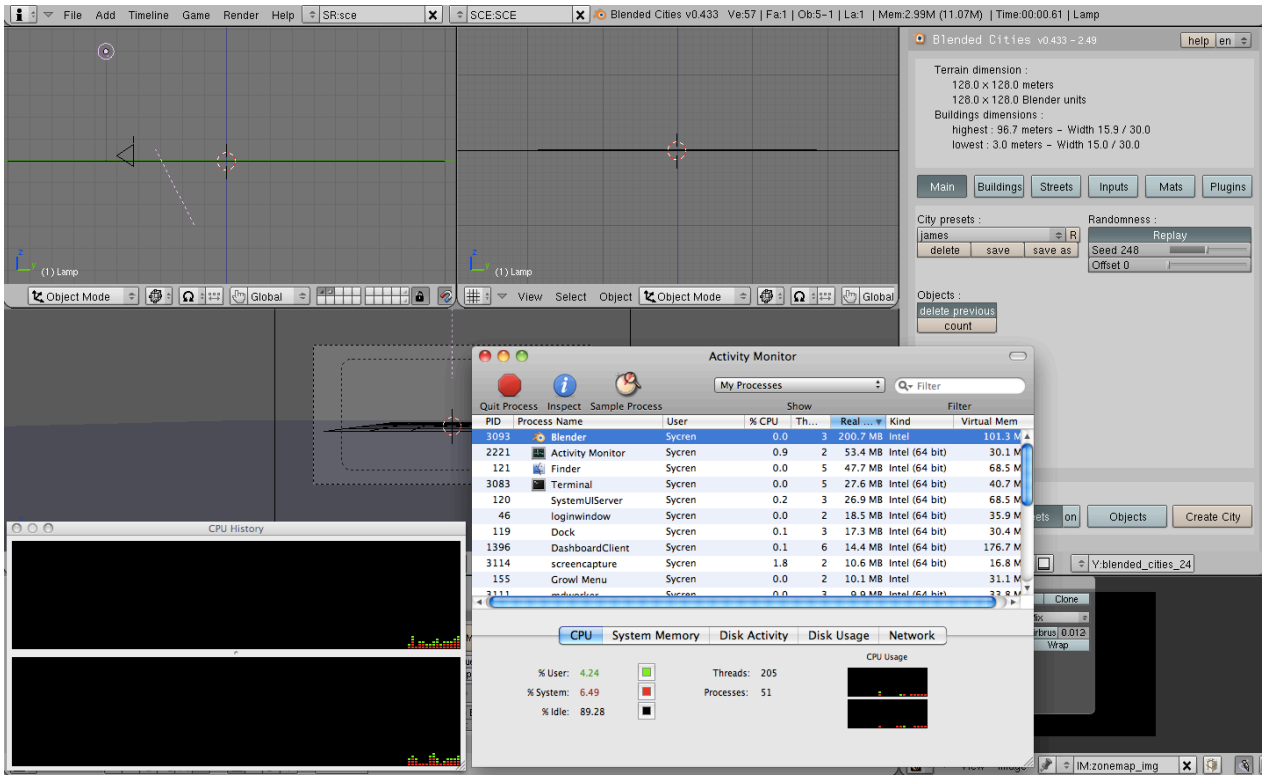


Figure x shows Blender loaded up before the city is being generated. Processor is at 0.0% for Blender with memory usage of 200.7MB. Figure y shows memory usage rise to 279.5MB with 1.34GB of virtual memory. Figure z shows processor at 94.1% and memory at 185.6%.

The data for memory seems skewed with figure 25, with lower memory than figure 23. This could be because it is hard to measure memory with generation as the memory required depends on the calculations being made and the storing of variables. It could be possible that figure 25 was also near the end of the city generation process and this could have had an effect on the outcome of the data.

Overall I think that this data does show that the script requires a lot of processing power to perform all the calculations needed.

Figure 24: Memory usage when running

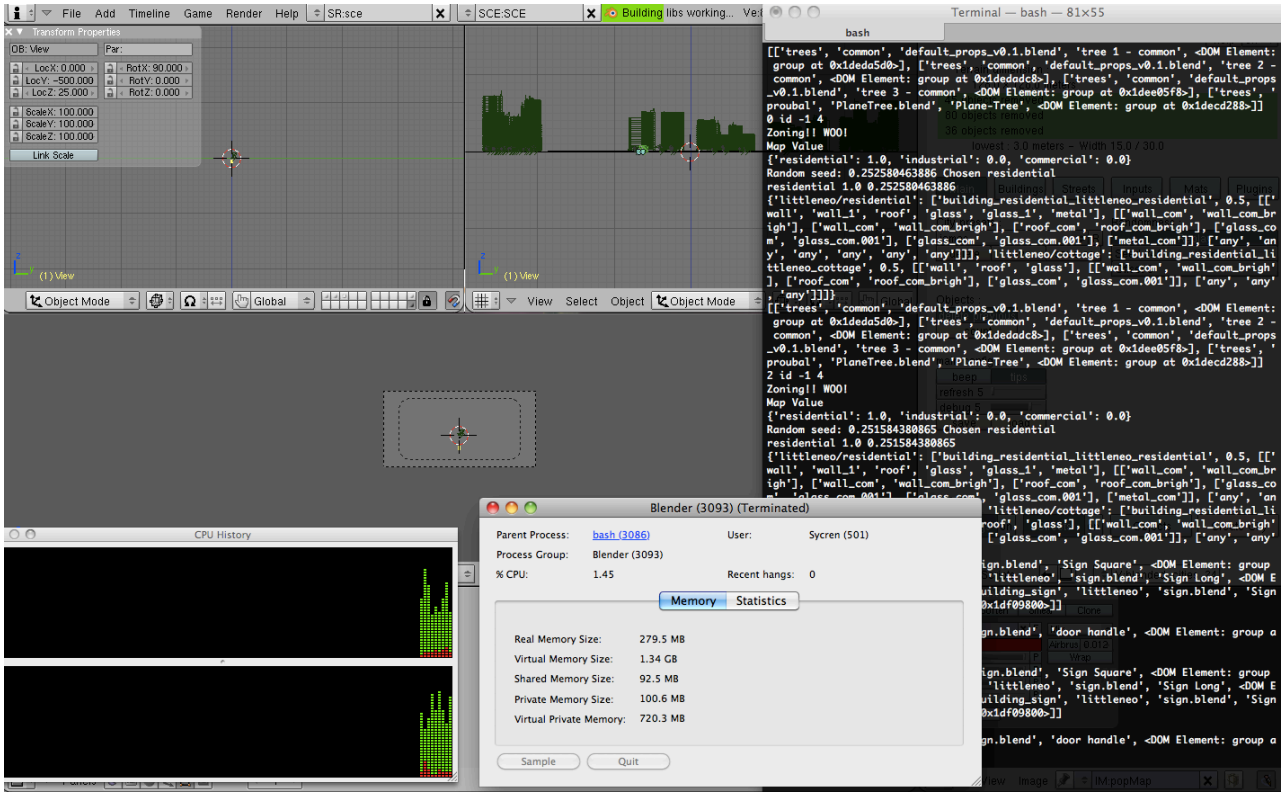
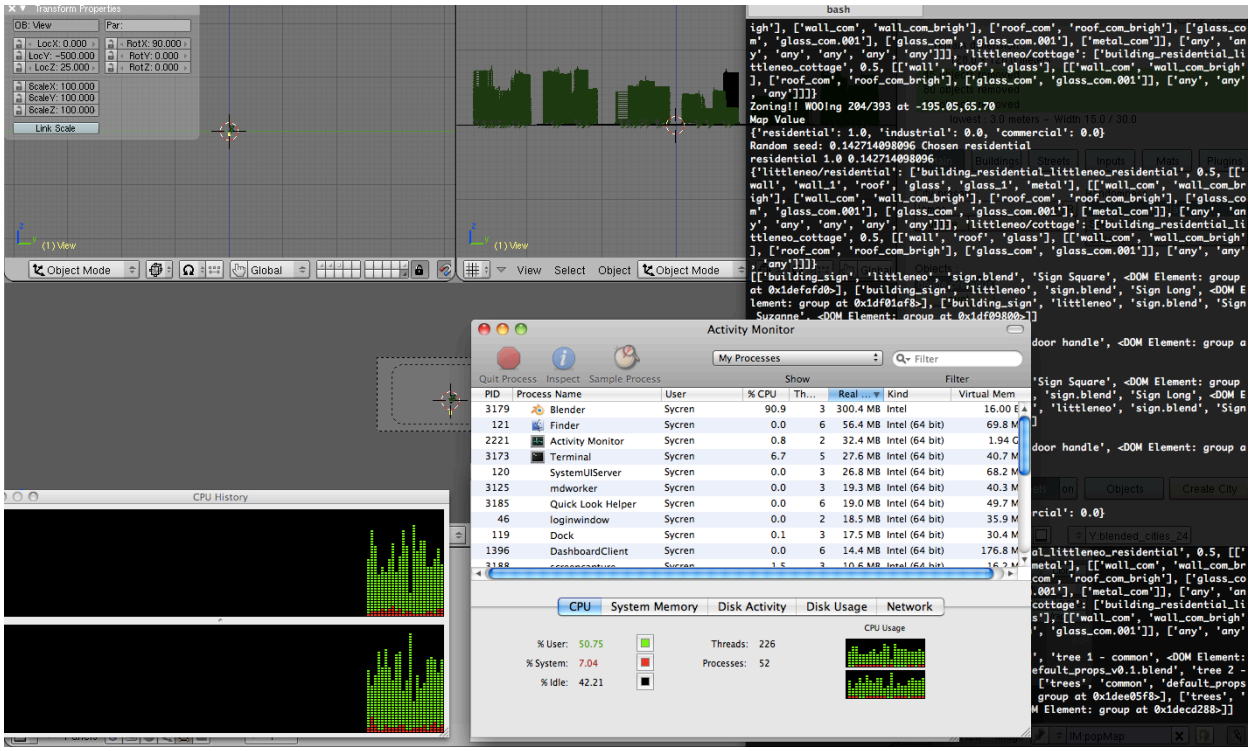


Figure 25: System usage when running



4.3.3 Black Box Testing

All tests performed as expected as can be seen from the test data (images and console traces) provided in the appendix and on the disk provided.

4.3.4 Conclusion

All the tests worked as expected.

5 Evaluation

5.1 User Testing

In order to test the script, it was first important to create some more building libraries to provide some more variety. The Blender community was asked if they could send some models of buildings. Many were happy to oblige and these models were packaged for inclusion in the script.

It was important to make sure that the buildings offered were under an attribution licence so that a user could use them under a commercial project without problems as long as they attributed the creation of the buildings to the relevant author.

When version 1.0 was working and ready for testing, a video tutorial was posted on Vimeo to show users how to use the script. Also a high quality video was also linked to through the forums for those with bad connections unable to use Vimeo.

The script was publicised through being published on Blendernation.

"BlenderNation is the central news site by and for the Blender community. It is a prominent source for Blender news, tutorials, reviews and art, and is visited by over 8,000 daily 3D enthusiasts and professionals." [28]

Within 2 hours of being published on blendernation [29], there were over 500 views of the tutorial on Vimeo [30] and 200 downloads of the high quality video. From the time of submission of this document there have been over 700 views of the video tutorial on Vimeo and over 300 downloads of the high quality version. This article was also linked to a survey where blender users would respond to the following questions:

- What they thought of the script
- How it could be used
- What they thought of the city zoning addition
- Improvements needed for the script
- Any problems
- Should it be ported to next version of Blender (2.5)
- Opinions on Procedural generation tools

Due to several answers to this survey being particularly large, the survey results can be found in the supplementary data on the disk.

The feedback from the survey has been almost entirely positive with comments such as:

When asked about the city zoning addition:

Michel Vilain: *"Seems like a natural addition for larger cities. (my first thoughts went to SimCity where zoning was very important)"*

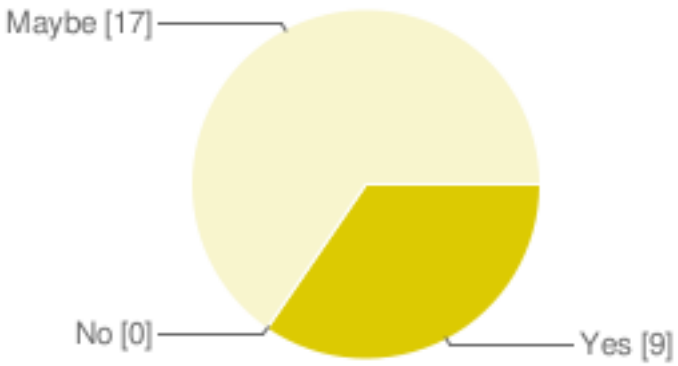
Woodman5k: *"Great addition, this can be used in concert with a land use polygon map very well."*

Anders Otte: *"A good way of quickly adding a decent level of "detail"."*

Tiago Allen: *"Perfect! Is a great way to make the script more suitable"*

When users were asked if they would use it:

Figure 26: Users who would use it for future projects



9 people said they would use it in the future and 17 said that perhaps they would use it meaning that it is a great addition to a current tool.

When users were asked if they thought it should be ported to Blender 2.5:

Figure 27: Users who think that it should be ported to Blender 2.5

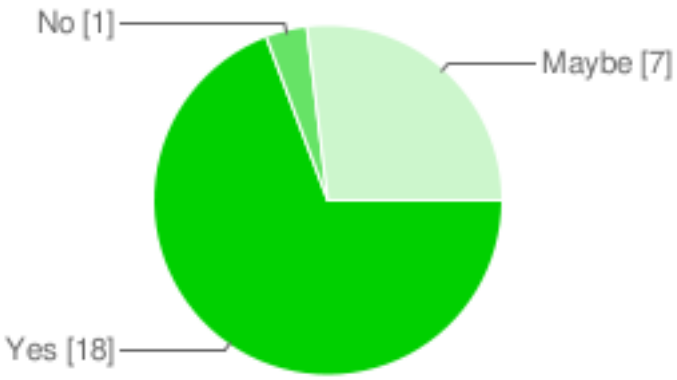
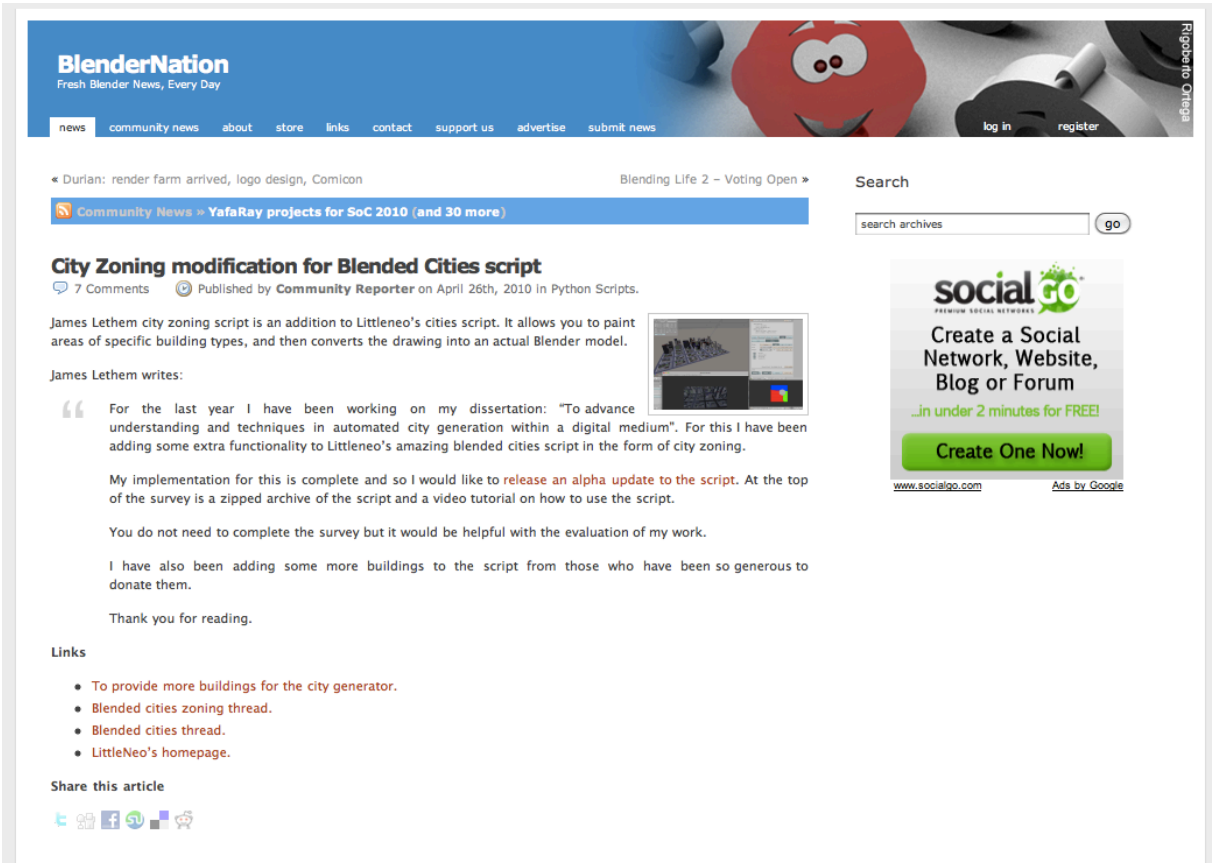


Figure 27 shows that people liked the script and would like it to be ported to the most recent version of Blender (2.5).

Figure 28: City Zoning article in Blendernation



5.2 Fulfilment of objectives

The project aim was split into 3 objectives:

- *"Investigate how existing tools and techniques are used for city generation"*

This objective was fulfilled through the research in section 2 through researching games, films and other projects as case studies. Interviews and correspondence with experts in the industry also competed the task.

- *"Investigate the structure of an existing city generation tool"*

This objective was fulfilled through the correspondence made with Jérôme Mahieux, the Blended Cities script author, research was also done by viewing the script and testing the different functions.

- *"Design, Implement and Document a city zoning algorithm with the Blended Cities script"*

This objective was fulfilled with version 1.0 from the methodology. Future releases between Jérôme Mahieux and myself will have more advanced capabilities. After university I will still work on this project and have taken on the task of documenting future releases for the community.

6 Conclusion

6.1 Personal Development

Throughout this project I have learned how to efficiently research the problem domain. This has been through the use of IRC, email and forum threads to contact the developers and artists directly. I have learnt that project management has been an essential part of this dissertation and soft deadlines must always be used when reaching important milestones. I have realised that developing for the open source community is a rich and rewarding experience that has helped me learn new ideas and paradigms.

There have been four areas of development where I have gained great insight:

1. Python Scripting – By scripting in python on Blended Cities and other scripts I have furthered my knowledge of such a powerful language.
2. Blender – By developing in blender I have furthered my knowledge of how the internals of Blender work along with how the user interacts with the tools developed.
3. Open Source Software – Through modification of Jérôme Mahieux's script, I have learnt how city generation works through contact of a seasoned developer in this field.
4. Problem solving – I have learnt how to better design algorithms to solve the problems found in the problem domain.

6.2 Future Work

Future developments could be done in several key areas:

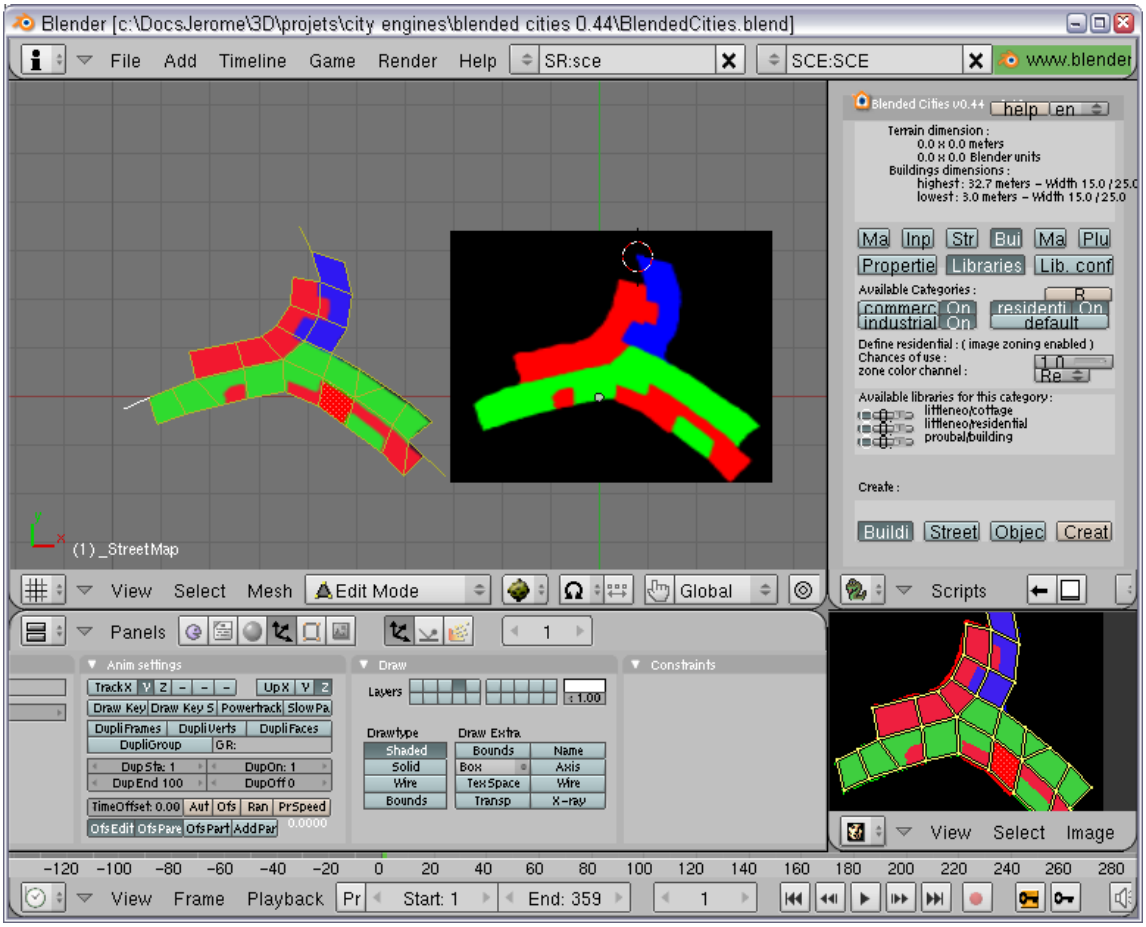
1. Porting to Blender 2.5 – Blender 2.5 is currently in alpha stage and so is quite unstable. However when the final release has been made for this version, porting the script to it will provide several advantages;
    a. The GUI has been revamped to make it easier for the user. It has also been developed to give more control to the developer. By porting we would use less code for it.
    b. The API is more powerful and straight-forward. Every function in Blender 2.5 now allows python to access it directly whereas in Blender 2.49, to access these functions specialised C code had to be compiled with blender.
    c. Python 3.0 – Use of this recent version of the python programming language enables developers a better toolset to develop with with the advantages of a more efficient language.
    d. BMesh [31]– A new system for control over the mesh data is being developed which would allow the developer to more easily make procedural building generators.
2. Integration with existing generators and scripts
    a. Interest has been expressed to merge the realistic L-system vegetation generator into Blended Cities to allow the creation of more realistic environments.

   b. The merging of the A.N.T generator to allow more realistic terrain
3. Additional zoning techniques
   a. Use of boundary containers for showing the different zones
   b. Use of particle systems by showing population density and zone through colour and number of particles.
4. Node based interface – where the user creates nodes and forms webs laying out how they want their city to look. Increases flexibility as linear menus can only hold so much before they become unwieldy to navigate.

6.3 Future Releases

Jérôme Mahieux is currently trying to bring version 1.1 out of the alpha stage. We are both currently working on adding new functionality to the script. Figure 29 shows the work that we are currently doing to allow users to paint the zones on the street map which is easier for the user to understand.

Figure 29: Painting zones on street map



References

[1] gamedeveloper Vol16 No 7 August 2009

[2] 2006. SpeedTree ® RT Essential to Oblivion ' s Environmental Excellence. Star, (803), 29072-29072.

[3]http://www.fitc.ca/events/speakers/speaker.cfm?event=102&speaker_id=12775

[4] http://www.quelsolaar.com/love/gameplay.html

[5] http://news.quelsolaar.com/

[6] http://www.infinity-universe.com/Infinity/index.php?option=com_smf&Itemid=75&topic=7426.0

[7] http://www.planetside.co.uk/content/view/15/27/

[8] http://www.theprodukkt.com/theprodukkt

[9] http://www.youtube.com/watch?v=wqu_IpkOYBg&fmt=22

[10] http://features.cgsociety.org/story_custom.php?story_id=5434&page=1

[11] Parish, Y.I. & Müller, P., 2001. Procedural Modeling of Cities. Population (English Edition), (August), 12-17.

[12] Watson, B. & Carolina, N., 2008. Procedural Urban Modeling in Practice. Star, (June), 18-26.

[13] Brenner, C., 2000. Towards fully automatic generation of city models. Building, XXXIII.

[14]White, C., 1933. King Kong – The Building of 1933 New York City. New York, 1933-1933.

[16] http://www.sc3000.com/knowledge/history_classic.cfm

[17] http://www.humantransit.org/2009/06/did-sim-city-make-us-stupid.html

[18] Lindenhof, W., 2009. Procedural generated design.

[19] http://web.me.com/rchrisw/workSite/bioN/bioN.html

[20] http://www.cgarchitect.com/upclose/article1_CW.asp

[21] http://pixelactive3d.com/Company/Management.php

[22] http://pixelactive3d.com/

[23] http://www.nucleardawnthegame.com/

[24] http://www.blender.org/

[25] http://blenderartists.org/forum/showthread.php?t=168836

[26] http://blenderartists.org/forum/showthread.php?t=169916

[27] http://arnaud.ile.nc/sce/

[28] http://www.blendernation.com

[29] http://www.blendernation.com/city-zoning-modification-for-blended-cities-script/

[30] http://vimeo.com/11231553

[31] http://bmeshblender.wordpress.com/about/

Glossary

IRC – Internet Relay Chat

Middleware - Software that connects other software together

OpenGL – Open Graphics Library

Matte Painting – Digital background used as backdrop

Bibliography

Alexander, B.C., 1966. A city is not a tree. City.

Alsweis, M. & Deussen, O., 2005. Modeling and Visualization of symmetric and asymmetric plant competition. Area.

Brown, S. et al., Accelerating the Scalable City. City.

Burgess, E.W., 2008. The Growth of the City : An Introduction to a Research Project. , 2008-2008.

Carrozzino, M. et al., 2004. Urban procedural modeling for real-time rendering.

City, N.Y., 1996. BY YEAR TABLE OF. Science, 149(1), 6-8.

Fritsch, D. & Kada, M., Visualisation using game engines.

Galin, E. et al., 2010. Procedural Generation of Roads. , 29(2).

Greuter, S. et al., 2003. GRAPHITE 2003 Real-time Procedural Generation of ' Pseudo Infinite ' Cities.

Greuter, S. et al., 2003. Undiscovered Worlds – Towards a Framework for Real-Time Procedural World Generation. Architecture.

Havemann, S., 2005. Generative Mesh Modeling.

Kelly, G. & Mccabe, H., A Survey of Procedural Techniques for City Generation. Techniques.

Kelly, G. & Mccabe, H., Citygen : An Interactive System for Procedural City Generation. Building.

Lechner, T. et al., Procedural City Modeling. Simulation.

Lindenhof, W., 2009. Procedural generated design.

M, P., Procedural Modeling of Buildings. Context.

Middel, A., Procedural 3D Modeling of Cityscapes. Architecture.

Mullane, K., 2007. Procedural Modeling of Cities implemented as a Blender Plugin Project Proposal.

Olsen, J., 2004. Realtime Procedural Terrain Generation. Synthesis.

Sensing, R. et al., 2002. AUTOMATIC GENERATION OF 3D CITY MODELS AND RELATED APPLICATIONS. Archives, XXXIV(Table 1).

Wonka, P., Wimmer, M. & Ribarsky, W., 2002. Instant Architecture. Grammars.

http://www.blender.org/documentation/249PythonDoc/

http://jerome.le.chat.free.fr/index.php/en/city-engine/

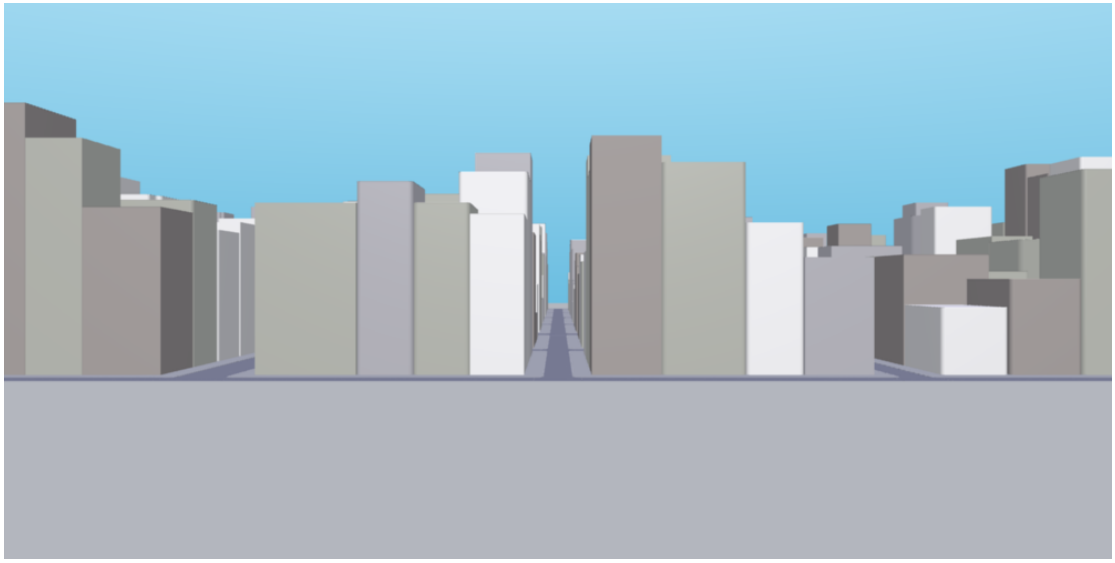
http://diveintopython.org/

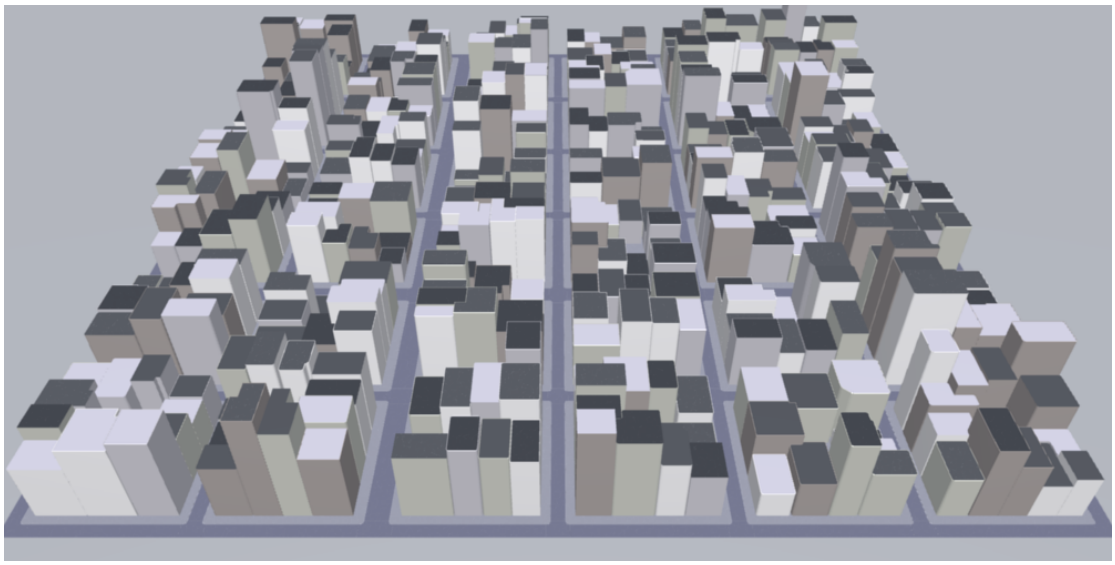
http://wiki.python.org/moin/BeginnersGuide

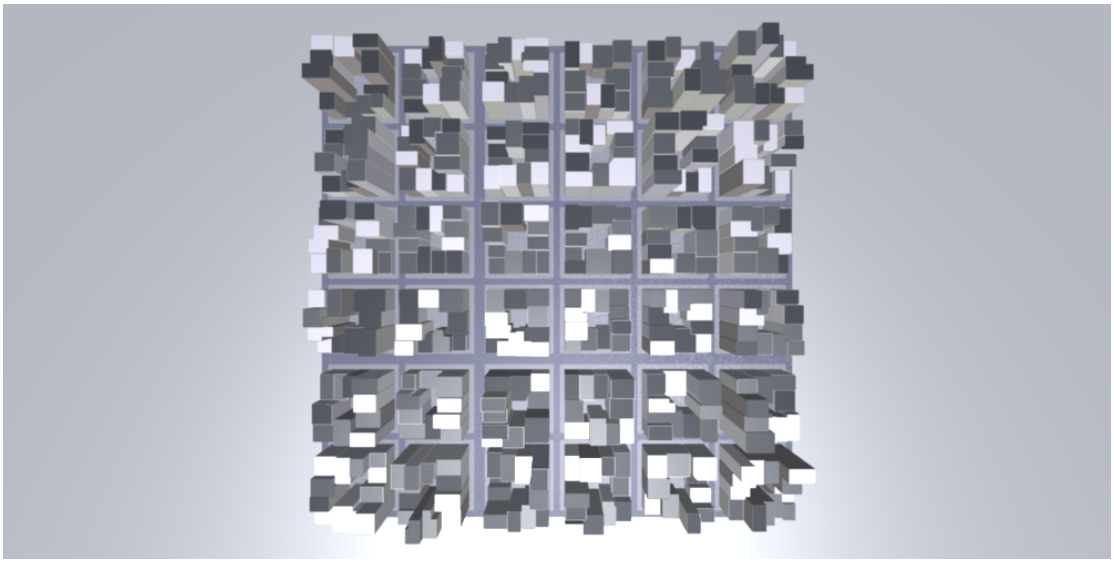
Appendix

Testing

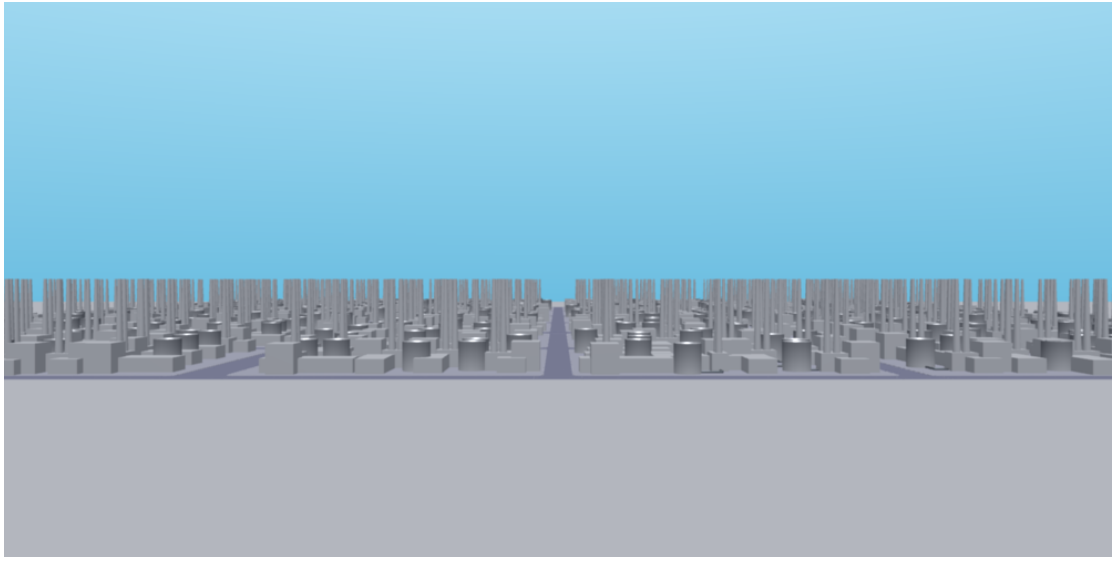
All Black

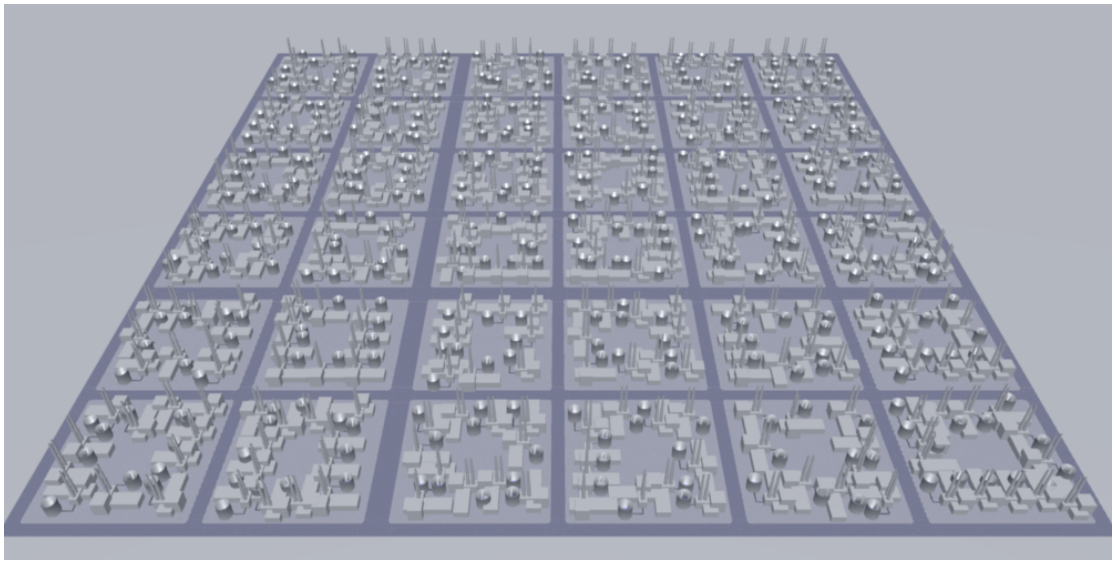


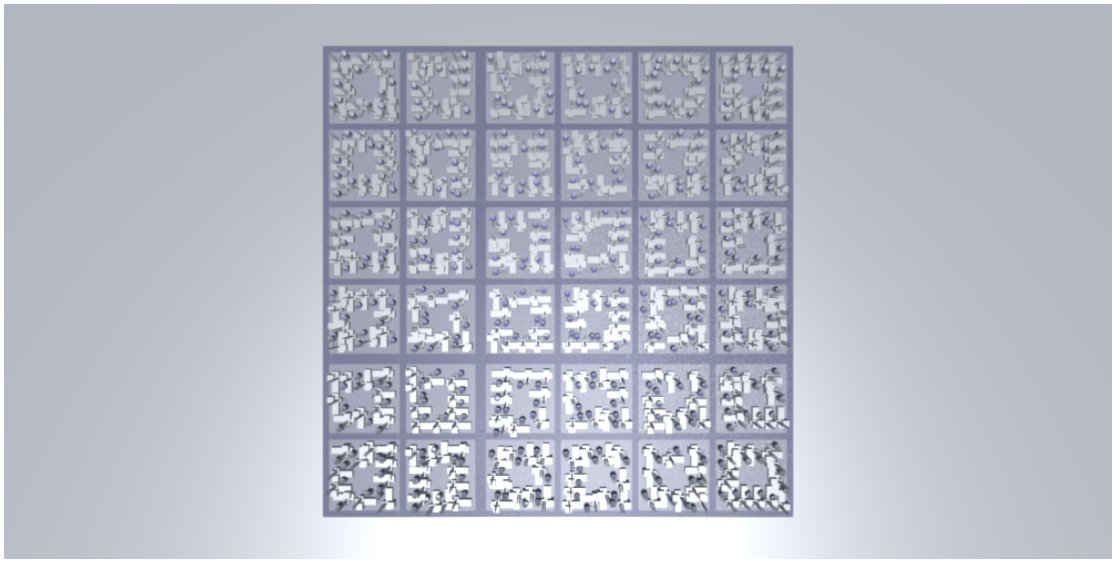




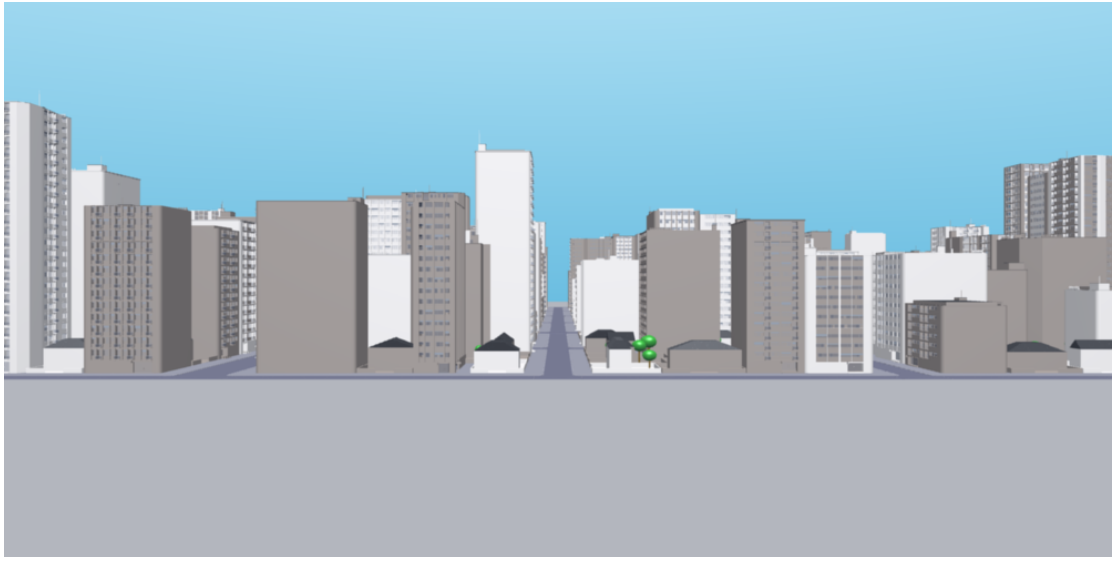
All Blue

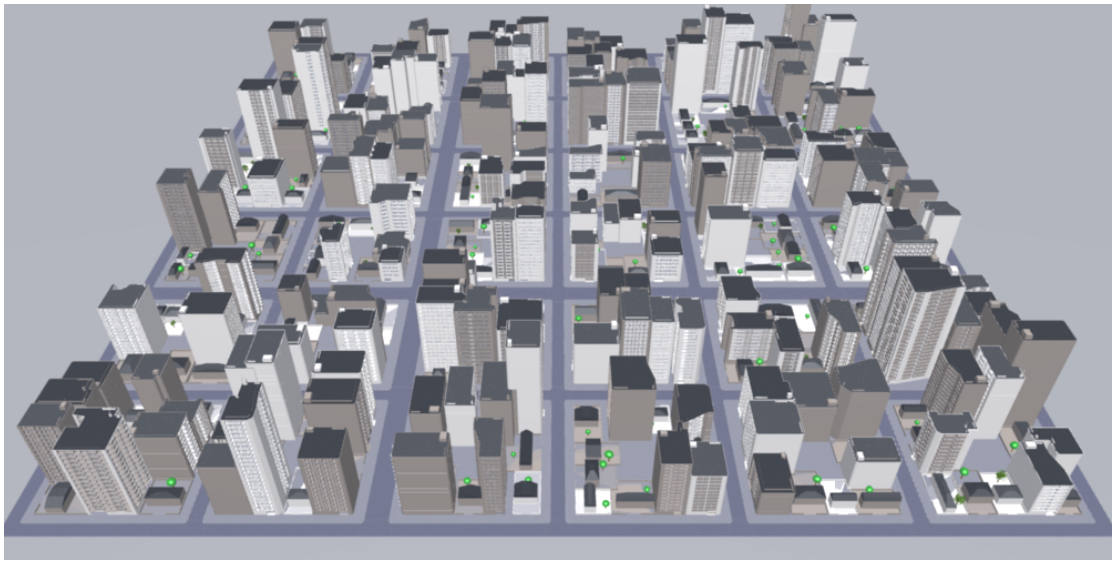


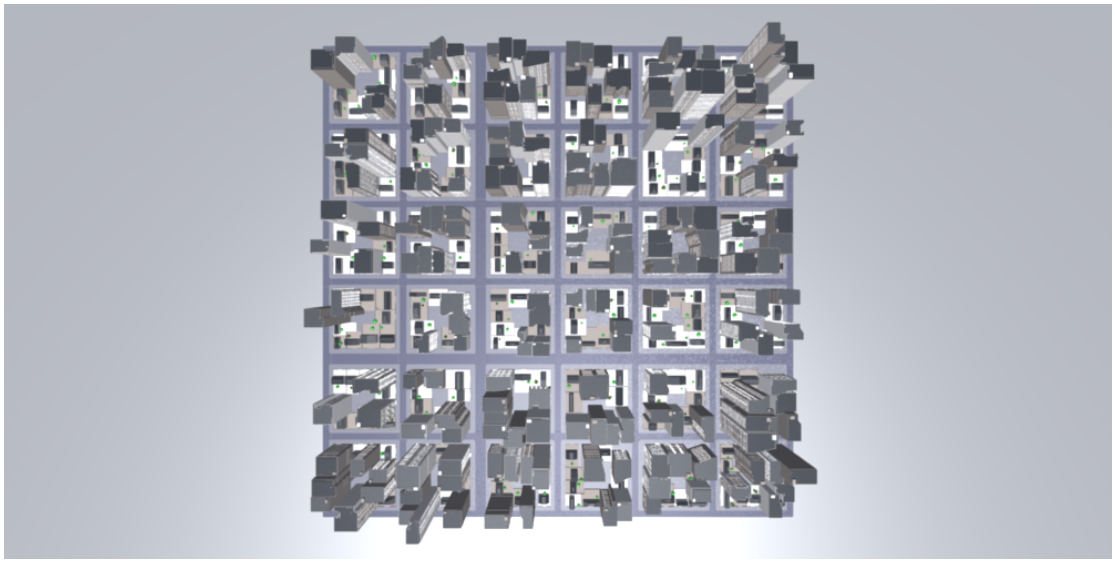




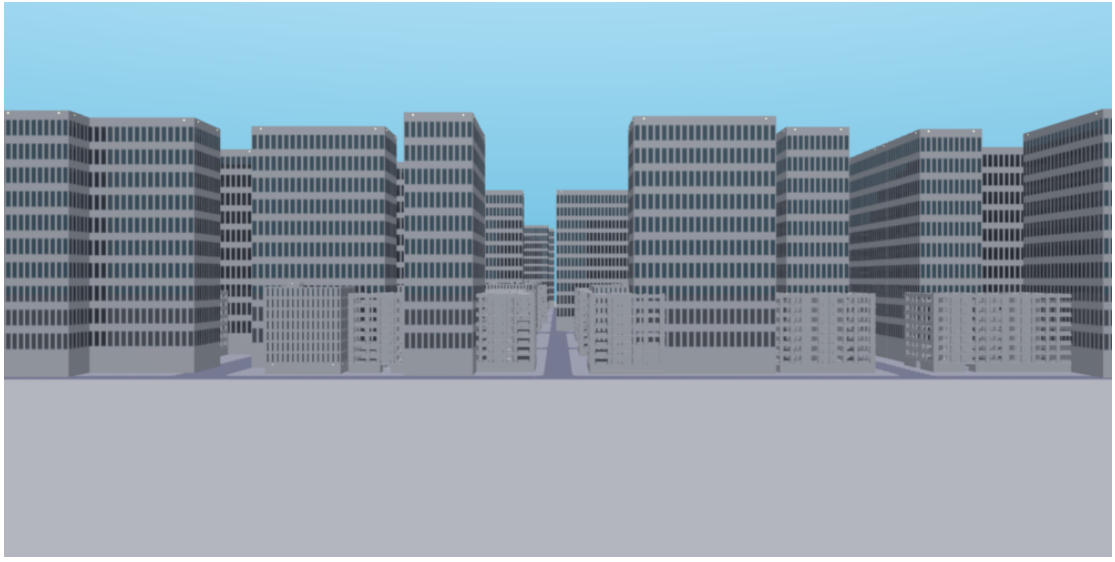
All Red

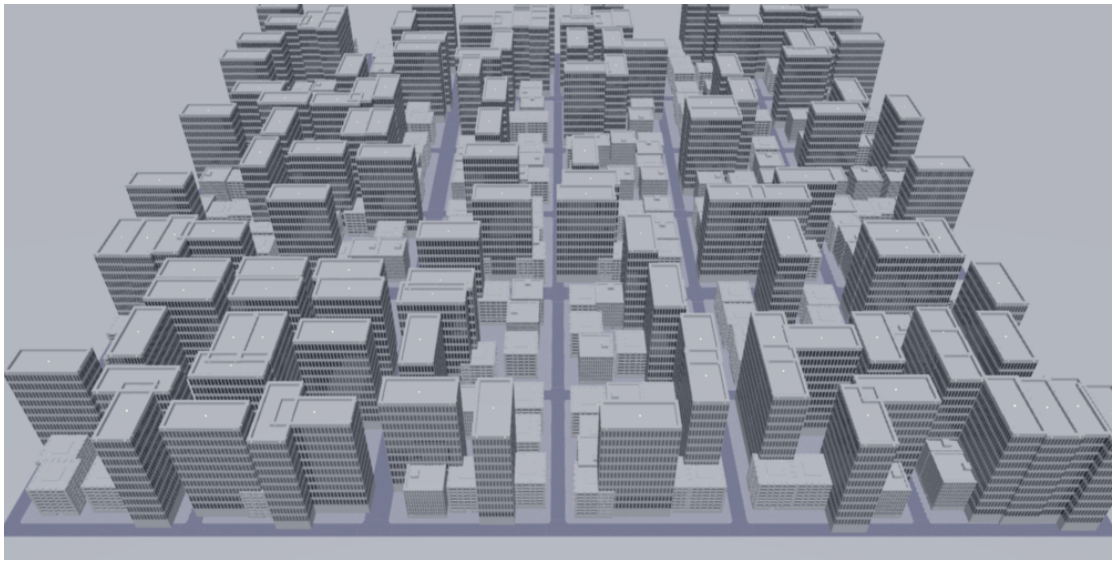


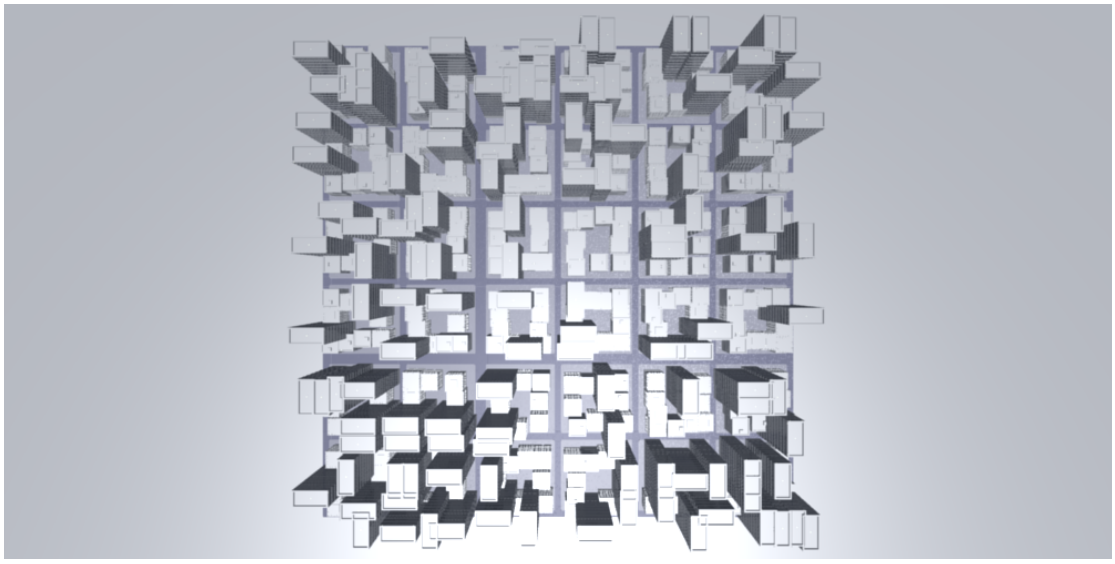




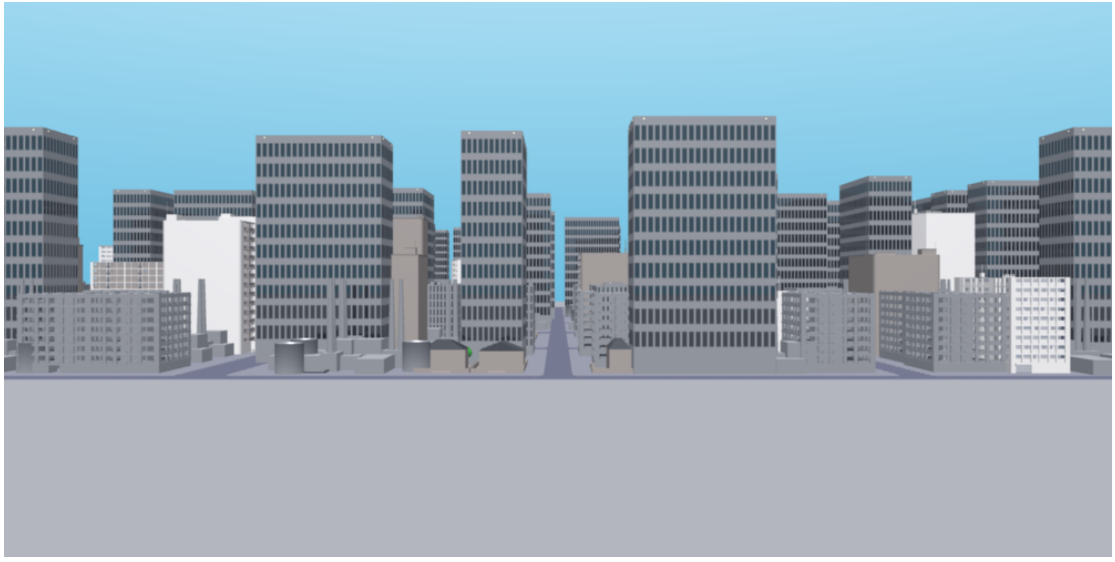
All Green

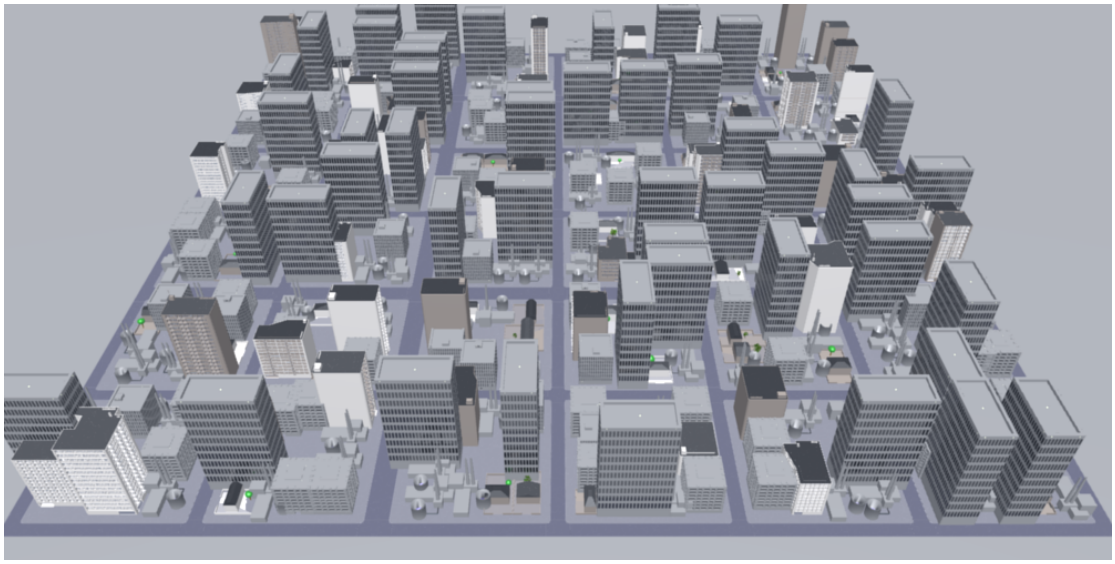


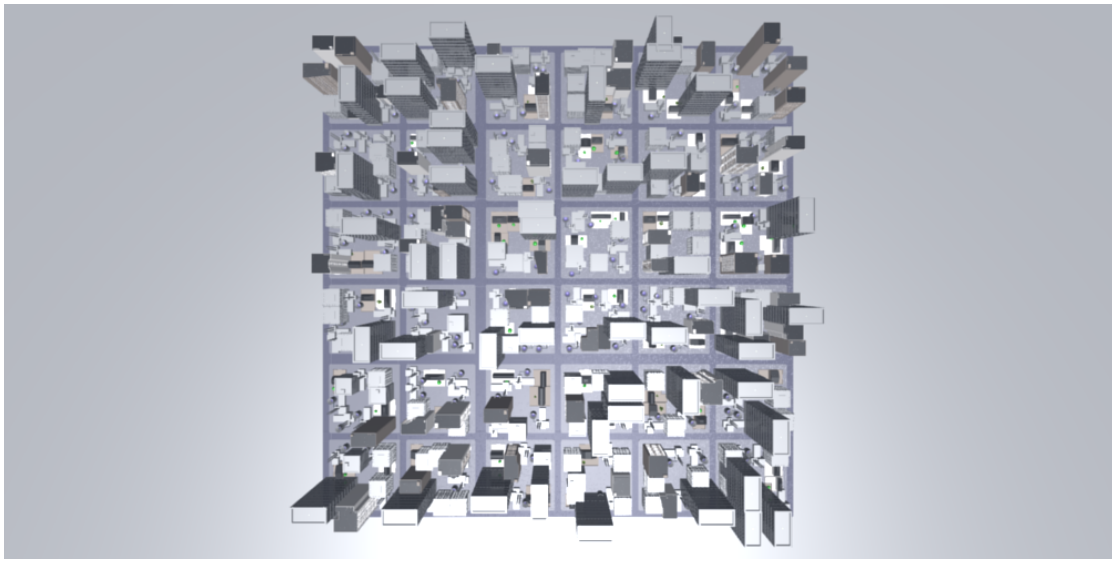




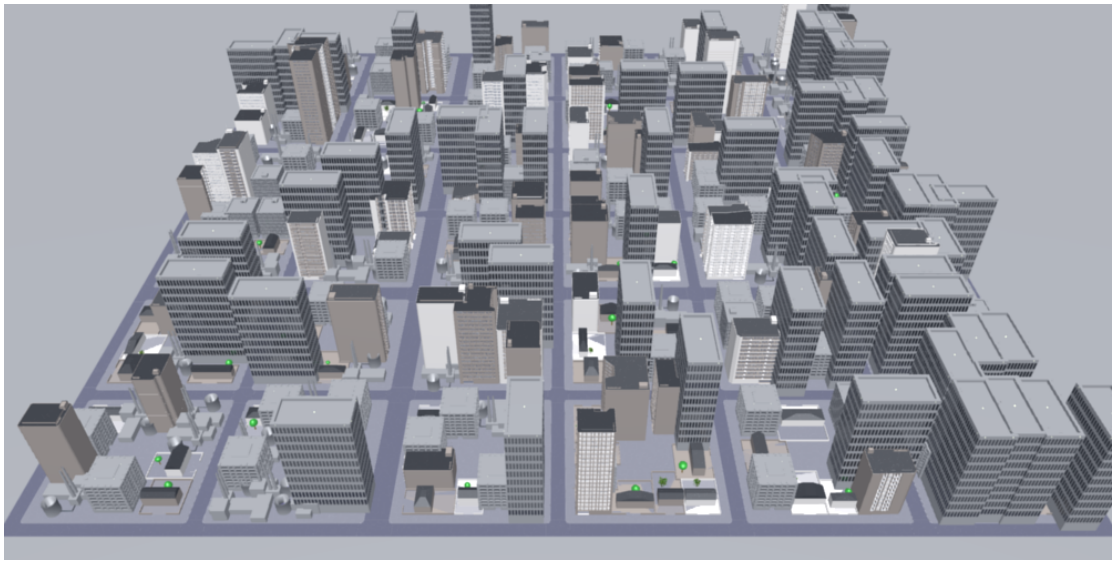
All white

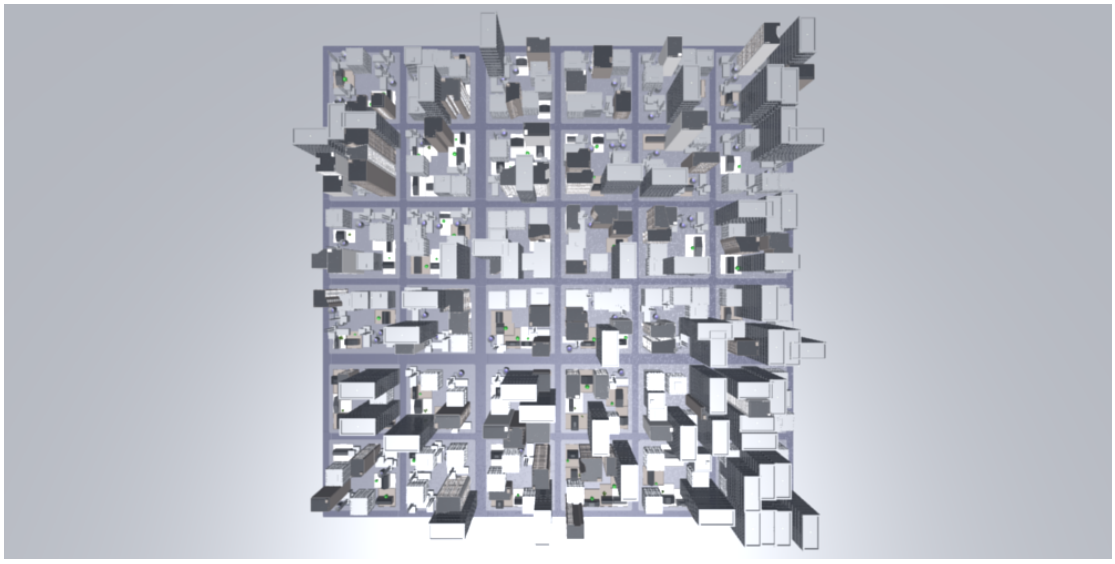




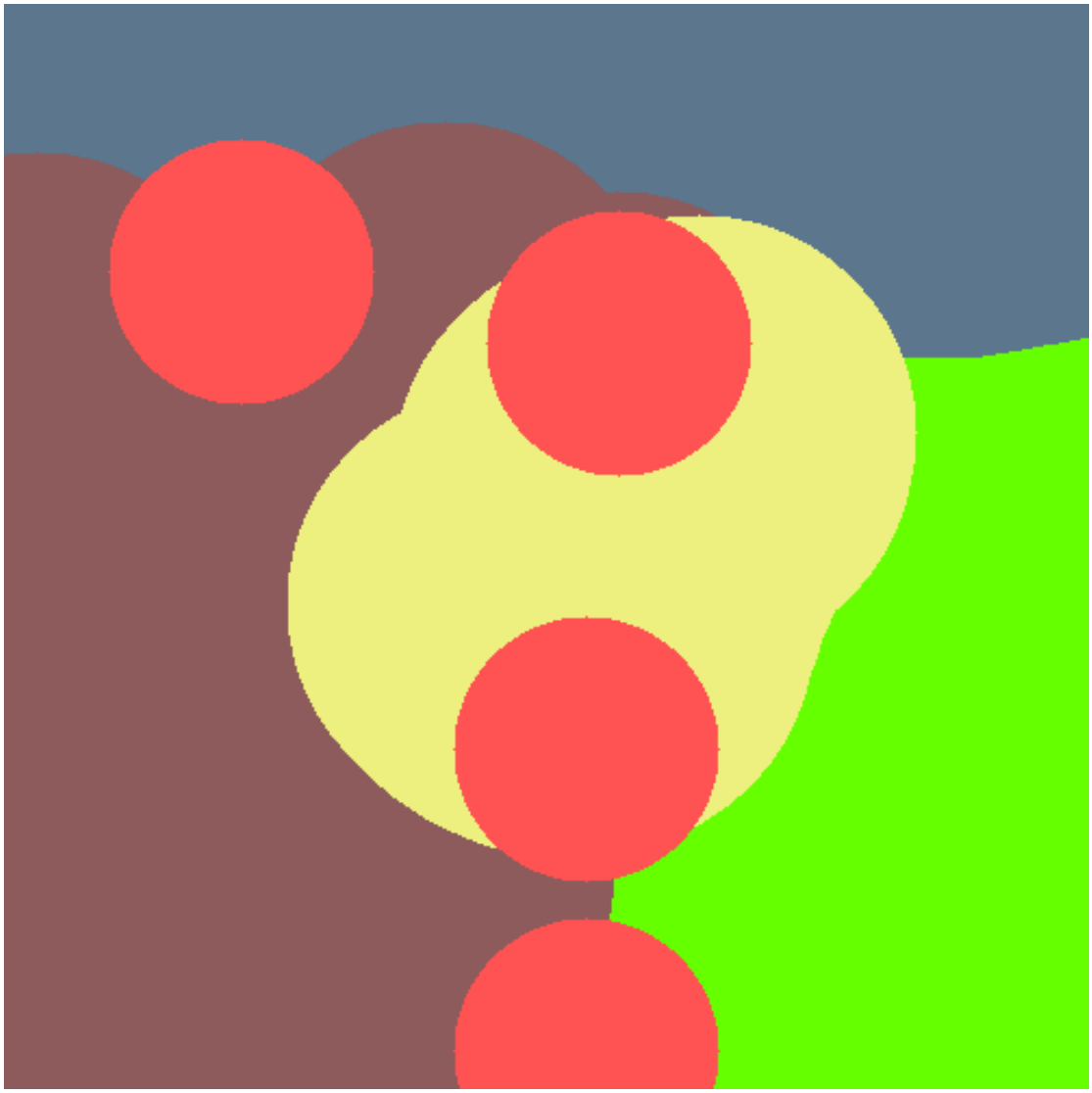


Colour Mix 1

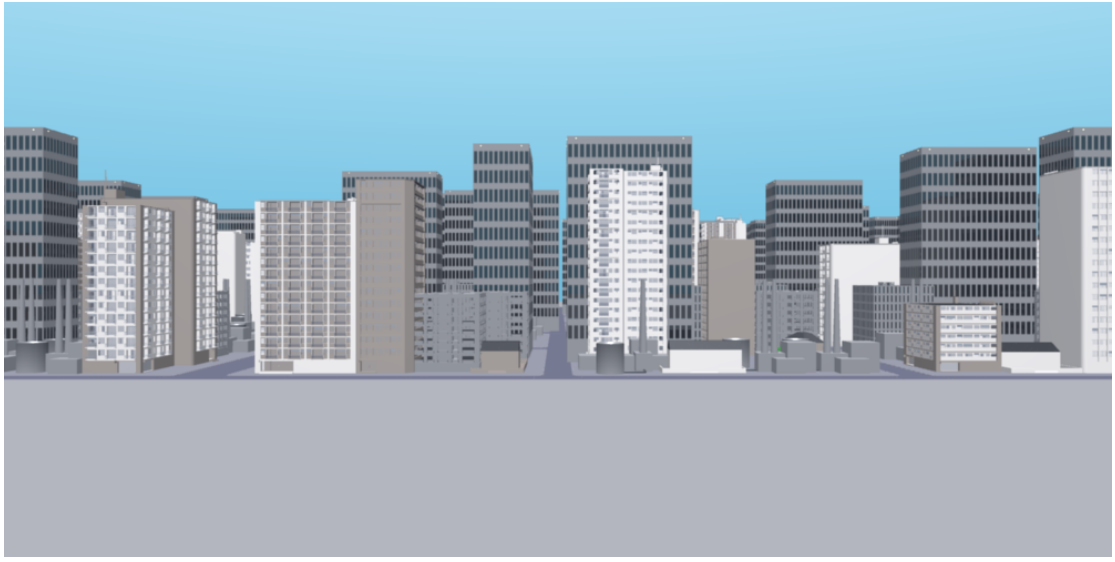


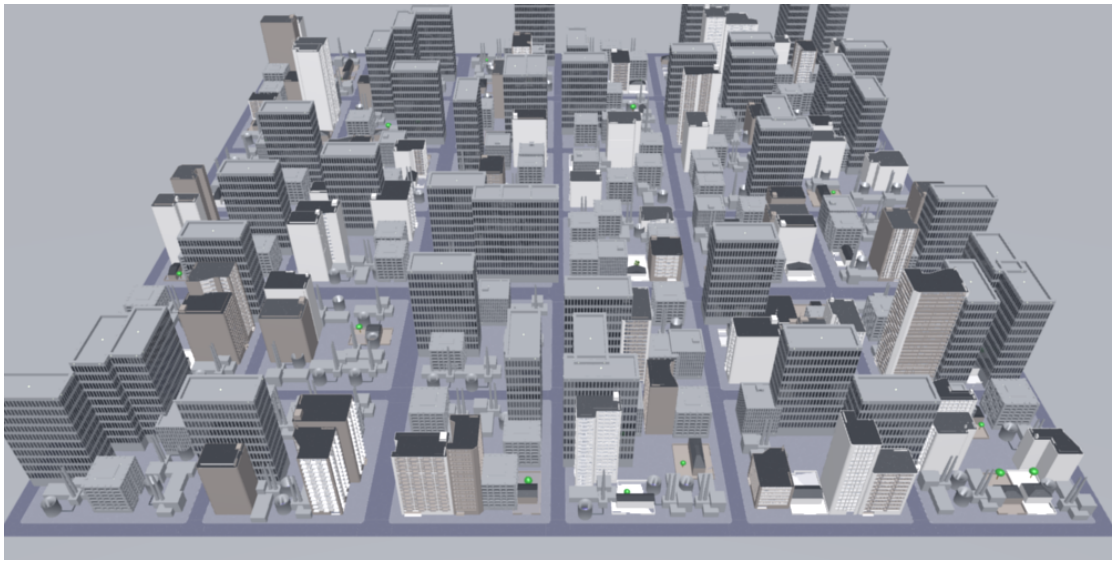


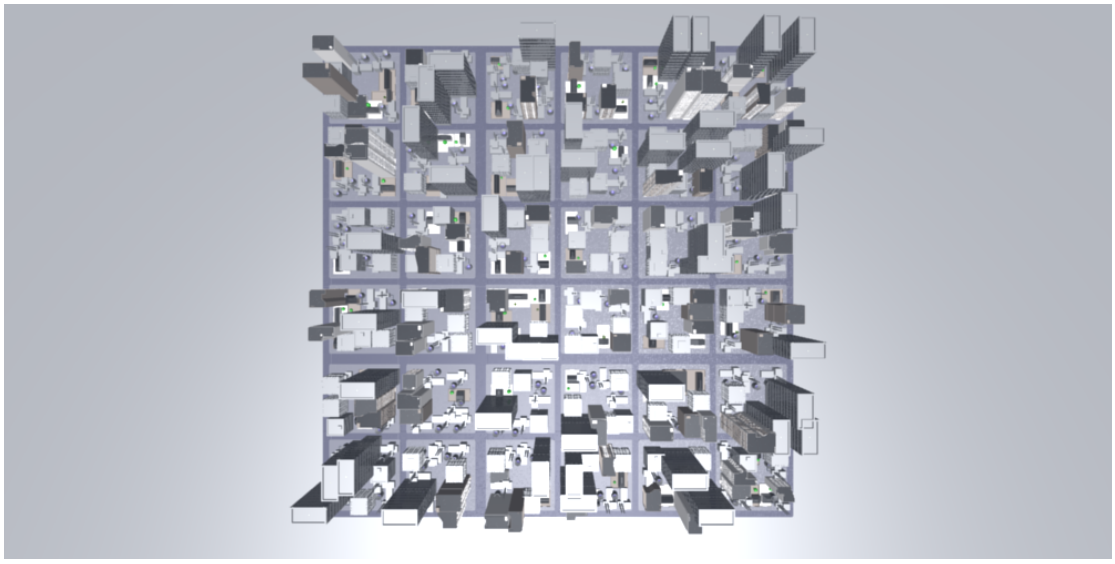
Zone map



Colour Mix 2

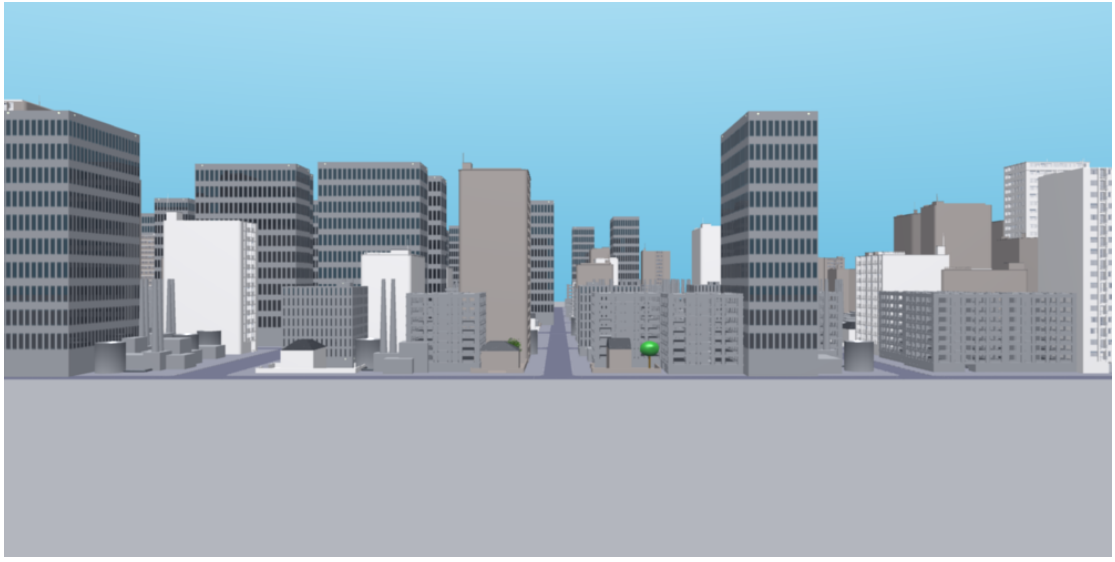


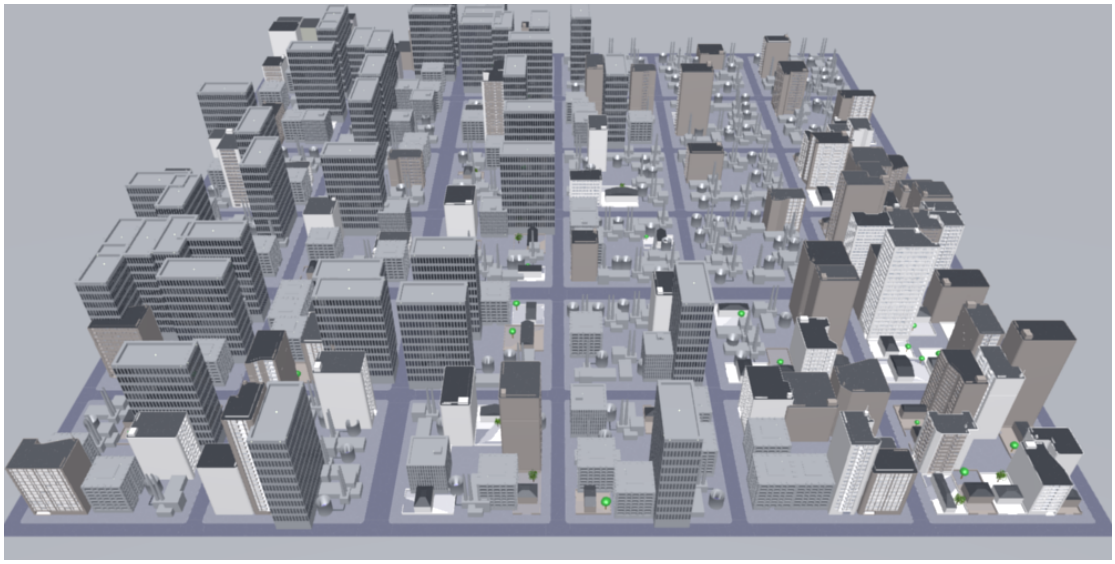


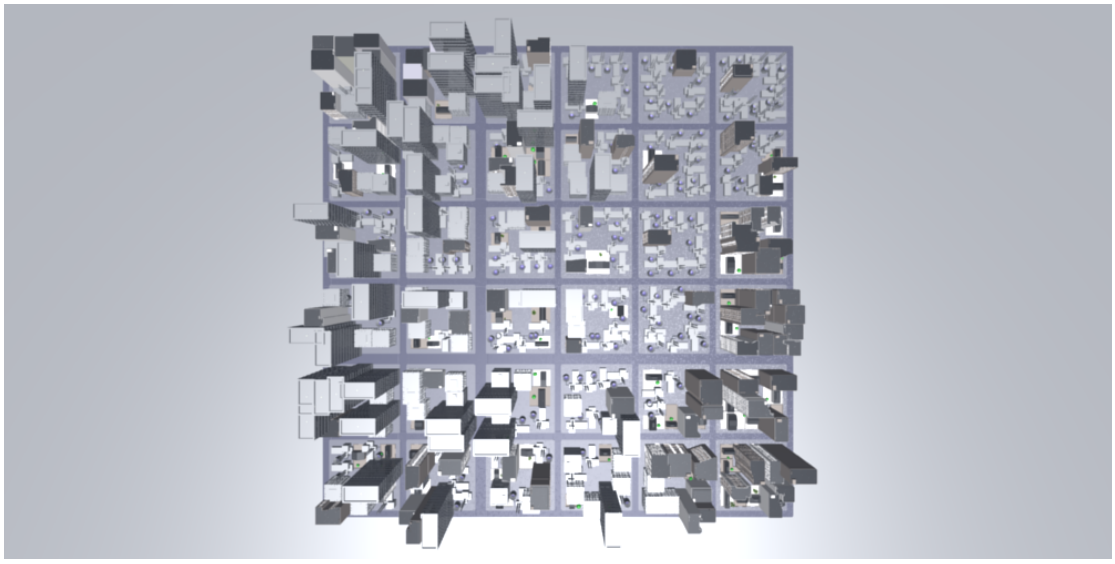


Zone map

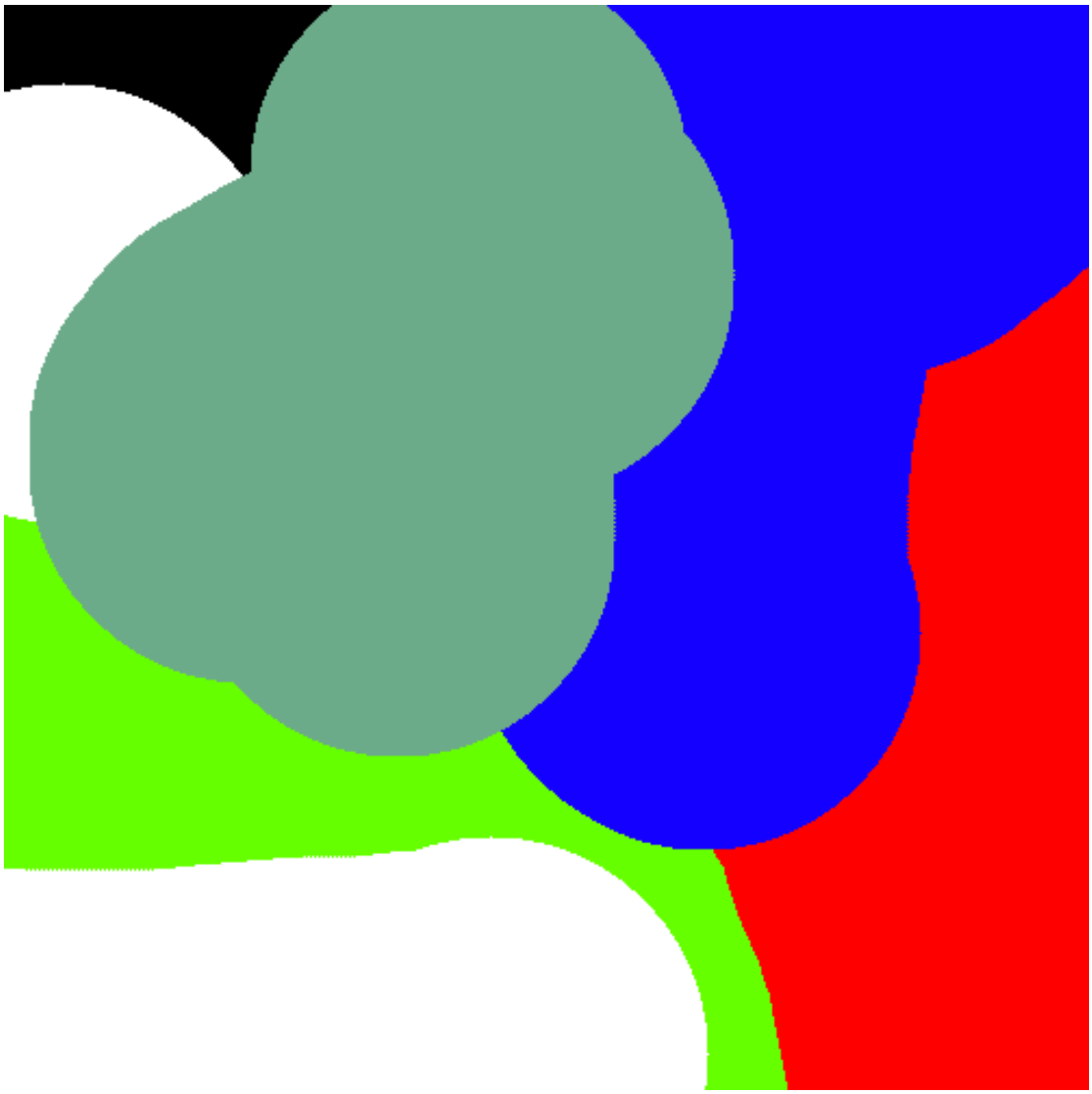
Colour Mix 3



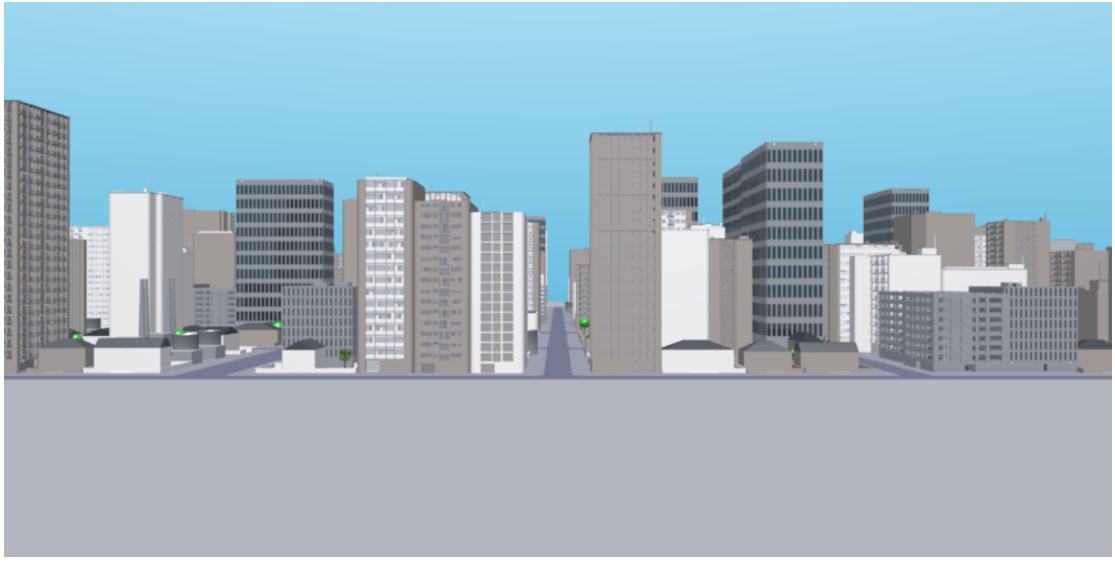


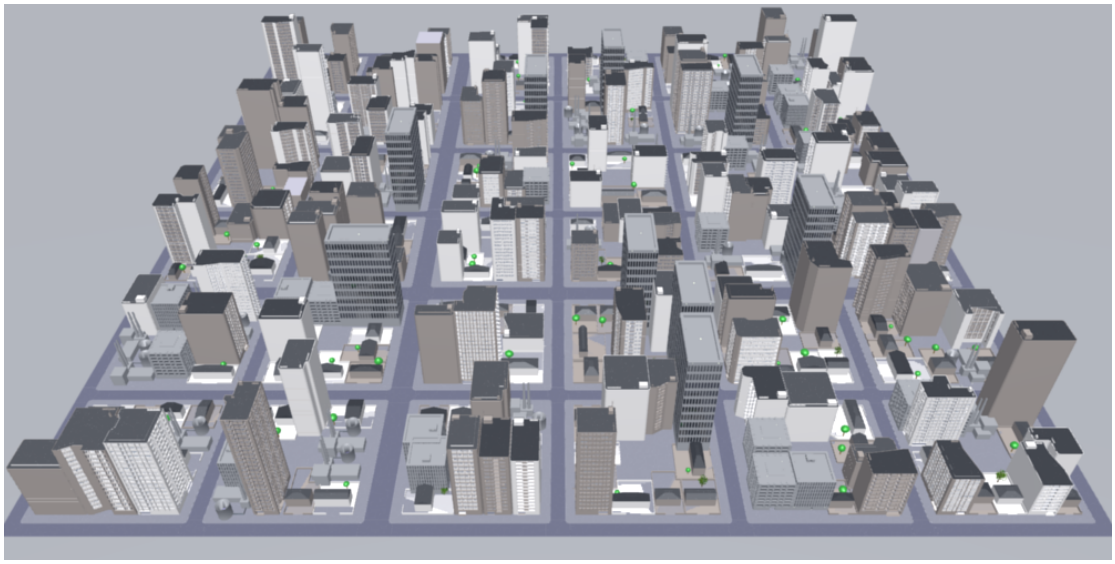


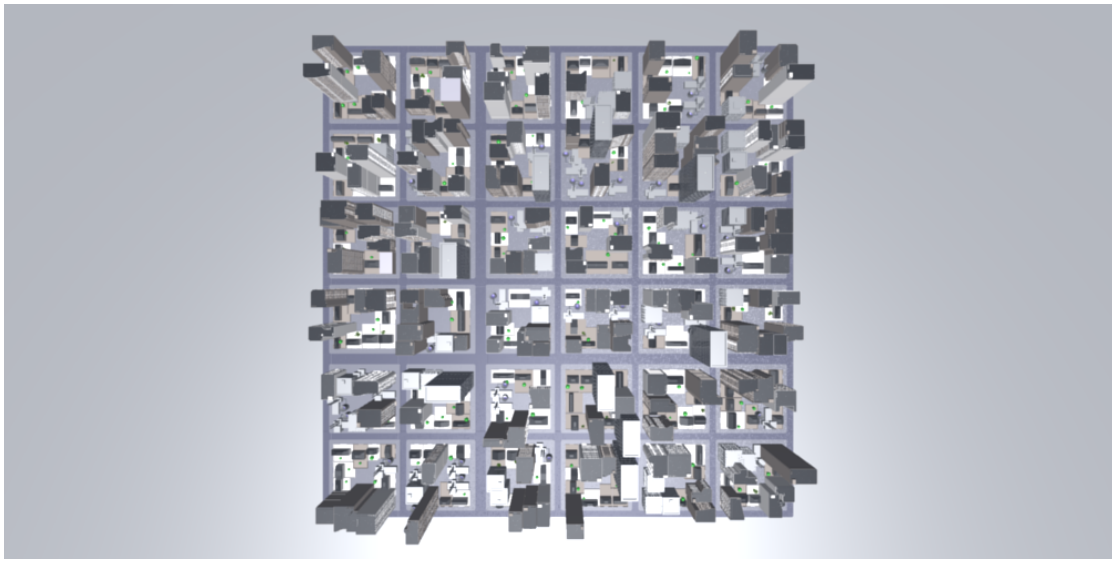
Zone map



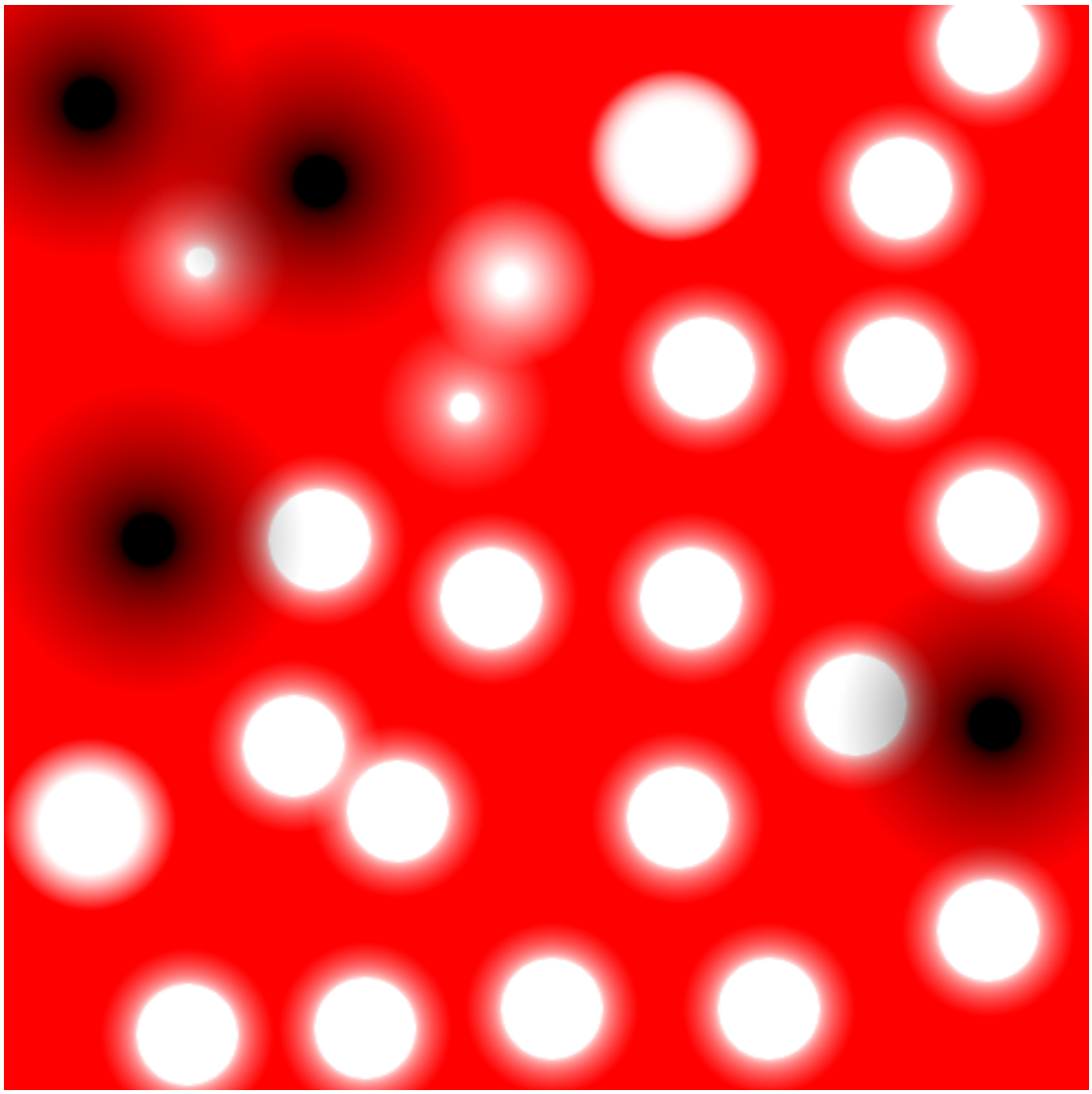
Colour Mix 4



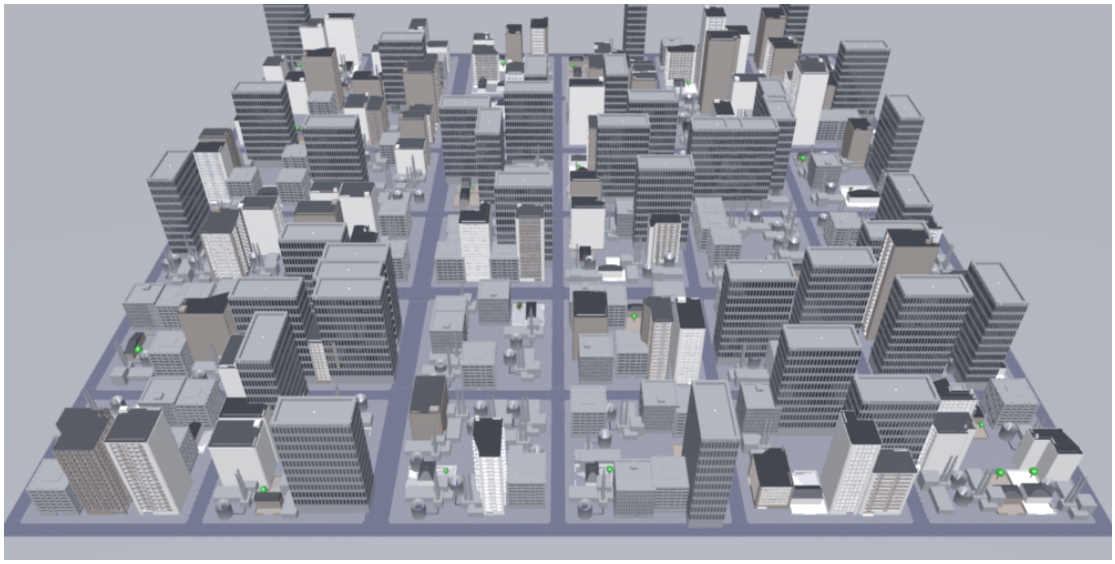


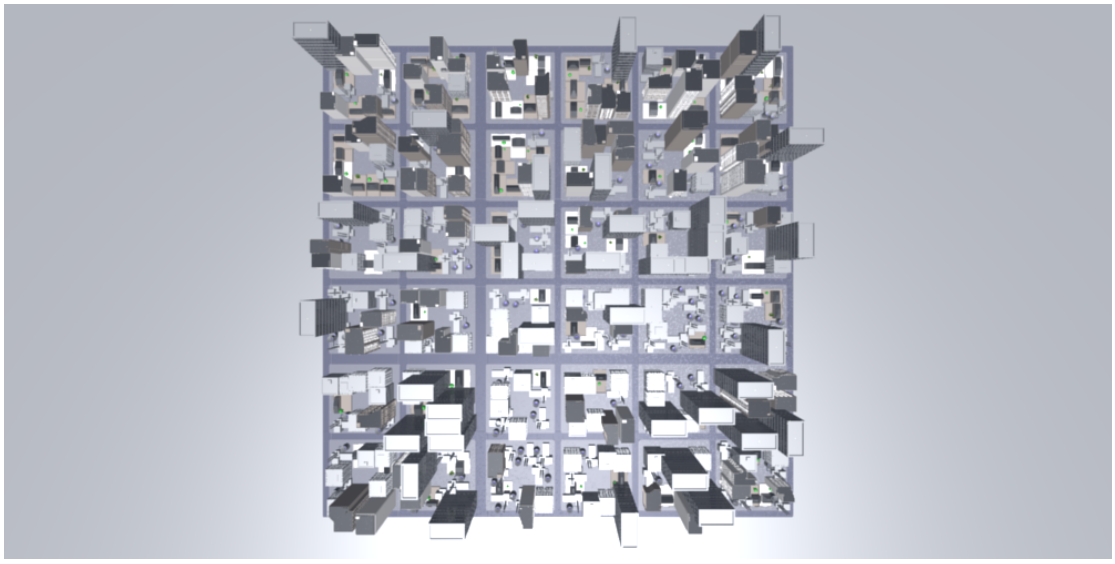


Zone map

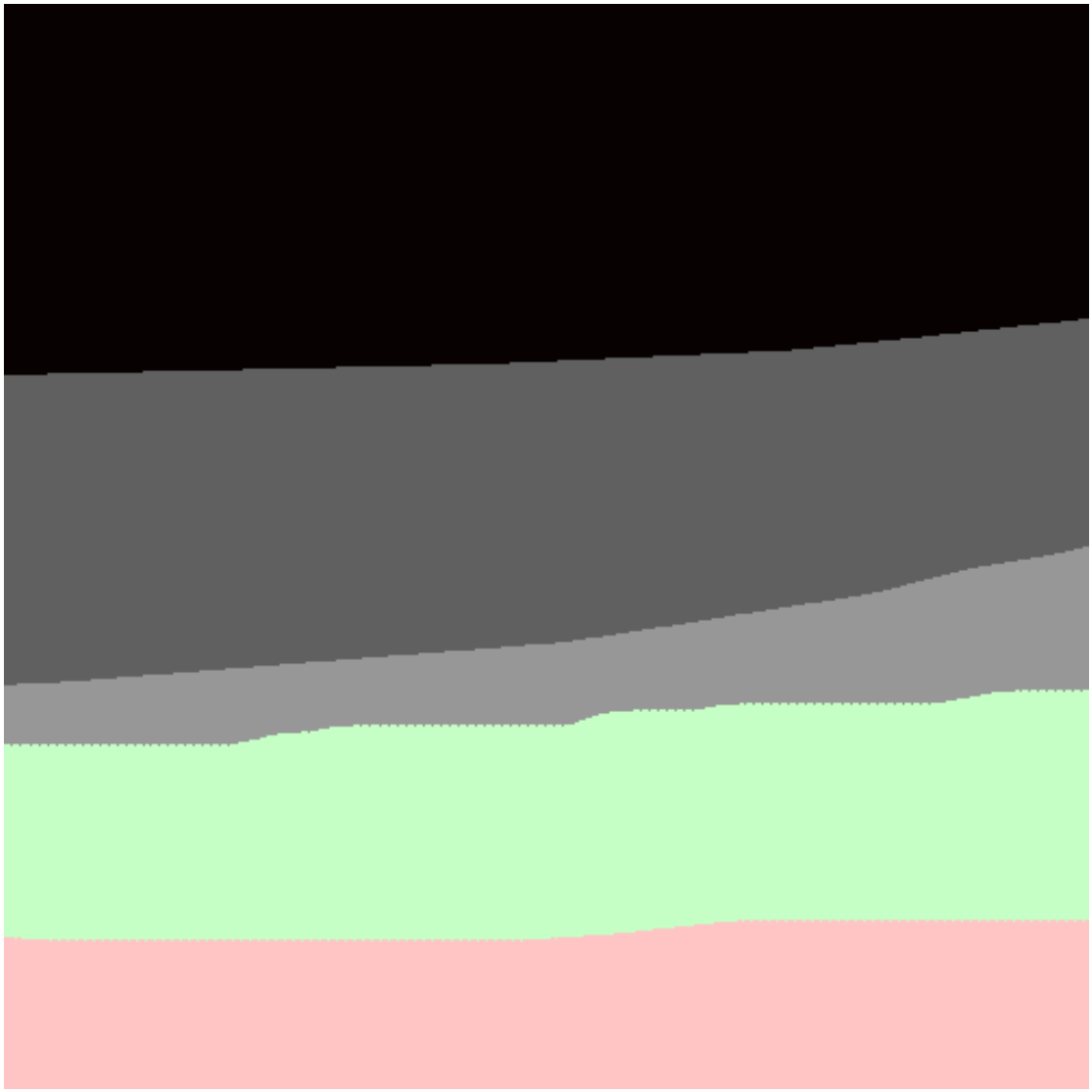


Colour Mix 5



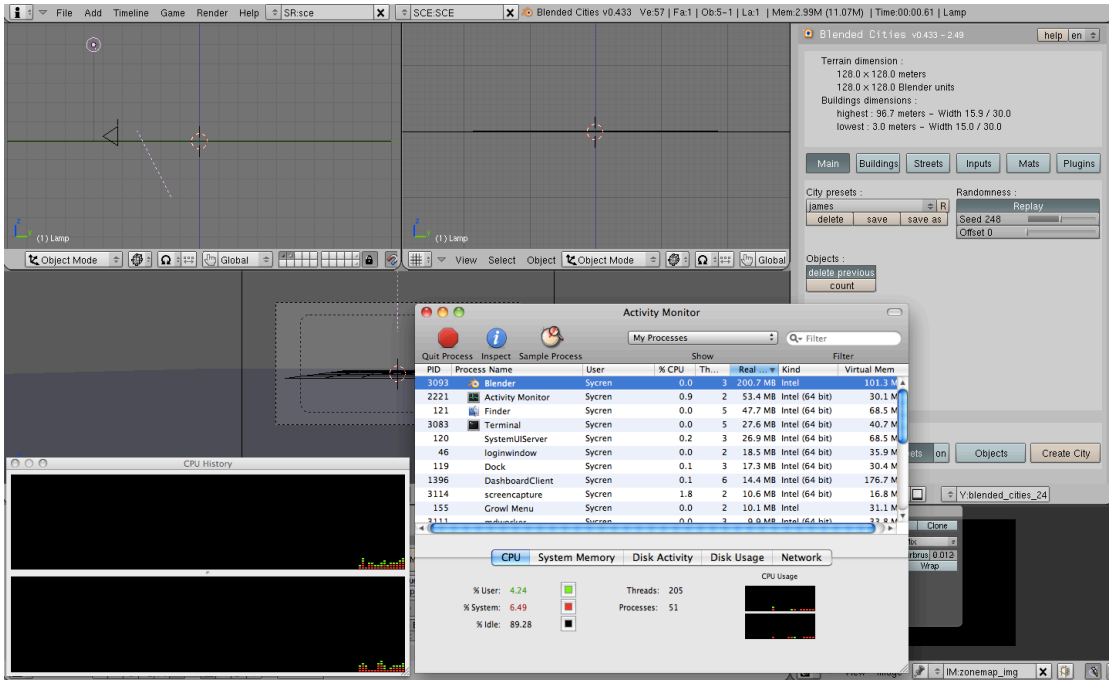


Zone map

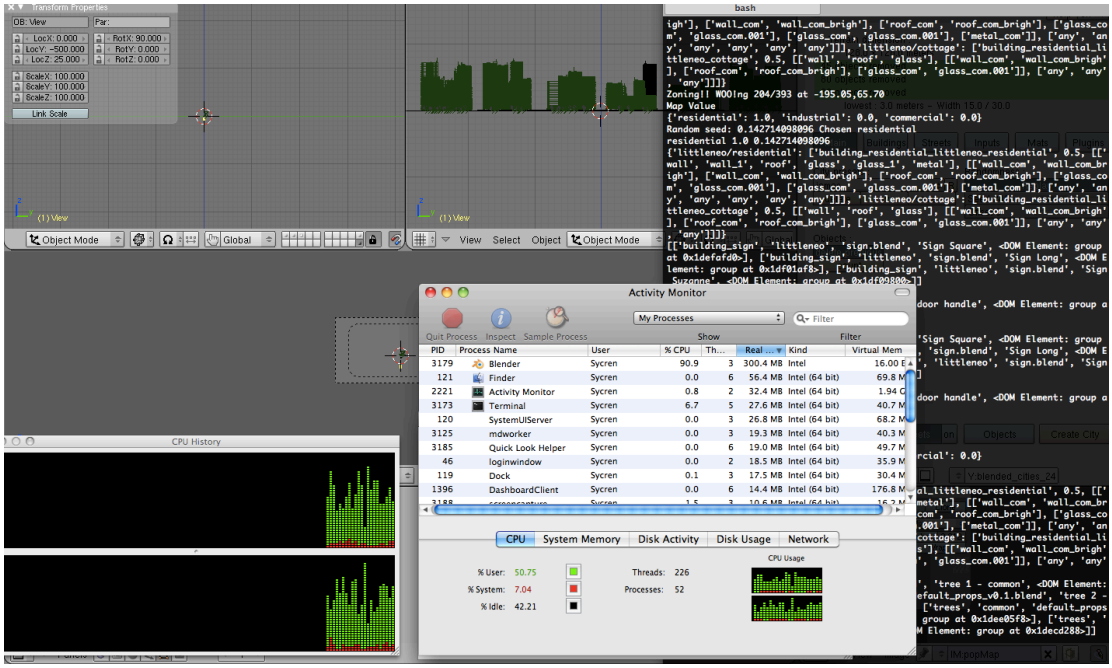


System Test

Base System



Blended Cities running



Interview: Igor Raffaele - Operations Director of Interwave studios

[2010-04-29 12:04:16] <Silverfell> Hello, James

[2010-04-29 12:04:39] <sycren> Im sorry if your really busy

Interviews and Correspondence

Igor Raffaele (Silverfell) - Operations Director of Interwave studios

[2010-04-29 12:04:46] <Silverfell> allow me to gather my thoughts for a few minutes :)

[2010-04-29 12:04:50] <sycren> thanks

[2010-04-29 12:05:28] <Silverfell> My main point is that procedural generation is more or less useful, in relation to the type of game you're making.

[2010-04-29 12:06:11] <Silverfell> I replied, as an example, with Nuclear Dawn, our own game, which is a multiplayer online shooter set in real-world cities, but after a devastating third world war.

[2010-04-29 12:06:38] <Silverfell> In such a context, procedural generation is of extremely limited use. It cannot be applied to the levels themselves for a number of reasons:

[2010-04-29 12:06:50] <Silverfell> 1) It cannot be used for main level blocking (since we're modelling on real cities)

[2010-04-29 12:07:36] <Silverfell> 2) It cannot be used to effectively create interactive objects such as cover or debris, since it would require an extreme amount of constraints - so much so that it would essentially be limited to a mechanism that only produces a few patterns that conform to the same requirements.

[2010-04-29 12:09:39] <Silverfell> This is mostly due to the fact that design-driven games, such as Nuclear Dawn, have to accomodate for an extremely high amount of player interaction and variation. Debris/cover, for example, would have to be both meaningful and allow for vehicle passage. It also needs to take into account the height and size of players, the nearby focal points (resource points for example).

[2010-04-29 12:11:02] <Silverfell> Finally, such placement would have to be FUN - which is not something you can leave to a procedural algorhythm. Each single prop in our game has been moved (often a few inches at a time) to be placed where they would best conduct specific gameplay situations (furious, fluid firefights, defense, last stand scenarios and so on)

[2010-04-29 12:12:10] <Silverfell> In such cases, procedural generation is too random, and not intelligent enough to make the most out of each situation. In order to be appealing, such procedural systems would have to be highly controllable, with a large amount of input variables. It is my opinion that once you're done inputting all the variables, you might as well have created a number of equally valid scenarios by hand.

[2010-04-29 12:12:49] <Silverfell> Finally, we talked about it for skybox generation, and I replied that likewise, a procedural skybox generator would have to give us a great amount of input in the parameters that it uses, so much so that we'd be approaching the previous scenario.

[2010-04-29 12:13:46] <sycren> For Point 1&2, I am interested in whether the artists created every building by hand. I have made a modification to an existing system to

allow city zoning, in a way that gives an artist greater control by allowing them to paint the different zones http://www.blendernation.com/city-zoning-modification-for-blended-cities-script/ I can see that it is very difficult to get a generator to give...

[2010-04-29 12:13:46] <sycren> ...good details of control, but perhaps they could be used in a similar way to my city zoning addition to allow the artist to paint the density of debris and for some kind of generator to choose premade models of debris based on the surrounding environment.

[2010-04-29 12:13:48] <Silverfell> In general, there is not a lot of space for procedural generation on large scale geometry also because of the lighting techniques used in gaming today. It's hard to bake an impressive lightmap when you don't know where your objects will be.

[2010-04-29 12:15:21] <Silverfell> The main issue is identity. With manual content generation, we can go in and inject an infinite amount of details into each scene. Procedural generation would mean sacrificing that kind of detail.

[2010-04-29 12:15:38] <Silverfell> It's all about the game, however. Imagine a driving shooter, for example

[2010-04-29 12:15:53] <Silverfell> One with a large number of cities that do not necessarily draw inspiration from real locations.

[2010-04-29 12:16:15] <Silverfell> For such a project, for example, a procedural approach, a highly customizable one, would be something we could consider.

[2010-04-29 12:17:48] <sycren> But I was also wondering about generation whilst developing the game rather than at runtime, so the generator may place objects and then the artists may move them. Be it prototyping...

[2010-04-29 12:18:49] <Silverfell> The main issue with that is efficiency. A skilled level designer can create all that, tailored to his needs and design, in 5-8 hours of work.

[2010-04-29 12:19:12] <Silverfell> The basic blockout for any of our maps (and they're pretty huge for shooters) took each of our level designers less than a day

[2010-04-29 12:19:38] <sycren> thats impressive, I would have thought that the levels would take a lot longer

[2010-04-29 12:19:59] <Silverfell> oh, they do. A level takes 4-6 months to complete.

[2010-04-29 12:20:10] <Silverfell> But that is endless iterations of detail, placement and optimization

[2010-04-29 12:20:13] <Silverfell> What you're proposing

[2010-04-29 12:20:26] <Silverfell> would only affect the prototyping which, in an active team takes no more than 2-3 days

[2010-04-29 12:20:52] <Silverfell> also remember that each of your buildings would have to be created to the same amount of detail

[2010-04-29 12:20:56] <Silverfell> and that's what takes time

[2010-04-29 12:21:36] <Silverfell> Like I said, I don't see many practical applications for first person shooters.

[2010-04-29 12:21:42] <Silverfell> You should look more into driving games

[2010-04-29 12:21:50] <Silverfell> where, for example, the player has less interaction with the blocks of buildings

[2010-04-29 12:22:03] <Silverfell> which are effectively just big obstacles to overcome

[2010-04-29 12:22:07] <sycren> very true

[2010-04-29 12:22:17] <Silverfell> or a city sim - designers could use your tools to quickly layout a huge city

[2010-04-29 12:22:30] <Silverfell> and then work backwards to put in the various zones, and working areas

[2010-04-29 12:22:39] <sycren> Chris Preston agrees with you, he also mentions one argument for it:

[2010-04-29 12:22:39] <sycren> In favour: Modern hardware is extremely good at doing the kind of work involved in procedural generation. If you want to make maximum use of a PS3, you really need to be using some procedural systems to prevent you being limited by the memory size - it can process a lot more data than it can hold in memory.

[2010-04-29 12:22:45] <sycren> Would you agree?

[2010-04-29 12:22:58] <Silverfell> Different scale.

[2010-04-29 12:23:07] <Silverfell> You're talking about procedurally generating the level

[2010-04-29 12:23:19] <Silverfell> there are a number or procedural systems in any game, and they do greatly help save resources.

[2010-04-29 12:23:35] <Silverfell> I agree completely, as long as we understand that you're not referring to procedural level generation.

[2010-04-29 12:24:05] <Silverfell> If you ever heard of Speedtree and similar products - plant generation in modern games is pretty much left entirely to procedural systems.

[2010-04-29 12:24:36] <Silverfell> Effects, even textures and materials, items and enemies in role playing games, random/queued environmental effects

[2010-04-29 12:24:42] <Silverfell> those are usually all procedural.

[2010-04-29 12:25:06] <Silverfell> In Nuclear Dawn, for example, we place blueprints procedurally, so that at each game, they're in different locations.

[2010-04-29 12:25:27] <Silverfell> Even that, however, is a guided procedural process, as we first define the areas of higher blueprint probability.

[2010-04-29 12:26:07] <sycren> blueprints being a pick up in the game

[2010-04-29 12:26:13] <sycren> ?

[2010-04-29 12:27:06] <Silverfell> yes

[2010-04-29 12:29:40] <sycren> Ok thanks for your time, last question.

[2010-04-29 12:29:40] <sycren> Do you see a future for this technology? What features would need to be prevalent in the software for your team to even consider using them?

[2010-04-29 12:29:52] <sycren> How would you like me to reference you?

[2010-04-29 12:34:03] <Silverfell> I already see applications for your technology today. In the right kind of game, it would be an invaluable prototyping tool. In order to be commercially appealing, you would have to provide a great amount of control in the procedural generation (such as the definition of weight maps, zone definition, cutoff points and several other parameters).

[2010-04-29 12:35:09] <Silverfell> I think that procedural generation will be much more appealing once modern engines move to a true real time lighting model. Once that happens, it will be feasible to generate levels that strike a balance between variations and playability, but that maintain the highest standard of graphical appeal.

Interview: Steve Rotenberg – CEO of PixelActive, CityScape

What kind of clients use your city generator software? How do they use it? (games, films..)

- Originally, our product was designed for video game development. Currently, the industry we do most business in is actually mapping/navigation, with military/simulation/training coming second, games third, and urban/civil planning last. Some of our specific clients are listed on our web page (http://pixelactive3d.com/Clients/), and we're not allowed to list some of the government & military customers.

Do you see a future for this area of development?

- Absolutely. Especially in the urban planning/management, civil engineering, and transportation engineering fields.

Any particular future plans?

- Bigger & bigger worlds. We're working on supporting Earth-sized models.

Would you ever think about including AI in future releases?

- We already generate pretty extensive AI information including a full traffic network, pedestrian navigation data, and navigation meshes. Check out our traffic mode for a demo.

One more thought: on the subject of 'procedural' city generation... our customers fall into two categories: ones that want to interactively design their own custom city, and those that want to build the real world as it exists today. In the 7 years we've been in business, no one has ever asked us for a 'random' city generation tool, 'rule-based' generation, or anything related to that.

I was wondering if you could tell me how you started developing your product? Whether you did any research and took a different direction from the various research papers out there,
(Most notable: Pascal Müller - Procedural Modelling of Cities, http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/PaperCities)

In the 1990's, I worked on several large city-based video games (Midtown Madness 1 & 2, Midnight Club 1, 2, & 3), and saw tons of money poured into large art teams building cities over and over. I felt that most of the labor

could be automated into a tool, and that's what motivated CityScape. I felt that the money would be better spent building a re-usable toolkit that allows rapid city modeling and rapid iteration, rather than thrown away over and over on manual labor.

We keep up with all current research in city modeling and have followed Pascal's work for many years. I would say that our primary motivation has been to build a tool that allows you to interactively and explicitly model a city exactly as you want it, rather than a tool that builds a 'random', or 'rule-based' city. This has caused us to focus more on fast interactivity, powerful user interface, reliable undo/redo capability, handling of very large environments, support for multiple simultaneous users, and other key features that our customers want. As I mentioned earlier, nobody has ever asked us for something to generate a random or rule-based city, as neat as that stuff might be. Generally, our customers don't want to write scripts or use grammars to generate cities. They want to point and click with a mouse.

If possible could you tell me which games have used your software?

Probably the best ones to list are:

Saints Row III (THQ - Volition)
True Crime: Hong Kong (Activision - United Front)

Note that neither of these has been released yet, but both are announced and public knowledge. It's possible that the True Crime game may have a slightly different title, as I don't think the title is finalized yet.

Chris Preston

From: Chris Preston

Sent: 10 March 2010 14:31

To: James Lethem

Subject: RE: student: procedural generation, placements

Hi James,

I can't really tell you about any research Ubisoft have done into procedural generation, except the things that have been made public (like the fire and trees in Far Cry 2). Various teams have looked at procedural generation of just about anything you can think of.

Personally, I think there are two main arguments against it and one main argument in favour.

Against 1: It's extremely hard to procedurally generate the kind of subtlety, detail and uniqueness that a human designer can put into an object. Whether that's gameplay nuances, visual detail or whatever. Not impossible, just very hard, and there aren't many examples of it having been done really well. It's the opposite of randomness or natural effects - objects in our world are specifically and carefully designed by people to deliver on a given purpose. For example, so-called 'new towns' were effectively procedurally generated - a strict pattern of development for their estates was laid out in central guidelines. Result - soulless places that only became nice as successive overseers made their personal marks.

Against 2: It's completely different from how most developers are used to working and thinking, and it's an area of sufficient complexity (if you want great results) that it's very hard for teams to make the jump. They set a certain quality bar with their manual processes and it's hard to guarantee they won't go backwards if they change to procedural generation.

In favour: Modern hardware is extremely good at doing the kind of work involved in procedural generation. If you want to make maximum use of a PS3, you really need to be using some procedural systems to prevent you being limited by the memory size - it can process a lot more data than it can hold in memory.

They're my views on why it has or has not found widespread adoption. I think it will become massive, it just hasn't yet.

As for placements or jobs, send a CV and a covering letter explaining the kind of role you'd be interested in and I'll forward on to the right people.

Cheers,

Chris.

-----Original Message-----

From: James Lethem [mailto:j.d.lethem@newcastle.ac.uk]

Sent: Sunday, February 21, 2010 8:35 PM

To: Chris Preston

Subject: student: procedural generation, placements

Hi Chris,

Thank you for giving the presentation on Thursday to us at Newcastle. I was wondering if you could perhaps tell me more about how you looked at procedural generation of cities that you mentioned after the presentation. Would be at all possible to see the research you made so that I could link it in to my dissertation?

When I questioned you cars being procedurally generated, you mentioned that procedural generation is only just being used more in game development. Is this due to the lack of imagination and artist input into designs that can fascinate the player?

Would you perhaps have any placements, research projects or jobs that I could go for after I graduate this year?

Kind regards,

James Lethem

Jérôme Mahieux

> If possible could you include a short bio on who you are and your motivation for blended cities?

Between two jobs as a network consultant, I discovered Blender by curiosity about 3d

modelling and animation. 4 years later, it becomes a passion and almost a job today. as I was experimented in programming and as Blender supports Python, I begun to learn this language, first to answer some special animation needs.

I looked several time these last years for an open source 3d tool able to build a big city according to some global properties. Ideally I was looking for a good Blender script since it's the 3d suite I'm the more experimented in. unfortunately I didn't find anything that matches my needs.
then Arnaud Couturier (Piiichan) published Suicidator City Engine 0.1 (sce), that was able to produce realistic background cities very quickly.

I had begun to modify his script in an informal way, adding some features here and here. This work started as a simple hack to answer my own need for another project. The first aim was to add streets and roads in it, the other first aim what to learn more about python…
As Arnaud had no time to develop it further during the year, and as the code and my thoughts go on alone, it becomes little by little a real personal project, and after almost one year, it became blended cities.

My motivations for this script are very various: create cities for games, architectural rendering or animation, traffic simulation… The main idea is to allow an artist/game developer/architect to focus on his main project, by designing relatively quickly a world around it that can answer most of his needs.

> Blended Cities:
>
> When you started developing this script, where did you start?

as I said it start as a sce modification (fork ?) : I added functions to produce streets and sidewalks starting from a simple image. But there was major limitations with images: it was hard to interpret curved streets from pixel curve, and limited to interpret road widths (and especially hard to interpret width of curved streets..). Also the generated buildings were always aligned to XY world coordinates and shaped rectangular.
So I begun to turn everything as vectors: I worked on a different building lot algorithm, and though the streets as a network, made of intersections and lines.
I also started very soon to design a modular concept, that would allow people to add their own building to the generator, or to create a new data source, or to append a third party code that can use the inner variables for specific need. I think modularity is primordial for that kind of open source big project in order it can live by its own.

> Any particular Inspirations?

Unfortunately I didn't have the occasion to test another city generator except sim city 3000 ;) and soon cityXL, whom game renders looks nice and realistic.
I like the intuitive, 3d way the objects are appended and edited in cityscape as shown in their video on the website. as far as I know the most impressive and complete city generator application I know is the Pascal Mueller city engine (the Procedural City Engine) especially, among numerous other thing, for his street and building algorithmic generation and the procedural language it uses.

> Where did you have the most complicated problems?

I think it was the way to build irregular crossings made of roads with different widths, and

the fight is still not over.
the building lot builder gives me also some severe headaches.

> As a summary, how does the script step through the code?
> eg) Does it check all the user inputs, make the roads, find the lots, choose libraries etc

(I think we covered this part. if you want you can send me your notes and I can check again)

> What is the structure of the script?
> eg) How are things modularised, data structures etc?

Data structures *tend to become* a coherent system, in order to meet the requirements for data exchanges. Python dictionaries and XML will be the ways.
as the project started as a fork of a little script, data structure was not an identified need, but is now mandatory.

The code is split for different reasons:

the same core code should be used in both blender 2.4 and 2.5. This to permit a smooth transition between them and maintaining them accordingly, as to my point of view blender 2.5 stable and complete versions won't appear before September and 2.49b will still be used a lot next year. as the blender function calls and the gui scheme differ I created to directories (/bin/24 and /bin/25) to group any blender python specific functions.

To allow users to add their own city objects, procedural building codes and premade meshes are apart from the core code, grouped by author or themes, and called through a unified library system.

A lot of procedural mesh creation function has been written is this script, and also a lot of common, multi-purpose functions, both can be reused in some other script; these functions are apart, in two different files.

The script is translatable, so there is different language files apart from the core.

I also would like to split the lot builder algorithm and the sources parsers and interpreters from the core, in order one can use other ones easily:
ideally the core script would be an empty shell, only responsible for the gui's in/out, the city data structure file operation, and creation process management, and would ask building algorithms, parsers, parametric objects to different modules: call this parser and interpret that street data source, retrieve streets data, build streets using this library, etc...
I think it is a good way to ensure the script evolution on a long-term perspective.

> What would you like the script be in the future?

a real community project, with a huge building and urban object library, able to build any kind of cities on undulating terrain, with bridges, railways and tunnels.

> Would you like it to be ported to Blender 2.5?

not only ported but taking advantages of the new 2.5 features. I think that 3d city edition as

in cityscape and micro edition would be easier to develop in 2.5, as the python script integration is 'by default' in 2.5, on the whole gui.

> If so, will the script need to be rewritten as to how it works?

I hope that the modular concept will help the 2.5 port.

> Do you think City Zoning helps users?

Absolutely, without zoning, building choice decision is based on the building lot properties: area size and height of the lot determine which kind of building to use. These lots attributes don't depend on their geographic position in town but essentially on the building elevation map, the block sizes, and the main lot properties defined by the user, so a kind of zoning can be made by playing with the building elevation map, but it's not the better way since elevation and zoning are two different city properties. for example one can have the flat and long shop of a car reseller and the 8 floors building of a big shop that both should grown in the 'commercial' areas of the city, but can't with the non-zoning way.
in order the zoning integration to be completed, the different zone must influence the building lot attributes : in other words, the used building library should be known before to create the lot. Nevertheless, using parametric/procedural buildings or flexible enough premade buildings it does the job already.

Robert Shinka

_____

From: Robert Shinka [grue@unknownlayer.net]

Sent: 17 April 2010 13:04

To: James Lethem

Subject: A cute little rant on L-systems

Firstly, since this digression is intended to address the suitability of L-systems for building cities, it should be noted that I have no professional experience at automated city generation: rather, my experience is limited to modeling of biological systems. However, the processes behind the two aren't particularly different.

The current trend for L-systems use is as a form of fractal generation, which (although similar to Lindenmayer's original usage) does not accurately model the "evolutionary" processes involved in an organisms life cycle. In the case of an algae, as new cells are generated, they're primarily an addition to the previous state, until the prior generations have matured sufficiently and begin to die off. The population decline is usually factored into the rules for growth rather than explicitly including decay, keeping the systems simple.

Deterministic context-free L-systems are useful for a simplistic modeling of, for example, the shape of a plant, or a pattern of roads within a city, but the other attributes (such as the thickness of a branch or a cells walls) are usually left to another system to describe. When coupled with the observable repeating patterns typically present in fractals, this separation of attribute generation causes the model to appear inconsistent and usually very unrealistic.

In a city, there is usually a discernible growth pattern where traffic on roads, the size of roads, the space reserved for parking, the number of residents, and the amount of, types of, and proximity of business are all very closely interrelated. An otherwise well-generated city may appear unrealistic if the building density is too sparse along the most easily-traveled intersections. Similarly, the vessels in a multi-cellular organism must agree with the needs of the surrounding tissue, roads leading to nowhere are seldom built, and leaves on a severed branch will quickly wither and die. Accurate modeling of problems such as these require a feedback loop that context-free systems lack.

Although they are typically more complicated, none of these problems need be present in context-sensitive L-systems that consider (and possible alter) prior generations, and the increase in complexity is made up for by ability of these systems to cleanly operate on a large number of state variables. Given enough rules, each of which can be expressed in an independent constraints-logic clause, you can run elaborate simulations on a set of values (such as a city) until it evolves to an acceptable level. In order to avoid an obvious repetitions of patterns, these systems can alter the existing dataset by transforming individual nodes (or groups of nodes), then use the resulting state to source dependent variables for the next iteration.

Another advantage of these systems is that they can be easily implemented via any preexisting graph-rewriting system simply by repeatedly applying the entire set of rules to the graph, which also happens to lend itself extremely well to functional approaches. A context-free system can achieve greater performance by keeping separate track of edge nodes, but doing so actually requires an overall more complex implementation, despite the lesser expressivity of the production rules.

The only significant disadvantage to the approach is the increase in resources needed to perform each iteration: in addition to generating the next generation of neighboring nodes, each previous generation of nodes must also be transformed. The cost is more than made up for by the increased realism granted by a quality implementation, particularly in the case of static content generation where there is no need to conserve cpu resources.